CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Modelagem Matemática e Computacional

GVNS e GRASP com *Path-Relinking* aplicado à solução do Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução

Dissertação de Mestrado, apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno : Warley Ribeiro de Freitas

Orientador : Prof Dr. Sérgio Ricardo de Souza

Co-Orientadora : Profa. Dra. Elisangela Martins de Sá

Co-Orientadora : Profa. Dra. Maria Amélia Lopes Silva

Belo Horizonte - MG
2026

Freitas, Warley Ribeiro de
F866g GVNS e GRASP com Path-relinking aplicado à solução do problema de sequenciamento em projetos com restrição em recursos e múltiplos modos de execução / Warley Ribeiro de Freitas. – 2026.
78 f.

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Sérgio Ricardo de Souza.
Coorientadora: Elisangela Martins de Sá.
Coorientadora: Maria Amélia Lopes Silva.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Algoritmos – Teses. 2. Estruturas de dados (Ciência da computação) – Teses. 3. Otimização combinatória – Teses. 4. Otimização matemática – Teses. 5. Meta-heurísticas – Teses. 6. Matemática discreta em Ciência da computação – Teses. I. Souza, Sérgio Ricardo de. II. Sá, Elisangela Martins de. III. Silva, Maria Amélia Lopes. IV. Centro Federal de Educação Tecnológica de Minas Gerais. V. Título.

CDD 519.6

Elaboração da ficha catalográfica pela bibliotecária Jane Marangon Duarte, CRB 6ª 1592 / Cefet/MG

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO DE MESTRADO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

"GVNS E GRASP COM PATH-RELINKING APLICADO À SOLUÇÃO DO PROBLEMA DE SEQUENCIAMENTO EM PROJETOS COM RESTRIÇÃO EM RECURSOS E MÚLTIPLOS MODOS DE EXECUÇÃO"

Dissertação de Mestrado apresentada por **Warley Ribeiro de Freitas**, em 20 de fevereiro de 2026, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Documento assinado digitalmente
gov.br SERGIO RICARDO DE SOUZA
Data: 27/03/2026 17:19:31-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Sérgio Ricardo de Souza (Orientador)
Centro Federal de Educação Tecnológica de Minas Gerais

Documento assinado digitalmente
gov.br ELISANGELA MARTINS DE SA
Data: 27/03/2026 18:08:45-0300
Verifique em https://validar.iti.gov.br

Profª. Drª. Elisangela Martins de Sá (Coorientadora)
Centro Federal de Educação Tecnológica de Minas Gerais

Documento assinado digitalmente
gov.br MARIA AMELIA LOPES SILVA
Data: 23/03/2026 07:35:50-0300
Verifique em https://validar.iti.gov.br

Profª. Drª. Maria Amélia Lopes Silva (Coorientadora)
Universidade Federal de Viçosa

Prof. Dr. Gustavo Alves Fernandes
Centro Universitário Una

Documento assinado digitalmente
gov.br GUSTAVO ALVES FERNANDES
Data: 12/03/2026 11:34:22-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Documento assinado digitalmente
gov.br MARCONE JAMILSON FREITAS SOUZA
Data: 16/03/2026 09:43:58-0300
Verifique em https://validar.iti.gov.br

Visto e permitida a impressão,

Prof. Dr. Alisson Marques da Silva
Coordenador do Programa de Pós-Graduação em
Modelagem Matemática e Computacional

Agradecimentos

Agradeço aos meus familiares pelo apoio incondicional, compreensão e incentivo nos momentos desafiadores. Aos meus orientadores pelas valiosas contribuições que foram essenciais para o desenvolvimento deste trabalho. Aos colegas e amigos, que diversas formas contribuíram com palavras de incentivo e trocas de ideias.

À instituição e aos professores, pelo conhecimento transmitido e pela oportunidade de crescimento acadêmico e pessoal.

À divisão de Apoio ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa para a realização da pesquisa.

Às agências CAPES, FAPEMIG e CNPq pelo apoio financeiro concedido ao longo do mestrado, fundamental para a continuidade desta pesquisa.

Por fim, a todos que, direta e indiretamente, contribuíram para a realização deste trabalho, o meu sincero agradecimento.

Resumo

Esta dissertação aborda a resolução do *Multi-mode Resource-Constrained Project Scheduling Problem* (MRCPSP), um problema que consiste em sequenciar e programar todas as atividades de um projeto, relacionadas por restrições de precedência e de recursos limitados. Cada atividade pode ser executada em diferentes modos, sendo que cada modo apresenta duração e consumo específicos de recursos renováveis e não renováveis, o que resulta em diferentes combinações viáveis possíveis para a execução do projeto. O objetivo do MRCPSP é minimizar o *makespan* total, determinando o período de término e o modo de execução para cada atividade. Este é um problema NP-difícil, portanto, a obtenção de soluções ótimas para instâncias de médio e grande por métodos exatos em tempo computacional razoável torna-se inviável. Nesse contexto, esta dissertação propõe a implementação das metaheurísticas *Variable neighborhood search* (VNS) e *Greedy Randomized Adaptive Search Procedure* (GRASP) para produzir soluções de alta qualidade de forma rápida. Adicionalmente, integra-se ao GRASP o procedimento de *Path Relinking* (PR) como estapa pós-otimização, com o objetivo de explorar regiões promissoras do espaço de busca. Duas variantes do PR são consideradas: o *Interior* PR (IPR), voltado à intensificação da busca ao longo do caminho que conecta soluções de alta qualidade, e o *Exterior* PR (EPR), direcionado à diversificação da busca por meio da exploração de regiões adjacentes a esse caminho. Na metodologia adotada a fase de construção do algoritmo GRASP é responsável por gerar soluções iniciais diversificadas, as quais são refinadas por um procedimento de busca local. Em seguida os procedimentos de PR são acionados, como estratégias de pós-otimização, para fazer um balanço entre diversificação e intensificação sobre as melhores soluções obtidas ao longo da busca. As análises das diferentes estratégias de busca e de pós-otimização indicam que as abordagens propostas apresentam desempenho competitivo ao da literatura, evidenciando o potencial dessas metodologias para a resolução do MRCPSP.

Palavras-chaves: Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução, *Greedy Randomized Adaptive Search Procedure*, *Path Relinking*, *Variable Neighborhood Search*.

Abstract

This dissertation addresses the resolution of the Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP), a problem that consists of sequencing and scheduling all project activities, related by precedence and limited resource constraints. Each activity can be executed in different modes, with each mode presenting specific duration and consumption of renewable and non-renewable resources, resulting in different feasible combinations for project execution. The objective of the MRCPSP is to minimize the total makespan, determining the completion period and execution mode for each activity. This is an NP-hard problem, therefore obtaining optimal solutions for medium and large instances by exact methods within reasonable computational time becomes unfeasible. In this context, this dissertation proposes the implementation of the metaheuristics Variable Neighborhood Search (VNS) and Greedy Randomized Adaptive Search Procedure (GRASP) to produce high-quality solutions quickly. Additionally, Path Relinking (PR) is integrated into GRASP as a post-optimization step, aiming to explore promising regions of the search space. Two variants of PR are considered: Interior PR (IPR), focused on intensification along the path connecting high-quality solutions, and Exterior PR (EPR), directed at diversification through the exploration of regions adjacent to this path. In the adopted methodology, the construction phase of the GRASP algorithm is responsible for generating diversified initial solutions, which are refined by a local search procedure. Subsequently, PR procedures are triggered as post-optimization strategies to balance diversification and intensification over the best solutions obtained throughout the search. Analyses of the different search and post-optimization strategies indicate that the proposed approaches present competitive performance compared to the literature, highlighting the potential of these methodologies for solving the MRCPSP.

Keywords: Multi-mode Resource-Constrained Project Scheduling Problem, Greedy Randomized Adaptive Search Procedure, Path Relinking, Variable Neighborhood Search.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Apresentação Geral

De acordo com Demeulemeester et al. (2002), um projeto pode ser entendido como um empreendimento único, de grande porte e complexidade, composto por diversas atividades interdependentes, cujo desenvolvimento exige considerável esforço financeiro e deve ser estruturado em um cronograma que respeite tanto os requisitos de precedência quanto as restrições de recursos. Exemplos típicos de projetos incluem: construção de edifícios e de infraestrutura, fabricação e montagem de produtos, reparo e manutenção, desenvolvimento e marketing de novos produtos; dentre outros.

Nesse contexto, Kolisch (2013) aponta que a gestão de projetos abrange três dimensões fundamentais: planejamento, programação (*scheduling*) e controle. O planejamento corresponde a definição das atividades necessárias para a conclusão do projeto, a estimativa de sua duração e custos, bem como a determinação das demandas globais de recursos. Já a programação consiste em organizar essas atividades no tempo, especificando a ordem em que devem ser executadas, alocando os recursos em cada etapa e estimando o tempo de conclusão. Por fim, o controle tem como objetivo monitorar a execução, comparando o cronograma planejado com o desempenho real, de modo a identificar desvios e aplicar correções.

De maneira geral, projetos consistem em três elementos principais: atividades (ou tarefas), relações de precedência e recursos. No entanto, existem diferenças significativas entre projetos justamente pelas características específicas que cada um desses elementos pode assumir. Em especial, no que se refere aos recursos, Błażewicz et al. (2001) os classificam em três categorias: recursos renováveis, recursos não renováveis e recursos duplamente restritos.

Recursos renováveis são aqueles cuja disponibilidade é limitada em cada período, mas que se renovam ao longo do horizonte de planejamento a medida que as tarefas são concluídas. Exemplos típicos incluem máquinas, equipamentos e mão de obra que, após a conclusão de alguma atividade, podem ser utilizados novamente por outras tarefas. Por outro lado, os recursos não renováveis possuem uma quantidade total disponível, que vai sendo consumida ao longo da execução do projeto sem possibilidade de reposição. Como exemplos, podem-se citar o orçamento de capital, fixado para todo o empreendimento, ou a disponibilidade de matérias-primas, como combustível, que, após o uso, não podem ser utilizados novamente. Já os recursos duplamente restritos combinam simultaneamente características de renováveis e não renováveis,

pois estão sujeitos tanto a uma limitação por período quanto a uma restrição global de disponibilidade. Um exemplo é o capital sujeito a restrições de fluxo de caixa em cada período e, ao mesmo tempo, a um limite máximo de investimento total. Outro exemplo é homem-hora por dia, em que a quantidade total de trabalhadores está disponível em todos os dias; contudo, existe a restrição do número total de horas trabalhadas por dia para cada trabalhador.

A caracterização dos recursos desempenha papel central na modelagem e no gerenciamento de projetos, uma vez que suas restrições influenciam diretamente a viabilidade e a qualidade do cronograma obtido. É justamente nesse contexto que surge o *Resource-Constrained Project Scheduling Problem* (RCPSP), considerado por Kolisch (2015) como um dos problemas mais estudados de sequenciamento de projetos. O RCPSP consiste em elaborar um cronograma viável para um conjunto de atividades de um projeto, sujeitas a relações de precedência e consumo de recursos renováveis. Apesar do grande potencial de aplicação, o RCPSP não contempla todas as situações práticas. Por essa razão, surgiram extensões e generalizações do problema capazes de lidar com diferentes cenários (Hartmann e Briskorn, 2022), entre elas, destaca-se o *Multi-mode Resource-Constrained Project Scheduling Problem* (MRCPSP).

Enquanto o RCPSP assume apenas um modo de execução por atividade, o MRCPSP permite que cada atividade seja realizada em diferentes modos, cada um caracterizado por uma combinação específica de duração e consumo de recursos renováveis e não renováveis. Essa generalização aumenta a complexidade do problema, de fato este problema é classificado como NP-difícil (Blazewicz et al., 1983), mas também amplia sua aplicabilidade a cenários reais, nos quais decisões sobre a sequência e o modo de execução das atividades impactam diretamente o tempo total de conclusão do projeto (*makespan*) e a utilização de recursos limitados.

De acordo com Demeulemeester et al. (2002), a flexibilidade introduzida pelo MRCPSP leva a necessidade de avaliar *trade-offs*, especialmente entre tempo e custo ou tempo e recursos. Em muitas situações práticas, por exemplo, uma atividade pode ser concluída mais rapidamente aumentando o número de trabalhadores, com uso de máquinas adicionais ou com mais investimentos financeiros. Essa aceleração, entretanto, tende a gerar custos extras que só se justificam se a redução no tempo total do projeto compensar o investimento. Por outro lado, optar por modos de execução mais econômicos pode reduzir o custo da atividade, mas prolongar o cronograma, resultando em atrasos e possíveis penalidades. A ponderação precisa desses ganhos e perdas representa, muitas vezes, um grande desafio (Demeulemeester et al., 2002). Nesse contexto, o MRCPSP tem como objetivo determinar um cronograma ótimo que minimize o tempo total de conclusão do projeto, respeitando simultaneamente as restrições de precedência e a disponibilidade dos recursos, além de definir o modo de execução mais adequado para cada atividade.

Segundo Etminaniesfahani et al. (2024), a aplicação de métodos exatos para instâncias do MRCPSP com mais de 20 tarefas torna-se inviável, devido ao alto custo computacional, o que motiva o desenvolvimento de abordagens heurísticas e metaheurísticas para problemas de maior escala.

Desse modo, este trabalho propõe a investigação de duas metaheurísticas para resolver o MRCPSP. Uma abordagem híbrida baseada na metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) com *Variable Neighborhood Descent* (VND) combinada com o método *Path Relinking* (PR), como etapa pós-otimização.

Outra abordagem utilizando uma variante da metaheurística *Variable Neighborhood Search* (VNS) denominada *General* VNS (GVNS), que possui como estrutura de busca local o método VND.

Proposta por Feo e Resende (1995), a GRASP é uma metaheurística *multistart* que constrói soluções iniciais viáveis de forma iterativa por meio de uma estratégia gulosa aleatória, seguida de uma fase de busca local. Nesta etapa, será incorporado o método VND, que explora sistematicamente diferentes vizinhanças, evitando a convergência prematura em ótimos locais e intensificando a exploração do espaço de soluções. O *Path Relinking* (PR), por outro lado, originalmente proposto em Glover (1997) como uma técnica de intensificação e diversificação no contexto das metaheurísticas *Tabu Search* e *Scatter Search*, busca explorar regiões intermediárias entre duas soluções de alta qualidade por meio da construção de um caminho que as conecta no espaço de soluções. A forma tradicional desse método é conhecida como *interior Path Relinking* (IPR). Uma variação menos explorada, denominada *exterior Path Relinking* (EPR) e introduzida por Glover (2014), tem por objetivo diversificar a busca ao explorar regiões não visitadas, externas a vizinhança das duas soluções.

O VNS, por sua vez, é uma metaheurística proposta por Mladenović e Hansen (1997) que busca explorar o espaço de soluções por meio da troca sistemática entre diferentes estruturas de vizinhança, evitando a estagnação em ótimos locais. Diferentemente do GRASP, que realiza um reinício aleatório a cada iteração, o VNS concentra a busca em torno de uma solução, aproveitando informações das iterações anteriores. Essa estratégia combina elementos de diversificação e intensificação, alternando entre a exploração de novas regiões do espaço de busca e o refinamento de boas soluções. Quando o processo de intensificação é conduzido pelo método VND, o algoritmo é denominado GVNS.

Devido a facilidade de implementação, aliado ao baixo número de parâmetros e ao bom desempenho reportado em diversos problemas de otimização, este trabalho propõe a investigação das metaheurísticas GRASP e GVNS, ainda pouco exploradas na resolução do MRCPSP. Além disso, em metodologias recentes, GRASP tem sido combinado com PR com o objetivo de aumentar sua eficácia, demonstrando bom desempenho na obtenção de soluções de alta qualidade (Laguna et al., 2025). Por esses resultados, este trabalho também propõe uma abordagem que integra o GRASP com o EPR e com IPR para a resolução do MRCPSP. A estratégia adotada consiste em realizar inicialmente uma amostragem das melhores soluções obtidas na fase construtiva do GRASP, formando um conjunto elite. Em seguida, regiões promissoras do espaço de busca são exploradas por meio da aplicação do PR na vizinhança dessas soluções, permitindo intensificar a busca a partir de combinações entre soluções de alta qualidade. Por fim, as diferentes variantes do PR são comparadas, com o objetivo de analisar quais estratégias de relinkagem se mostram mais eficazes para o problema em estudo.

1.2 Objetivos: Geral e Específicos

O objetivo geral desta dissertação é propor três algoritmos para a resolução do MRCPSP. Dois deles visam a integração do procedimento GRASP com a busca local guiada pelo VND e com o EPR/IPR como etapa de pós-otimização. Outro avalia

o desempenho do GVNS para encontrar boas soluções. Para atingir esse objetivo, definem-se os seguintes objetivos específicos:

- Avaliar os principais métodos discutidos na literatura para construir soluções viáveis para o MRCPSP;
- Implementar e avaliar o impacto de diferentes configurações do VND;
- Implementar os métodos proposto;
- Analisar e comparar os resultados produzidos pelos dois método propostos;
- Realizar comparações dos resultados obtidos em relação aos trabalhos presentes na literatura.

1.3 Metodologia

Inicialmente, realizou-se uma revisão abrangente sobre o MRCPSP e as principais abordagens encontradas na literatura, com ênfase em técnicas heurísticas e metaheurísticas. A partir dessa análise, foram adaptadas e implementadas heurísticas construtivas capazes de gerar soluções viáveis para o GRASP e para o VNS. Em paralelo, foram levantadas as estruturas de vizinhança mais recorrentes para o MRCPSP, que, após testes preliminares, foram integradas ao esquema de busca local VND a fase de diversifição do VNS. Com base em uma revisão da metaheurística PR, definiu-se sua adaptação para o problema em destaque.

Após o desenvolvimento dos algoritmos, foram definidos os parâmetros de execução e selecionados conjuntos de instâncias testes tradicionalmente utilizados na literatura. Os experimentos computacionais foram realizados em ambiente controlado, repetindo-se cada execução diversas vezes para reduzir os efeitos da aleatoriedade e assegurar robustez dos resultados. Por fim, os resultados foram analisados quantitativamente, comparando-se as diferentes etapas do método e contrastando-os com resultados reportados na literatura.

1.4 Justificativa

O MRCPSP é um problema de grande relevância prática em áreas como construção civil, manufatura, logística e desenvolvimento de software, pois decisões sobre a sequência de atividades e os modos de execução impactam diretamente no tempo de conclusão do projeto e na utilização de recursos limitados. Entretanto, sua natureza NP-difícil pode tornar inviável a resolução exata de instâncias de porte real, o que reforça a necessidade do desenvolvimento de técnicas heurísticas e metaheurísticas capazes de produzir soluções de boa qualidade em tempo computacional reduzido.

Nesse contexto, o GRASP e o GVNS apresentam-se como metaheurísticas promissoras devido sua simplicidade e bom desempenho em problemas combinatórios, enquanto o VND e o PR se destacam como estratégias complementares de intensificação e diversificação da busca. A avaliação dessas técnicas no âmbito do MRCPSP constitui uma lacuna relevante na literatura, segundo nosso conhecimento, e pode

contribuir tanto para a melhoria da qualidade das soluções quanto para a uma nova investigação de métodos metaheurísticos de resolução. Assim, este trabalho justifica-se pela proposta de investigar e avaliar o GVNS e a combinação entre GRASP, VND e EPR/IPR, explorando seu potencial para gerar soluções competitivas em diferentes classes de instâncias e ampliando o conhecimento sobre abordagens híbridas aplicadas ao problema.

1.5 Organização da Dissertação

O restante desta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta a descrição do problema, a formulação matemática, a notação adotada, as instâncias recorrentes utilizadas e a revisão da literatura sobre o MRCPSP, com ênfase nas principais metaheurísticas aplicadas à sua resolução. No Capítulo 3, são discutidos os fundamentos teóricos dos métodos encontrados na literatura e empregados neste trabalho. O Capítulo 4 descreve as implementações dos algoritmos propostos para a resolução do MRCPSP. Em seguida, o Capítulo 5 apresenta os resultados obtidos, acompanhados de comparações com os melhores trabalhos reportados na literatura. Por fim, no Capítulo 6, são discutidas as conclusões do estudo, bem como perspectivas para pesquisas futuras relacionadas ao tema.

Capítulo 2

Descrição do Problema

Este capítulo descreve o Problema de Sequenciamento de Projetos com Restrição de Recursos e Múltiplos Modos de Execução (*Multi-mode Resource-Constrained Project Scheduling Problem* – MRCPSP), contextualizando-o a partir da versão clássica do Problema de Sequenciamento de Projetos com Restrição de Recursos (*Resource-Constrained Project Scheduling Problem* – RCPSP). Inicialmente, são discutidos os fundamentos do RCPSP, destacando suas principais aplicações e variações (Seção 2.1). Em seguida, apresenta-se a formulação matemática que descreve o MRCPSP (Seção 2.2). Posteriormente, a Seção 2.3 descreve as principais classes de instâncias disponíveis na literatura, que servem como base para a experimentação e comparação de métodos de solução. Por fim, na Seção 2.4, é realizada uma revisão da literatura sobre os principais métodos empregados na resolução do MRCPSP, com ênfase nas abordagens baseadas em metaheurísticas.

2.1 Problema de Sequenciamento em Projetos com Restrição em Recursos

Para formalizar a apresentação do problema de otimização investigado neste trabalho, será adotada a notação resumida mostrada na Tabela 2.1, a qual servirá como referência ao longo da dissertação.

O RCPSP pode ser formalmente definido da seguinte maneira. Considere um projeto composto por n atividades enumeradas de $j = 1, \ldots, n$. Cada atividade j possui uma duração $d_j \in \mathbb{N}$ e um conjunto de predecessores imediatos P_j, de modo que a atividade só pode ser iniciada após a conclusão de todas as atividades $i \in P_j$. Além disso, uma vez iniciada, a execução de uma atividade não pode ser interrompida. O consumo de recursos é modelado por r_{jk}, que representa a quantidade do recurso renovável $k \in \{1, \ldots, K\}$ requerida pela atividade j, sendo que a disponibilidade de cada recurso é limitada a R_k^ρ unidades por período. Sem perda de generalidade, consideramos duas atividades fictícias adicionais $j = 0$ e $j = n + 1$, representando o início e o término do projeto, respectivamente. A essas atividades, denominadas *dummy*, são atribuídos consumo de tempo e de recursos nulos.

O objetivo do problema consiste em determinar os instantes $t \in \{1, \cdots, T\}$ de início das atividades, de forma que todas as restrições de precedência e de capacidade de recursos sejam atendidas e minimizando o tempo total de conclusão do projeto,

Tabela 2.1: Notação utilizada para a descrição do MRCPSP

Símbolo	Descrição
J	Conjunto de atividades presentes no projeto ($\|J\| = n$ representa o número de tarefas)
$j \in J$	Atividade j presente no projeto
M_j	Conjunto de modos de execução da atividade $j \in J$
$m \in M_j$	modo m de execução da atividade j do conjunto M_j
P_j	Conjunto de predecessores da atividade j
S_j	Conjunto de sucessores da atividade j
d_{jm}	Tempo de processamento da atividade j no modo m
K	Quantidade de recursos presentes no projeto
$k = 1, \ldots, K$	índice do recurso presente no projeto
R_k^ρ	Quantidade disponível do recurso renovável do tipo k
R_k^ν	Quantidade disponível do recurso não renovável do tipo k
r_{jmk}^ρ	Quantidade do recurso renovável do tipo k consumida por j no modo m
r_{jmk}^ν	Quantidade do recurso não renovável do tipo k consumida por j no modo m
EST_j	Tempo de início mais cedo da atividade j
EFT_j	Tempo de término mais cedo da atividade j
LST_j	Tempo de início mais tardio da atividade j
LFT_j	Tempo de término mais tardio da atividade j
SLK_j	Folga da atividade j
C_{max}	*Makespan* do projeto
T	Horizonte do projeto

denominado *makespan*. O parâmetro T define o horizonte de planejamento.

O modelo matemático para o RCPSP, apresentado em Pritsker et al. (1969), pode ser expresso por uma formulação com variáveis indexadas no tempo. Seja a variável de decisão x_{jt}, que assume valor $x_{jt} = 1$ se a tarefa j é finalizada no tempo t, e valor $x_{jt} = 0$, caso contrário. O modelo matemático é dado então por:

$$\min \sum_{t=\mathrm{EFT}_{|J|}}^{\mathrm{LFT}_{|J|}} t x_{|J|t} \tag{2.1}$$

$$\text{suj. a:} \sum_{t=\mathrm{EFT}_j}^{\mathrm{LFT}_j} x_{jt} = 1, \qquad j \in J \tag{2.2}$$

$$\sum_{t=\mathrm{EFT}_j}^{\mathrm{LFT}_j} (t - d_j) x_{jt} - \sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i} t x_{it} \geq 0, \qquad j \in J; i \in \mathcal{P}_j \tag{2.3}$$

$$\sum_{j \in J} \sum_{s=t}^{t+d_j-1} r_{jk}^\rho x_{js} \leq R_{kt}^\rho, \qquad k = 1, \ldots, R^\rho; t = 1, \ldots, T \tag{2.4}$$

$$x_{jt} \in \{0,1\}, \qquad j \in J; t = \mathrm{EFT}_j, \ldots, \mathrm{LFT}_j \tag{2.5}$$

De acordo com essa formulação, a expressão (2.1) tem por objetivo minimizar o tempo de término da última tarefa. A restrição (2.2) garante que cada atividade j será finalizada exatamente uma vez em um tempo t específico. A expressão (2.3) assegura que a tarefa j inicie apenas após todas as suas predecessoras terem terminado. As expressões (2.4) limitam a quantidade de recursos renováveis utilizados. Finalmente, a restrição (2.5) expressa o domínio binário das variáveis de decisão x_{jt}.

Para cada tarefa j, são definidos os tempos de término mais cedo (EFT_j) e mais tardio (LFT_j), que estabelecem um intervalo no qual a tarefa pode ser concluída sem violar as relações de precedência e causar atrasos no projeto. Esses valores são determinados pelo *Critical Path Method* (CPM) (Kelley Jr e Walker, 1959).

O RCPSP é uma generalização do clássico problema *Static Job Shop Problem* (JSP) (Kolisch, 2013) apud (Schrage, 1970). Em Kolisch (2013) é apresentado como é feita a redução do RCPSP ao JSP. Devido ao fato de que o RCPSP é uma generalização do JSP, que é um problema NP-difícil, ele também pertence a classe de problemas NP-difíceis (Blazewicz et al., 1983).

Hartmann e Briskorn (2022) observaram que, embora o RCPSP seja já um modelo poderoso, ele não é capaz de abranger todas as situações que ocorrem na prática em planejamento de projetos. Portanto, muitos pesquisadores desenvolveram problemas de agendamento de projetos mais gerais, muitas vezes utilizando o RCPSP padrão como ponto de partida. Hartmann e Briskorn (2010, 2022) apresentam uma revisão sobre as variantes e extensões do RCPSP propostas na literatura como:

(i) Conceitos de atividade generalizados.

  (a) *Preemptive scheduling*: permite a interrupção e retomada de atividades em diferentes instantes (Afshar-Nadjafi, 2018);

  (b) *Resource requests varying with time*: demandas de recursos que variam ao longo da execução de uma mesma atividade (Klein, 2000);

  (c) *Setup times*: tempos adicionais necessários para preparar recursos ou atividades antes de sua execução (Vanhoucke e Coelho, 2019);

  (d) *Multiple modes* (MRCPSP): atividades que podem ser executadas em diferentes modos, cada um associado a distintas durações, custos e consumos de recursos (Fernandes e de Souza, 2021);

  (e) *Trade-off problems* envolvendo *discrete time-cost trade-off problem* (DTCTP) (Khalili-Damghani et al., 2015) e *discrete time-resource trade-off problem* (DTRTP) (Colak e Azizoglu, 2014).

(ii) Restrições temporais generalizadas.

  (a) *Minimal and maximal time lags:* são conriderados intervalos de tempo mínimos e máximos entre pares de atividades (Ballestín et al., 2011);

  (b) *Release dates and deadlines:* inclui janelas de tempo e prazos de entrega (Hill et al., 2019);

  (c) *Generalized network structures and logical dependencies:* execução opcional de um conjunto de atividades (Kellenbrink e Helber, 2015).

(iii) Restrições de recursos generalizadas.

  (a) *Nonrenewable and partially renewable resources*: a disponibilidade dos recursos está associada a um subconjunto de períodos do horizonte de planejamento e as atividades que exigem o recurso apenas o consomem se forem processadas nesses períodos (Watermeyer e Zimmermann, 2020);

(b) *Resources with multiple skills* (MS-RCPSP): cada atividade precisa de certas habilidades para ser realizada e os recursos possuem apenas algumas dessas habilidades. Assim, um recurso só pode ser alocado a uma atividade se tiver as habilidades necessárias e estiver disponível no período (Zabihi et al., 2019);

(c) *Continuous resources*: recursos que podem ser fracionados em quantidades contínuas (Waligóra, 2014);

(d) *Resource capacities varying with time*: capacidade disponível de um recurso muda ao longo do horizonte do projeto (Hartmann, 2014).

2.2 Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução

Formalmente, o MRCPSP pode ser caracterizado da seguinte forma. Considere um projeto composto por n tarefas, sendo que cada tarefa $j = 1, \ldots, n$ deve ser processada sem interrupção, respeitando a relação de precedência entre elas. Isso significa que uma tarefa j só pode ser iniciada após a conclusão de todas as suas predecessoras $i \in P_j$, em que P_j é o conjunto de predecessores da tarefa j. Cada tarefa j pode ser executada em um conjunto de modos M_j, em que cada modo $m \in M_j$ define um tempo de processamento d_{jm} e um consumo específico de recursos renováveis r^{ρ}_{jmk} e não renováveis r^{ν}_{jmk} do tipo k, para $k = 1, \ldots, |K|$. Os recursos renováveis possuem uma disponibilidade fixa R^{ρ}_k em cada período t, enquanto os não renováveis possuem uma quantidade total limitada ao longo de todo o horizonte de planejamento T, denotada por R^{ν}_k. As tarefas $j = 1$ e $j = n$ são fictícias, representando, respectivamente, o início e o término do projeto; a elas é atribuído um único modo de operação, com duração e consumo de recursos nulos.

O objetivo do problema é encontrar um sequenciamento viável ótimo que atribua a cada tarefa um modo de execução e um instante de início e término, de modo a minimizar a duração total do projeto.

O modelo matemático mais comumente utilizado do MRCPSP foi introduzido por Talbot (1982) e é baseado no modelo do RCPSP de Pritsker et al. (1969), sendo formulado da seguinte maneira. Seja a variável de decisão $x_{jmt} = 1$ se a tarefa j é executada no modo m e finalizada no tempo t; e $x_{jmt} = 0$, caso contrário. O modelo

matemático é, então, dado:

$$\min \sum_{m\in M_{|J|}} \sum_{t=\mathrm{EFT}_{|J|}}^{\mathrm{LFT}_{|J|}} t x_{|J|mt} \tag{2.6}$$

$$\textbf{suj a:} \sum_{m\in M_j} \sum_{t=\mathrm{EFT}_j}^{\mathrm{LFT}_j} x_{jmt} = 1, \qquad j \in J \tag{2.7}$$

$$\sum_{m\in M_j} \sum_{t=\mathrm{EFT}_j}^{\mathrm{LFT}_j} (t - d_{jm}) x_{jmt} - \sum_{m\in M_i} \sum_{t=\mathrm{EFT}_i}^{\mathrm{LFT}_i} t x_{imt} \geq 0, \quad j \in J; i \in \mathcal{P}_j \tag{2.8}$$

$$\sum_{j\in J} \sum_{m\in M_j} \sum_{s=t}^{t+d_{jm}-1} r^{\rho}_{jmk} x_{jms} \leq R^{\rho}_{kt}, \qquad k = 1,\ldots,R^{\rho}; t = 1,\ldots,T \tag{2.9}$$

$$\sum_{j\in J} \sum_{m\in M_j} \sum_{s=\mathrm{EFT}_j}^{\mathrm{LFT}_j} r^{\nu}_{jmk} x_{jms} \leq R^{\nu}_{kt}, \qquad k = 1,\ldots,R^{\nu} \tag{2.10}$$

$$x_{jmt} \in \{0,1\}, \qquad j \in J; m \in M; t = \mathrm{EFT}_j,\ldots,\mathrm{LFT}_j \tag{2.11}$$

De acordo com essa formulação, a expressão (2.6) tem, por objetivo, minimizar o tempo de término da última tarefa. A restrição (2.7) garante que cada atividade j será finalizada exatamente uma vez, em um único modo m e em um tempo t específico. A expressão (2.8) assegura que a tarefa j inicie apenas após todas as suas predecessoras terem terminado. As expressões (2.9) e (2.10) limitam a quantidade de recursos renováveis e não renováveis utilizados, respectivamente. Finalmente, a restrição (2.11) expressa o domínio binário das variáveis de decisão x_{jmt}.

Para cada tarefa j, são definidos os tempos de término mais cedo (EFT_j) e mais tardio (LFT_j), que estabelecem um intervalo no qual a tarefa pode ser concluída sem violar as relações de precedência e causar atrasos no projeto. Esses valores são determinados pelo *Critical Path Method* (CPM) (Kelley Jr e Walker, 1959). Para o MRCPSP, essas valores são estimados a partir do caminho crítico mínimo, conforme discutido na Seção 3.2.

O MRCPSP é um problema NP-difícil, porém, quando há múltiplos recursos, Blazewicz et al. (1983) provou que é um problema NP-completo. Nos trabalhos de Hartmann e Briskorn (2022); Mika et al. (2015); Noori et al. (2018) são apresentados levantamentos bibliográficos acerca da aplicação do MRCPSP, bem como suas extensões e variações. O Seção 2.4 apresenta uma breve revisão dos principais métodos para resolução desse problema.

2.3 Instâncias de Referência para o MRCPSP

Esta seção apresenta os principais conjuntos de instâncias de referência utilizadas na literatura para o MRCPSP. Segundo Fernandes (2017) apud Patterson et al. (1989), problemas de otimização podem ser descritos como um conjunto I de instâncias. Cada instância é representada por um par (F, c), em que F é o conjunto de soluções viáveis e c é uma função de custo definida como $c : F \to \mathbb{R}$. O objetivo consiste em encontrar um elemento $f \in F$ tal que $c(f) \leq c(y)$, $\forall y \in F$, no caso de problemas de minimização. Esse elemento f é denominado ótimo global da instância,

ou simplesmente solução ótima. Dessa forma, uma instância é caracterizada por seus dados de entrada, que contêm todas as informações necessárias para a obtenção de uma solução.

Como destacado por Vanhoucke e Coelho (2018), a padronização no uso de dados é fundamental para permitir comparações consistentes e a realização de *benchmarking* de novos métodos em relação ao estado da arte. Essas instâncias servem como referência, assegurando que os resultados obtidos possam ser comparados com aqueles reportados em outros estudos.

Os três principais conjuntos de dados para o MRCPSP são as classes de problemas BOCTOR, PSLIB e MMLIB, descritas a seguir.

- PSLIB: proposto em Kolisch et al. (1992), este conjunto de dados possui sete subconjuntos de dados (J10, J12, J14, J16, J18, J20 e J30), cada uma contendo 640 instâncias com 10, 12, 14, 16, 18, 20 e 30 atividades, respectivamente geradas pelo *project generator* (ProGen) (Kolisch, 2013). Cada atividade possui 3 modos de execução e utiliza 2 recursos renováveis e 2 não renováveis. Este conjunto de dados pode ser encontrado no endereço `https://www.om-db.wi.tum.de/psplib/data.html`.

- BOCTOR: proposto em Boctor (1993), esse conjunto de dados é composto por 240 problemas, gerados aleatoriamente, divididos em dois subconjuntos principais: 120 instâncias com 50 atividades (Boctor50) e 120 com 100 atividades (Boctor100), cada um gerado a partir de redes com, em média, dois sucessores imediatos por atividade e diferentes configurações de recursos renováveis (1, 2 ou 4 tipos). O número de modos de execução varia uniformemente de 1 a 4, com durações e requisitos de recursos definidos por distribuições uniformes e ajustes para manter consumo médio semelhante entre modos. Este conjunto de dados pode ser encontrado no endereço `https://www.om-db.wi.tum.de/psplib/data.html`.

- MMLIB: proposto em Van Peteghem e Vanhoucke (2014), este conjunto de dados possui três subconjuntos de instâncias: MMLIB50 (50 atividades), MMLIB100 (100 atividades) e MMLIB+ (50 e 100 atividades). As duas primeiras classes possuem características semelhantes às da PSLIB, com cada atividade contendo três modos de execução e disponibilidade de dois recursos renováveis e dois não renováveis, totalizando 540 instâncias por classe. A classe MMLIB+, mais complexa, contém 2 ou 4 recursos renováveis e não renováveis, e cada atividade possui 3, 6 ou 9 modos de execução, totalizando 3240 instâncias. O conjunto está disponível em `http://www.projectmanagement.ugent.be/`.

No trabalho de Van Peteghem e Vanhoucke (2014), os autores destacam que, apesar do crescente interesse em desenvolver metaheurísticas para o MRCPSP, a comparação entre métodos enfrentava naquela época dificuldades significativas. O principal motivo estava na inconsistência dos conjuntos de dados de referência e na variação dos critérios de parada adotados. Com base nessa constatação e a partir de análises das instâncias PSLIB e BOCTOR, os autores propuseram um novo conjunto de dados, denominado MMLIB, com o objetivo de permitir avaliações mais consistentes e confiáveis dos métodos.

As análises realizadas em Van Peteghem e Vanhoucke (2014) consideram um conjunto de parâmetros descritos na literatura para avaliar a complexidade dos projetos, contemplando tanto as características da rede quanto a escassez de recursos. Para a complexidade da rede, foram utilizadas as métricas:

- CNC (*coefficient of network complexity*): mensura o número total de relações de precedência sobre o número total de atividades do projeto. Essa medida quantifica a complexidade geral da estrutura da rede do projeto.
- OS (*order strength*): define o número total de relações de precedência (incluindo relações de precedência indiretas) sobre o número máximo teórico de relações de precedência, indica o grau em que as relações de precedência entre as atividades restringem sua possível ordenação. Uma força de ordem mais alta sugere uma rede mais restrita ou rígida.
- I2 (*complexity index*): mensura o quão perto o projeto está de uma rede paralela/serial. Para I2=0, a rede possui todas as atividades em paralelo, enquanto que, para I2=1, todas as atividades estão em série.

Já a escassez de recursos foi mensurada pelas métricas:

- RF (*resource factor*): fornece uma indicação da intensidade do uso de recursos dentro do projeto, refletindo a média de recursos utilizados por recurso. Se RF=1, então cada atividade requer a utilização de todos os recursos; por outro lado, se RF=0, então nenhuma atividade requer qualquer tipo de recurso.
- RS (*resource strengh*): mede a escassez de recursos. Um valor menor de RS indica maior escassez de recursos, o que significa que os recursos são mais restritos.

Para mais detalhes ver Kolisch et al. (1992); Vanhoucke et al. (2008).

No estudo de Van Peteghem e Vanhoucke (2014), são apresentadas estatísticas dessas características para as classes BOCTOR e PSLIB. Observou-se que os valores de OS (J30 e Boctor100) e de RS (Boctor100) possuem faixa restrita. A PSLIB, segundo os autores, apesar da diversidade de variação para os parâmetros de recursos RS(0,25-1) e incorporar dois tipos de recursos, apresenta quatro limitações principais:

(i) nem todas as instâncias possuem solução viável (média de 549 viáveis por subconjunto);

(ii) baixa variação de OS (0,35–0,60), implicando baixa diversidade na complexidade topológica das redes;

(iii) instâncias resolvidas facilmente por métodos existentes, oferecendo pouco espaço para alcançar melhorias significativas com novos procedimentos; e

(iv) presença de modos ineficientes, eliminados no procedimento de pré-processamento de Sprecher et al. (1997), o que altera os parâmetros de recursos e reduz a média para 2,88 modos por atividade, em vez de 3.

Para Boctor100, Van Peteghem e Vanhoucke reconhecem como vantagem o maior número de atividades por instância, mas apontam três limitações:

(i) ausência de recursos não renováveis;

(ii) baixa restrição de recursos (RS $\leq$ 0,25); e

(iii) alta força de precedência (OS = 0,8-0,95), caracterizando redes quase totalmente seriais.

Diante dessas inconsistências, segundo Van Peteghem e Vanhoucke, o processo de geração da MMLIB buscou atender a quatro condições:

(i) garantir diversidade tanto na estrutura da rede quanto nas características dos recursos;

(ii) assegurar que todas as instâncias possuam pelo menos uma solução viável;

(iii) evitar a exclusão de modos no pré-processamento; e

(iv) considerar simultaneamente recursos renováveis e não renováveis.

Para gerar essas instâncias, foi utilizado o gerador *RanGen*, desenvolvido por Vanhoucke et al. (2008), explorando diferentes combinações de parâmetros. O conjunto de dados MMLIB, ainda segundo Van Peteghem e Vanhoucke, fornece instâncias mais desafiadoras para testar e comparar algoritmos para o MRCPSP, retirando as limitações existentes nos *benchmarks* anteriores.

Considerando a diversidade e complexidade das instâncias utilizadas na avaliação dos algoritmos para a resolução do MRCPSP, torna-se relevante realizar uma análise das abordagens propostas na literatura para esse problema. Nesse contexto, a Seção 2.4 apresenta uma revisão dos principais trabalhos relacionados, com ênfase em métodos baseados em metaheurísticas, de modo a identificar as estratégias mais recorrentes e as lacunas de pesquisa relacionados ao MRCPSP.

2.4 Trabalhos Relacionados

A resolução do MRCPSP tem sido amplamente estudada nas últimas décadas, com diversas abordagens propostas na literatura, incluindo métodos exatos, heurísticos, metaheurísticos e matheurísticos. Nesta seção, apresentamos um levantamento dos principais trabalhos relacionados ao problema.

Uma revisão de pesquisas realizadas entre 2010 e 2020 é apresentada em Hartmann e Briskorn (2022). Esse trabalho realiza um levantamento bibliográfico dos métodos utilizados para tratar o RCPSP e suas variantes.

Diversas metaheurísticas foram adaptadas para a resolução do MRCPSP (Van Peteghem e Vanhoucke, 2014). Dentre elas destacam-se na literatura: *genetic algorithm* (GA), *scatter search* (SS), *tabu search* (TS), *simulated annealing (SA)*, *particle swarm optmization* (PSO), *ant colony optmization* (ACO), *differencial evolution algorithm* (DEA), *multi-agent learning algorithm* (MAL), *estimation of distribution algorithm* (EDA), *path relinking* (PR), *Modified Variable Neighborhood Search Heuristic* (MVNSH), *Variable Neighborhood Search* (VNS), *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Iterated Local Search* (ILS).

Hartmann (2001) introduziu o uso de GA para resolver o MRCPSP, desenvolvendo uma representação específica para os componentes genéticos e hibridizando o GA com heurísticas de busca local baseadas no *multi-mode left shift* de Sprecher (1994). Essa proposta estendeu a abordagem já aplicada por Hartmann (1998) ao RCPSP. Partindo de uma linha de pesquisa semelhante, de estudos anteriores sobre o RCPSP, Alcaraz et al. (2003) implementaram o GA para o MRCPSP e, com base nas observações do trabalho de Hartmann (2001), propuseram modificações na função objetivo para lidar com soluções inviáveis.

De forma análoga, Lova et al. (2009) apresentaram uma versão híbrida do GA com busca local, utilizando uma extensão do *forward–backward improvement method (FBI)* para o caso *multi-mode*, descrito em Li e Willis (1992). Assim como em Alcaraz et al. (2003), os autores também propuseram uma nova função para avaliar os indivíduos da população. No entanto, Lova et al. buscaram maximizar a probabilidade de gerar indivíduos viáveis, atribuindo os modos de execução de forma enviesada, por meio da atribuição de pesos na escolha dos modos, em contraste com a atribuição aleatória utilizada nos trabalhos anteriores de Hartmann (2001); Alcaraz et al. (2003). Trabalhos mais recentes, como os de Peteghem e Vanhoucke (2010); Okada et al. (2014); Afshar et al. (2022), também exploram a hibridização do GA baseada no FBI, mas Okada et al. e Afshar-Nadjafi focaram nas tarefas críticas do projeto. Já Machado-Domínguez et al. (2021) desenvolveram um algoritmo memético, combinando componentes do GA e do VNS.

O SS foi explorado por Van Peteghem e Vanhoucke (2011), que implementaram procedimentos de busca local baseados em Hartmann (2001) e Peteghem e Vanhoucke (2010), avaliando o impacto de cada um no desempenho do método. De forma semelhante, Ranjbar et al. (2009) também aplicaram o SS, porém ao MRCPSP com apenas recursos renováveis.

Józefowska et al. (2001) investigaram a aplicação do SA em duas abordagens distintas: a primeira considerando apenas soluções viáveis em relação aos recursos e a segunda permitindo soluções inviáveis, tratadas por meio da função de penalização proposta por Hartmann (2001). Segundo os autores, a segunda abordagem, com o uso da função de penalização, apresentou desempenho superior à primeira. De forma complementar, Bouleimen e Lecocq (2003) implementaram o SA empregando o *serial schedule generation scheme* (SSS) (Kolisch, 1996) para o sequenciamento das atividades, e propuseram a exploração da vizinhança em dois estágios: o primeiro voltado para a exploração da vizinhança dos modos e o segundo voltado para a vizinhança baseada em atividades.

Zhang et al. (2006) introduziram uma metodologia baseada em PSO para resolver o MRCPSP, na qual cada solução, representada por uma partícula, era ajustada para tratar inviabilidades e posteriormente convertida em um cronograma por meio do SSS. Posteriormente, Jarboui et al. (2008) propuseram uma variação denominada *Combinatorial* PSO (CPSO), que considera soluções inviáveis, mas as penaliza. Em comparação, o CPSO apresentou desempenho superior ao PSO convencional e também superou, em termos de número de problemas resolvidos de forma ótima e de desvio médio, os resultados obtidos por Bouleimen e Lecocq (2003).

Chiang et al. (2008) e Li e Zhang (2013) implementaram o ACO para resolver o MRCPSP. Em Li e Zhang (2013), são propostos dois tipos de feromônio para guiar as formigas: um relacionado a seleção do modo de execução das atividades e

outro ao seu sequenciamento, acompanhados das respectivas heurísticas e algoritmos específicos para cada tipo de feromônio. Em Damak et al. (2009), foi implementado o DEA com foco no desempenho, buscando resolver o problema em menor tempo de processamento. Posteriormente, Nguyen e Kachitvichyanukul (2012) propuseram o *Efficient* DEA (EDEA) para o MRCPSP. A proposta consistia em uma versão aprimorada do DEA que utiliza um fator de cruzamento decrescente linear e um custo de penalidade adaptativo, com o objetivo de acelerar a busca por soluções de melhor qualidade, evitando ficar preso em ótimos locais e explorando o espaço de busca de forma mais eficiente. Por fim, Zhang et al. (2015) apresentaram uma abordagem híbrida combinando PSO e DEA. O método adota uma codificação em dois níveis: no nível superior, o PSO otimiza a sequência das atividades; no nível inferior, o DEA determina o modo de execução de cada tarefa.

O EDA foi abordado por Wang e Fang (2012), que utilizaram o SSS aliado a um novo modelo probabilístico e FBI como busca local. De forma híbrida, Soliman e Elgendi (2014) combinaram EDA com uma busca local baseada no operador *delete-then-insert* associado ao *Random Walk* (RW). Esse procedimento remove uma atividade da sua posição atual em uma lista de atividades e a insere em uma posição elegível aleatória e a aceitação dessa mudança é feita com base em uma taxa de aceitação (RW), que controla a aleatoriedade e a exploração. Abordagens baseadas em MAL foram exploradas por Wauters et al. (2009), Wauters et al. (2011) e Jędrzejowicz e Ratajczak-Ropel (2015), utilizando aprendizagem para construção de cronogramas.

No trabalho de Cravo (2009), o GRASP é aplicado a um estudo de caso de Programação de Manutenção Industrial, modelado como um problema de programação de projetos. Já em Alvarez-Valdés et al. (2008), o GRASP é integrado ao PR e empregado em uma variação do MRCPSP que considera recursos parcialmente renováveis. Em Boctor e d'Avignon (2005) e Mika et al. (2008) aplicaram a TS para resolver o MRCPSP, sendo que, no trabalho de Mika et al., foi considerada a versão do problema com tempos de *setup*. Em Tchao e Martins (2008), a TS foi hibridizada com o PR como estratégia de pós-otimização. Segundo os autores, essa abordagem foi capaz de encontrar soluções superiores às obtidas pela TS isolada e com baixo custo de tempo de processamento adicional. Por sua vez, Muritiba et al. (2018) também utilizaram o PR, mas como metaheurística principal. Diferentemente de Tchao e Martins (2008), o método proposto não parte de um conjunto de soluções de elite pré-existente; em vez disso, integra o PR como elemento central da busca no espaço de soluções, aliado a um procedimento de busca local para melhorar a exploração. Em outras abordagens, Fernandes (2017) implementaram o ILS, Ramos et al. (2022) o *multi-start* ILS (MS-ILS), enquanto Chakrabortty et al. (2019, 2020) propuseram o MVNSH, uma variação do VNS que integra elementos inspirados da TS a fim de evitar a recorrência de soluções repetidas.

Vale destacar que os resultados obtidos em Muritiba et al. (2018) são os melhores reportados na literatura para metaheurísticas aplicadas às bibliotecas PSLIB e MMLIB, exceto para os problemas MMLIB+ da classe MMLIB, nos quais Chakrabortty et al. (2020) apresentam desempenho superior.

Adicionalmente, técnicas guiadas por matheurísticas também têm sido aplicadas com sucesso ao MRCPSP. Por exemplo, Ramos et al. (2025) exploraram uma generalização da decomposição de Benders, enquanto Fernandes e de Souza (2021)

utilizaram a metaheurística *Local Branching*, obtendo soluções promissoras. Outra abordagem relevante é a de Changchun et al. (2018), que propuseram um algoritmo de sequenciamento baseado em geração de colunas. Já Etminaniesfahani et al. (2024) apresentaram uma solução através de *Relax-and-Solve*, consolidando-se como um dos trabalhos de referência para instâncias da classe MMLIB e PSLIB.

A Tabela 2.2 apresenta, de forma resumida, algumas das metaheurísticas citadas anteriormente para a resolução do MRCPSP, organizadas de acordo com autores, métodos e instâncias utilizadas.

Autores	**Abordagem**	**Instância**
Hartmann (2001)	GA	PSLIB
Alcaraz et al. (2003)	GA	BOCTOR/PSLIB
Lova et al. (2009)	GA	BOCTOR/PSLIB
Peteghem e Vanhoucke (2010)	GA	BOCTOR/PSLIB
Okada et al. (2014)	GA	PSLIB
Machado-Domínguez et al. (2021)	GA/VNS	PSLIB/MMLIB
Afshar et al. (2022)	GA	PSLIB
Ranjbar et al. (2009)	SS	PSLIB
Van Peteghem e Vanhoucke (2011)	SS	PSLIB
Józefowska et al. (2001)	SA	PSLIB
Bouleimen e Lecocq (2003)	SA	PSLIB
Zhang et al. (2006)	PSO	PSLIB
Jarboui et al. (2008)	CPSO	PSLIB
Zhang et al. (2015)	PSO / DEA	PSLIB
Chiang et al. (2008)	ACO	PSLIB
Li e Zhang (2013)	ACO	PSLIB
Damak et al. (2009)	DEA	PSLIB
Nguyen e Kachitvichyanukul (2012)	EDEA	PSLIB
Wang e Fang (2012)	EDA	PSLIB
Soliman e Elgendi (2014)	EDA	PSLIB
Wauters et al. (2009)	MAL	PSLIB
Wauters et al. (2011)	MAL	PSLIB
Jędrzejowicz e Ratajczak-Ropel (2015)	MAL	PSLIB
Cravo (2009)	GRASP	PSLIB
Boctor e d'Avignon (2005)	TS	BOCTOR
Tchao e Martins (2008)	TS /PR	PSLIB
Muritiba et al. (2018)	PR	BOCTOR/PSLIB/MMLIB
Chakrabortty et al. (2019)	MVNSH	PSLIB
Chakrabortty et al. (2020)	MVNSH	PSLIB/MMLIB
Fernandes (2017)	ILS	PSLIB
Ramos et al. (2022)	MS-ILS	PSLIB/MMLIB
Este trabalho	GRASP/PR	PSLIB/MMLIB
Este trabalho	G-VNS	PSLIB/MMLIB

Tabela 2.2: Resumo dos trabalhos revisados, com autores, abordagens e instâncias aplicadas ao MRCPSP.

A partir da Tabela 2.2, observa-se uma predominância de abordagens evolucionárias e populacionais, em especial algoritmos genéticos, em comparação às metaheurísticas baseadas em trajetórias. Nota-se também que grande parte das contribuições utiliza principalmente as instâncias da PSLIB; as instâncias de BOCTOR aparecem em menor escala, sobretudo porque consideram apenas recursos renováveis. Já as instâncias da MMLIB, mais recentes e de maior complexidade, têm despertado crescente interesse por possibilitarem a exploração de novas estratégias de resolução. Além disso, destaca-se na literatura a ampla utilização do método SSS na construção de soluções viáveis e do procedimento FBI como mecanismo de busca local, frequentemente combinado de forma híbrida com algoritmos genéticos.

Observa-se uma exploração ainda limitada de metaheuríticas baseadas em trajetória para o MRCPSP. Nesse contexto, este trabalho se insere na literatura ao propor e contribuir com uma nova abordagem metaheurística híbrida para a resolução do MRCPSP, combinando o GRASP com *Path Relinking* como etapa pós-otimização. Adicionalmente, são comparadas duas variações do PR com propostas distintas de exploração do espaço de busca: o *interior* PR com ênfase na intensificação ao longo do caminho que conecta soluções de elite, e o exterior PR, voltado à diversificação por meio da exploração de regiões adjacentes a esse caminho. A análise comparativa entre essas duas estratégias permite avaliar o impacto de diferentes mecanismo de exploração do espaço de busca no desempenho global do algoritmo.

Dessa forma, esta dissertação propõe não apenas uma abordagem híbrida baseada em metaheurísticas, mas também fornece evidências empíricas sobre o impacto do PR como mecanismo de intensificação e diversificação no contexto do MRCPSP. Além disso, o uso de heurísticas já consolidadas na literatura e instâncias recorrentes para o problema, permite que os resultados obtidos sejam comparados diretamente com estudos anteriores e, eventualmente, posteriores, reforçando a relevância e o potencial de contribuição da proposta.

Capítulo 3

Fundamentação Teórica

Neste capítulo, são apresentados os fundamentos teóricos necessários para o desenvolvimento deste trabalho. Inicialmente, na Seção 3.1, descreve-se uma instância exemplo do MRCPSP, a qual servirá de base para a construção e a apresentação dos modelos. Na sequência, a Seção 3.2 aborda conceitos fundamentais da estrutura de redes de projetos, com ênfase na identificação das atividades críticas e em suas principais características. A Seção 3.3 trata da representação computacional do problema enquanto a Seção 3.4 trata da função de avaliação utilizada. Em seguida, a Seção 3.5 apresenta os métodos construtivos mais relevantes descritos na literatura para a resolução do MRCPSP. Posteriormente, discutem-se heurísticas de refinamento, com destaque para o VND. Por fim, a Seção 3.7 examina as metaheurísticas aplicadas neste trabalho, enfatizando o GRASP, PR e o VNS, detalhando suas principais características no processo de busca por soluções de qualidade.

3.1 Exemplo de Projeto

Para ilustrar de forma clara as etapas de construção e resolução do modelo, os exemplos apresentados ao longo deste trabalho serão baseados na instância j105_4, pertencente à biblioteca PSPLIB. Essa instância faz parte do conjunto de problemas com 10 atividades reais, acrescidas de 2 tarefas fictícias. Ela contempla restrições de precedência entre as atividades, múltiplos modos, além de restrições de capacidade de recursos renováveis e não renováveis, características típicas do MRCPSP.

A Tabela 3.1 apresenta os modos de execução das atividades, com seus respectivos tempos de processamento e consumo de recursos, para $K = 4$ tipos de recursos, sendo $k = 1, 2$ representando os recursos renováveis e $k = 3, 4$ representando os recursos não renováveis. As informações sobre as relações de precedência entre as tarefas são apresentadas na Tabela 3.2.

Tabela 3.1: Instância j105_5 da PSLIB

Tarefa (j)	Modo (m)	Duração (d_{jm})	r^{ρ}_{jm1}	r^{ρ}_{jm2}	r^{ν}_{jm3}	r^{ν}_{jm4}
1	1	0	0	0	0	0
2	1	6	8	0	9	4
	2	6	0	5	9	3
	3	8	10	0	9	1
3	1	5	9	0	8	4
	2	7	7	0	7	4
	3	8	0	6	6	4
4	1	4	9	0	4	9
	2	5	0	9	4	7
	3	10	0	3	4	7
5	1	2	8	0	6	4
	2	3	0	8	6	4
	3	9	0	4	4	3
6	1	2	7	0	6	5
	2	4	6	0	6	3
	3	9	6	0	5	3
7	1	1	6	0	6	3
	2	2	0	8	6	3
	3	4	3	0	6	3
8	1	1	0	6	5	2
	2	2	0	5	4	1
	3	4	0	5	3	1
9	1	1	0	9	9	10
	2	4	0	5	8	6
	3	4	0	4	9	6
10	1	2	3	0	3	4
	2	5	0	7	2	4
	3	9	0	4	2	2
11	1	4	0	3	9	6
	2	4	4	0	10	8
	3	6	0	3	9	5
12	1	0	0	0	0	0
Recursos Disponíveis (R^{ρ}_1,R^{ρ}_2,R^{ν}_3,R^{ν}_4)			**10**	**9**	**58**	**39**

Tabela 3.2: Relações de precedência da instância j105_5 da PSLIB

Atividades	#modos	#successores	successores
1	1	3	2 3 4
2	3	2	5 6
3	3	1	7
4	3	1	7
5	3	3	8 9 11
6	3	3	9 10 11
7	3	1	10
8	3	1	10
9	3	1	12
10	3	1	12
11	3	1	12
12	1	0	–

3.2 Método do Caminho Crítico

Desenvolvido no final da década de 1950 e proposto por Kelley Jr e Walker (1959), o *Critical Path Method* (Método do Caminho Crítico - CPM) surgiu em um contexto de crescente complexidade no planejamento de grandes projetos de engenharia aliado à ausência de técnicas eficazes para lidar com essa complexidade. Segundo Kelley Jr e Walker (1959), uma das principais dificuldades das abordagens utilizadas na época era a realização simultânea das etapas de planejamento e agendamento, que exigia que os gestores considerassem, de forma simultânea, grande quantidade de informações relacionadas à tecnologia, precedência, prazos, calendários e custos. Nesse cenário, o CPM foi proposto como uma técnica sistemática para decompor essas funções em etapas estruturadas, permitindo identificar uma sequência de atividades críticas, atividades que determinam diretamente a duração total do projeto e o tempo mínimo necessário para a conclusão de um projeto.

Para Demeulemeester et al. (2002), na execução do CPM a representação do projeto é fundamental, pois, a partir dela, as características topológicas do projeto são estabelecidas. Segundo os mesmos autores, uma representação comum apresentada na literatura é a utilização da estrutura em rede *Activity on Node* (AoN), que permite representar, por meio de um grafo, as relações de ordem entre as atividades. Nesta representação, cada atividade representa um nó, enquanto os arcos mostram a sequência na qual as atividades no projeto devem ser executadas, isto é, representam as relações de precedência. A Figura 3.1 apresenta a representação dessa rede para a instância j105_4 a partir da Tabela 3.2.

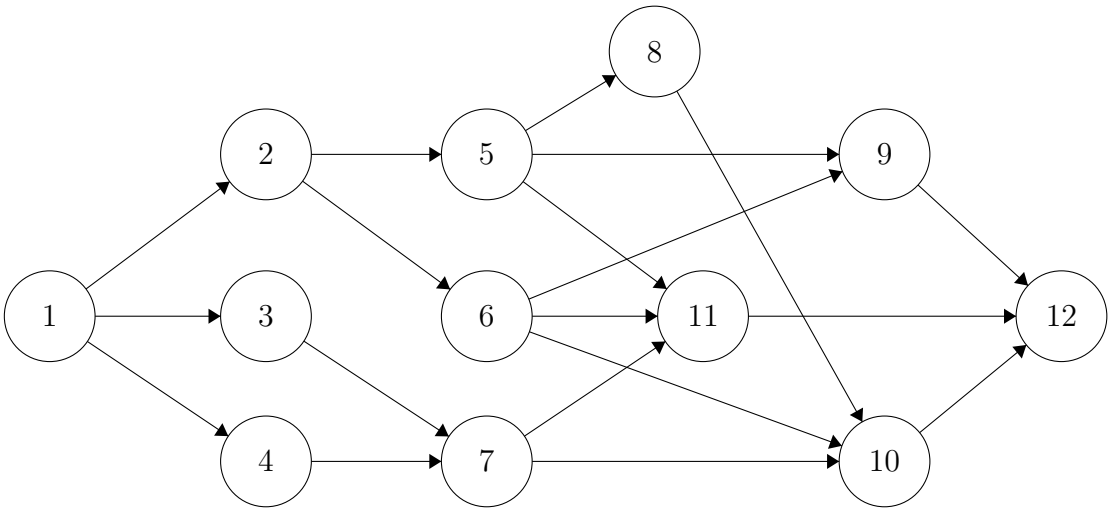


Figura 3.1: Estrutura de rede AoN da instância j105_4.

Formalmente essa representação pode ser definida por meio de um grafo direcionado acíclico $G = (N, A)$, em que:

- N: conjunto de nós, em que cada nó representa uma atividade do projeto;
- $A \subseteq N \times N$: conjunto de arestas direcionadas, em que cada arco $(i, j) \in A$ indica relação de precedência entre as atividades, ou seja, indica que atividade j não pode ser iniciada até que a atividade i seja concluída.

Por meio dessa estrutura, o CPM estima os tempos mais cedo e mais tarde de início e término de cada atividade, considerando apenas seus tempos de processamento. Segundo Demeulemeester et al. (2002), o cálculo padrão desses tempos assume que uma atividade, representada por um nó, pode ser iniciada somente após a conclusão de todas as atividades predecessoras conectadas por arestas incidentes àquele nó.

Seja então um projeto P, representado na estrutura AoN, com n atividades. Denotamos por EST_j, *(Earliest Start Time)* como o tempo mais cedo que uma determinada tarefa j pode começar e por d_j sua respectiva duração. Conforme apresentado em Kelley Jr e Walker (1959), podemos calculá-lo da seguinte maneira.

$$\begin{cases} EST_1 = 0, \\ \\ EST_j = \max\{EST_i + d_i \mid (i,j) \in P\}, \quad 2 \leq j \leq n. \end{cases} \tag{3.1}$$

De maneira similar, podemos calcular o tempo de término mais tardio de uma atividade j, LFT_j *(Latest Finish Time)*. Considerando T como horizonte de planejamento de P, para $T \geq EFT_n$, obtemos:

$$\begin{cases} LFT_n = T, \\ \\ LFT_i = \min\{LFT_j - d_j \mid (i,j) \in P\}, \quad 1 \leq j \leq n-1. \end{cases} \tag{3.2}$$

Algorithm 1 Método do Caminho Crítico

1: $EST_1 \leftarrow 0$;
2: **for** $j \leftarrow 2$ até n **do**
3: $\quad EST_j \leftarrow \max\{EST_i + d_i \mid i \in P_j\}$;
4: **end for**
5: $LFT_n \leftarrow T$;
6: **for** $j \leftarrow n-1$ até 1 **do**
7: $\quad LFT_j \leftarrow \min\{LFT_i - d_i \mid i \in S_j\}$;
8: **end for**

Em Demeulemeester et al. (2002) os autores representam esse procedimento de forma iterativa por meio do Algoritmo 1 através do procedimento denominado *Forward and Backward pass*. Nesse caso, P_j agora representa o conjunto de nós (atividades) que precedem imediatamente o nó j, enquanto S_j representa o conjunto de nós que sucedem imediatamente j. Ainda segundo Demeulemeester et al. (2002), a complexidade computacional desse algoritmo é da ordem de $O(n^2)$, sendo n o número de atividades.

Após o cálculo dos tempos EST_j e LFT_j da atividade j é possível definir seus respectivos tempos de término mais cedo e de início mais tardio, EFT_j e LST_j, sendo:

$$EFT_j = EST_j + d_j;$$

$$LST_j = LFT_j - d_j$$

A diferença entre esses tempos define a folga total (ou *total slack*) de cada atividade, isto é:

$$SLK_j = LST_j - EST_j = LFT_j - EFT_j \tag{3.3}$$

Quando a folga total de uma atividade é igual a zero, ela é classificada como uma tarefa crítica. Esse termo se justifica pelo fato de que qualquer atraso em sua execução resulta em um atraso equivalente na conclusão do projeto. Segundo Kelley Jr e Walker (1959), um projeto apresentará tarefas críticas somente quando $T = EST_n$, ou seja, quando a duração total do projeto for igual ao tempo de início mais cedo da última atividade. Além disso, se um projeto possui tarefas críticas, ele também contém pelo menos um caminho contíguo formado por essas atividades críticas, ligando o início ao fim do projeto. Esse caminho é denominado caminho crítico.

Como destacado em Fernandes (2017), um projeto pode possuir vários caminhos críticos. No contexto do MRCPSP, ao atribuir a cada atividade o modo com menor duração e aplicar o CPM, obtém-se o chamado Caminho Crítico Mínimo. Por outro lado, ao considerar o modo com maior duração para cada atividade, obtém-se o Caminho Crítico Máximo. Essas variações oferecem uma visão dos extremos possíveis para a estrutura crítica de um projeto no MRCPSP.

Para exemplificar a execução do Algoritmo 1, a Tabela 3.3 apresenta os valores de EST, EFT, LST, LFT e SLK para uma determinada configuração de modos. A partir dessa tabela, observa-se que as tarefas 1, 2, 5, 8, 10 e 12 compõem o caminho crítico ($SLK_j = 0$), cujo tamanho é de 30 períodos. Nota-se ainda que a tarefa 3 possui uma folga de 13 períodos, o que significa que sua execução pode ser adiada em

Tabela 3.3: Exemplo do resultado da execução do CPM

Atividade	Modo	EST	EFT	LST	LFT	SLK = LFT - EFT
1	1	0	0	0	0	0
2	3	0	8	0	8	0
3	2	0	7	13	20	13
4	2	0	5	15	20	15
5	3	8	17	8	17	0
6	1	8	10	19	21	11
7	1	7	8	20	21	13
8	3	17	21	17	21	0
9	2	17	21	26	30	9
10	3	21	30	21	30	0
11	1	17	21	26	30	9
12	1	30	30	30	30	0

até 13 unidades de tempo além do seu EFT, sem comprometer a duração mínima do projeto. Como o método CPM considera apenas restrições de precedência e tempos de processamento, desconsiderando limitações de recursos, caso todas as tarefas sejam iniciadas o mais cedo possível, a duração mínima do projeto será de 30 períodos. Dessa forma, se alguma tarefa for programada fora da respectiva janela $[EFT, LFT]$, ocorrerá atraso no cronograma.

3.3 Representação da Solução

Para explorar o espaço de soluções de um problema de otimização, é fundamental definir uma forma adequada de representar computacionalmente cada solução. Neste trabalho, adota-se uma codificação para soluções viáveis do MRCPSP, representadas por $L(x) = (LJ, LM, LT)$, na qual a solução x é descrita por meio de três listas, sendo:

- LJ: Lista de tarefas
- LM: Lista de modos de execução
- LT: Lista de tempos de término

A solução x é construída de forma iterativa através de um esquema de sequenciamento. Ou seja, a cada iteração i escolhe-se uma atividade para adicionar a LJ. Durante o processamento, para cada tarefa j na posição i de LJ, seu modo de execução é atribuído de acordo com a posição correspondente em LM, e o tempo de término é verificado na posição i de LT.

A lista LJ deve ser viável em relação à restrição de precedência. Isto é, para uma tarefa j na posição i de LJ, todos os seus predecessores $p_j \in P_j$ e sucessores $s_j \in S_j$ devem estar nas posições i_p e i_s, de forma que $i_p < i < i_s$. A sequência das atividades define uma ordem de prioridades nas quais os recursos serão alocados.

Na Figura 3.2, é apresentado um exemplo de solução viável para o MRCPSP, codificada por meio das listas $L(x)$. A primeira atividade da lista, a tarefa 1, é

executada no modo 1 e termina no instante 0, indicando o início do projeto. Em seguida, a atividade 3, posicionada na segunda posição de LJ, é realizada no modo 2, com término no tempo 7. Observa-se, por exemplo, que a atividade 11, localizada na nona posição de LJ, só é iniciada e posicionada após a conclusão das atividades 5, 6 e 7, respeitando as restrições de precedência.

LJ	1	3	4	2	5	6	7	8	11	9	10	12
LM	1	2	2	3	3	1	1	3	1	2	3	1
LT	0	7	5	15	24	17	18	28	28	32	37	37

Figura 3.2: Exemplo de uma solução viável para o MRCPSP codificada por meio de listas.

Essa solução também pode ser visualizada na Figura 3.3 por meio de um cronograma, em que cada retângulo representa uma atividade, sua posição na linha do tempo indica o instante de início e duração. Essa representação gráfica torna evidente a estrutura de prioridades e a alocação temporal das atividades dentro do projeto. Atividades sem restrições de precedência, como as tarefas 4 e 3, podem ser executadas em paralelo, enquanto atividades com precedência direta, como a tarefa 6, devem ser iniciadas somente após a conclusão de sua predecessora 2.

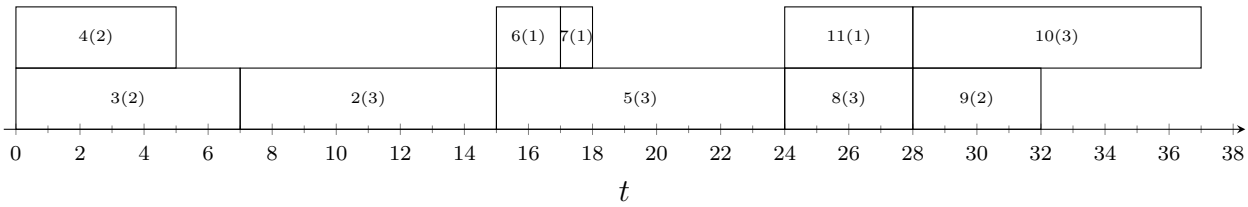


Figura 3.3: Representação visual da solução na Figura 3.2

3.4 Função de Avaliação

Para medir a qualidade das soluções, a principal métrica considerada é o *makespan*, representado como $C_{\max}(s)$, que indica o tempo total necessário para a conclusão do projeto em uma solução s. Contudo, essa métrica se torna inadequada quando a solução é inviável, pois o valor de avaliação dessa solução não pode ser melhor que soluções viáveis. Para contornar essa situação, Hartmann (2001) definiu a seguinte função de avaliação:

$$f(s) = \begin{cases} C_{\max}(s), & \text{se } L_k^{\nu}(\mu) = 0, \\ T + L^{\nu}(\mu), & \text{caso contrário.} \end{cases}$$

em que $L^{\nu}(\mu)$ representa o excesso de consumo de recursos não renováveis considerando o conjunto de modos atribuídos μ, sendo definida como:

$$L^{\nu}(\mu) = \sum_{k=1}^{R^{\nu}} \max\{0, \sum_{j=1}^{J} r_{j\mu(j)k}^{\nu} - R_k^{\nu}\}. \tag{3.4}$$

Se $L^{\nu}(\mu) > 0$, a demanda por recursos não renováveis excede a disponibilidade, tornando a solução inviável. Nesse caso, aplica-se uma penalização à avaliação da solução, que passa a ser dada pelo horizonte T do projeto somado ao módulo da quantidade de recursos não renováveis excedida.

Contudo, Alcaraz et al. (2003) identificou uma limitação nessa abordagem. O autor observou que, como o valor de avaliação de uma solução inviável depende apenas do excesso de demanda em relação aos recursos não renováveis, soluções com o mesmo excesso de recursos, mas diferentes *makespan*, acabam recebendo a mesma avaliação. Para corrigir esse problema, ele propôs uma nova função de avaliação:

$$f(s) = \begin{cases} C_{\max}(s), & \text{se } L_k^{\nu}(\mu) = 0, \\ \max(C_{max}^{viavel}(POP)) + C_{max}(s) - CP_{min} + L^{\nu}(\mu), & \text{caso contrário.} \end{cases}$$

em que $\max(C_{max}^{viavel}(POP))$ representa o maior *makespan* entre os sequenciamentos viáveis da população na geração atual, servindo como um limitante superior para todos os indivíduos daquela geração, e CP_{min} corresponde ao caminho crítico mínimal, obtido ao atribuir os modos de menor duração. De acordo com Alcaraz et al. (2003), essa nova abordagem apresentou melhores resultados em comparação à proposta de Hartmann (2001).

Nesta dissertação serão consideradas apenas soluções viáveis. Portanto, a métrica utilizada para medir a qualidade das soluções propostas é o C_{max}. Ele é avaliado pela atividade na qual é atribuído o maior tempo de término. Nesse ponto, justifica-se a inserção das atividades fictícias no projeto. Como elas delimitam o início e o término do sequenciamento, é possível, por meio delas, acessar diretamente quando o projeto será finalizado.

Seja FT_n o tempo de finalização da tarefa fictícia n. Como essa atividade possui duração nula, seu tempo de término é atribuído ao maior tempo de término de suas atividades predecessoras, ou seja:

$$FT_n = max\{FT_i : \forall i \in P_n\}.$$

De forma prática, o *makespan* do projeto, FT_n, pode ser obtido diretamente pela codificação da solução apresentada, utilizando a seguinte função:

$$f(x) = FT_n = LT[n], \tag{3.5}$$

em que n é a tarefa fictícia que representa o término do sequenciamento.

Dessa forma, $LT[n]$ fornece diretamente o tempo de término do projeto. Portanto, a partir do exemplo de solução da Figura 3.2, é possível identificar que o *makespan* do projeto é $LT[n] = LT[12] = 37$.

3.5 Heurística Construtiva: Esquema de Sequenciamento Serial

Proposto por Kelley (1963), o Esquema de Sequenciamento Serial (*Serial Scheduling Scheme* – SSS) é um método construtivo amplamente utilizado na geração de

cronogramas viáveis para o RCPSP. O método é composto por $n = 1, \cdots, |J|$ estágios, nos quais, a cada estágio, uma atividade é sequenciada, adicionada à sequência de execução, e programada, atribuído um tempo de início, com base nas restrições de precedência do projeto, em um critério de seleção e na disponibilidade de recursos.

Além disso, em cada estágio n, estão associados dois conjuntos disjuntos de atividades:

- O conjunto completo C_n, que contém todas as atividades já sequenciadas, ou seja, aquelas que fazem parte do cronograma parcial até o estágio n;
- O conjunto de decisão D_n, que reúne todas as atividades ainda não programadas cujos predecessores pertencem ao conjunto completo C_n.

O conjunto de decisão pode ser formalmente definido por:

$$D_n = \{j \mid j \notin C_n,\ P_j \subseteq C_n\} \tag{3.6}$$

em que P_j representa o conjunto de predecessores da atividade j.

A cada estágio n, uma atividade $j \in D_n$ é selecionada com base em um critério de seleção. Então, a essa atividade é atribuído o menor tempo de início viável, respeitando o término de todos os seus predecessores e a disponibilidade dos recursos renováveis durante toda a sua execução. Após ser programada, a atividade é removida de D_n e adicionada a C_n, indicando que a tarefa foi completada. O método encerra quando todas as atividades do projeto tiverem sido sequenciadas, isto é, quando $C_n = J$.

Seja πR_{kt} a capacidade residual do recurso renovável k no período t, definida por:

$$\pi R_{kt} = R_k^{\rho} - \sum_{j \in A_t} r_{jk}^{\rho} \tag{3.7}$$

em que A_t representa o conjunto das atividades em execução no período t e r_{jk}^{ρ} é o consumo do recurso k pela atividade j. Considere também $v(j)$ o valor da prioridade associado à atividade $j \in D_n$.

Assim, conforme apresentado por Fernandes e de Souza (2021), o SSS pode ser formalmente descrito de acordo com o Algoritmo 2.

Algorithm 2 Serial Scheduling Scheme (SSS)

1: $n \leftarrow 1$;
2: $C_n \leftarrow \emptyset$;
3: $\pi R_{kt} \leftarrow R_k^{\rho}$, para $k = 1, \ldots, K$, $t = 1, \ldots, T$;
4: **while** $|C_n| \leq |J|$ **do**
5: **Calcular** D_n;
6: $j^* \leftarrow \min_{j \in D_n}\{j | v(j) = \min_{i \in D_n}\{v(i)\}\}$;
7: $EFT_{j^*} \leftarrow \max\{FT_i \mid i \in P_{j^*}\} + d_{j^*}$;
8: $FT_{j^*} \leftarrow \min\{t : EFT_{j^*} \leq t \leq T, r_{j^*k} \leq \pi R_{k\tau}, \tau = t - d_{j^*}, \ldots, t, k = 1, \cdots, K^{\rho}\}$;
9: $C_{n+1} \leftarrow C_n \cup \{j^*\}$;
10: $\pi R_{kt} \leftarrow \pi R_{kt} - r_{j^*k}^{\rho}$, para $k = 1, \ldots, K$, $t = FT_{j^*} - d_{j^*}, \ldots, FT_{j^*}$;
11: **end while**

O Algoritmo 2 apresenta o pseudocódigo do algoritmo SSS. Este inicializa, nas linhas 1-3, o estágio atual $n = 1$, o conjunto de atividades completas C_n como vazio, e define a capacidade residual de cada recurso renovável πR_{kt} como igual à capacidade total disponível R_k^ρ, para todos os recursos e períodos. Em seguida, o laço das linhas 4-10 se repete enquanto o número de elementos do conjunto C_n for menor ou igual à quantidade de atividades do projeto.

Na linha 5 é calculado o conjunto de decisão D_n, garantindo o respeito às restrições de precedência. Em seguida, na linha 6, é selecionada uma atividade $j^* \in D_n$ que apresenta menor valor de $v(j), \forall j \in D_n$; em caso de empate, escolhe-se a atividade com menor índice. Determina-se, então, na linha 7, o menor tempo de término viável EFT_{j^*}, considerando o maior tempo de término entre seus predecessores somado à sua duração. A atividade é alocada na linha 8 no menor tempo t que respeite o consumo de recursos durante todo o período de execução de j^*. Após sua alocação, a atividade j^* é incluída no conjunto C_{n+1}, conforme linha 9, e as capacidades residuais dos recursos são atualizadas na linha 10. Esse processo é repetido até que todas as atividades tenham sido programadas, o que ocorre no estágio $n = |J|$, resultando em um cronograma viável para o projeto.

A Tabela 3.4 mostra, passo a passo, como a solução apresentada na Figura 3.2 foi construída por meio do SSS. Cada linha da tabela corresponde a uma iteração do algoritmo, enquanto as colunas indicam: o estágio atual n, o conjunto de decisões D_n, a tarefa j^* escolhida em D_n, o modo de execução selecionado m_{j^*}, o menor tempo de término EFT_{j^*}, o tempo de término efetivamente atribuído FT_{j^*} e o conjunto C_n de atividades concluídas até o momento. O sequenciamento obtido é viável em relação aos recursos renováveis e não renováveis e resulta em um *makespan* de 37 períodos. Para essa combinação de modos (ver Tabela 3.3), as tarefas 1, 2, 5, 8, 10 e 12 se configuram como críticas. Um ponto interessante a observar é que a tarefa 2, cuja janela de folga é $[EFT,LFT] = [8,8]$, foi programada no tempo $FT_2 = 15$. Esse atraso propagou-se para as atividades que dependiam direta ou indiretamente dela, fazendo com que também ultrapassassem suas respectivas janelas de folga, como no caso da atividade 5. Isso ilustra como atrasos em tarefas críticas podem comprometer a rede de precedências.

Tabela 3.4: Exemplo de execução passo-a-passo do Algoritmo SSS.

n	D_n	j^*	m_{j^*}	EFT_{j^*}	FT_{j^*}	C_n
1	{1}	1	1	0	0	{1}
2	{2,3,4}	3	2	8	15	{1,3}
3	{2,4}	4	2	7	7	{1,3,4}
4	{2,7}	2	3	5	5	{1,3,4,2}
5	{5,6,7}	5	3	17	24	{1,3,4,2,5}
6	{6,7,8}	6	1	10	17	{1,3,4,2,5,6}
7	{7,8,9,}	7	1	8	18	{1,3,4,2,5,6,7}
8	{8,9,11}	8	3	21	28	{1,3,4,2,5,6,7,8}
9	{9,10,11}	11	1	21	28	{1,3,4,2,5,6,7,8,11}
10	{9,10}	9	2	30	32	{1,3,4,2,5,6,7,8,11,9}
11	{10}	10	3	21	37	{1,3,4,2,5,6,7,8,11,9,10}
12	{12}	12	1	30	37	{1,3,4,2,5,6,7,8,11,9,10,12}

3.6 Heurísticas de Refinamento

Esta seção apresenta as heurísticas de refinamento adotadas nesta dissertação, cujo objetivo é a partir de uma solução previamente obtida, identificar soluções factíveis de melhor qualidade. Após uma breve conceituação das heurísticas de refinamento, bem como as principais estratégias de movimentação utilizadas para guiar o processo de melhoria das soluções. Descreve-se alguns métodos recorrentes na literatura para o MRCPSP, com destaque para o aprimoramento da atribuição de modos por meio do *Mode Assignment Compression* (Subseção 3.6.2) e para a melhoria do sequenciamento das atividades utilizando o *Forward–Backward Improvement* (Subseção 3.6.3). Além da descrição dos métodos e dos movimentos empregados, apresenta-se também a rotina de exploração VND (Subseção 3.6.4), amplamente utilizada em diversos problemas, discutindo-se seus principais conceitos e variações.

3.6.1 Busca Local

Heurísticas de refinamento, também conhecidas como técnicas de busca local (BL), consistem em explorar sistematicamente a vizinhança de uma solução s (Souza, 2008). A vizinhança $N(s)$ de s é definida como o conjunto de soluções que podem ser obtidas a partir de s usando pequenos movimentos. Cada solução $s' \in N(s)$ é denominada vizinha de s. A BL inicia-se a partir de uma solução inicial s e, a cada iteração, percorre-se $N(s)$ em busca de soluções com melhor valor de função objetivo. A busca local é encerrada quando não há mais vizinhos capazes de melhorar a solução corrente segundo o critério de avaliação adotado, resultando, assim, em um ótimo local.

Dentro dessa abordagem, duas estratégias se destacam para guiar a movimentação entre os vizinhos:

(i) *Best Improvement Method* (BI);

(ii) *First Improvement Method* (FI).

Na estratégia BI, a partir de uma solução s, todas as soluções vizinhas $s' \in N(s)$ são avaliadas, e a cada iteração o algoritmo move-se para aquela que apresenta o melhor valor de função objetivo em relação à solução corrente s. Esse procedimento é repetido até que não seja possível encontrar um vizinho melhor, isto é, o processo se encerra quando encontra um ótimo local. O Algoritmo 3 ilustra esse procedimento para um problema de minimização.

Algorithm 3 Best Improvement Method

Entrada: $N(.)$, s

1: $V = \{s' \in N(s) \mid f(s') < f(s)\}$
2: **while** $V \neq \emptyset$ **do**
3: $\quad s' = \text{argmin}\{f(s') \mid s' \in V\}$
4: $\quad s \leftarrow s'$
5: $\quad V = \{s' \in N(s) \mid f(s') < f(s)\}$
6: **end while**
7: **return** s

O método FI segue uma lógica semelhante ao BI, mas ao invés de avaliar toda a vizinhança da solução corrente de forma exaustiva, a busca por um vizinho é interrompida assim que encontra a primeira solução vizinha que apresenta melhoria em relação à atual e move-se para ela. Como toda a vizinhança é explorada ao não se encontrar um vizinho melhor, este método encerra-se no primeiro ótimo local encontrado. Vale destacar que os ótimos locais são melhores do que todos os seus vizinhos, mas que não representam necessariamente a melhor solução possível, isto é, o ótimo global (Voudouris et al., 2010).

Segundo Souza (2008), a definição da vizinhança desempenha um papel fundamental, pois, a partir de qualquer solução do espaço, deve ser possível alcançar qualquer outra em um número finito de passos, utilizando determinado tipo ou tipos de movimentos. Nas seções seguintes, apresentamos os principais métodos de busca local aplicados neste trabalho.

3.6.2 *Mode Assignment Compression* (MAC)

Apresentado em Muritiba et al. (2018) o procedimento *Mode Assignment Compression* (MAC) é um método de busca local que visa melhorar a combinação atual dos modos de execução das atividades de um projeto. Esse método atua apenas nos atributos da solução relacionados aos modos e tem como objetivo principal comprimir (ou reduzir) os tempos de processamento das atividades sem violar as restrições de recursos não renováveis.

A ideia central do MAC consiste em explorar trocas entre os modos de execução de pares de atividades relacionadas por precedência, conforme ilustrado no Algoritmo 4. Inicialmente, na linha 3, as atividades são embaralhadas aleatoriamente. Em seguida, para cada par de atividades (i,j) tal que $j \in P_i$, são avaliadas todas as possíveis combinações de modos de execução $(\bar{m},\bar{h}) \in M_i \times M_j \mid \bar{m} \neq m, \bar{h} \neq h$, sendo m e h os modos atualmente atribuídos às atividades i e j, respectivamente, como mostrado nas linhas 4–6. A troca entre os modos m e h será considerada apenas se a nova combinação resultar em uma redução na soma dos tempos de processamento (linha 7) e se for viável em relação aos recursos não renováveis, verificada na linha 8. Caso ambas as condições sejam atendidas, os modos são trocados (linhas 9-10) e o processo continua. Esse procedimento é repetido iterativamente até que não sejam mais encontradas trocas viáveis que resultem em melhorias.

Algorithm 4 Mode Assignment Compression (MAC)

repeat
  $stop \leftarrow$ **true**;
  SHUFFLE(J);
  for all $i \in J$ **do**
    for all $j \in J \mid j \in P_i$ **do**
      for all $(\bar{m},\bar{h}) \in M_i \times M_j \mid \bar{m} \neq m, \bar{h} \neq h$ **do**
        if $p_{im} + p_{jh} > p_{i\bar{m}} + p_{j\bar{h}}$ **then**
          if SWAPFEASIBLE$((m,\bar{m});(h,\bar{h}))$ **then**
            SWAP$(m,\bar{m})$;
            SWAP$(h,\bar{h})$;
            $stop \leftarrow$ **false**;
          end if
        end if
      end for
    end for
  end for
until *stop*
return S

Segundo Muritiba et al. (2018), o procedimento MAC não realiza o agendamento das atividades, ou seja, não determina seus instantes de início ou término, mas apenas ajusta os modos de execução das tarefas para obter uma configuração mais eficiente em termos de tempo e utilização de recursos. Entretanto, ainda de acordo com os autores, o MAC não é indicado em instâncias em que os recursos são ilimitados, pois, nessas situações, o procedimento tende a retornar a mesma atribuição de modos.

3.6.3 *Forward–Backward Improvement*

De forma geral, os métodos heurísticos de construção de cronogramas factíveis para resolver o RCPSP que utilizam rotinas como SSS resolvem o problema em duas fases principais (Li e Willis, 1992):

Fase 1: nesta fase constrói-se uma lista ordenada de atividades a partir de critérios específicos ou regras heurísticas. Essa lista define a ordem em que as atividades serão consideradas na etapa seguinte;

Fase 2: as atividades são processadas e programadas sequencialmente, recebendo tempo de início por meio da ordem apresentada na lista.

Dentro da segunda fase, destacam-se ainda duas técnicas distintas para a programação das atividades:

(a) *Forward-loading*: as atividades são iniciadas o mais cedo possível;

(b) *Backward-loading*: as atividades são programadas para iniciar o mais tarde possível, a partir da estimativa de duração do projeto.

Li e Willis (1992) observaram empiricamente que procedimentos de agendamento que empregam *backward-loading* às vezes podem gerar cronogramas mais eficientes do que os produzidos por procedimentos que empregam uma técnica *forward-loading*. Os autores ainda observam que, para casos em que as disponibilidades de recursos variam ao longo do tempo, técnicas *backward-loading* geralmente podem levar a cronogramas mais curtos.

Porém, essa técnica apresenta diversas limitações práticas: pode gerar atividades alocadas no passado quando a data estimada de término é subestimada; torna todas as atividades críticas, de modo que qualquer atraso impacta diretamente a duração do projeto; em alguns casos, resulta em um cronograma mais longo do que o obtido pelo *forward-loading*; e enfrenta sérias dificuldades na presença de recursos não renováveis, duplamente restritos ou com disponibilidade variável, frequentemente produzindo cronogramas inviáveis (Li e Willis, 1992).

Um ponto adicional observado pelos autores está no padrão de utilização dos recursos. Enquanto no *forward-loading* o consumo tende a ser mais intenso no início e reduzir-se ao final, no *backward-loading* ocorre o inverso; o consumo é reduzido nas etapas iniciais e concentrado na fase final. Essa assimetria gera o que os autores chamam de *hysteresis loop*, ou ciclo de histerese, caracterizado por períodos de má utilização dos recursos que poderiam ser evitados caso o consumo fosse distribuído de forma mais uniforme ao longo do horizonte do projeto.

Com base nessas observações, Li e Willis (1992) propuseram um procedimento iterativo, denominado *forward–backward improvement* (FBI), que combina as técnicas *forward* e *backward-loading* para gerar cronogramas curtos e viáveis. O processo iterativo é definido em Fernandes (2017) pelos passos:

Passo 1: Inicialmente, estabelece-se uma lista de atividades e constrói-se um sequenciamento *forward*, por exemplo pelo SSS, no qual cada atividade recebe o menor tempo de início viável, respeitando as restrições do problema. A partir dessa programação, obtém-se a duração do projeto C^f_{max}, que passa a ser utilizada como limite de referência;

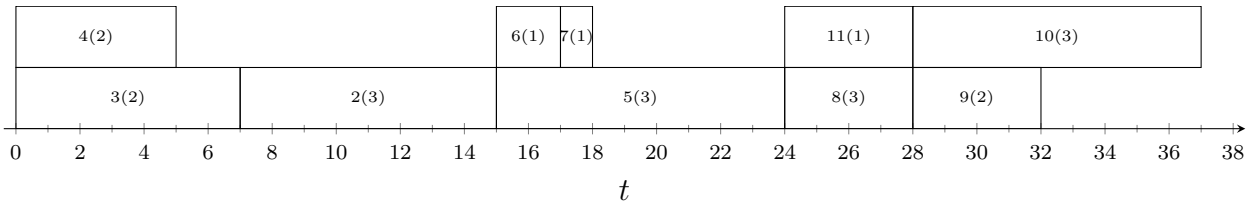
Passo 2: Em seguida, as atividades são reordenadas de forma crescente, de acordo com seus tempos de término definidos no sequenciamento *forward*. A partir dessa ordenação, percorre-se a lista no sentido inverso, da última atividade para a primeira, e, para cada atividade j, atribui-se o tempo de término FT_j mais tardio possível, conforme a Equação 3.8. Consequentemente, o tempo de início é atualizado como $ST_j = FT_j - d_j$. Ao final dessa etapa, obtém-se o valor C^b_{max}, calculado como $C^b_{max} = C^f_{max} - ST_1$, em que 1 corresponde à atividade inicial do projeto. Caso $C^b_{max} = C^f_{max}$, significa que não há possibilidade de redução adicional na duração do projeto e o procedimento é encerrado. Por outro lado, se $C^b_{max} < C^f_{max}$, o processo prossegue para a próxima etapa;

$$FT_j = \begin{cases} C_{\max}(s), & \text{se } S_j = \{\emptyset\}, \\ \min\{ST_i : \forall i \in S_j\}, & \text{caso contrário.} \end{cases} \tag{3.8}$$

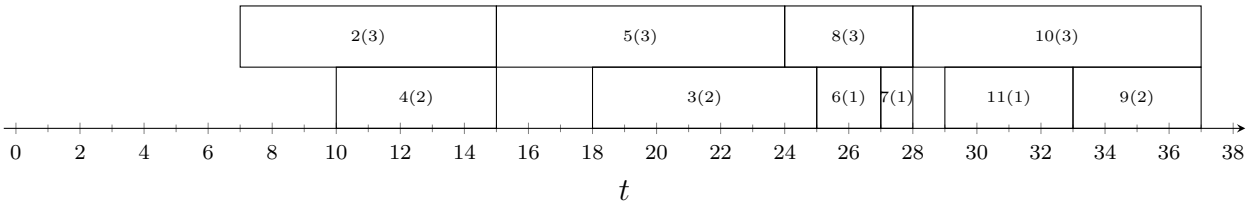
Passo 3: As atividades são então novamente ordenadas, agora de forma crescente segundo os tempos de início obtidos no sequenciamento *backward*. A partir dessa lista, percorrem-se as atividades da primeira até a última, aplicando-se novamente o procedimento *forward* descrito no passo inicial. Dessa forma, é obtido um novo valor de C^f_{max}. Se a igualdade $C^f_{max} = C^b_{max}$ for satisfeita, conclui-se que não há mais possibilidade de melhoria, encerrando o processo iterativo. Caso contrário, o método retorna ao passo anterior, repetindo a alternância entre as construções *forward* e *backward*.

Ao término da execução do procedimento, o sequenciamento final é definido a partir da última programação *forward*, e a duração final do projeto $C_{\max}$ corresponde ao último valor de C^f_{max} calculado.

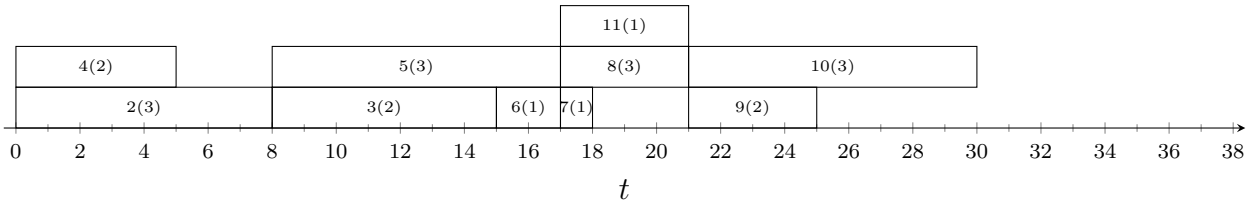
Para ilustrar o funcionamento da heurística FBI, a Figura 3.4 apresenta, usando um gráfico de Gantt, as iterações do sequenciamento obtido a partir da solução mostrada na Figura 3.2. Na primeira iteração (Figura 3.4 a), tem-se a solução inicial construída pelo sequenciamento *forward* SSS, cujo *makespan* é de 37 períodos. A partir dessa solução, na segunda iteração (Figura 3.4 b), as atividades são ordenadas de acordo com o tempo de término e reprogramadas pelo sequenciamento *backward*, resultando em um *makespan* de 30 períodos. Na terceira iteração (Figura 3.4 c), as atividades são novamente programadas pelo sequenciamento *forward*, mantendo o mesmo *makespan* de 30 unidades. Por fim, na quarta iteração (Figura 3.4 d), aplica-se novamente o sequenciamento *backward*, sem redução adicional do *makespan*. Esse resultado indica que não é possível melhorar a solução, encerrando-se o processo. Assim, a configuração final é representada pela solução da Figura 3.4 c.



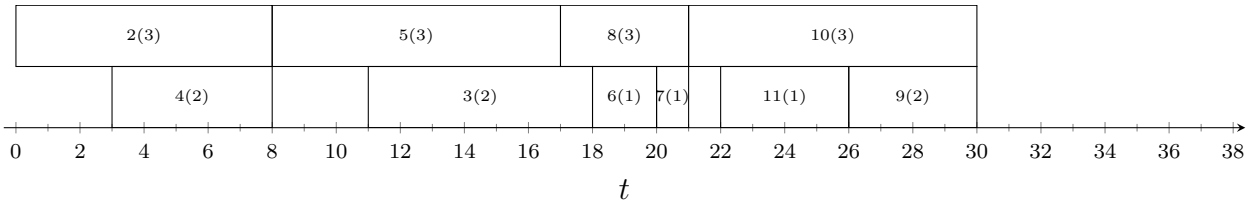
a) Gráfico de Gantt refrente à primeira iteração do procedimento FBI.



b) Gráfico de Gantt referente à segunda iteração do procedimento FBI.



c) Gráfico de Gantt referente à terceira iteração do procedimento FBI.



d) Gráfico de Gantt referente à quarta iteração do procedimento FBI.

Figura 3.4: Exemplo de sequenciamento obtido pela heurística FBI

LJ	1	2	4	5	3	8	6	7	10	11	9	12
LM	1	3	2	3	2	3	1	1	3	1	2	1
LT	0	8	5	17	15	21	17	18	30	21	25	30

Figura 3.5: Sequenciamento *forward* final obtido pela heurística FBI.

A solução obtida, codificada por listas na Figura 3.5, mostra que todas as tarefas, incluindo as críticas para essa configuração de modos (tarefas 1, 2, 5, 8, 10 e 12), foram programadas dentro de suas respectivas janelas de folga $[EFT,LFT]$, conforme mostra a Tabela 3.3. Dessa forma, nenhuma atividade sofreu atraso, resultando em um cronograma com a mesma duração do que o estimado pelo CPM. Vale destacar que o CPM fornece um limite inferior para a conclusão do projeto, pois é derivado

da relaxação das restrições de recursos do problema. Assim, o fato de a solução da Figura 3.5 atingir esse limite indica que, para a configuração de modos considerada, o cronograma não pode ser melhorado.

3.6.4 Variable Neighborhood Descent (VND)

Proposto por Mladenović e Hansen (1997), o método *Variable Neighborhood Descent* (VND) é uma heurística de refinamento que explora o espaço de busca por meio da troca sistemática entre diferentes estruturas de vizinhança. O VND parte do princípio de que uma solução localmente ótima para uma determinada vizinhança não é, necessariamente, ótima para outras vizinhanças (Ribeiro et al., 2023).

Segundo Hansen et al. (2019), soluções localmente ótimas em uma ou mais vizinhanças frequentemente fornecem informações sobre o ótimo global. Portanto, segundo os mesmos autores, é mais provável que um ótimo global seja alcançado ao explorar diferentes vizinhanças do que com uma única estrutura.

Seja S o espaço de soluções viáveis e $f : S \to \mathbb{R}$ a função objetivo de um problema de minimização. Suponha um conjunto ordenado de k estruturas de vizinhança $V = \{N^{(1)}, N^{(2)}, \ldots, N^{(k)}\}$, em que cada $N^{(i)}(s) \subseteq S$ representa a vizinhança da solução s segundo a i-ésima estrutura. A escolha e ordenação dessas vizinhanças são fatores cruciais para o desempenho do método. Segundo Souza (2008), uma abordagem comum é ordenar as vizinhanças com base em sua complexidade de exploração, priorizando inicialmente aquelas que usam movimentos de menor complexidade.

A ideia central do VND consiste em aplicar busca local sequencialmente em cada vizinhança, reiniciando o processo a partir da primeira vizinhança sempre que uma melhoria é identificada. O algoritmo é finalizado quando nenhuma melhoria é encontrada em nenhuma das estruturas de vizinhança consideradas. O pseudocódigo desse método é apresentado no Algoritmo 5.

A busca é iniciada a partir de uma solução inicial s e de um conjunto de vizinhanças V. O índice da estrutura a ser explorada é inicializado com valor 1 na linha 2. O laço de repetição entre as linhas 3 e 11 percorre todas as vizinhanças de forma consecutiva. Na linha 4, aplica-se uma busca local a solução atual, retornando o melhor vizinho s' pertencente à vizinhança $N^{(k)}(s)$ correspondente ao índice k. A condição da linha 5 verifica se s' representa uma melhoria em relação a solução incumbente s. Em caso positivo, a solução é atualizada, e o índice da vizinhança é reinicializado para 1 na linha 7. Caso contrário, a linha 9 incrementa o índice k, permitindo que o algoritmo investigue a próxima vizinhança. Por fim, a linha 12 retorna uma solução que é localmente ótima em relação a todas as vizinhanças exploradas.

Algorithm 5 VND

Entrada: $V(\cdot), s$

1: $r \leftarrow$ Num_Vizinhanças
2: $k \leftarrow 1$
3: **while** $k \leq r$ **do**
4: Encontre o melhor vizinho $s' \in N^{(k)}(s)$
5: **if** $f(s') < f(s)$ **then**
6: $s \leftarrow s'$
7: $k \leftarrow 1$
8: **else**
9: $k \leftarrow k + 1$
10: **end if**
11: **end while**
12: **return** s

O procedimento descrito pelo pseudocódigo 5 é denominado *Basic* VND (B-VND), segundo Duarte et al. (2018); Hansen et al. (2019). Outras estruturas de implementação de VND também são apresentadas nessas referências, utilizando diferentes regras para selecionar a próxima estrutura de vizinhança a ser explorada caso ocorra melhoria na solução corrente:

- *Basic* VND (B-VND): se há melhoria, retorna para a primeira estrutura vizinhança da lista;
- *Pipe* VND (P-VND): se há melhoria, continua a exploração na mesma estrutura de vizinhança;
- *Cyclic* VND (C-VND): independente se houver ou não melhoria, a busca continua na próxima estrutura de vizinhança;
- *Union* VND (U-VND): uma única vizinhança é obtida pela união de todas as vizinhanças predefinidas. Ou seja, não percorre vizinhanças uma a uma, mas na mesma iteração explora todas as vizinhanças ao mesmo tempo.

Em Duarte et al. (2018) é realizado um estudo empírico dessas diferentes variantes do VND para resolver o problema do caixeiro viajante (TSP). Os autores observaram que o U-VND apresentou desempenho ligeiramente superior aos demais métodos no que diz respeito aos melhores valores médios de solução obtidos. No entanto, esse ganho em qualidade foi acompanhado de um custo elevado em termos de tempo computacional, já que o U-VND, a cada iteração, realiza uma exploração de uma porção muito maior do espaço de soluções. Esse comportamento torna o método adequado para alcançar boas soluções finais, mas pouco eficiente quando o tempo de execução é um fator crítico.

Além disso, ao comparar as versões que recentralizam a busca em seu *loop* interno (B-VND, P-VND e C-VND), os autores verificaram que o B-VND foi o mais eficaz em termos de qualidade de solução, produzindo os melhores valores médios. No que diz respeito ao tempo de processamento médio consumido por cada um para atingir tais resultados, o P-VND se destacou como o mais rápido, seguido do B-VND e, por

último, do C-VND, que foi o mais lento entre os três. Contudo, a diferença de tempo entre P-VND e B-VND mostrou-se pouco significativa, especialmente considerando a superioridade do B-VND em termos de qualidade de solução.

3.7 Metaheurísticas

Esta seção apresenta as principais metaheurísticas utilizadas neste projeto. Serão discutidas, inicialmente, as principais características do GRASP (Subseção 3.7.1) e do PR (Subseção 3.7.2), seguida de suas variações *Interior Path Relinking* (Subseção 3.7.3) e *Exterior Path Relinking* (Subseção 3.7.4). Na sequência, apresentam-se algumas extensões do PR (Subseção 3.7.5), bem como sua hibridização com o GRASP (Subseção 3.7.6), destacando o papel central do conjunto elite nesse processo (Subseção 3.7.7). Por fim, a Subseção 3.7.8 apresenta a metaheurística VNS, discutindo sua estrutura geral, bem como suas variantes e extensões (Subsubseção 3.7.8.1).

3.7.1 GRASP

A metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*) é um procedimento amplamente aplicado na resolução de problemas de otimização combinatória (Festa e Resende, 2009). Proposto em Feo e Resende (1995), este método é baseado em um processo iterativo composto por duas fases principais: (*i*) construção e (*ii*) busca local. No fim das iterações, o algoritmo retorna a melhor solução obtida ao longo do processo. O pseudocódigo descrito no Algoritmo 6 ilustra esse procedimento para um problema de minimização.

As iterações do GRASP ocorrem nas linhas 2-9 do Algoritmo 6 e são encerradas quando atingirem o número máximo de iterações $GRASP_{max}$. A linha 3 corresponde à fase de construção do GRASP, enquanto a linha 4 representa a fase de busca local. Caso uma solução melhor seja encontrada, a solução incumbente é atualizada na linha 6. A seguir, apresentamos uma descrição mais detalhada de cada fase.

Algorithm 6 Algoritmo GRASP

Entrada: α, $Grasp_{max}$

1: $f^\star \leftarrow \infty$
2: **for** $Iter = 1, 2, \ldots, GRASP_{\max}$ **do**
3: s ← Construção $(g(.), \alpha)$
4: BuscaLocal $(f(.), N(.), s)$
5: **if** $f(s) < f^\star$ **then**
6: $s^\star \leftarrow s$
7: $f^\star \leftarrow f(s)$
8: **end if**
9: **end for**
10: $s \leftarrow s^\star$
11: **return** s

Fase de Construção - GRASP

Na fase de construção, parte-se de uma solução s inicialmente vazia e, a cada iteração, acrescenta-se um elemento de modo guloso e aleatório. Para isso, identifica-se uma lista de candidatos LC e ordenam-se seus elementos por uma função de avaliação gulosa $g(\cdot)$, que estima o benefício de incluir cada candidato na solução parcial. A partir desses valores, define-se a lista de candidatos restrita LCR, contendo os melhores elementos de LC. Em seguida, seleciona-se aleatoriamente um elemento da LCR para compor s. O nível de aleatoriedade do procedimento é controlado por um parâmetro $\alpha \in [0,1]$. Esse processo é repetido até que a solução esteja completamente construída.

No Algoritmo 7 é descrita essa fase da construção. O laço de repetição das linhas 2-9 é repetido até que a LC esteja vazia; em outras palavras, esse processo é repetido até que uma solução s seja totalmente construída. Na linha 5 é formada a LCR. Em seguida, na linha 6, um elemento $t \in LCR$ é escolhido aleatoriamente e adicionado à solução parcial s na linha 7. Por fim, atualiza-se a LC na linha 8 e reinicia-se o processo. Retornando ao final uma solução s construída.

Algorithm 7 Construção

Entrada: α, s

1: Inicializar LC
2: **while** $LC \neq \emptyset$ **do**
3: $\quad t_{\min} = \min\{g(t) \mid t \in LC\}$
4: $\quad t_{\max} = \max\{g(t) \mid t \in LC\}$
5: $\quad \text{LCR} = \{t \in LC \mid g(t) \leq t_{\min} + \alpha \cdot (t_{\max} - t_{\min})\}$
6: $\quad$ Selecionar, aleatoriamente, um elemento $t \in \text{LCR}$
7: $\quad s \leftarrow s \cup \{t\}$
8: $\quad$ Atualizar LC
9: **end while**
10: **return** s

Fase de Busca Local - GRASP

Após a construção da solução inicial, aplica-se procedimento de busca local, linha 4 do Algoritmo 6, com o objetivo de refiná-la, explorando soluções vizinhas na tentativa de encontrar um ótimo local. Para isso, define-se uma estrutura de vizinhança $N(s)$, que representa todas as soluções que podem ser obtidas a partir da solução atual s por meio de pequenas modificações. No Algoritmo 8 é descrito um procedimento básico de busca local.

Algorithm 8 BuscaLocal

Entrada: $N(.)$, s

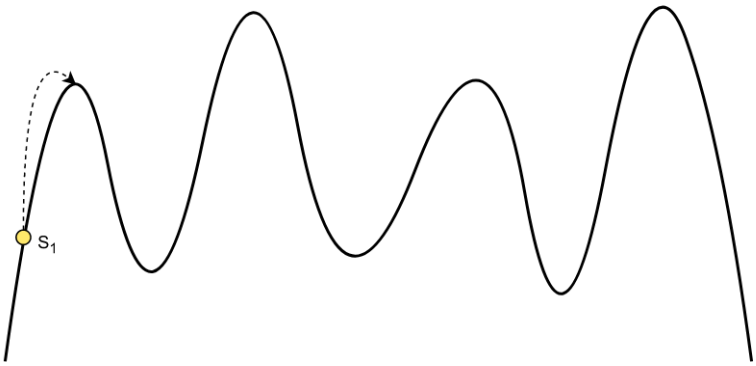
1: $V = \{s' \in N(s) \mid f(s') < f(s)\}$
2: **while** $V \neq \emptyset$ **do**
3: $\quad s' = \text{argmin}\{f(s') \mid s' \in V\}$
4: $\quad s \leftarrow s'$
5: $\quad V = \{s' \in N(s) \mid f(s') < f(s)\}$
6: **end while**
7: **return** s

Comentários gerais - GRASP

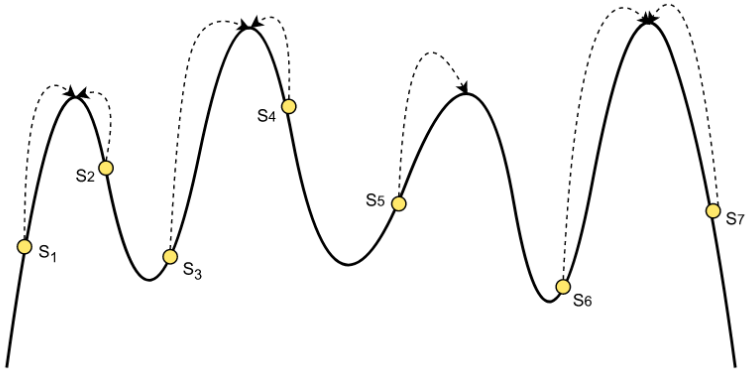
Segundo Feo e Resende (1995), uma característica atraente da metodologia GRASP é sua facilidade de implementação, uma vez que requer poucos parâmetros a serem ajustados, como o fator de aleatoriedade α e o número máximo de iterações $GRASP_{\max}$. Os autores também discutem o impacto que cada um desses parâmetros pode ter na qualidade das soluções obtidas. O parâmetro α controla o nível de gulosidade e aleatoriedade do procedimento. Para $\alpha = 0$, o procedimento torna-se puramente guloso, selecionando sempre o melhor candidato, enquanto, para $\alpha = 1$, a escolha entre todos os candidatos é completamente aleatória. Valores intermediários permitem um equilíbrio entre essas duas abordagens e contribuem para que as soluções geradas sejam diversas e de qualidade. Já o número de iterações $GRASP_{\max}$ está relacionado à intensificação da busca, permitindo uma exploração mais ampla do espaço de soluções.

Essencialmente, o GRASP pode ser interpretado como uma técnica de amostragem sobre o espaço de soluções viáveis, na qual cada iteração constrói uma nova solução inicial e a refina por meio da busca local. A repetição desse processo possibilita explorar diferentes regiões do espaço de busca, aumentando as chances de encontrar soluções de alta qualidade. Dessa forma, o GRASP combina as vantagens dos algoritmos puramente gulosos com os de procedimentos aleatórios de construção de soluções (Souza, 2008).

No exemplo apresentado pela Figura 3.6, extraído de Lozano-Osorio et al. (2023), os autores, de forma visual, comparam a vantagem do GRASP em relação a um algoritmo puramente guloso combinado com BL. Neste exemplo, no caso do algoritmo guloso, apenas uma solução inicial s_1 é gerada, e a busca local converge para um ótimo local e não explora outras soluções. Já no GRASP, a fase de construção gera múltiplas soluções iniciais diversificadas $(s_1, \ldots, s_7)$, explorando diferentes regiões do espaço de busca. Por exemplo, mesmo que s_6 seja a pior solução inicial em termos de qualidade, a aplicação da busca local sobre ela pode levar a uma solução final melhor do que a obtida pelo algoritmo guloso, exemplificando como a diversificação aumenta a probabilidade de alcançar ótimos locais de maior qualidade.



a)Exemplo de um algoritmo guloso



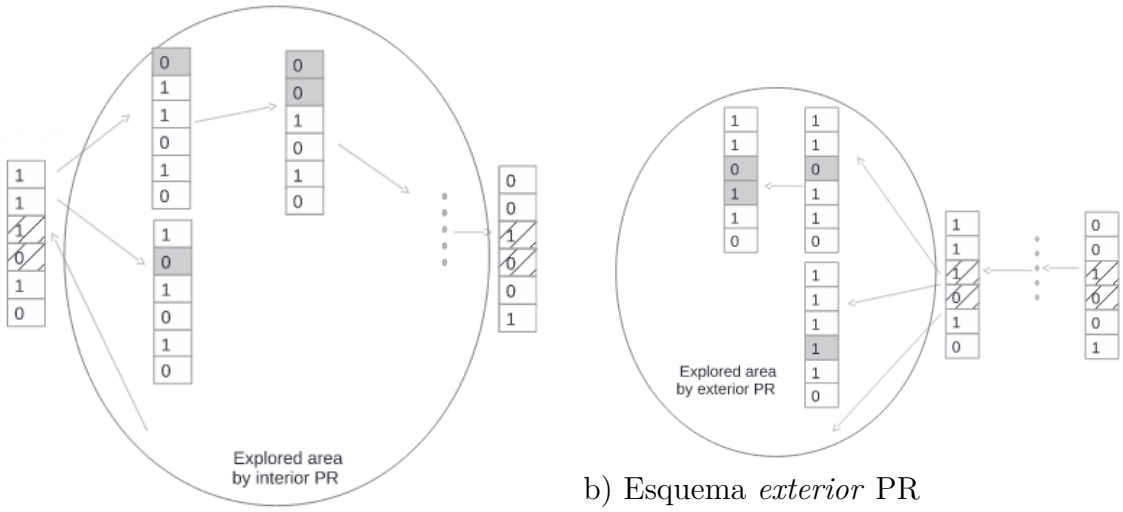
b) Exemplo da performance do GRASP

Figura 3.6: Comparação entre os desempenhos de algoritmos puramente gulosos e do GRASP, para um problema de maximização, extraído de Lozano-Osorio et al. (2023).

3.7.2 *Path Relinking*

O procedimento *Path Relinking* (reconexão por caminhos - PR) foi introduzido por Glover (1997), originalmente proposto como uma estratégia de intensificação e diversificação no contexto das metaheurísticas *Tabu Search* e *Scatter Search*. A técnica consiste em criar caminhos explorando trajetórias que conectam pares de soluções elite de alta qualidade, com o objetivo de encontrar soluções potencialmente melhores. Segundo Laguna et al. (2023), como soluções de boa qualidade frequentemente compartilham atributos em comum, o PR parte da premissa de que soluções ainda melhores podem ser encontradas ao explorar vizinhanças situadas entre ou ao redor dessas soluções de alta qualidade.

De acordo com Rodriguez et al. (2017), duas abordagens principais de implementação do PR são estudadas na literatura: o *Interior Path Relinking* (IPR) e o *Exterior Path Relinking* (EPR). De modo geral, o IPR busca intensificar a exploração do espaço de busca contido entre duas soluções de alta qualidade, enquanto o EPR investiga regiões fora desse caminho direto, diversificando a busca com o intuito de não ficar preso em ótimos locais. Rodriguez et al. (2017) propuseram uma implementação híbrida do GRASP com EPR para resolver o *Multidimensional Two-Way Number Partitioning* – MDTWNP. No estudo, os autores investigam diferentes es-



a) Esquema *interior* PR

b) Esquema *exterior* PR

Figura 3.7: Representação dos esquemas *interior* e *exterior Path Relinking* extraído de Rodriguez et al. (2017). A figura a) ilustra o espaço de exploração entre duas soluções para o IPR, enquanto a figura b) evidencia qua a vizinhança explorada do EPR investiga regiões exteriores as duas soluções.

tratégias do PR, comparando o PR interior e o PR exterior. Essa comparação é ilustrada na Figura 3.7, que, de forma visual, mostra a diferença conceitual entre os dois métodos.

Na Figura 3.7 a), observa-se o PR interior, em que a trajetória é construída progressivamente dentro da região limitada por duas soluções, uma inicial e uma guia. Esse procedimento gera um caminho mais restrito, explorando apenas soluções intermediárias próximas a ambas as extremidades. Já na Figura 3.7 b), é mostrado o PR exterior, no qual o caminho é construído de forma expandida, explorando áreas externas às soluções inicial e guia. Essa abordagem amplia a diversidade das soluções visitadas, aumentando as chances da busca não ficar presa em ótimos locais e de encontrar soluções de maior qualidade. Assim, na comparação entre as duas estratégias, Rodriguez et al. (2017) destaca que, enquanto o PR interior foca em intensificação ao explorar soluções dentro do espaço limitado por duas soluções, o PR exterior foca na diversificação, permitindo alcançar regiões do espaço de busca que não seriam acessadas pelo método tradicional.

3.7.3 *Interior Path Relinking* (IPR)

Na abordagem do IPR, um caminho é construído a partir de uma solução inicial S^i, que é progressivamente transformada em uma solução guia S^g por meio da introdução gradual de atributos presentes em S^g. Essa transformação ocorre iterativamente, com a substituição de atributos de S^i por atributos de S^g, resultando em uma sequência de soluções intermediárias. Os atributos comuns a ambas as soluções são mantidos inalterados ao longo do processo.

O Algoritmo 9 apresenta um pseudocódigo genérico do IPR aplicado a um par de soluções (S^i, S^g), considerando um problema de minimização. Inicialmente, calcula-se

o grau de diversidade ou diferença simétrica $|\Delta(S^i, S^g)|$ entre S^i e S^g , que representa o número de movimentos necessários para transformar S^i em S^g, em que $\Delta(S^i, S^g) = \{j : S^i_j \neq S^g_j\}$, representando as posições, por exemplo, em que as duas soluções diferem. A partir desse conjunto, um caminho de soluções é construído ligando S^i e S^g.

Algorithm 9 Path Relinking Interior (IPR)

Entrada: S^i, S^g

1: Calcular $\Delta(S^i, S^g)$ ▷ diferença simétrica;
2: **while** $\Delta(S^i, S^g) \neq \emptyset$ **do**
3: $m^* \leftarrow \arg\min\{f(S^i \oplus m) : m \in \Delta(S^i, S^t)\}$;
4: $S^i \leftarrow S^i \oplus m^*$;
5: $\Delta(S^i, S^g) \leftarrow \Delta(S^i, S^g) \setminus \{m^*\}$;
6: **end while**

Em cada etapa do procedimento são verificados todos os movimentos $m \in \Delta(S^i, S^g)$ da solução atual S^i e selecionado o movimento que melhor contribui para solução, isto é, que minimiza $f(S^i \oplus m)$ na linha 3. Para cada posição $m \in \Delta$, definimos $S^i \oplus m$ como a solução obtida a partir S^i ao mudar o elemento na m-ésima posição pela sua correspondente em S^g. O melhor movimento m^* é realizado, resultando na solução $S^i \oplus m^*$ definida na linha 4. O conjunto de movimentos é atualizado na linha 5. O procedimento termina quando $|\Delta(S^i, S^g)| = 0$.

Vale destacar que, após cada solução intermediária, procedimentos de busca local podem ser incorporados para intensificar a exploração do espaço de soluções. Em Muritiba et al. (2018), por exemplo, a busca local é aplicada a cada solução intermediária gerada, refinando progressivamente o caminho construído. Já em Lozano-Osorio et al. (2023), a intensificação ocorre sobre a melhor solução encontrada no percurso.

Além disso, diferentes variações do PR têm sido exploradas em conjunto com outras metaheurísticas. O PR interior, por exemplo, foi aplicado por Peidro et al. (2024) ao problema de *Uncapacitated Facility Location* (UFLP), em combinação com a *Tabu search*. De maneira semelhante, Calvete et al. (2025) integraram o IPR ao *Scatter Search* para tratar problemas *lineares bilevel*, enquanto Benfedel et al. (2025) propuseram sua hibridização com *Simulated Annealing* para resolver o *Two-Echelon Production Routing Problem with Multiple Delivery Modes* (2E-PRP-MDM).

3.7.4 *Exterior Path Relinking* (EPR)

O EPR seque uma ideia oposta ao do IPR. Dadas duas soluções, uma inicial S^i e a outra guia S^g, o EPR iterativamente inclui elementos em S^i que não estão em S^g, com o objetivo de alcançar novas soluções que são diversas em relação a ambas as soluções. Diferentemente do IPR, os atributos comuns das soluções são alterados durante a exploração e aqueles que permanecem fixos são os atributos distintos.

Para adaptar o Algoritmo 9 para o EPR, basta considerar que os movimentos serão realizados sobre o conjunto $\Delta(S^i, S^g)'' = \{j : S^i_j = S^g_j\}$, que representa as posições em que os atributos de S^i e S^g coincidem nas duas soluções.

O EPR tem sido aplicado com sucesso em diferentes contextos. Uribe et al. (2024), por exemplo, abordam o *bi-objective Double Floor Corridor Allocation Problem* (bDF-

CAP), utilizando o PR como algoritmo principal e combinando as estratégias EPR e IPR para a construção e exploração das soluções. Já, Pérez-Peló et al. (2020) integram o GRASP ao EPR para resolver o *α-separator problem*. Na mesma linha, Duarte et al. (2015) e Sánchez-Oro et al. (2021) também propuseram a combinação entre GRASP e EPR, aplicada respectivamente ao *Differential Dispersion Problem* e ao *Multiobjective k-Balanced Center Location Problem*.

3.7.5 *Extensões do Path Relinking*

A estratégia de seleção de movimentos apresentada pelo Algoritmo 9 de natureza gulosa, selecionando sempre o melhor movimento, pode ser computacionalmente custosa, pois exige a geração e a avaliação de todas as soluções viáveis a cada passo do caminho. Para contornar esse custo, uma alternativa comumente utilizada é a seleção aleatória de movimentos (Laguna et al., 2025), que, além de reduzir o tempo computacional, também contribui para aumentar a diversidade dos caminhos construídos. Como observado em Resende e Ribeiro (2016), esta estratégia pode ser útil quando o PR é aplicado ao mesmo par de soluções mais de uma vez.

Além das estratégias para seleção de movimentos, outras variações na forma de construção do caminho entre S^i e S^g também são discutidas na literatura, como descrito a seguir:

- *Forward relinking*: considera-se a pior entre as soluções x^1 e x^2 tal que $f(x^1) \geq f(x^2)$ como ponto de inicial e a outra como solução guia, isto é $S^i = x^1$ e $S^g = x^2$, buscando trajetórias que partem de soluções de baixa qualidade em direção a soluções melhores. Em que $f(i)$ representa uma função objetivo a ser minimizada;

- *Backward relinking*: adota-se a melhor entre as soluções x^1 e x^2 tal que $f(x^1) \leq f(x^2)$ como solução inicial, transformando-a gradualmente na pior, isto é $S^i = x^2$ e $S^g = x^1$;

- *Back and forward relinking*: explora duas trajetórias distintas, uma a partir de x^1 em direção a x^2 e outra na direção oposta, aumentando a diversidade das soluções intermediárias geradas;

- *Mixed relinking*: ambas as trajetórias, partindo de x^1 e x^2, são exploradas simultaneamente até encontrarem uma solução intermediária equidistante entre elas;

- *Greedy randomized adaptive relinking*: em vez de selecionar sempre o melhor movimento disponível, escolhe-se aleatoriamente um movimento promissor de uma lista de candidatos;

- *Truncated relinking:* a trajetória completa entre x^1 e x^2 não é totalmente explorada. Apenas parte do caminho é percorrida.

De acordo com Resende e Ribeiro (2003), explorar simultaneamente duas trajetórias distintas na forma *Back and forward relinking*, para cada par de soluções, exige aproximadamente o dobro do tempo em comparação à exploração de apenas uma

trajetória. Os autores destacam ainda que, quando se opta por investigar apenas uma trajetória, os melhores resultados são alcançados ao iniciar o processo de construção do caminho a partir da melhor dentre as duas soluções, ou seja, utilizando o *backward relinking*. Isso ocorre, segundo o autor, porque a vizinhança da solução inicial é explorada de forma mais intensa que a da solução guia, ampliando as chances de identificar soluções promissoras. Em outras palavras, o algoritmo concentra a busca na vizinhança da solução mais promissora, o que justifica a interrupção antecipada da trajetória do *Truncated relinking* antes de se alcançar a solução destino, pois evita custos computacionais adicionais, sem comprometer a qualidade dos resultados.

Segundo Resende e Ribeiro (2016), a combinação do PR com outras metaheurísticas, de forma híbrida, tem levado a avanços significativos tanto na qualidade das soluções quanto na redução do tempo computacional em diversos problemas de otimização combinatória.

3.7.6 GRASP e *Path Relinking*

Cada iteração do GRASP, como apresentado na Subseção 3.7.1, ocorre de forma independente, sem utilizar informações das iterações anteriores, caracterizando-o como um procedimento sem memória. Para complementar essa abordagem, algumas extensões do GRASP incorporam mecanismos de memória para melhorar a qualidade das soluções. Dentre essas extensões, destaca-se a combinação do GRASP com o PR, aplicado de forma hibridizada, durante a execução do algoritmo, ou como uma etapa de pós-otimização, após a execução do algoritmo (Resende e Ribeiro, 2005). Em ambas as abordagens, um conjunto ES, denominado conjunto de soluções elite, é incorporado ao GRASP para armazenar as melhores soluções obtidas e, então, o método PR explora as trajetórias entre e através das soluções desse conjunto.

A abordagem híbrida, também denominada *dynamic* PR (DPR) (Lozano-Osorio et al., 2023), foi inicialmente proposta em Laguna e Marti (1999) para o problema de minimização *Straight-Line Crossings in a 2-Layer Graph*, como uma abordagem de intensificação. Para essa proposta, um conjunto elite é utilizado para coletar soluções de alta qualidade produzidas durante a busca e cada solução localmente ótima produzida pela fase de BL do GRASP é combinada utilizando PR. Em outras palavras, o PR é aplicado a cada par de soluções (x,y), em que x é uma solução localmente ótima obtida pela busca local e y uma solução do conjunto elite ES, podendo y ser escolhida aleatoriamente ou com base em critério de seleção, como a diferença simétrica entre as soluções (Resende e Ribeiro, 2003). As soluções intermediárias geradas ao longo dessa trajetória são avaliadas e, caso apresentem desempenho superior, podem substituir as piores soluções do conjunto ES. Assim, o conjunto elite é atualizado dinamicamente a cada iteração, mantendo um equilíbrio entre qualidade e diversidade. Na Subseção 3.7.7, é apresentado como é construído e atualizado o conjunto elite.

Algorithm 10 *Dynamic* PR aplicado ao GRASP

```
1: f* ← ∞
2: while critério de parada não for sastifeito do
3:     S ← Construcao(α)
4:     S ← Busca_Local(S)
5:     if |ES| > 0 then
6:         S' ← Selecionar_Solução(ES)
7:         S ← PR(S',S)
8:         ES ← Atualizar_Elite(S, ES)
9:     else
10:        ES ← {S}
11:    end if
12:    if f(S) < f* then
13:        S* ← S
14:        f* ← f(S*)
15:    end if
16: end while
17: return S*
```

O procedimento DPR pode ser resumido pelo Algoritmo 10 que integra o PR durante as iterações do GRASP. A cada iteração, uma solução S é construída (linha 3) e, em seguida, submetida a um procedimento de busca local (linha 4). Caso $ES \neq \emptyset$ (linha 5), uma solução S' é selecionada a partir do conjunto elite ES (linha 6) e o processo de PR é aplicado para explorar o espaço de soluções intermediárias entre S' e S (linha 7). A nova solução gerada é então considerada para atualizar o conjunto elite (linha 8). Caso contrário, quando se trata da primeira iteração, o conjunto elite é inicializado com a solução corrente. Ao final de cada iteração (linha 12), se a solução encontrada apresentar valor de função objetivo melhor do que o atual f_*, esta passa a ser considerada a melhor solução S_*. O processo se repete até que o critério de parada seja satisfeito, retornando como saída a melhor solução obtida.

Quando aplicado como estratégia de pós-otimização, o PR é executado somente após o término das iterações do GRASP, partindo de um conjunto ES fixo, formado pelas melhores soluções obtidas durante a busca. Após o término das iterações, todos os pares de soluções desse conjunto são reconectados por meio do PR, e a melhor solução encontrada ao fim do processo é retornada, como ilustrado pelo Algoritmo 11. Note que, nessa abordagem, o conjunto inicial de soluções é gerado e depois combinado, mas as novas soluções encontradas no caminho não são consideradas para atualizar o conjunto elite, que permanece estático. Esta abordagem é conhecida também como *Static PR* (SPR) (Lozano-Osorio et al., 2023).

O Algoritmo 11 apresenta o pseudocódigo do SPR. A primeira parte (linhas 3-11) corresponde à execução do GRASP, cujo objetivo é realizar a amostragem de soluções de boa qualidade. Nesse estágio, uma solução inicial é construída (linha 4), posteriormente refinada por meio da busca local (linha 5), e, então, pode ser incorporada ao conjunto elite ES (linha 6). Concluída a fase de iterações do GRASP, inicia-se a segunda parte do algoritmo (linhas 12-18), dedicada ao PR. Nela, cada par de soluções do conjunto elite é reconectado por meio do PR, gerando novas soluções interme-

diárias (linha 13). Sempre que uma solução obtida nessa fase apresenta qualidade superior à melhor solução corrente, esta é substituída (linha 14). Por fim, a melhor solução encontrada ao longo de todo o procedimento é retornada (linha 19).

Algorithm 11 *Static* PR com GRASP

```
1: ES ← ∅
2: f* ← ∞
3: while stopping criterion is not satisfied do
4:     S ← Construcao(α)
5:     S ← Busca_Local(S)
6:     ES ← Atualizar_Elite(ES)
7:     if f(S) < f* then
8:         S* ← S
9:         f* ← f(S*)
10:    end if
11: end while
12: for S,T ∈ ES do
13:    S ← PR(S,T)
14:    if f(S) > f* then
15:        S* ← S
16:        f* ← f(S*)
17:    end if
18: end for
19: return S*
```

Em contrapartida, no *Evolutionary* PR, proposto em Resende e Werneck (2004), o conjunto elite passa a ser atualizado dinamicamente. Nesse caso, as soluções geradas durante o PR são avaliadas para possível inclusão no conjunto elite. A cada atualização, novos pares de soluções são formados e o PR é reaplicado, repetindo-se o processo até que não seja mais possível atualizar ES e todas as soluções tenham sido reconectadas. Essa abordagem amplia a cobertura do espaço de busca e aumenta as chances de encontrar soluções de melhor qualidade, combinando processos de intensificação e diversificação.

O Algoritmo 12 apresenta o pseudocódigo do GRASP combinado com o *Evolutionary* PR. Inicialmente, o GRASP é executado para gerar soluções de boa qualidade (linha 1), formando o conjunto elite ES (linha 2). Em seguida, enquanto existirem pares de soluções ainda não conectados (linha 3), aplica-se o PR (linha 4), gerando soluções intermediárias que podem atualizar tanto ES quanto a melhor solução corrente (linhas 5 e 6). O processo se repete até que todos os pares tenham sido explorados, e ao final, a melhor solução encontrada é retornada (linha 11).

Algorithm 12 GRASP com Evolutionary PR

```
1: Aplicar GRASP
2: Seja ES o conjunto elite resultante
3: while ∃ R,T ∈ ES que não foram conectadas do
4:     S ← PR(R,T)
5:     ES ← Atualizar_Elite(ES,S)
6:     if f(S) < f* then
7:         S* ← S
8:         f* ← f(S*)
9:     end if
10: end while
11: return S*
```

Segundo Laguna et al. (2025), o PR promove melhorias significativas no GRASP, tanto em termos de tempo de resolução quanto na qualidade das soluções. A hibridização entre essas duas metodologias tem sido amplamente explorada na literatura, com aplicações em diferentes problemas de otimização. Por exemplo, Sudarsan e Pugalendhi (2025) aplicaram o GRASP-PR para resolver o *Max k-Cut Problem*, enquanto Martí e Sandoya (2013) investigaram sua utilização no *Equitable Dispersion Problem*. De forma semelhante, Cuellar-Usaquén et al. (2023) empregaram a técnica para o *Traveling Purchaser Problem* (TPP). Já Muritiba et al. (2018) aplica PR de maneira distinta da abordagem tradicional; enquanto normalmente ele é utilizado como complemento a outra metaheurística, os autores o desenvolveram como método principal de busca. Nessa perspectiva, o PR deixa de atuar como estratégia auxiliar de intensificação e passa a constituir o núcleo da exploração do espaço de soluções, conduzindo a busca de forma independente, sem a necessidade de um conjunto de soluções previamente definido. Outras abordagens podem ser vistas em Laguna et al. (2023), nesse trabalho os autores apresentam uma revisão abrangente das recentes implementações acerca da hibridização GRASP-PR, destacando seus avanços e aplicações em diferentes contextos.

3.7.7 Conjunto Elite para o GRASP

Um conjunto elite ES corresponde a um subconjunto formado por soluções de alta qualidade obtidas ao longo da execução do GRASP, limitado a no máximo `maxElite` elementos. A função desse conjunto é preservar soluções promissoras que, além de boas, sejam diversificadas, de modo a representar diferentes regiões do espaço de busca.

O esquema básico para manter um conjunto elite ES para um problema de minimização é apresentado em Resende e Ribeiro (2016) e pode ser apresentado como se segue pelo Algoritmo 13. Esse algoritmo recebe uma solução candidata S e decide se ela deve ser adicionada ao conjunto ES. Caso inserida, o algoritmo identifica qual solução deverá ser removida do conjunto ES, se necessário, para manter seu tamanho fixo.

Se o conjunto elite ES estiver vazio (linha 1), a solução candidata S é automaticamente adicionada ao conjunto. Caso o conjunto ainda não esteja completo, ou seja, se $|ES| <$ `maxElite`, a solução S será adicionada desde que seja diferente de todas

Algorithm 13 Atualizar_Elite

Entrada: ES, S

1: **if** $|ES| <$ `maxElite` **then**
2: **if** $|ES| = \emptyset$ **then**
3: $ES \leftarrow ES \cup \{S\}$
4: **else**
5: $\delta \leftarrow \min\{\Delta(S, S') : S' \in ES\}$
6: **if** $\delta > 0$ **then**
7: $ES \leftarrow ES \cup \{S\}$
8: **end if**
9: **end if**
10: **else**
11: $f^+ \leftarrow \max\{f(S') : S' \in ES\}$
12: $\delta \leftarrow \min\{\Delta(S, S') : S' \in ES\}$
13: **if** $f(S) < f^+$ **and** $\delta > 0$ **then**
14: $S^- \leftarrow \arg\min\{\Delta(S, S') : S' \in ES \mid f(S') \geq f(S)\}$
15: $ES \leftarrow ES \cup \{S\} \setminus \{S^-\}$
16: **end if**
17: **end if**
18: **return** ES

as soluções já presentes em ES (linhas 4-9). Para verificar essa diferença, calcula-se a diferença simétrica $\Delta(S, S')$ entre S e cada $S' \in ES$, que corresponde ao conjunto de elementos distintos entre as duas soluções. O valor mínimo dessas diferenças é armazenado em δ, que serve como critério de diversidade.

Por outro lado, se o conjunto elite já estiver completo, qualquer nova inserção exige a remoção de uma solução existente, de forma a manter o tamanho máximo `MaxElite`. Esse caso é tratado pelas linhas 10 a 17 do algoritmo. Na linha 11, identifica-se o valor f^+, correspondente ao pior valor da função objetivo entre todas as soluções em ES. Em seguida, na linha 12, calcula-se o valor mínimo δ entre as diferenças simétricas entre S e os elementos de ES. A solução S será adicionada ao conjunto elite apenas se for melhor que a pior solução atual (isto é, se $f(S) < f^+$) e suficientemente diferente das demais (ou seja, se $\delta > 0$), conforme estabelecido na linha 13.

Se essas condições forem satisfeitas, a linha 14 identifica a solução $S^- \in ES$ mais semelhante a S (aquela com a menor $\Delta(S, S')$, entre as que têm valor da função objetivo maior ou igual a $f(S)$). Essa solução S^- é então removida do conjunto na linha 15, e S é adicionada. O conjunto atualizado é retornado na linha 18.

Segundo Resende e Ribeiro (2016) o algoritmo pode ser ajustado para exigir uma diversidade mínima maior ao modificar as condições nas linhas 6 e 13. Nessa versão, a condição $\delta > 0$ é substituída por $\delta \geq \delta_{\min}$, em que $\delta_{\min} > 0$ é um parâmetro predefinido. Com isso, a solução candidata só será aceita se for suficientemente diferente das demais, conforme o critério estabelecido pelo valor de $\delta_{\min}$.

3.7.8 *Variable Neighborhood Search* (VNS)

Variable Neighborhood Search (VNS) é uma metaheurística proposta por Mladenović e Hansen (1997), que consiste em explorar o espaço de soluções pela alteração sistemática das estruturas de vizinhança. De acordo com Hansen et al. (2017), o método fundamenta-se em três observações: (i) um ótimo local em relação a uma determinada estrutura de vizinhança não é necessariamente ótimo local para outra; (ii) um ótimo global é ótimo local em relação a todas as estruturas de vizinhança;(iii) para diversos problemas, a maioria dos ótimos locais encontram-se relativamente próximos entre si.

Partindo dessas observações, o VNS explora vizinhanças progressivamente mais distantes da solução incumbente, concentrando a busca em torno de uma nova solução somente quando é realizado um movimento que resulta em melhoria. Esse processo é ilustrado na Figura 3.8. Desse modo, o método tende a preservar características promissoras do ótimo local, como variáveis que já se encontram em seus valores ótimos, (Hansen e Mladenović, 2001). Essa exploração é conduzida seguindo três passos que combinam elementos de diversificação e intensificação. Esses passos incluem fases de perturbação, de refinamento (ou busca local) e mudança de vizinhança, repetidos até algum critério de parada ser atingido.

As estruturas de vizinhança empregadas nos procedimentos de perturbação e de busca local podem ser distintas ou as mesmas. Assim, adota-se a notação N para representar vizinhanças utilizadas na busca local e $\mathcal{N}$ para as utilizadas para perturbação.

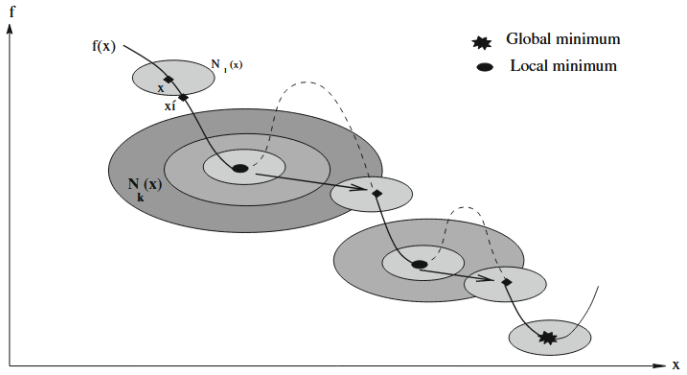


Figura 3.8: Ilustração da exploração do VNS, extraída de Brimberg et al. (2010).

O Algoritmo 14 apresenta o funcionamento geral do VNS. Nesse algoritmo, parte-se de uma solução inicial x qualquer e a cada iteração seleciona-se aleatoriamente um vizinho x' pertencente a vizinhança $\mathcal{N}_k(x)$ da solução x corrente por meio do procedimento de perturbação. Esse vizinho é então submetido a um procedimento de busca local. A solução é avaliada por algum critério de aceitação e, com base na estratégia de exploração adotada, define-se qual estrutura de vizinhança será utilizada na próxima iteração.

Algorithm 14 *Variable Neighborhood Search*

Entrada: solução inicial x, conjunto de vizinhanças $\mathcal{N}(.)$, $N(.)$

1: **while** critério de parada não satisfeito **do**
2: $k \leftarrow 1$
3: **while** $k \leq |\mathcal{N}|$ **do**
4: $x' \leftarrow$ Perturbação(x,k,$\mathcal{N}$)
5: $x'' \leftarrow$ Busca_Local(x,N)
6: Mudar_Vizinhança(x,x',k)
7: **end while**
8: **end while**
9: **return** s

Dentro da estrutura do VNS, o procedimento de perturbação tem como objetivo escapar do ótimo local corrente, permitindo a exploração de diferentes regiões do espaço de soluções. Esse procedimento consiste em selecionar uma solução aleatória da k-ésima estrutura de vizinhança, ou seja, gera uma solução $x' \in \mathcal{N}_k(x)$, em que $\mathcal{N}_k \in \mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$ e $1 \leq k \leq k_{\max}$. A busca local, por sua vez, tem por objetivo explorar de forma intensiva a vizinhança da solução corrente em busca de melhores soluções. Por fim, a etapa de mudança de vizinhança tem como propósito orientar a busca do VNS durante a exploração do espaço de soluções. Essa fase decide qual estrutura de vizinhança será explorada a seguir e se uma determinada solução será aceita como a nova solução corrente.

Segundo Hansen et al. (2017), os procedimentos de mudança de vizinhança mais utilizados são os seguintes:

(a) **Mudança de vizinhança sequencial:** Caso encontre uma melhor solução, a busca é retomada para a primeira vizinhança e a solução incumbente é atualizada; caso contrário, a busca continua na próxima vizinhança (ver Algoritmo 15).

Algorithm 15 Mudança de vizinhança sequencial (x,x',k)

1: **if** $f(x') < f(x)$ **then**
2: $x \leftarrow x'$
3: $k \leftarrow 1$
4: **else**
5: $k \leftarrow k + 1$
6: **end if**

(b) **Mudança de vizinhança cíclica:** Independente de haver ou não melhoria, a busca continua na próxima estrutura. Apenas a solução incumbente é atualizada (ver Algoritmo 16).

Algorithm 16 Mudança de vizinhança cíclica (x,x',k)

1: $k \leftarrow k + 1$
2: **if** $f(x') < f(x)$ **then**
3: $x \leftarrow x'$
4: **end if**

(c) **Mudança de vizinhança *pipe*:** Caso ocorra atualização na solução incumbente, a exploração continua naquela vizinhança que gerou melhoria. Caso contrário, a busca continua na próxima vizinhança, como apresentado pelo Algoritmo 17.

Algorithm 17 Mudança de vizinhança *pipe* (x,x',k)

1: **if** $f(x') < f(x)$ **then**
2: $\quad x \leftarrow x'$
3: **else**
4: $\quad k \leftarrow k+1$
5: **end if**

(d) **Mudança de vizinhança *skewed*:** Nesta estratégia de exploração, a atualização da solução incumbente pode ocorrer mesmo que encontre uma solução pior avaliada, isto é, que não gera melhoria. Esse processo tem como objetivo explorar regiões distantes da solução incumbente. Para isso, cada solução encontrada é avaliada não apenas pela sua função objetivo, mas também pela sua distância em relação a melhor solução encontrada. Essa avaliação é auxiliada pelo uso da função $\rho(x,x')$ que mensura a distância entre as soluções x e x'. Desse modo a função de avaliação pode ser definida, para um problema de minimização, como:

$$g(x,x') = f(x) - f(x') - \alpha\rho(x,x')$$

em que α é um parâmetro positivo que deve ser escolhido para permitir o movimento para regiões mais distantes de x quando $f(x') > f(x)$. Essa função de avaliação pode ser integrada nas mudanças de vizinhanças descritas anteriormente. O Algoritmo 18 exemplifica essa integração com a mudança de vizinhança sequencial.

Algorithm 18 Mudança de vizinhança *skewed* (x,x',k)

1: **if** $f(x') - f(x) < \alpha\rho(x',x)$ **then**
2: $\quad x \leftarrow x'$
3: $\quad k \leftarrow 1$
4: **else**
5: $\quad k \leftarrow k+1$
6: **end if**

(e) **Mudança de vizinhança adaptativa:** Constitui uma estratégia dinâmica para selecionar qual estrutura de vizinhança será explorada na próxima iteração. Nesse método, cada vizinhança $\mathcal{N}_k$ recebe um valor de mérito, que representa sua eficiência durante a busca. Isto é, sempre que é encontrada uma melhoria em determinada vizinhança, o mérito da mesma é aumentado, caso contrário, ele é reduzido. As vizinhanças são então ordenadas de acordo com seus méritos, de modo que, na iteração seguinte, a vizinhança com maior mérito é selecionada para ser explorada.

Ao introduzir esse componente adaptativo, o método tende a privilegiar as estruturas de vizinhança que demonstram melhor desempenho ao longo das iterações. Permitindo que o algoritmo direcione o esforço computacional para as vizinhanças mais eficazes (Brimberg et al., 2023).

3.7.8.1 Variantes do VNS

O VNS tem sido aplicado a diferentes problemas de otimização (ver, por exemplo, De Armas e Moreno-Pérez (2025); Guo et al. (2025); Karakostas e Sifaleras (2025); Lan et al. (2025)). Diversas variantes da metaheurística foram propostas com o objetivo de aprimorar seu desempenho e adaptá-lo a diferentes tipos de problemas. Algumas dessas aplicações e extensões podem ser vistas em Sleptchenko et al. (2023); Brimberg et al. (2023). Essas variantes diferenciam-se, principalmente, pela forma como combinam as fases de perturbação, busca local e mudança de vizinhança, bem como pela profundidade de exploração e estratégias de intensificação e diversificação adotadas.

A seguir, são apresentadas algumas das principais variantes do VNS descritas na literatura.

(i) *Basic* VNS (B-VNS)

O *basic* VNS alterna entre duas etapas principais; procedimento de perturbação e de busca local simples (descrita no Algoritmo 3), seguido por uma mudança de vizinhança. Esse processo é repetido até um critério de parada ser atendido. O Algoritmo 14 exemplifica esse processo.

(ii) *Reduced* VNS (R-VNS)

O *reduced* VNS é uma variação mais simples do B-VNS. No R-VNS, após cada perturbação, a solução encontrada é avaliada diretamente, sem aplicação da busca local. Esse procedimento, descrito no Algoritmo 19, reduz o custo computacional, sendo útil para encontrar soluções para instâncias grandes e procedimentos de busca local custosos computacionalmente (Hansen et al., 2019).

Algorithm 19 Reduced Variable Neighborhood Search

Entrada: solução inicial x, conjunto de vizinhanças $\mathcal{N}(.)$

```
1: while critério de parada não satisfeito do
2:     k ← 1
3:     while k ≤ |N| do
4:         x' ← Perturbação(x,k,N)
5:         Mudar_Vizinhança(x,x',N)
6:     end while
7: end while
8: return s
```

Por não incluir fase de melhoria, o R-VNS pode ficar preso por muitas iterações em uma mesma solução incumbente. Contudo, como as iterações são rápidas, o método consegue mapear regiões amplas do espaço de busca e encontrar boas

soluções (Brimberg et al., 2010). Como, por exemplo, nos estudos de Brandimarte e Fadda (2024), aplicado ao *just-in-time job shop scheduling problem*, e de Xiao et al. (2011) voltado ao *uncapacitated multilevel lot-sizing problems*. Este último, combinado com movimentos de perturbação eficazes conseguiram obter novas melhores soluções em relação as reportadas na literatura.

(iii) *General* VNS (G-VNS)

O *general* VNS é uma variação do B-VNS, o que difere as duas variantes é que o G-VNS emprega o método VND como estrutura de busca local. Nesse esquema, representado pelo Algoritmo 20, após a fase de perturbação, aplica-se o VND como intensificação, seguido da mudança de vizinhança.

Algorithm 20 General Variable Neighborhood Search

Entrada: solução inicial x, conjunto de vizinhanças $\mathcal{N}(.)$

```
1: while critério de parada não satisfeito do
2:     k ← 1
3:     while k ≤ |N| do
4:         x' ← Perturbação(x,k,N)
5:         x' ← VND(x',N)
6:         Mudar_Vizinhança(x,x',N)
7:     end while
8: end while
9: return s
```

Em trabalhos recentes, Todosijević et al. (2025) aplicaram o G-VNS ao problema *capacitated single allocation hub maximal covering*, observando que o método foi capaz de encontrar todas as soluções ótimas conhecidas do problema e fornecer melhores limitantes em comparação ao CPLEX, com baixo custo computacional. Além disso, em Dai e Cheng (2019) o G-VNS foi combinado com PR para resolver o *multi-skill resources-constrained project scheduling problem with multiple restrictions* com o objetivo de melhorar a eficiência da busca local.

(iv) *Nested* VNS

A *Nested Variable Neighborhood Search* (VNS) consiste em aplicar o VNS em cada ponto de uma grande estrutura de vizinhança composta por várias vizinhanças, em vez de aplicar um procedimento de busca local. Considerando $\mathcal{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_{k_{\max}}\}$ um conjunto de vizinhanças, a vizinhança composta a ser explorada é definida como $\mathcal{N}^* = \mathcal{N}_1 \circ \mathcal{N}_2 \circ \cdots \circ \mathcal{N}_{k_{\max}}$. Essa rotina é apresentada pelo Algoritmo 21.

Algorithm 21 Variable neighborhood decomposition search

Entrada: solução inicial x, conjunto de vizinhanças $\mathcal{N}(.)$

1: $\mathcal{N}^* \leftarrow N_1 \circ N_2 \circ \cdots \circ N_{k_{\max}}$
2: $x' \leftarrow x$
3: **while** critério de parada não satisfeito **do**
4: $x \leftarrow x'$
5: **for** cada $y \in \mathcal{N}^*(x)$ **do**
6: $x'' \leftarrow \text{VNS}(y, \mathcal{N})$
7: **if** $f(x'') < f(x')$ **then**
8: $x' \leftarrow x''$
9: **end if**
10: **end for**
11: **end while**
12: **return** x'

Segundo (Hansen et al., 2017) para uma melhor performance do *nested* VNS é importante que o VNS aplicado seja muito rápido. Em razão disso, um critério de parada utilizado para o VNS interno é um pequeno tempo limite para a exploração.

(v) *Variable neighborhood decomposition search* (VNDS)

O *Variable neighborhood decomposition search* baseia-se na decomposição do problema original em subproblemas menores. Essa abordagem resulta em uma estrutura de dois níveis, um nível que controla a decomposição, e outro que resolve cada subproblema gerado utilizando o VNS. Então, diferentemente das outra variantes do VNS, o VNDS não realiza a fase de melhoria em todo espaço de soluções, mas do espaço reduzido correspondente ao subproblema derivado do original. O Algoritmo 22 ilustra esse procedimento.

Após a fase de perturbação, a solução é decomposta gerando uma nova solução y que difere de x em k atributos, por exemplo. Sobre essa solução y aplica-se um procedimento de busca local, mas restrito ao subespaço definido pela decomposição. Após, a solução é reintegrada a solução original, combinando com os atributos inalterados da solução anterior $x' \backslash y$.

Algorithm 22 Variable neighborhood decomposition search

Entrada: solução inicial x, conjunto de vizinhanças $\mathcal{N}(.)$

1: **while** critério de parada não satisfeito **do**
2: $k \leftarrow 1$
3: **while** $k \leq |\mathcal{N}|$ **do**
4: $x' \leftarrow$ Perturbação(x,k,$\mathcal{N}$)
5: $y \leftarrow x' \backslash x$
6: $y' \leftarrow$ Busca_Local(y,N)
7: $x'' = (x' \backslash y) \cup y'$
8: Mudar_Vizinhança(x,x'',k)
9: **end while**
10: **end while**
11: **return** s

Em Hansen e Mladenović (2001), o VNDS foi aplicado ao problema *p-median*, sendo comparado com as variantes B-VNS e o R-VNS. Os autores observaram que, para instâncias de médio porte, os resultados do B-VNS apresentaram melhor desempenho. No entanto, para instâncias de larga escala, o VNDS se mostrou superior, fornecendo soluções de melhor qualidade em menor tempo. Já em Seeanner et al. (2013), foi proposto um método que combina o VNDS com a heurística Fix&Optimize capaz de resolver instâncias de grande porte de problemas de *multi-level lot-sizing* e *scheduling problems*. Nesse estudo, os critérios de decomposição do VNDS são utilizados para auxiliar a fixação de variáveis do Fix&Optimize.

Capítulo 4

Métodos de Solução Propostos

Neste capítulo, são apresentadas as implementações e adaptações realizadas para a integração da GRASP com o EPR e com o IRP, além do algoritmo GVNS visando a solução de instâncias do MRCPSP. Inicialmente, na Seção 4.1, apresenta-se o algoritmo GRASP com PR proposto. Detalha-se as fases da GRASP, com ênfase nos procedimentos de construção e de busca local, estruturados com base nas estratégias discutidas no Capítulo 3. É abordada a heurística construtiva utilizada, bem como as estruturas de vizinhança aplicadas no VND. Em seguida, é apresentada a implementação do EPR e do IPR definindo os componentes nas quais serão construídos os caminhos e a rotina de exploração pelo *evolutionary* PR. De forma análoga, na Seção 4.2 são detalhados os elementos que compõem o VNS, incluindo a construção da solução inicial, as estruturas de vizinhanças aplicadas na perturbação e a estratégia de busca local.

4.1 GRASP com PR para o MRCPSP

Nesta seção é apresentada a integração da GRASP com o procedimento de PR implementado neste trabalho. Inicialmente, descreve-se a implementação da GRASP, detalhando procedimento de construção da solução inicial (Subseção 4.1.1), as estruturas de vizinhança empregadas na busca local, bem como a rotina do VND adotada (Subseção 4.1.2). Em seguida, na Subseção 4.1.3, apresenta-se o processo de construção dos caminhos no PR, tanto do EPR quanto do IPR, além do critério de diversificação adotado (Subseção 4.1.3). Por fim, é descrito o funcionamento geral do algoritmo proposto (Subseção 4.1.4).

4.1.1 Fase de construção da GRASP

A resolução do MRCPSP busca definir um modo de execução para cada atividade e determinar um tempo de término viável para elas. Resolver esses problemas de forma simultânea, analisando todas as restrições e avaliando o impacto de cada escolha, é uma tarefa complexa. Por isso, a estratégia adotada foi dividi-los em dois subproblemas menores, que podem ser analisados separadamente. A solução obtida para o primeiro subproblema é utilizada como entrada para o segundo, e a saída deste fornece uma solução para o problema original.

Essa abordagem pode ser descrita em duas etapas. Na primeira etapa, busca-se uma atribuição de modos que seja viável, garantindo que o consumo de recursos não renováveis não exceda sua capacidade disponível. Na segunda etapa, as atividades são sequenciadas, atribuindo-se tempos de término viáveis para minimizar o *makespan*, enquanto se respeitam as relações de precedência e a disponibilidade dos recursos renováveis em cada período de tempo. Essa etapa considera a atribuição de modos já realizada na primeira fase.

Ao final dessas duas etapas, é gerada uma solução, codificada como $L(x)$. O procedimento utilizado para a atribuição de modos e a programação é detalhado nas seções a seguir.

4.1.1.1 Atribuição de Modos (AM)

Quando a instância possui mais de um tipo de recurso não renovável, a atribuição de modos torna-se um problema NP-Completo (Blazewicz et al., 1983). Para lidar com essa complexidade, foi adotado o método proposto por Hartmann (2001), que busca obter uma atribuição de modos viável, se possível, dentro de um número máximo de iterações e com baixo custo computacional. Por se tratar de um método heurístico, não há garantia de que uma configuração viável será encontrada. Assim, caso o procedimento não consiga obter um atribuição viável de modos que satisfaça as restrições de recursos não renováveis dentro do limite estabelecido de iterações, a instância não é considerada para as etapas seguintes de resolução.

Inicialmente, os modos são atribuídos aleatoriamente, sem verificação prévia da viabilidade. Caso a configuração gerada seja inviável, um procedimento de reparação é chamado para tentar ajustar a seleção dos modos.

Esse processo de reparação opera iterativamente, buscando minimizar o excesso de consumo de recursos. A cada iteração, uma atividade j é escolhida aleatoriamente para ter seu modo de execução alterado de m_j para $\overline{m_j}$, também selecionado de forma aleatória. Se a troca reduzir o excesso de consumo, o novo modo $\overline{m_j}$ é mantido. O procedimento continua até que a inviabilidade seja eliminada ou até que $|J|$ tentativas consecutivas de reparação não tenham sucesso.

Apesar dessa estratégia ser capaz de gerar soluções viáveis, procedimentos puramente aleatórios frequentemente geram soluções de baixa qualidade. Diante disso, após garantir a viabilidade dos modos, aplica-se o procedimento de busca local *Mode Assignment Compression* (MAC), apresentado na Subseção 3.6.2, para ajustar a seleção dos modos de execução para uma configuração que consuma menos tempo.

Ao final dessa etapa, é gerada uma solução codificada como μ, em que $\mu(j)$ representa o modo atribuído à tarefa j.

4.1.1.2 Sequenciamento das atividades

Garantida a viabilidade de modos, aplica-se a heurística construtiva FBI, apresentada na Subseção 3.6.3, para definir os tempos de término das atividades do projeto. Para integrar o FBI à fase construtiva do GRASP, sua fase inicial, construída pelo SSS, foi adaptada para introduzir diversidade nas soluções geradas. Em lugar de sempre selecionar o melhor candidato de forma determinística, a escolha passa a ser feita de maneira gulosa e aleatória a partir de uma LCR. Neste trabalho, foram implementadas duas regras de prioridade: a regra SLK, que seleciona a atividade com

menor folga, e a regra LFT, que escolhe a atividade com o menor tempo de término LFT_j. Essas métricas são obtidas pelo método CPM apresentado na Seção 3.2. Outras regras de prioridades podem ser vistas em Boctor (1993); Fernandes e de Souza (2021). A cada execução da fase construtiva do GRASP, uma dessas regras é escolhida aleatoriamente.

A aplicação da busca local MAC na atribuição de modos e a escolha aleatória do critério de seleção no SSS foram estratégias utilizadas para garantir a diversidade e a qualidade das soluções geradas.

4.1.2 Fase de Busca Local da GRASP

A fase de BL tem como objetivo aprimorar a solução construída inicialmente, explorando sua vizinhança em busca de melhorias. As estruturas de vizinhanças adotadas neste trabalho baseiam-se nas adaptações propostas em Afshar et al. (2022), Muritiba et al. (2018), Geiger (2017) e Fernandes e de Souza (2021), incorporando estratégias específicas para o MRCPSP. Vale destacar que somente soluções viáveis foram consideradas.

4.1.2.1 Estruturas de vizinhança

Para construir a vizinhança $N(x)$ centrada em x, são considerados três tipos de vizinhanças:

(i) Vizinhanças baseadas em modos:

- *Single Mode Change (SMC)*:
  Esse movimento consiste em alterar o modo de execução de uma única atividade. Para isso, seleciona-se uma atividade j e troca-se o modo atual atribuído m por $\bar{m} \neq m$, tal que $\bar{m} \in M_j$. A alteração é aceita apenas se resultar em uma configuração viável de modos; caso contrário, a solução vizinha encontrada não é avaliada.
  A cardinalidade do conjunto de vizinhança gerado por SMC é limitada pelo número de atividades n e pela quantidade de modos disponíveis para cada atividade j, denotado por $|M_j|$:

$$|SMC| \leq \sum_{j=1}^{n}(|M_j| - 1)$$

  Na Figura 4.1 é apresentado um exemplo de um vizinho gerado nessa estrutura. Nesse movimento, a solução corrente s é modificada a partir da escolha da atividade $j = 2$, alterando o seu modo de execução de $m = 2$ para $m = 3$, resultando na solução s'. Essa modificação possibilitou a redução do *makespan*. Embora o tempo de duração do modo 3 seja maior que o do modo 2 para a atividade 2 ($d_{22} = 6 < d_{23} = 8$), a atividade pôde ser programada em um instante anterior em relação à configuração com menor tempo de duração. Isso ocorreu porque, antes do movimento, as atividades 2 e 4 competiam pelo recurso renovável R_2^ρ. Após a alteração,

essa competição deixou de existir, permitindo que ambas fossem executadas em paralelo, o que possibilitou a antecipação de outras atividades e, consequentemente, a redução do *makespan*.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	3	3	2	3	1	3	3	2	3	1
LT	0	5	8	13	12	22	14	26	28	33	37	37

Figura 4.1: Movimento SMC aplicado à solução corrente s levando a um vizinho s'.

- *Two Modes Change* (TMC):

  Duas atividades i e j são selecionadas, e seus modos atuais m_i e m_j são trocados por novos modos $\bar{m}_i \neq m_i \in M_i$ e $\bar{m}_j \neq m_j \in M_j$, respectivamente. Assim como no movimento SMC, apenas soluções viáveis são consideradas para avaliação.

  A cardinalidade da vizinhança gerada pelo movimento TMC, considerando todos os pares distintos de atividades j e j', com $j \neq j'$, e seus respectivos conjuntos de modos $|M_j|$ e $|M_{j'}|$, é limitada por:

$$|TMC| \leq \frac{1}{2} \sum_{j=1}^{n} \sum_{\substack{j'=1 \\ j' \neq j}}^{n} (|M_j| - 1)(|M_{j'}| - 1)$$

  Na Figura 4.2 é apresentado um exemplo de um vizinho gerado nessa estrutura. Esse movimento altera simultaneamente os modos de execução de duas atividades distintas. Nesse movimento, a solução s é modificada a partir da seleção das atividades $j = 3$ e $j = 9$, cujos modos de execução foram alterados para $m_3 = 2$ e $m_9 = 2$, respectivamente. Essa alteração resultou em uma configuração viável com menor tempo de processamento. Vale ressaltar que, após o movimento, as atividades são reprogramadas pelo SSS a partir da primeira posição afetada, garantindo a consistência da solução obtida.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	2	2	3	1	3	3	2	2	1
LT	0	5	11	7	15	20	8	24	26	31	35	35

Figura 4.2: Movimento TMC aplicado à solução s resultando no vizinho s'.

(ii) Vizinhanças baseadas em atividades;

- *Shift* em uma atividade (SH):
  SH realoca a posição de uma atividade j na lista LJ. Para isso, determina-se um intervalo $I_j = [i_p + 1, i_s - 1]$, onde i_p e i_s para $p \in P_j$ e $s \in S_j$ são, respectivamente, as posições das atividades predecessora e sucessora mais próximas de j no sequenciamento LJ atual. Esse intervalo define as posições para as quais a realocação de j para uma nova posição é viável, respeitando as restrições de precedência.
  Cada movimento de realocação de j para uma posição $i \in I_j$, com $i \neq j$, gera um novo vizinho. A cardinalidade de SH pode ser determinada como:

$$|SH| \leq n(n-1)$$

  Na Figura 4.3 é apresentado um exemplo de um vizinho gerado nessa estrutura. Nesse movimento, a solução s é alterada a partir da escolha da atividade $j = 3$ para ser movimentada na sequência LJ. Para isso, identificaram-se seus predecessores e sucessores imediatos mais próximos, $p_3 = 1$ e $s_3 = 7$ nas respectivas posições $i_p = 1$ e $i_s = 7$, de modo a realocá-la em uma nova posição dentro desse intervalo (1,7). Esse procedimento altera a ordem de prioridade na alocação de recursos, possibilitando que determinadas tarefas sejam antecipadas, como ocorreu com a tarefa 5.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	4	2	6	5	3	7	8	11	10	9	12
LM	1	2	2	2	3	3	1	3	3	2	3	1
LT	0	5	11	15	20	28	29	32	26	37	32	37

Figura 4.3: Movimento SH aplicado à solução s, resultando no vizinho s'.

- *Swap* entre atividades (SW):
  SW consiste em trocar a posição de duas atividades da lista LJ. De maneira similar ao SH, inicialmente escolhe-se uma atividade j e determina-se um intervalo $I_j = [i_p + 1, i_s - 1]$ das atividades predecessoreas e sucessoras mais proximas, onde $i_p \in P_j$ e $i_s \in S_j$. Esse intervalo define as posições para as quais trocas das atividades em I_j com j serão avaliadas. A troca será aceita, se somente se não há violação de precedência.
  A cardialidade de SW é expressa como:

$$|SH| \leq \frac{1}{2}n(n-1)$$

  Na Figura 4.4 é apresentado um exemplo de um vizinho gerado nessa estrutura. Esse movimento altera simultaneamente os modos de execução de duas atividades distintas. Nesse movimento, a solução s é alterada a partir da escolha da atividade $j = 5$ para trocar de posição com outra atividade na sequência LJ. Para isso, identificaram-se seus predecessores e sucessores imediatos mais próximos, $p_5 = 2$ e $s_5 = 8$ nas respectivas posições $i_p = 3$ e $i_s = 8$. O movimento consistiu em selecionar a atividade $j = 3$ e trocá-la de posição com a tarefa 5. Dentro do intervalo de posições (3,8), esse movimento não viola as restrições de precedência. Por outro lado, trocar a atividade 5 com a 7 causaria violação, pois a atividade 7 iniciaria antes da atividade 3.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	4	2	5	6	3	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	20	15	28	29	32	26	37	32	37

Figura 4.4: Movimento SW aplicado à solução s, resultando no vizinho s'.

(iii) Vizinhanças baseadas em atividades e modos.

- *Shift* em atividade e *Single Mode Change* (SH-SCM):
  Este movimento combina as vizinhanças SH e SMC. Após a seleção de uma atividades j determina-se o intervalo de movimentação I_j. Cada movimento de realocação de j para uma posição $i \in I_j$ gera um novo vizinho. Após o movimento, troca-se o modo de execução de j para um $m' \neq m$ desde que sejam atendidas as restrições de capacidade de recursos não renováveis.
  Esse processo avalia cada atividade $j = 1, \ldots, n$, cada intervalo I_j bem como todos os modos de execução de j, sua cardinalidade :

$$|SH - SCM| \leq n(n-1)\sum_{j=1}^{n}(|M_j| - 1)$$

  Na Figura 4.5 é apresentado um exemplo de um vizinho gerado nessa estrutura. Esse movimento altera simultaneamente os modos de execução de duas atividades distintas. Nesse movimento foram consideradas duas operações: a realocação da atividade e a troca de modos. Primeiramente, selecionou-se a atividade $j = 2$, localizada na posição 3 da lista LJ, e identificou-se o intervalo de movimentação (1,5), definido pelos predecessores e sucessores mais próximos, ou seja, $p_2 = 1$ e $s_2 = 6$. A atividade é então realocada para a segunda posição da sequência, e em seguida avaliam-se as possíveis trocas do modo atual pelos modos disponíveis. Dentre essas opções, escolheu-se aquela que proporcionou a maior melhoria no *makespan* do projeto; nesse caso, a substituição do modo $m = 2$ pelo $m = 3$. Esse movimento foi aceito, uma vez que não viola as restrições de precedência nem de recursos não renováveis.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	2	4	3	6	5	7	8	11	10	9	12
LM	1	3	2	2	2	3	1	3	3	2	2	1
LT	0	8	5	13	12	22	14	26	28	33	37	37

Figura 4.5: Movimento SH-SMC aplicado à solução s, resultando no vizinho s'.

- *Swap* entre atividades e *Single Change Mode* (SW-SCM):
  Este movimento combina as vizinhanças SW e SCM. Após a seleção de uma atividade j determina-se o intervalo de movimentação I_j. Cada movimento de troca de j com um atividade $\bar{j}$ dentro do intervalo I_j gera um novo vizinho. Após o movimento, troca-se o modo de execução de j para um $m' \neq m$ desde que sejam atendidas as restrições de capacidade de recursos não renováveis.

  De modo semelhante ao SH-SCM, esse processo avalia para cada atividade $j = 1, \ldots, n$, cada intervalo $I_j = [i_p+1, i_s-1]$ e todos os modos de execução de j. Existem,

$$|SW - SCM| \leq \frac{1}{2} n(n-1) \sum_{j=1}^{n} (|M_j| - 1)$$

  Na Figura 4.6 é apresentado um exemplo de um vizinho gerado nessa estrutura. Esse movimento altera simultaneamente os modos de execução de duas atividades distintas. Nesse movimento, foram aplicadas duas operações: a troca de posição entre atividades e a troca de modos. Inicialmente, selecionou-se a atividade $j = 11$, localizada na posição 11 da lista LJ, e determinou-se o intervalo de movimentação (7,12), definido pelos predecessores e sucessores mais próximos ($p_{11} = 7$ e $s_{11} = 12$). Em seguida, a atividade foi reposicionada por meio da troca com a tarefa 9 dentro desse intervalo. Posteriormente, avaliaram-se as possíveis substituições do modo atual pelos modos disponíveis. Entre as alternativas, escolheu-se aquela que proporcionou a maior redução do *makespan* do projeto, correspondendo à troca do modo $m = 2$ pelo $m = 3$. O movimento foi aceito, pois não violou as restrições de precedência nem de recursos não renováveis.

Solução s

LJ	1	4	2	3	6	5	7	8	11	10	9	12
LM	1	2	2	3	2	3	1	3	3	2	3	1
LT	0	5	11	19	15	28	20	32	34	39	43	43

Solução s'

LJ	1	4	2	3	6	5	7	8	9	10	11	12
LM	1	2	2	3	2	3	1	3	2	2	3	1
LT	0	5	11	19	15	28	20	32	36	41	34	41

Figura 4.6: Movimento SW-SMC aplicado à solução s, resultando no vizinho s'.

(iv) Vizinhanças baseadas em atividades críticas e modos.

- *Critical tasks* SH-SCM (SH-SCM-CT): Realiza os movimentos da vizinhança SH-SCM, porém restringindo a movimentação apenas às tarefas críticas;
- *Critical tasks* SW-SCM (SW-SCM-CT): Executa os movimentos da vizinhança SW-SCM, limitando-se às tarefas críticas.

Essa estruturas são uma modificação das vizinhanças SH-SCM e SW-SCM. A principal diferença introduzida está na etapa de seleção das atividades que podem ser movimentadas. Enquanto nas versões originais qualquer atividade pode ser escolhida para compor os movimentos, nessas novas estruturas a escolha é restrita exclusivamente às atividades críticas do projeto.

Essa modificação se justifica pelo fato de que as atividades críticas exercem impacto direto sobre a duração total do projeto. Dessa forma, concentrar os movimentos nessas atividades tende a direcionar a busca para regiões mais promissoras do espaço de soluções, evitando esforços computacionais em alterações que não influenciariam a qualidade da solução. Assim, essas vizinhanças mantém a lógica básica das estruturas SH-SCM e SW-SCM, mas adapta o critério de seleção de atividades com o objetivo de intensificar a busca em pontos mais relevantes da solução corrente.

4.1.2.2 Descrição da Busca local

A rotina de busca local é realizada pelo B-VND apresentado na Subseção 3.6.4. Para a implementação, foram propostos seis VNDs com diferentes combinações de estruturas de vizinhas e diferentes estratégias de exploração com a seguinte ordenação:

(i) VND_1 : [`SMC (BI), TMC (BI), SH-SCM (FI), SW-SCM (FI)`]

(ii) VND_2 : [`SH (BI), SW (BI), SH-SCM (FI), SW-SCM (FI)`]

(iii) VND_3 : [`SMC (BI), TMC (BI), SH (BI), SW (BI)`]

(iv) VND_4 : [`SMC (BI), TMC (BI), SH-SCM (FI)`]

(v) VND_5 : [`SMC (BI), TMC (BI), SW-SCM (FI)`]

(vi) VND_6 : [`SMC (BI), TMC (BI), SW-SCM-CT (BI), SH-SCM-CT (BI)`]

A notação SMC (BI)/(FI) indica a estrutura de vizinhança *Single Mode Change* (SMC) explorada pela estratégia *Best Improvement* (BI)/ *First Improvement* (FI). Na seção de resultados, será apresentada a combinação de estruturas de vizinhança N que se mostrou mais eficiente. Devido ao elevado custo computacional de explorar todas as vizinhanças em uma única execução, especialmente aquelas baseadas em SW-SCM e SM-SCM com maior cardinalidade, a busca foi dividida em múltiplos procedimentos de VND. Essa divisão permitiu gerenciar de forma mais eficiente o tempo de execução, mantendo a capacidade de explorar as vizinhanças mais promissoras.

O Algoritmo 23 apresenta a implementação proposta, que recebe como entrada uma solução viável x e retorna a melhor solução encontrada dentre todas as vizinhanças. Vale destacar que, sempre que a solução incumbente é atualizada, a busca reinicia a partir da primeira vizinhança,

Algorithm 23 VND adaptado

Entrada: $N(.)$,x

1: $N \leftarrow [N^1, \cdots, N^r]$
2: $k \leftarrow 1$
3: $f^* \leftarrow f(x)$
4: **while** $k \leq |N|$ **do**
5: Encontre o melhor vizinho $x' \in N^{(k)}(x)$
6: **if** $f(x') < f^*$ **then**
7: $x \leftarrow x'$
8: $f^* \leftarrow f(x')$
9: $k \leftarrow 1$
10: **else**
11: $k \leftarrow k + 1$
12: **end if**
13: **end while**
14: **return** x

4.1.3 *Path Relinking* para o MRCPSP

Para a construção dos caminhos no procedimento de PR, é necessário definir, inicialmente, o critério utilizado para diferenciar duas soluções, bem como o tipo de movimento empregado na geração das soluções intermediárias ao longo do caminho. No contexto do MRCPSP, duas soluções são consideradas distintas se existir ao menos uma atividade cuja atribuição de modo de execução difira entre elas. Com base nesse critério, a construção dos caminhos é realizada por meio de movimentos que atuam exclusivamente sobre os atributos de modo das atividades.

No EPR, são identificados os atributos de modo comuns entre as soluções de referência, e os movimentos são aplicados de forma a modificar esse subconjunto, promovendo diversificação. Em contrapartida, no IPR, são considerados os atributos

de modo distintos entre as soluções, sendo realizados movimentos com o objetivo de aproximar gradualmente uma solução da outra ao longo do caminho.

Além disso, para que o PR, de maneira geral, seja capaz de encontrar soluções de boa qualidade, é fundamental manter um conjunto elite ES, composto por soluções de alta qualidade e suficientemente diversas entre si. Ao final de cada iteração da GRASP, a solução obtida é avaliada como candidata a integrar ES. Caso o conjunto ainda não esteja completo, a solução é adicionada desde que seja distinta das já existentes. Quando ES estiver preenchido, a solução candidata somente será incorporada se satisfizer pelo menos uma das seguintes condições: (i) apresentar valor melhor que a melhor solução atualmente armazenada; ou (ii) superar a pior solução e, adicionalmente, apresentar diversidade adequada em relação ao conjunto. Em caso de inserção, a pior solução de ES é removida.

O procedimento EPR, descrito na Subseção 3.7.4, tem como objetivo gerar soluções intermediárias entre uma solução inicial S^i e uma solução guia S^g. Essas soluções são geradas a partir de movimentos na solução inicial tendo como referência a solução guia. Isto é, a cada iteração, modificam-se atributos dos modos da solução inicial de forma que a solução resultante permaneça distinta da solução guia.

Para isso, define-se uma lista auxiliar MA, em que $MA[j] = m$ indica o modo de execução $m \in M_j$ atribuído a atividade $j \in J$.

Dadas duas soluções S^i e S^g, com sua respectivas listas de modos MA_i e MA_g, dizemos que $S^i \neq S^g$ se existir ao menos uma atividade $j \in J$ tal que $MA_i[j] \neq MA_g[j]$. Definimos então o conjunto

$$\Omega(S^i, S^g) = \{j \in J : MA_i[j] = MA_g[j]\},$$

e a diferença simétrica como

$$\Delta(S^i, S^g) = J \setminus \Omega(S^i, S^g),$$

a qual expressa o conjunto de atividades cujos modos diferem entre as duas soluções.

A conexão entre S^i e S^g é realizada sobre os componentes de $\Omega(S^i, S^g)$, alterando gradualmente os modos de execução das atividades em S^i, enquanto os elementos $j \in \Delta(S^i, S^g)$ permanecem inalterados durante todo o processo.

O Algoritmo 24 descreve o funcionamento desse procedimento aplicado ao par de soluções x^s e x^t. Inicialmente, define-se a solução inicial S^i como aquela com melhor valor da função objetivo entre x^s e x^t (linha 1), enquanto S^g é a solução guia, ou seja, a de pior qualidade entre as duas (linha 2). Para cada solução, extraem-se os modos de execução das atividades, representados por MA_i e MA_g, respectivamente (linhas 3-4). A melhor solução corrente e seu valor objetivo são armazenados em x^* e f^*, ambos inicializados com os valores de S^i (linhas 5-6).

Em seguida (linha 7), é calculado o conjunto $\Omega(S^i,S^g)$. Enquanto esse conjunto não estiver vazio, seleciona-se aleatoriamente um elemento $k \in \Omega$ (linha 9), e o modo da atividade correspondente em MA_i é substituído aleatoriamente por um novo modo dentro do conjunto de modos M_k (linha 10). Caso a atribuição MA_i seja viável (linha 11), a partir dessa modificação, gera-se uma nova solução S por meio do SSS (linha 12) e aplica-se uma busca local guiada pelo VND (linha 13), obtendo um ótimo local S'. Se o valor da função objetivo dessa nova solução for melhor do que o atual f^* (linha 14), atualizam-se tanto x^* quanto f^* (linhas 15-16). Após essa avaliação, o

Algorithm 24 Exterior Path Relinking adaptado (EPR)

Entrada: x_s, x_t

1: $S^i \leftarrow arg\min(f(x_s),f(x_t))$
2: $S^g \leftarrow arg\max(f(x_s),f(x_t))$
3: $MA_i \leftarrow modos(S^i)$
4: $MA_g \leftarrow modos(S^g)$
5: $f^* \leftarrow f(S^i)$
6: $x^* \leftarrow S^i$;
7: Calcular $\Omega(S^i, S^g)$
8: **while** $\Omega(S^i, S^g) \neq \emptyset$ **do**
9: $k \leftarrow Rand(j : j \in \Omega)$;
10: $MA_i[k] \leftarrow Rand(m : m \in M_k)$
11: **if** MA_i viável **then**
12: $S \leftarrow SSS(MA_i)$
13: $S' \leftarrow VND(S)$
14: **if** $f(S') < f^*$ **then**
15: $x^* \leftarrow S$
16: $f^* \leftarrow f(S')$
17: **end if**
18: **end if**
19: $\Omega(S^i, S^g) \leftarrow \Omega(S^i, S^g) \setminus \{k\}$
20: **end while**
21: **return** x^*

índice k é removido do conjunto $\Omega(S^i,S^g)$ (linha 19), e o processo continua até que não existam mais componentes a serem explorados entre S^i e S^g.

Ao final, a melhor solução encontrada x^* é retornada (linha 21). Ressalta-se que a busca local não é aplicada a todos os componentes da solução intermediária: a cada iteração do PR, a tarefa incorporada a solução inicial é fixada, e apenas os atributos ainda não fixados permanecem sujeitos a modificações na busca local.

A Figura 4.7 ilustra a evolução das iterações do EPR em uma instância composta por 12 atividades. Nesse exemplo, considera-se um par de soluções: a solução inicial S_i e a solução guia S_g. Inicialmente, identifica-se o conjunto de atividades que possuem atributos em comum, destacado em cinza (atividades 1, 3, 4, 6, 7, 11 e 12). A partir dessa lista, e respeitando a ordem topológica, busca-se modificar o modo de execução dessas atividades, sempre que possível. As atividades inicial e final são fictícias e possuem apenas um modo de execução, não sendo passíveis de alteração. A próxima atividade da lista, $j = 3$, teve seu modo alterado aleatoriamente para o modo 3, originando a solução intermediária S_1. Após essa modificação, a atividade é removida da lista. Esse procedimento é repetido de forma iterativa: para cada atividade elegível, verifica-se a possibilidade de troca de modo e, caso haja, seleciona-se um modo distinto do atual. É importante destacar que a modificação é realizada mesmo que a nova configuração resulte em uma solução inviável. Ao final das iterações (Iteração 5), obtém-se a solução intermediária S_5, a qual difere completamente da solução guia, com exceção das atividades fictícias.

Solução Inicial S_i	1	2	2	3	2	3	1	3	3	2	3	1
Iteração 1 (S_1)	1	2	1	3	2	3	1	3	3	2	3	1
Iteração 2 (S_2)	1	2	1	2	2	3	1	3	3	2	3	1
Iteração 3 (S_3)	1	2	1	2	2	1	1	3	3	2	3	1
Iteração 4 (S_4)	1	2	1	2	2	1	2	3	3	2	3	1
Iteração 5 (S_5)	1	2	1	2	2	1	2	3	3	2	1	1
Solução Guia S_g	1	1	2	3	1	3	1	2	1	3	3	1

Figura 4.7: Evolução das soluções intermediárias até a solução guia.

O procedimento do IPR, descrito na Subseção 3.7.3, para o MRCPSP, apresenta uma implementação conceitualmente semelhante a do EPR. A principal diferença reside no conjunto de atributos sobre os quais os movimentos são aplicados. No IPR, a conexão entre S^i e S^g é realizada a partir dos elementos de $\Delta(S^i, S^g)$, isto é, dos atributos de modo nos quais as duas soluções diferem. Em cada iteração, seleciona-se uma atividade $j \in \Delta(S^i, S^g)$ e o modo de execução dessa atividade em S^i é modificado de forma a coincidir com aquele atribuído em S^g. Esse procedimento reduz progressivamente a distância entre as soluções, conduzindo a busca por soluções intermediárias localizadas no interior do caminho que conecta S^i e S^g. O processo prossegue até que S^i se torne idêntica a S^g, em relação a atribuição de modos.

4.1.4 Procedimento geral GRASP+EPR

Os procedimentos GRASP+EPR e GRASP+IPR, conforme mencionado anteriormente, possuem implementações bastante semelhantes, diferenciando-se apenas pelo conjunto de atributos sobre o qual os movimentos de PR são realizados. Sendo assim, a seguir é apresentado apenas o procedimento geral do GRASP+EPR. Para o entendimento da abordagem GRASP+IPR, basta considerar que, em vez de aplicar os movimentos sobre o conjunto de atributos comuns às soluções inicial e guia, como no EPR, os movimentos são realizados sobre o conjunto de atributos nos quais essas soluções diferem, conduzindo a busca ao longo do interior do caminho que conecta ambas. Todas as demais etapas do algoritmo, incluindo a geração do conjunto elite, os critérios de aceitação, a aplicação da busca local e as condições de parada, permanecem inalteradas.

Nesse contexto, de maneira geral, o algoritmo GRASP+EPR combina a metaheurística GRASP com o método EPR como etapa pós-otimização. Inicialmente, as soluções são construídas dividindo o problema em dois subproblemas: na primeira etapa, atribuem-se aleatoriamente os modos de execução às atividades, sendo essa atribuição refinada pelo algoritmo MAC; em seguida, as atividades são programadas pelo FBI, cuja etapa inicial utiliza uma versão adaptada do SSS para gerar soluções parcialmente gulosas. A utilização do MAC e do FBI tem como objetivo produzir soluções diversas e de qualidade.

A etapa de busca local é conduzida pelo B-VND, explorando diferentes vizinhanças definidas na literatura. Durante as iterações do GRASP, as melhores e mais diversas

soluções encontradas são armazenadas em um conjunto elite. Após o término das iterações, aplica-se o EPR a cada par de soluções do conjunto elite. O EPR é realizado sobre os atributos dos modos atribuídos, utilizando o *backward* PR como mecanismo de busca, enquanto a seleção dos novos modos é feita de forma aleatória. Além disso, após cada solução intermediária gerada, aplica-se novamente o B-VND, mas restringindo os movimentos apenas as variáveis não fixadas. Por fim, o processo retorna a melhor solução encontrada, conforme apresentado pelo Algoritmo 25.

Algorithm 25 Pseudocódigo GRASP+EPR

Entrada: $maxIT, \alpha$

$ES \leftarrow \emptyset$
$f_* \leftarrow \infty$
while critério de parada $maxIT$ não foi sastifeito **do**
    $S \leftarrow$ Construção(α)
    $S' \leftarrow$ Busca_Local(S)
    $ES \leftarrow$ Atualizar_Elite(ES,S')
    if $f(S) < f_*$ **then**
        $S_* \leftarrow S'$
        $f_* \leftarrow f(S')$
    end if
end while
while $\exists\, S, T \in ES$ não foi conectado **do**
    $I \leftarrow$ EPR(S,T)
    $ES \leftarrow$ Atualizar_Elite(ES, I)
    if $f(S) < f_*$ **then**
        $S_* \leftarrow S$
        $f_* \leftarrow f(S_*)$
    end if
end while
return S_*

4.2 GVNS para o MRCPSP

Nesta seção será apresentado o algoritmo GVNS implementado neste trabalho, descrevendo-se o procedimento de construção da solução inicial (Subseção 4.2.1), as estruturas de vizinhança empregadas tanto na fase de perturbação quanto na busca local (Subseção 4.2.3), bem como o funcionamento geral do algoritmo (Subseção 4.2.4).

4.2.1 Solução inicial GVNS

Para o MRCPSP, a solução inicial do GVNS é construída em dois estágios. No primeiro, os modos de execução são atribuídos utilizando a regra SFM (*Shortest Feasible Mode*), que atribui a atividade o modo de execução com o menor tempo de processamento, seguida de um procedimento de reparação, caso necessário, para garantir a viabilidade em relação aos recursos não renováveis (ver Subsubseção 4.1.1.1).

No segundo estágio, aplica-se a heurística construtiva FBI, apresentada na Subseção 3.6.3, para determinar os tempos de término das atividades do projeto. No início do procedimento do FBI, é utilizada a regra de prioridade LFT (*Latest finish start*), que seleciona a atividade com o menor tempo de término possível para ser sequenciada pelo SSS. Ao final desse processo, obtém-se uma solução inicial S codificada conforme descrito na Seção 3.3.

4.2.2 Estruturas de vizinhança

As estruturas de vizinhança definidas nesta seção são responsáveis por mover a solução atual para uma nova solução, por meio de um conjunto de operações ou movimentos. No algoritmo proposto, o interesse principal está em alterar os modos de execução das atividades, uma vez que essas modificações impactam diretamente o consumo de tempo e recursos do projeto. Para o GVNS, foram definidas duas estruturas de vizinhança:

(i) **Vizinhança $\mathcal{N}_1$ - Modificação de um modo:** Dada uma solução s, um vizinho é obtido alterando o modo de execução de uma única atividade.

(ii) **Vizinhança $\mathcal{N}_2$ - Modificação de dois modos:** Os vizinhos são gerados alterando os modos de execução de duas atividades distintas.

4.2.3 Perturbação e Busca local

Com o objetivo de explorar regiões da vizinhança além de ótimos locais, o procedimento de perturbação é utilizado para gerar aleatoriamente uma solução s' pertencente à k-ésima vizinhança de s. No algoritmo proposto, as estruturas $\mathcal{N}_1$ e $\mathcal{N}_2$ são utilizadas nesse processo.

Ao modificar o modo de execução de uma ou mais atividades, a solução resultante pode tornar-se inviável em relação aos recursos não renováveis. Para lidar com esse problema, foram consideradas e testadas duas estratégias:

(i) gerar apenas soluções vizinhas viáveis;

(ii) caso a solução resultante seja inviável, manter fixos os movimentos que causaram a inviabilidade e aplicar um procedimento de reparação modificando os demais atributos não fixados.

Entre essas abordagens, a estratégia (ii) apresentou melhor desempenho. Observou-se que, ao incorporar elementos aleatórios durante o processo de reparação, essa estratégia contribuiu para uma maior diversificação da busca, reduzindo a probabilidade de estagnação em ótimos locais. Os resultados comparativos dessas estratégias serão apresentados na Seção 5.5. O procedimento de reparação é descrito na Subsubseção 4.1.1.1.

Após a fase de perturbação, a solução gerada é refinada por um processo de intensificação, conduzido pelo procedimento $B\text{-}VND$, utilizando as estruturas de vizinhança N_1 e N_2 descritas anteriormente na Subsubseção 4.1.2.2.

4.2.4 Procedimento geral GVNS

O Algoritmo 26 apresenta o método GVNS proposto com mudança de vizinhança sequencial. A solução inicial é gerada conforme o procedimento descrito na Subseção 3.6.3, sendo em seguida refinada por meio do B-VND (ver Subsubseção 4.1.2.2). Observou-se empiricamente que a qualidade da solução inicial influencia significativamente o desempenho do método; portanto, o processo de construção e refinamento inicial tem como objetivo fornecer uma solução de boa qualidade para o início da busca.

O algoritmo recebe como entrada o conjunto de vizinhanças $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2\}$, ordenadas de acordo com sua cardinalidade crescente. Durante a execução, o procedimento alterna entre fases de perturbação e busca local, guiada pelo B-VND, até que o critério de parada seja satisfeito. Retornando ao final do processo, a melhor solução encontrada.

Algorithm 26 GVNS para o MRCPSP

Entrada: conjunto de vizinhanças $\mathcal{N}(.), N(.)$

1: $\mathcal{N} = [\mathcal{N}_1, \mathcal{N}_2]$
2: $s \leftarrow$ Gerar solução inicial utilizando *FBI*
3: $s \leftarrow$ Refinar a solução com $B\text{-}VND(N, s)$
4: $f^* \leftarrow f(s)$
5: **while** critério de parada não satisfeito **do**
6: $k \leftarrow 1$
7: **while** $k \leq |\mathcal{N}|$ **do**
8: Gerar um vizinho $s' \in \mathcal{N}^{(k)}(s)$
9: $s'' \leftarrow B\text{-}VND(N, s')$
10: **if** $f(s'') < f^*$ **then**
11: $s \leftarrow s''$
12: $f^* \leftarrow f(s'')$
13: $k \leftarrow 1$
14: **else**
15: $k \leftarrow k + 1$
16: **end if**
17: **end while**
18: **end while**
19: **return** s

Capítulo 5

Resultados Computacionais

Nesse capítulo são apresentados os testes computacionais realizados para os algoritmos propostos. Após, são avaliados os resultados produzidos por meio de comparações entre os melhores resultados apresentados na literatura, bem como as conclusões e perspectiva de trabalhos futuros.

5.1 Descrição dos Experimentos

Os testes computacionais foram realizados no *cluster* da Universidade Federal de Viçosa – UFV (`https://dct.ufv.br`). Utilizou-se um computador (nó de cálculo) com um processador Intel(R) Xeon(R) CPU X5650 (12M de cache, 2,66 GHz, 6,40 GT/s Intel(R) QPI, seis núcleos, 12 threads), totalizando 24 núcleos de execução, com o sistema operacional Linux CentOS.

Para a avaliação dos métodos, foram adotadas as classes de instâncias PSLIB (Kolisch et al., 1992) e MMLIB (Van Peteghem e Vanhoucke, 2014). Na Tabela 5.1 apresentam-se as principais características desses conjuntos de dados.

Tabela 5.1: Características da classe PSLIB e MMLIB.

	PSLIB							MMLIB	
Classe	J10	J12	J14	J16	J18	J20	J30	J50	J100
# Instâncias viáveis	536	547	551	550	552	554	552	540	540
# Modos de execução	3	3	3	3	3	3	3	3	3
# Recursos renováveis	2	2	2	2	2	2	2	2	2
# Recursos não renováveis	2	2	2	2	2	2	2	2	2

Cada instância foi executada 10 vezes, sendo reportadas as médias do desvio percentual e do tempo de processamento. As soluções ótimas de todas as instâncias da classe J30, J50 e J100 são desconhecidas, desse modo, o desempenho dos algoritmos foi avaliado por meio da média de desvio em relação ao limite inferior obtido pelo caminho crítico mínimo (gap_{cpm}). Esse indicador é definido como:

$$gap_{cpm} = \frac{\bar{f} - f_{cpm}}{f_{cpm}}, \tag{5.1}$$

em que f_{cpm} corresponde ao limite inferior do caminho crítico mínimo de cada instância e $\bar{f}$ representa o valor médio do *makespan* obtido pelo algoritmo. Para as classes J10-J20, cujos valores ótimos são conhecidos, na Equação 5.1 substitui-se f_{cpm} por f^*, que representa a solução ótima, resultando na Equação 5.2.

$$gap_{opt} = \frac{\bar{f} - f^*}{f^*}, \tag{5.2}$$

Os valores ótimos dessas instâncias foram obtidas com o solver CPLEX 22.11.

5.2 Pré-Processamento PSLIB

Conforme discutido na Seção 2.3, a classe de instâncias PSLIB apresenta algumas limitações, como soluções inviáveis e informações redundantes. Em vista desse problema, Sprecher et al. (1997) desenvolveram um método de pré-processamento destinado a excluir modos e/ou recursos não renováveis do conjunto de dados dessas instâncias.

Primeiramente, são definidos dois parâmetros auxiliares, $kmin^{\nu}_{jr}$ e $kmax^{\nu}_{jr}$, para cada atividade j e recurso não renovável r. Representam, respectivamente, o menor e o maior consumo do recurso não renovável r entre todos os modos disponíveis de cada atividade j.

Adicionalmente, definem-se os seguintes conceitos:

- **Modo não executável**: um modo m_j é dito não executável em relação a um recurso renovável r se o consumo máximo desse recurso, no respectivo modo, excede a capacidade disponível K^{ρ}_{r}.

  Analogamente, um modo é dito não executável com relação a um recurso não renovável r, se a soma do consumo mínimo dos demais modos e do consumo desse modo ultrapassa a disponibilidade K^{ν}_{r}. Isto é, se

$$\sum_{\substack{i=1 \\ i \neq j}}^{J} kmin^{\nu}_{jr} + k^{\nu}_{jm_jr} > K^{\nu}_{r}, \quad \text{para } r \in R^{\nu}$$

- **Modo ineficiente**: um modo m_j é dito ineficiente se existir outro modo da mesma atividade que domina o primeiro, ou seja, que possui menor ou igual duração e menor ou igual consumo em relação a todos os recursos.

- **Recurso não renovável redundante**: um recurso não renovável é dito redundante se o consumo máximo possível desse recurso, somando todas as atividades, for menor que a capacidade disponível. Isso significa que o recurso não é um limitante e, portanto, pode ser eliminado no modelo sem afetar o conjunto de soluções viáveis.

Após a definição desses conceitos, é executada a seguinte rotina:

- Passo 1: Remover todos os modos não executáveis do projeto.
- Passo 2: Remover todos os recursos não renováveis redundantes do projeto.

- Passo 3: Remover todos modos ineficientes.
- Passo 4: Se algum modo for removido no passo 3, vá para o passo 2.

Caso alguma atividade não possua qualquer modo de execução, a instância é inviável. De modo geral, esse pré-processamento busca eliminar informações desnecessárias ou inviáveis, reduzindo o número de variáveis e restrições do modelo. Como não altera o conjunto de soluções ótimas possíveis, mas acelera o processo de resolução ao simplificar o problema original, essa rotina foi utilizada na dissertação.

Após a aplicação do pré-processamento, 88 instâncias da classe J30 da PSLIB foram identificadas como inviáveis e removidas do conjunto de testes. Além disso, observou-se uma redução na média de modos por atividade, que passou de 3 para 2,88 modos por instância.

5.3 Calibragem dos Parâmetros

O algoritmo G-VNS proposto possui dois parâmetros principais que delimitam número de iterações da método e da busca local VND, denotados por `it_max` e `itVND_max`, respectivamente. Já o algoritmo GRASP possui dois parâmetros centrais: o número de iterações, denotado por `Grasp_max`, e o nível de aleatoriedade, representado por α. O parâmetro α controla o tamanho da LCR, enquanto `Grasp_max` define o critério de parada do algoritmo, ou seja, o número máximo de soluções construídas. Como o procedimento VND foi incorporado ao GRASP, o número de iterações da busca local também foi parametrizado, denotado por `VND_max`. Para o EPR, somente o parâmetro `delta` foi calibrado, este representa o grau de diversidade mínimo requerido, o parâmetro `maxELITE` que indica o tamanho do conjunto elite foi fixado em 5, valor reportado em algumas literaturas para essa heurística.

A calibração desses parâmetros foi realizada de forma automática utilizando o pacote IRACE (López-Ibáñez et al., 2016). Para esse processo, foram selecionadas aleatoriamente 330 instâncias da PSLIB e MMLIB , sendo 50 de cada classe J30, J50 e J100 e 30 de cada classe J10, J12, J14, J16, J18 e J20.

Tabela 5.2: Combinação de valores para a calibração dos parâmetros

Parâmetros	Níveis
`it_max`	$(20, 30, 40)$
`itVND_max`	$(20, 30, 40)$
α	$(0.1, 0.3, 0.5, 0.7, 0.9)$
`Grasp_max`	$(20, 30, 40, 50)$
`VND_max`	$(10, 20, 30, 40)$
`delta`	$(0.00, 0.05, 0.10, 0.15, 0.20)$

A Tabela 5.2 apresenta os conjuntos de valores utilizados para avaliar a influência dos parâmetros de cada algoritmo proposto. Os resultados dessa calibração são os seguintes:

(i) GVNS:

- `it_max` = 20;
- `itVND_max` = 40.

(ii) GRASP:

- α = 0.3;
- `Grasp_max` = 30;
- `VND_max` = 20.

(iii) EPR e IPR:

- `delta` = 0.05;
- `maxELITE` = 5.

Em razão da cardinalidade alta do conjunto de soluções das instâncias J100, foi estipulado um critério de parada de 100s para a execução dos algoritmos dessa classe. Para as abordagens com pós-otimização, foi definido o mesmo limite para cada uma das fases.

5.4 Experimentos VND

Para avaliar o desempenho das estruturas de VND apresentadas na Subsubseção 4.1.2.2, foram selecionadas 30 instâncias aleatórias da classe MMLIB50 com soluções ótimas conhecidas. Cada instância foi resolvida 10 vezes, e a partir desses experimentos calcularam-se as médias do *makespan* obtido e do tempo de resolução para cada estrutura de VND integrada a rotina do GRASP.

A Tabela 5.3 resume os resultados, apresentando o desvio médio em relação ao ótimo conhecido ($\%gap_{opt}$) e o tempo médio de resolução (em segundos).

Tabela 5.3: Resultados médios das estruturas de VND aplicadas ao GRASP.

	VND1	VND2	VND3	VND4	VND5	VND6
$\%gap_{opt}$	5,26	8,14	5,21	5,12	5,13	4,71
Tempo (s)	58,36	74,55	6,61	32,50	28,15	3,73

A Figura 5.1 apresenta os *boxplots* dos tempos de resolução das instâncias em estudo. Pode-se observar que o VND3 e o VND6 apresentam uma variabilidade relativamente baixa nos tempos de execução, indicando consistência no esforço computacional entre diferentes instâncias. Em contraste, o VND1 e o VND2 apresentam maior dispersão, evidenciada pelas diferenças entre os quartis, o que sugere que algumas instâncias demandaram significativamente mais tempo de processamento. Essa análise permite concluir que, em termos de desempenho computacional, o VND3 e o VND6 são mais consistentes, enquanto o VND1 e o VND2 podem apresentar resultados mais sensíveis as características específicas das instâncias.

A Tabela 5.3 evidencia, mesmo em uma análise preliminar, o desempenho superior da estrutura VND6 em comparação as demais, apresentando simultaneamente o menor desvio médio e o menor tempo de resolução. As estruturas de vizinhança VND1 e

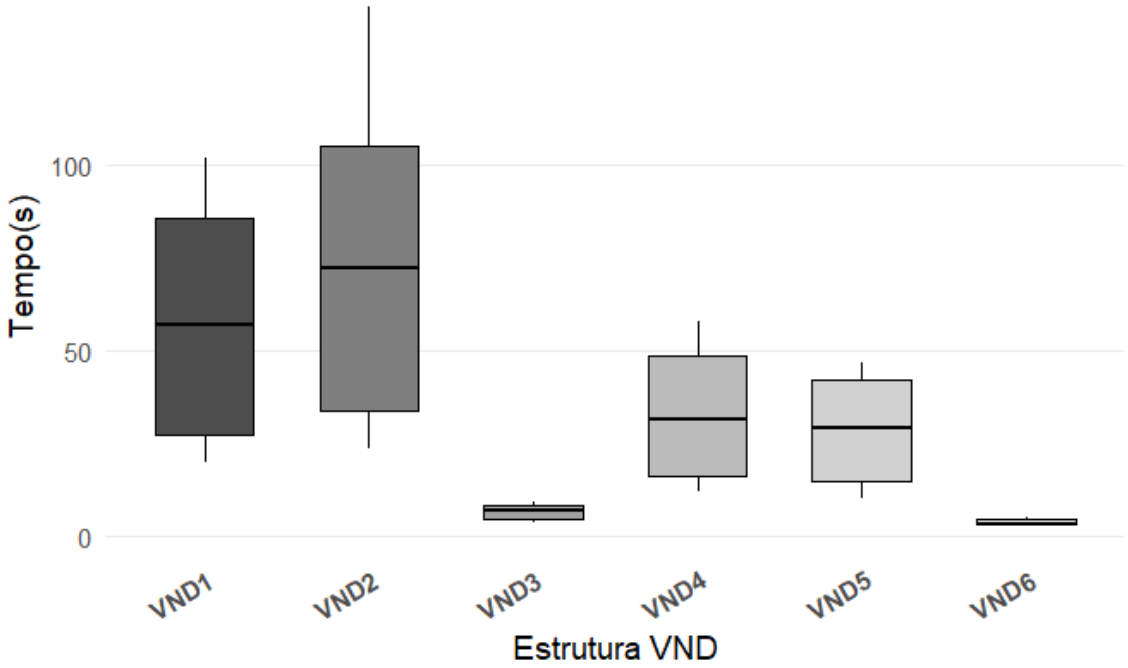


Figura 5.1: Boxplot do tempo de resolução de cada VND

VND2 seguem uma lógica semelhante a do VND6, combinando uma busca intensiva para aprimorar a configuração dos modos de execução com movimentos de troca e realocação de atividades para otimizar o sequenciamento das tarefas. No entanto, apresentaram desempenho inferior, sobretudo em termos de tempo de execução. A principal limitação dessas estruturas decorre da elevada cardinalidade das vizinhanças SH-SCM e SW-SCM, que demandou a adoção da estratégia de *first improvement*, interrompendo a busca assim que uma melhoria é encontrada.

Por outro lado, as vizinhanças SH-SCM-CT e SW-SCM-CT foram projetadas para explorar o espaço de busca de forma mais seletiva, concentrando-se em movimentos que afetam de maneira mais significativa a estrutura do projeto, isto é, as tarefas críticas. Com isso, além de reduzir o espaço de busca e viabilizar a adoção da estratégia *best improvement*, essas vizinhanças direcionam o processo para movimentos mais promissores, evitando esforços em alterações pouco relevantes. Entretanto, essa estratégia também possui limitações, pois, ao restringir o espaço de busca, há o risco de descartar soluções potencialmente melhores que poderiam ser identificadas pelas vizinhanças completas.

O objetivo desse experimento foi identificar a estrutura mais eficiente, considerando tanto o tempo computacional quanto o desvio médio em relação a média, para posterior aplicação nos algoritmo GRASP+EPR, GRASP+IPR e VNS. Nessa perspectiva, a VND6 apresentou desempenho superior e, por esse motivo, foi selecionada para a etapa seguinte dos experimentos.

5.5 Experimentos G-VNS

Nesta seção, apresentamos os experimentos correspondentes aos itens na Subseção 4.2.3. Serão avaliadas duas estratégias de diversificação para o G-VNS, que se diferenciam pela forma com que tratam a geração de soluções inviáveis durante a fase de perturbação. A Estratégia 1 considera apenas a construção de soluções vizinhas viáveis; enquanto a Estratégia 2 quando a solução gerada é inviável, mantém fixos os movimentos responsáveis pela inviabilidade (para evitar ciclagem) e aplica um procedimento de reparação, modificando os demais atributos não fixados.

O objetivo desse experimento é comparar o desempenho das diferentes estratégias propostas. Para isso, foram selecionadas uma amostra de 30% de instâncias das classes J10-J20, cujas soluções ótimas são conhecidas, e avaliou-se o desvio médio em relação a solução ótima (gap_{opt}) e o tempo computacional médio ($T(s)$) necessário para resolver cada grupo de instâncias. A Tabela 5.4 apresenta os valores médios de gap_{opt} e $T(s)$ para ambas as estratégias, enquanto a Figura 5.2 ilustra graficamente os resultados. O boxplot da Figura 5.3 exibe a distribuição do tempo médio para resolver todas as instâncias consideradas para cada est.

Tabela 5.4: Resultados das estratégias por classe

	Classes											
	J10		J12		J14		J16		J18		J20	
	$\%gap_{opt}$	$T(s)$	$\%gap_{opt}$	$T(s)$	$\%gap_{opt}$	$T(s)$	$\%gap_{opt}$	$T(s)$	$\%gap_{opt}$	$T(s)$	$\%gap_{opt}$	$T(s)$
Estratégia 1	4,7	0,39	5,0	0,71	4,5	0,99	6,2	1,063	5,5	1,324	5,4	1,864
Estratégia 2	3,6	0,71	4,8	0,89	3,7	1,02	5,7	1,436	4,0	1,868	4,9	2,813

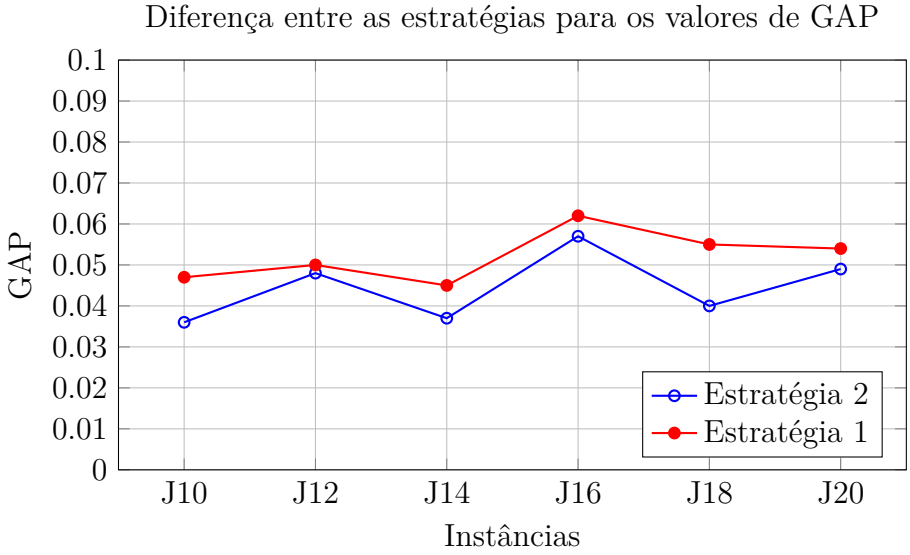


Figura 5.2: Diferença entre as estratégias para os valores de GAP em relação às soluções ótimas por instância solucionada.

Os valores apresentados na Tabela 5.4 e no gráfico da Figura 5.2 indicam que a Estratégia 2 apresenta melhor desempenho em termos de qualidade das soluções, alcançando menor desvio médio em relação ao ótimo em todos conjuntos de dados avaliados quando comparado com a Estratégia 1. Por outro lado, percebe-se que a

Estratégia 1 exigiu menor esforço computacional e apresentou menor variância nos tempos de resolução para resolver as instâncias que a Estratégia 2, como verificado pelo boxplot da Figura 5.3.

Observando os resultados acima, destaca-se as características vantajosas de cada estratégia. A Estratégia 1 é mais rápida em solucionar o problema, enquanto a Estratégia 2, embora mais custosa, é superior em apresentar melhores soluções. Vale destacar que o maior custo da Estratégia 2 é esperado, pois o processo de reparação adiciona etapas extras a busca. No entanto, esse procedimento, que incorpora uma rotina aleatória responsável por modificar os atributos dos modos até garantir a viabilidade da solução, intensifica a perturbação das soluções, favorecendo uma exploração mais ampla do espaço de busca. Como consequência, observa-se que um maior nível de diversificação contribui significativamente para a obtenção de soluções de melhor qualidade. Apesar de demandar mais tempo do que a Estratégia 1, o tempo necessário para a convergência da Estratégia 2 permanece dentro de limites aceitáveis.

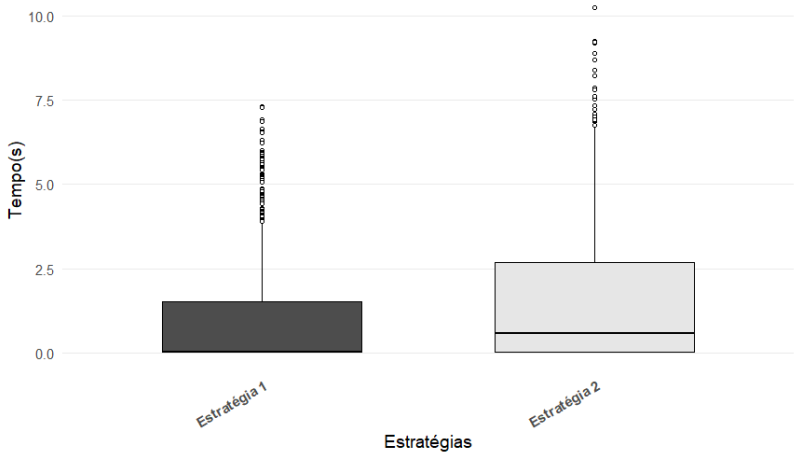


Figura 5.3: Boxplot para a diferença entre os tempos obtidos pelas duas estratégias.

5.6 Resultados

Nesta seção apresentamos os resultados para os algoritmos propostos após a calibração dos parâmetros e a discussão das análises dos experimentos do VND e do G-VNS. O objetivo é comparar o desempenho dos algoritmos G-VNS, GRASP (de forma isolada), GRASP com *Path Relinking Interior* (GRASP+IPR) e GRASP com *Path Relinking Exterior* (GRASP+EPR) para a combinação de parâmetros apresentados.

Para essa comparação, todas as classes da PSLIB, MMLIB50 e MMLIB100 foram resolvidas, sendo reportados o gap_{opt} (para instâncias com ótimos conhecidos J10-J20 na Tabela 5.5) e gap_{cpm} (para instâncias J30,J50 e J100 na Tabela 5.6) além do tempo computacional médio $T(s)$ necessário para resolver cada conjunto de dados. Para as instâncias em que a solução exata é conhecida, foi reportado a porcentagem de soluções ótimas encontradas (Ótimos(%)) por cada algoritmo. Enquanto para as instâncias com soluções exatas desconhecidas, reporta-se o percentual de soluções

que atingiram o valor do limitante inferior fornecido pelo CPM (Ótimos$_{cpm}$(%)). Os valores apresentados pela GRASP representam a média dos resultados obtidos pela primeira fase dos algoritmos GRASP+EPR e GRASP+IPR.

Tabela 5.5: Comparação entre G-VNS, GRASP, GRASP+EPR e GRASP+IPR para as classes J10–J20

Classes	Algoritmo	Ótimos (%)	Gap$_{opt}$ (%)	Tempo (s)
J10	G-VNS	88,82	0,563	1,502
	GRASP	92,16	0,138	1,649
	GRASP+EPR	92,91	0,135	2,076
	GRASP+IPR	93,84	0,134	3,259
J12	G-VNS	84,46	0,429	2,131
	GRASP	85,65	0,264	2,377
	GRASP+EPR	87,02	0,233	3,804
	GRASP+IPR	89,95	0,186	4,697
J14	G-VNS	71,68	0,927	2,719
	GRASP	73,23	0,556	3,101
	GRASP+EPR	74,41	0,518	5,714
	GRASP+IPR	81,95	0,353	7,360
J16	G-VNS	67,09	0,831	3,327
	GRASP	65,45	0,719	4,008
	GRASP+EPR	66,18	0,688	9,065
	GRASP+IPR	67,64	0,513	10,499
J18	G-VNS	64,67	1,000	4,349
	GRASP	62,32	0,967	5,000
	GRASP+EPR	63,04	0,931	8,935
	GRASP+IPR	69,57	0,671	12,170
J20	G-VNS	60,01	1,176	5,291
	GRASP	56,86	1,213	6,139
	GRASP+EPR	57,04	1,162	12,492
	GRASP+IPR	63,54	0,791	16,000

As Tabela 5.5 e Tabela 5.6 reportam os resultados obtidos pelas abordagens G-VNS, GRASP, GRASP+EPR e GRASP+IPR. Observa-se que a proposta baseada no G-VNS apresentou desempenho inferior tanto no número de soluções ótimas encontradas quanto nos valores médios de desvio em relação ao ótimo. Observa-se ainda que, à medida que o número de atividades da instância aumenta, o desempenho desse algoritmo tende a diminuir, indicando que, em espaços de solução mais amplos, como a J100 que, além de apresentar o maior desvio médio, não produziu nenhuma solução ótima segundo o critério adotado, as estruturas de vizinhança adotadas não foram suficientes para conduzir a busca a soluções de maior qualidade.

Naturalmente, o GRASP isolado apresenta desempenho inferior aos algoritmos que incorporam uma fase de pós-otimização. No entanto, ao compará-lo diretamente com o G-VNS, verifica-se que, para instâncias de pequeno porte, o GRASP é mais efetivo, alcançando melhores resultados com tempos de resolução semelhantes. A medida que o tamanho das instâncias aumenta, o desempenho dos dois algoritmos

Tabela 5.6: Comparação entre G-VNS, GRASP, GRASP+EPR e GRASP+IPR para as classes J30, J50 e J100

Classes	Algoritmo	Ótimos$_{cpm}$(%)	Gap$_{cpm}$ (%)	T(s)
J30	G-VNS	46,56	14,7	10,240
	GRASP	47,19	15,4	13,228
	GRASP+EPR	46,92	15,4	13,237
	GRASP+IPR	48,37	14,1	43,582
J50	G-VNS	16,67	29,2	34,265
	GRASP	22,87	30,5	45,456
	GRASP+EPR	23,15	30,4	95,652
	GRASP+IPR	23,70	28,5	121,200
J100	G-VNS	0,00	47,4	41,138
	GRASP	16,67	34,3	55,676
	GRASP+EPR	17,41	34,2	100,841
	GRASP+IPR	17,41	32,5	146,229

torna-se mais próximo, sendo o G-VNS levemente superior ao GRASP para as classes J30 e J50. Para a classe J100, entretanto, o G-VNS apresenta desempenho consideravelmente inferior considerando o mesmo critério de parada por tempo, exibindo uma diferença de aproximadamente de 13% entre os valores de GAP obtidos pelas duas abordagens, conforme evidenciado na Tabela 5.6 .

A partir das Tabela 5.5 e Tabela 5.6, observa-se ainda que o algoritmo GRASP+IPR obteve melhor desempenho dentre as abordagens propostas tanto na quantidade de soluções ótimas encontradas quanto nos menores valores médios de GAP para todas as classes de instâncias avaliadas. Esses resultados indicam, em particular, que a estratégia utilizando o interior PR foi melhor que a estratégia baseada no exterior PR em relação a essas métricas. Apesar da diferença não ser expressiva para instâncias pequenas, isso se dá principalmente porque a fase inicial do GRASP já produz um número elevado de soluções ótimas ou muito próximas do ótimo, com gap_{opt} inferior a 1%, limitando o potencial de melhoria na fase de pós-otimização.

Entretanto, conforme aumenta a complexidade do problema, as vantagens do GRASP+IPR tornam-se mais evidentes, especialmente nas classes J50 e J100, nas quais apresentou uma diferença média de 2% no gap$_{cpm}$ em relação ao GRASP+EPR. Ressalta-se, contudo, que essa melhoria na qualidade das soluções é acompanhada por um aumento significativo no tempo computacional, conforme indicado pela coluna de tempo de resolução T(s).

Embora os dois algoritmos de pós-otimização atuem sobre conjuntos de soluções elite distintos, os resultados evidenciam o potencial dessas abordagens em produzir soluções superiores aquelas obtidas apenas na fase inicial do GRASP. Na próxima seção, será apresentada uma análise detalhada do desempenho individual de cada algoritmo.

5.6.1 Resultados GRASP+IPR e GRASP+EPR

Os algoritmos que incorporam uma fase de pós-otimização foram executados em experimentos distintos. Como consequência, os caminhos gerados pelas diferentes

versões do PR baseiam-se em conjuntos de soluções elite também distintos. Dessa forma, para garantir uma comparação justa entre as abordagens, a análise não se restringiu apenas ao valor do desvio médio obtido, mas considerou igualmente o potencial de cada algoritmo em produzir soluções de melhor qualidade, bem como o desempenho da fase inicial do GRASP em cada variante.

As Tabela 5.7 e Tabela 5.8 apresentam, respectivamente, os resultados obtidos pelos algoritmos GRASP+EPR e GRASP+IPR. Em ambas as tabelas são reportados: o tempo de resolução da fase GRASP, em segundos, $T(s)$; o desvio médio obtido ao final dessa fase, denotado por $Gap_{grasp}(\%)$; o tempo total de execução das versões do PR, $T_{epr}(s)$ e $T_{ipr}(s)$; e os desvios médios finais $Gap_{epr}(\%)$ e $Gap_{ipr}(\%)$.

Tabela 5.7: Desempenho do algoritmo GRASP+EPR

Classes	T(s)	$\text{Gap}_{grasp}(\%)$	T_{epr}(s)	$\text{Gap}_{epr}(\%)$
J10	1,652	0,142	2,076	0,135
J12	2,380	0,243	3,804	0,233
J14	3,106	0,545	5,714	0,518
J16	4,008	0,721	9,065	0,688
J18	5,000	0,967	8,935	0,931
J20	6,148	1,210	12,492	1,162
J30	13,237	15,483	39,950	15,417
J50	45,575	30,530	95,652	30,437
J100	55,7386	34,24	100,841	34,14

Tabela 5.8: Desempenho do algoritmo GRASP+IPR

Classes	T(s)	$\text{Gap}_{grasp}(\%)$	T_{ipr}(s)	$\text{Gap}_{ipr}(\%)$
J10	1,646	0,134	3,259	0,134
J12	2,374	0,284	4,697	0,186
J14	3,097	0,567	7,360	0,353
J16	3,979	0,717	10,499	0,513
J18	4,946	0,944	12,170	0,671
J20	6,129	1,215	16,000	0,791
J30	13,219	15,491	43,582	14,158
J50	45,336	30,525	121,200	28,513
J100	55,613	34,350	146,229	32,58

Inicialmente, vale destacar que o desempenho da GRASP em ambas algoritmos são consistentes, apresentando desempenho médio semelhante e com pouca variabilidade, tanto em tempo de resolução quando no desvio médio obtido. Considerando essa consistência, percebe-se pela Tabela 5.8 que mesmo em conjuntos de dados com pior avaliação, como J12, J14, J20, J30 e J100 a estratégia utilizando o *interior* PR foi capaz de retornar melhores soluções mesmo nesses cenários. No entanto, é considerável o custo computacional associado especialmente para instâncias maiores, como a J100 com diferença de aproximadamente 46 segundos.

A Tabela 5.9 apresenta, adicionalmente, o percentual de soluções ótimas encontradas em cada etapa (opt_{grasp}, opt_{epr} e opt_{ipr}) bem como o percentual de soluções

efetivamente aprimoradas durante a fase de pós-otimização. Esses valores são indicados por EPR+(%) e IPR+(%), correspondendo, respectivamente, às abordagens que utilizam *Exterior Path Relinking* e *Interior Path Relinking*.

O cálculo desses indicadores é realizado da seguinte forma: dentre as soluções que não atingiram o valor ótimo ao término da fase GRASP, contabiliza-se a proporção daquelas que apresentaram melhoria após a aplicação do procedimento de pós-otimização. Esse percentual é obtido pela razão entre o número de soluções melhoradas pelo PR e o total de soluções não ótimas consideradas, multiplicada por 100.

Para as instâncias das classes J30, J50 e J100, cujas soluções ótimas são desconhecidas, o critério adotado consistiu em verificar as instâncias que não atingiram o valor do limitante inferior fornecido pelo CPM. Esses indicadores permitem avaliar e comparar a eficácia do procedimento de pós-otimização em aprimorar soluções de boa qualidade, evidenciando o impacto de cada versão do PR no desempenho global do algoritmo proposto.

Tabela 5.9: Desempenho dos algoritmos GRASP+EPR e GRASP+IPR

	GRASP+EPR			GRASP+IPR		
	$\text{opt}_{grasp}(\%)$	$\text{opt}_{EPR}(\%)$	EPR+(%)	$\text{opt}_{grasp}(\%)$	$\text{opt}_{IPR}(\%)$	IPR+(%)
J10	92,54	92,91	15,00	91,79	93,84	43,18
J12	86,29	87,02	10,67	85,01	89,95	69,51
J14	73,87	74,41	18,06	72,60	81,31	63,58
J16	65,45	66,18	16,32	65,45	67,64	48,95
J18	62,32	63,04	19,42	62,86	69,57	65,85
J20	56,50	57,04	18,99	57,22	63,54	74,68
J30	46,74	46,92	25,85	47,64	48,37	85,47
J50	22,96	23,15	23,56	22,78	23,70	89,69
J100	17,22	17,41	28,41	16,11	17,41	84,26
Média	58,21	58,68	19,58	57,94	61,70	69,46

A Tabela 5.9 indica que o *Interior* PR apresentou desempenho superior ao *Exterior* PR segundo as métricas avaliadas. Observa-se que o algoritmo GRASP+IPR foi capaz de encontrar, em média, um percentual maior de soluções ótimas em comparação à fase anterior do GRASP, atingindo aproximadamente 3,76%, enquanto o GRASP+EPR obteve cerca de 0,47% de novas soluções ótimas. Além disso, o GRASP+IPR conseguiu aprimorar, em média, 69,46% das soluções pertencentes ao conjunto elite, ao passo que no GRASP+EPR esse percentual foi substancialmente menor, alcançando apenas 19,8%. Esse comportamento se mantém independentemente do tamanho da instância considerada.

Esse elevado percentual de soluções melhoradas na fase de pós-otimização contribui, em partes, para explicar o maior tempo computacional observado para o GRASP+IPR, uma vez que os caminhos construídos pelo PR entre soluções são explorados de forma mais intensiva e interrompidos apenas quando não encontra mais melhorias, conforme apresentado pelo Algoritmo 25.

Apesar do melhor desempenho do GRASP+IPR segundo as métricas analisadas, esses resultados não implicam que o método de *Interior* PR seja superior ao *Exterior*

PR. O que se observa é que, para o critério e grau de diversificação adotado e para a estratégia de construção dos caminhos utilizada neste trabalho, o IPR mostrou-se mais eficaz. Esses resultados sugerem que preservar os atributos comuns entre soluções elite, em particular os modos de execução, favorece a intensificação da busca em regiões promissoras do espaço de soluções, em detrimento de estratégias que promovem alterações mais agressivas nesses atributos.

5.7 Comparação com a Literatura

Para fins de comparação com os trabalhos da literatura, foram utilizados os resultados obtidos pela metaheurística GRASP+IPR, que apresentou melhor desempenho dentre os algoritmos implementados. A tabela Tabela 5.10, extraída de Chakrabortty et al. (2020) com o acréscimo do trabalho mais recente de Ramos et al. (2022), reúne os trabalhos utilizados como referência para comparação, enquanto a Tabela 5.11 reporta os resultados de diferentes algoritmos aplicados ao conjunto de dados PSLIB, com no minimo 5.000 cronogramas gerados, e a Tabela 5.12 apresenta os resultados da MMLIB50 e MMLIB100, considerando os critérios de paradas de no mínimo 1.000, 5.000 e 50.000 cronogramas gerados. Vale destacar que, nos estudos considerados, foram reportados resultados correspondentes apenas a soluções viáveis.

Tabela 5.10: Algoritmos da literatura considerados para comparação da resolução do MRCPSP.

Algoritmo	Autores	Instâncias
GA[1]	(Peteghem e Vanhoucke, 2010)	PSLIB/MMLIB
GA[2]	(Lova et al., 2009)	PSLIB/MMLIB
GA[3]	(Alcaraz et al., 2003)	PSLIB
GA[4]	(Hartmann, 2001)	PSLIB
SA	(Józefowska et al., 2001)	PSLIB
SAT	(Coelho e Vanhoucke, 2011)	PSLIB
EDA[1]	(Ayodele et al., 2017)	PSLIB
EDA[2]	(Soliman e Elgendi, 2014)	PSLIB
EDA[3]	(Wang e Fang, 2012)	PSLIB/MMLIB
LSA	(Geiger, 2017)	MMLIB
SCIPBest	(Schnell e Hartl, 2017)	MMLIB
PSO	(Jarboui et al., 2008)	PSLIB/MMLIB
EA	(Elloumi e Fortemps, 2010)	MMLIB
DE	(Damak et al., 2009)	MMLIB
GLS	(Tseng e Chen, 2009)	MMLIB
SS	(Van Peteghem e Vanhoucke, 2011)	MMLIB
SS[1]	(Ranjbar et al., 2009)	PSLIB
PR	(Muritiba et al., 2018)	PSLIB/MMLIB
MVNSH	(Chakrabortty et al., 2020)	PSLIB/MMLIB
MS-ILS	(Ramos et al., 2022)	PSLIB
GRASP+ERP	nossa abordagem	PSLIB/MMLIB

Em Van Peteghem e Vanhoucke (2014), o número de cronogramas gerados é de-

finido como a soma das vezes em que cada atividade do projeto recebe um horário de início viável, dividida pelo número total de atividades. Assim, cada atribuição de um modo viável a uma atividade corresponde a uma fração $\frac{1}{|J|}$ de um cronograma completo.

Nesta dissertação, esse critério de parada não foi adotado, pois poderia resultar no encerramento prematuro dos algoritmos antes da execução da fase de pós-otimização nos métodos GRASP+EPR e GRASP+IPR. Dessa forma, optou-se por executar os algoritmos utilizando suas melhores configurações de parâmetros, sem interrupção antecipada. Como o objetivo principal é comparar esses métodos com o G-VNS, o mesmo critério foi aplicado a esse algoritmo. Ainda assim, os resultados obtidos mostraram-se comparáveis àqueles reportados na literatura para execuções com, no mínimo, 5.000 cronogramas gerados.

Tabela 5.11: Comparação com outras abordagens do desvio médio (%) para a classe PSLIB após pelo menos 5.000 cronogramas gerados

Autores	J10 (%)	J12 (%)	J14 (%)	J16 (%)	J18 (%)	J20 (%)	J30 (%)
GA[1]	0,01	0,09	0,22	0,32	0,42	0,57	13,75
GA[2]	0,06	0,17	0,32	0,44	0,63	0,87	14,58
PSO	0,03	0,09	0,36	0,44	0,89	1,10	18,63
SS[1]	0,18	0,65	0,89	0,95	1,21	1,64	16,21
GA[3]	0,24	0,73	1,00	1,12	1,43	1,91	21,83
SA	1,16	1,73	2,60	4,07	5,52	6,74	11,76
SAT	0,07	0,16	0,32	0,48	0,56	0,80	13,75
EDA[1]	0,06	0,17	0,28	0,40	0,48	0,64	13,95
EDA[2]	0,09	0,12	0,36	0,42	0,85	1,09	25,30
EDA[3]	0,12	0,14	0,43	0,59	0,90	1,28	12,90
PR	0,01	0,00	0,05	0,03	0,09	0,06	12,85
MVNSH	0,15	0,62	0,81	0,15	0,89	0,27	13,63
GRASP+IPR	0,13	0,18	0,35	0,51	0,67	0,79	14,10

Para a classe de instâncias da PSLIB, observa-se, a partir da Tabela 5.11, que os trabalhos de Peteghem e Vanhoucke (2010) e, principalmente, de Muritiba et al. (2018) se destacam pela obtenção dos menores valores de GAP médio. O algoritmo GRASP+IPR proposto, por sua vez, apresenta desempenho competitivo ao estado da arte do MRCPSP. Quando comparada a heurísticas de Lova et al. (2006) e Chakrabortty et al. (2020), além de superar, em algumas instâncias, algoritmos populacionais, como os propostos por Alcaraz et al. (2003) e Jarboui et al. (2008).

Tabela 5.12: Comparação com outras abordagens do desvio médio (%) para a classe MMLIB50 e MMLIB100

Algoritmo	MMLIB50			MMLIB100		
	Número de Sequenciamentos			Número de Sequenciamentos		
	1000	5000	50,000	1000	5000	50,000
MVNSH	42,5	35,6	31,92	73,2	65,7	60,24
PR	25,09	24,24	23,52	26,20	25,18	24,01
LSA	42,96	33,02	28,35	53,26	44,11	32,91
SCIPBest	–	–	26,08	–	–	45,58
EDA[2]	43,17	31,95	28,76	52,94	38,55	30,45
SS	28,17	25,45	23,79	29,77	26,51	24,02
GA[1]	34,07	27,12	24,93	37,58	29,55	25,63
EA	43,84	32,47	29,92	56,21	40,22	30,02
DE	46,19	32,46	26,27	52,31	36,87	28,92
GA[2]	34,16	28,59	26,69	36,29	31,01	27,89
GLS	65,17	37,92	29,44	72,95	66,04	37,04
PSO	49,98	38,86	32,01	91,85	49,41	40,23
GA[4]	35,40	30,61	26,81	39,69	33,98	29,04
MS-ILS	–	–	28,08	–	–	23,56
GRASP+IPR	–	**28,51**	–	–	–	**32,58**

No contexto do MRCPSP, observa-se que grande parte das contribuições da literatura explora algoritmos populacionais e evolucionários, incluindo abordagens baseadas em algoritmos genéticos, enxame de partículas e estratégias de dispersão. Entre os métodos reportados na Tabela 5.12 para as instâncias da MMLIB50, destaca-se o trabalho de Muritiba et al. (2018), que, mesmo com pelo menos 1.000 cronogramas gerados, alcança resultados substancialmente superiores aos demais algoritmos avaliados. Esse método também apresenta o melhor desempenho médio para a classe MMLIB50 e é superado apenas por Ramos et al. (2022) na classe J100, considerando um critério de parada de pelo menos 50.000 cronogramas gerados.

Em contraste, o algoritmo GRASP+IPR adota um critério de parada baseado em tempo, fixado em 100 segundos para cada fase do algoritmo, mesmo com esse critério o desempenho é comparável, em termos de esforço computacional, a abordagens que geram aproximadamente 50.000 cronogramas.

Para verificar se as diferenças observadas entre os algoritmos são estatisticamente significativas e avaliar o desempenho global dos métodos reportados na Tabela 5.10, aplicou-se o teste não paramétrico de Friedman (Zimmerman e Zumbo, 1993), sugerido por Zaman et al. (2020), sobre os desvios médios apresentados nas Tabela 5.11 e Tabela 5.12. Segundo Etminaniesfahani et al. (2024), esse teste permite identificar se existem diferenças significativas entre múltiplos tratamentos avaliados em blocos, por meio de uma abordagem de comparação múltipla baseado em postos (ou ranks), sendo frequentemente aplicado quando não se pode assumir normalidade dos dados.

Para esse teste, a hipótese nula H_0 estabelece que não há diferenças significativas entre os algoritmos comparados e será rejeitada se o $p-valor$ for menor que o nível de significância de 5%. A Tabela 5.13 apresenta os ranks médios obtidos pelos algoritmos nas instâncias PSLIB e da MMLIB, bem como os respectivos valores de $p-valor$

associados ao teste. Foram obtidos $p - valores$ iguais a $2{,}06 \times 10^{-4}$ para a PSLIB e 0,032 para a MMLIB. Como ambos são inferiores a 0,05, rejeita-se H_0, sugerindo a existência de diferenças estatisticamente significativas entre os métodos avaliados.

Tabela 5.13: Ranks médios dos algoritmos para as instâncias PSPLIB e MMLIB

Testes instâncias PSPLIB		Testes instâncias MMLIB	
Algoritmo	Rank médio	Algoritmo	Rank médio
PR	1,2	PR	1
GA_1	2,8	SS	2
EDA_1	4,7	GA_1	3
SAT	5,7	GA_2	4,5
GA_2	6,3	GA_4	6
MVNSH	6,6	DE	6,5
PSO	6,9	**GRASP+IPR**	6,5
EDA_2	7,4	EDA_2	7,5
GRASP+IPR	7,4	EA	8
EDA_3	7,9	LSA	10
SS_1	10,9	GLS	11,5
SA	11,3	MVNSH	12
GA_3	12	PSO	12,5
p-valor	$2{,}06 \times 10^{-4}$	**p-valor**	0,032

A Tabela 5.13 também apresenta os ranks relativos de cada método, ordenados para cada conjunto de dados considerado. Observa-se que o algoritmo PR (Muritiba et al., 2018) obteve o melhor desempenho médio tanto nas instâncias da PSPLIB quanto da MMLIB. Embora a abordagem proposta (GRASP+IPR) não tenha alcançado a primeira posição no ranqueamento, ela apresentou desempenho intermediário consistente, o que evidencia sua robustez. Esses resultados reforçam a competitividade do método proposto, especialmente quando comparado a algoritmos consolidados na literatura, demonstrando sua capacidade de produzir soluções de qualidade com baixo custo computacional, inclusive em instâncias de maior complexidade.

Capítulo 6

Conclusão e Trabalhos Futuros

Esta dissertação apresentou uma implementação das metaheurísticas G-VNS, GRASP, combinada com o procedimento *Exterior Path Relinking* e com *Interior Path Relinking* como método pós-otimização, como estratégias para resolver o Problema de Escalonamento de Projetos com Múltiplos Modos e Recursos (MRCPSP). Para a construção das soluções iniciais, foram realizadas adaptações em métodos determinísticos clássicos da literatura, de modo a integrá-los à fase construtiva do GRASP. Além disso, foram desenvolvidas estratégias de busca local baseadas em VND, especificamente adaptadas às características do problema. Para realizar os experimentos, as classes de bibliotecas públicas PSLIB e MMLIB foram utilizadas para mensurar a qualidade da estratégia proposta.

A partir de estruturas de vizinhança recorrentes em diversos trabalhos para resolver o MRCPSP, foi proposto e implementado algumas variações no conjuntos de vizinhanças para o VND. Essas variações consistiam em modificar a combinação de estruturas de vizinhança do método bem como a estratégia exploração. Dentre as estruturas experimentadas a estratégia de priorizar movimentos sobre as tarefas críticas do projeto mostrou-se promissora para melhorar a solução de forma rápida. No entanto, restringir a busca exclusivamente a essas tarefas mostrou-se um limitante o processo de busca. Isso ocorre porque a noção de criticidade está diretamente vinculada à configuração corrente dos modos, ou seja, uma tarefa pode ser crítica em determinada combinação e deixar de ser em outra. Essa característica pode ter contribuído para que o algoritmo convergisse prematuramente para uma região em que os movimentos restritos sobre uma determinada solução não fossem capazes de promover melhorias adicionais.

Como alternativa de trabalhos futuros, pode-se adotar uma abordagem híbrida, em que parte dos movimentos seja direcionada às tarefas críticas, mas sem eliminar completamente a exploração das demais tarefas. Dessa forma, seria possível manter o foco em alterações mais promissoras, ao mesmo tempo em que se reduz o risco de perder soluções relevantes que só poderiam ser alcançadas por meio de movimentos aplicados a tarefas não críticas.

O GVNS teve uma implementação simples, consistindo, na fase de perturbação, apenas a modificação dos atributos de modos. Verificou-se que essa estratégia é mais eficaz quando considera-se estratégias adicionais de reparação do que apenas a geração de soluções viáveis. Indicando que movimentos que geram maiores pertubações, como possivelmente modificar também a ordem das atividades, podem obter resultados

melhores aos encontrados

No que se diz respeito ao Exterior PR, até onde se tem conhecimento, não foram identificados na literatura trabalhos que explorem essa estratégia especificamente para o MRCPSP. Embora os resultados obtidos tenham sido modestos, a aplicação do EPR ao problema não deve ser descartada. O que se pode concluir é que os critérios de diversificação adotados mostraram-se inadequados para esse método.Por outro lado, o *Interior Path Relinking*, empregado como procedimento de pós-otimização, destacou-se como a estratégia mais eficaz entre os métodos propostos, sendo capaz de encontrar soluções melhores do que aquelas obtidas na fase inicial do GRASP. No entanto, quando comparado tanto à literatura quanto o próprio GRASP, os ganhos observados não se mostraram suficientemente significativos para justificar o elevado custo computacional.

Observou-se que o método apresentou dificuldades em explorar novas regiões do espaço de busca, convergindo de forma prematura para o mesmo ótimo local, independentemente da estrutura de vizinhança utilizada no VND. Esse comportamento indica baixa diversidade no conjunto elite. Mesmo com a parametrização do critério de diversidade por meio do IRACE, os valores permaneceram baixos, em torno de 5%. Empiricamente, constatou-se que, para valores de diversidade superiores a 20% o conjunto elite não era completamente preenchido. Esse comportamento é explicado pelo fato de o critério de diversidade considerar apenas a atribuição de modos e pela baixa cardinalidade do conjunto de modos disponíveis para cada atividade nas instâncias analisadas, o que naturalmente leva diferentes soluções a compartilharem os mesmos modos de execução. Assim, uma alternativa promissora consistiria na definição de critérios de diversidade mais abrangentes, capazes de considerar outras características estruturais das soluções.

Como direções para trabalhos futuros, sugere-se o aprimoramento da fase construtiva, em particular no que se refere à atribuição de modos, bem como o refinamento das estratégias de busca local por meio de abordagens menos restritivas, mas que ainda priorizem tarefas críticas. Nesse mesmo contexto, o conceito de criticidade pode ser explorado no próprio processo de PR, identificando tarefas críticas comuns entre duas soluções, ou críticas em apenas uma delas, e direcionando movimentos sobre esse subconjunto. O objetivo seria reduzir o número de tarefas críticas no cronograma, uma vez que projetos com maior quantidade de atividades críticas tendem a ser mais sensíveis a pequenas alterações, dificultando a obtenção de melhorias adicionais.

Referências Bibliográficas

Afshar, Mohammad Reza; Shahhosseini, Vahid e Sebt, Mohammad Hassan. (2022). A genetic algorithm with a new local search method for solving the multimode resource-constrained project scheduling problem. *International Journal of Construction Management*, v. 22, n. 3, p. 357–365.

Afshar-Nadjafi, Behrouz. (2018). A solution procedure for preemptive multi-mode project scheduling problem with mode changeability to resumption. *Applied computing and informatics*, v. 14, n. 2, p. 192–201.

Alcaraz, Javier; Maroto, Concepcion e Ruiz, Ruben. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, v. 54, p. 614–626.

Alvarez-Valdés, Ramón; Crespo, Enric; Tamarit, José Manuel e Villa, Fulgencia. (2008). Grasp and path relinking for project scheduling under partially renewable resources. *European Journal of Operational Research*, v. 189, n. 3, p. 1153–1170.

Ayodele, Mayowa; McCall, John e Regnier-Coudert, Olivier. (2017). Estimation of distribution algorithms for the multi-mode resource constrained project scheduling problem. *2017 IEEE Congress on Evolutionary Computation (CEC)*, p. 1579–1586. IEEE, (2017).

Ballestín, Francisco; Barrios, Agustín e Valls, Vicente. (2011). An evolutionary algorithm for the resource-constrained project scheduling problem with minimum and maximum time lags. *Journal of scheduling*, v. 14, n. 4, p. 391–406.

Benfedel, Rachida; Belkaid, Fayçal e Brahimi, Nadjib. (2025). Production routing decisions in a two-echelon supply chain with multiple delivery modes. *International Transactions in Operational Research*, v. n/a, n. n/a, p. 1–37. doi: https://doi.org/10.1111/itor.70019.

Blazewicz, J.; Lenstra, J.K. e Kan, A.H.G.Rinnooy. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, v. 5, p. 11–24.

Błażewicz, Jacek; Ecker, Klaus H.; Pesch, Erwin; Schmidt, Günter e Węglarz, Jan. (2001). *Scheduling under Resource Constraints*, p. 317–365. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-662-04363-9. doi: 10.1007/978-3-662-04363-9_9. URL `https://doi.org/10.1007/978-3-662-04363-9_9`.

Boctor, Fayez F e d'Avignon, Gilles. (2005). A tabu search algorithm for the multiple mode resource-constrained project scheduling problem. *Gestion des Opérations et Production*, v. 28, p. 89.

Boctor, Fayez Fouad. (1993). Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *The international journal of production research*, v. 31, n. 11, p. 2547–2558.

Bouleimen, KLEIN e Lecocq, HOUSNI. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, v. 149, p. 268–281.

Brandimarte, Paolo e Fadda, Edoardo. (2024). A reduced variable neighborhood search for the just in time job shop scheduling problem with sequence dependent setup times. *Computers & Operations Research*, v. 167, p. 106634.

Brimberg, Jack; Hansen, Pierre e Mladenovic, Nenad. 06(2010). Attraction probabilities in variable neighborhood search. *4OR quarterly journal of the Belgian, French and Italian Operations Research Societies*, v. 8, p. 181–194. doi: 10.1007/s10288-009-0108-x.

Brimberg, Jack; Salhi, Said; Todosijević, Raca e Urošević, Dragan. (2023). Variable neighborhood search: The power of change and simplicity. *Computers & Operations Research*, v. 155, p. 106221.

Calvete, Herminia I.; Galé, Carmen; Iranzo, José A. e Laguna, Manuel. (2025). Scatter search with path relinking for linear bilevel problems. *European Journal of Operational Research*, v. 326, n. 3, p. 439–450. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2025.04.043. URL `https://www.sciencedirect.com/science/article/pii/S0377221725003285`.

Chakrabortty, Ripon; Abbasi, Alireza e Ryan, Mike. 07(2019). A modified variable neighbourhood search heuristic for the multi-mode resource constrained project scheduling problem with a case study. *INCOSE International Symposium*, v. 29, p. 1124–1139.

Chakrabortty, Ripon K; Abbasi, Alireza e Ryan, Michael J. (2020). Multi-mode resource-constrained project scheduling using modified variable neighborhood search heuristic. *International Transactions in Operational Research*, v. 27, p. 138–167.

Changchun, Liu; Xi, Xiang; Canrong, Zhang; Qiang, Wang e Li, Zheng. (2018). A column generation based distributed scheduling algorithm for multi-mode resource constrained project scheduling problem. *Computers & Industrial Engineering*, v. 125, p. 258–278. ISSN 0360-8352.

Chiang, Chuan-Wen; Huang, Yu-Qing e Wang, Wen-Yen. (2008). Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling. *Journal of Intelligent & Fuzzy Systems*, v. 19, p. 345–358.

Coelho, José e Vanhoucke, Mario. (2011). Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. *European Journal of Operational Research*, v. 213, n. 1, p. 73–82.

Colak, Erdem e Azizoglu, Meral. (2014). A resource investment problem with time/resource trade-offs. *Journal of the Operational Research Society*, v. 65, n. 5, p. 777–790.

Cravo, Gildásio Lecchi. (2009). Escalonamento de projetos com restrições de recursos e múltiplos modos de processamento: soluções heurísticas e uma aplicação à programação de manutenção industrial. Dissertação de mestrado, Universidade Federal do Espírito Santo, Vitória. Curso de Informática.

Cuellar-Usaquén, Daniel; Gomez, Camilo e Álvarez-Martínez, David. (2023). A grasp/path-relinking algorithm for the traveling purchaser problem. *International Transactions in Operational Research*, v. 30, n. 2, p. 831–857.

Dai, H e Cheng, W. (2019). A general variable neighborhood search based on path relinking algorithm for multi-skill resource-constrained project scheduling problem with multiple restrictions. *Journal of Physics: Conference Series*, volume 1341, p. 052006. IOP Publishing, (2019).

Damak, Najeh; Jarboui, Bassem; Siarry, Patrick e Loukil, Taicir. (2009). Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research*, v. 36, n. 9, p. 2653–2659.

De Armas, Jésica e Moreno-Pérez, José A. (2025). A survey on variable neighborhood search for sustainable logistics. *Algorithms*, v. 18, n. 1, p. 38.

Demeulemeester, Erik Leuven; Herroelen, Willy e Herroelen, Willy S. (2002). *Project scheduling: a research handbook*, volume 49. Springer Science & Business Media.

Duarte, Abraham; Sánchez-Oro, Jesús; Mladenović, Nenad e Todosijević, Raca. (2018). *Variable Neighborhood Descent*, p. 341–367. Springer International Publishing, Cham. ISBN 978-3-319-07124-4. doi: 10.1007/978-3-319-07124-4_9. URL `https://doi.org/10.1007/978-3-319-07124-4_9`.

Duarte, Abraham; Sánchez-Oro, Jesús; Resende, Mauricio GC; Glover, Fred e Martí, Rafael. (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, v. 296, p. 46–60.

Elloumi, Sonda e Fortemps, Philippe. (2010). A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 205, n. 1, p. 31–41.

Etminaniesfahani, Alireza; Gu, Hanyu; Naeni, Leila Moslemi e Salehipour, Amir. (2024). An efficient relax-and-solve method for the multi-mode resource constrained project scheduling problem. *Annals of Operations Research*, v. 338, p. 41–68.

Feo, Thomas A e Resende, Mauricio GC. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, v. 6, p. 109–133.

Fernandes, Gustavo Alves. *Metaheurísticas para o Problema de Sequenciamento em Projetos com Restrição em Recursos e Múltiplos Modos de Execução*. PhD thesis, CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS, (2017).

Fernandes, Gustavo Alves e deSouza, Sérgio Ricardo. (2021). A matheuristic approach to the multi-mode resource constrained project scheduling problem. *Computers & Industrial Engineering*, v. 162, p. 107592.

Festa, Paola e Resende, Mauricio GC. (2009). An annotated bibliography of grasp–part ii: Applications. *International Transactions in Operational Research*, v. 16, p. 131–172.

Geiger, Martin Josef. (2017). A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem. *European Journal of Operational Research*, v. 256, n. 3, p. 729–741.

Glover, Fred. (1997). *Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges*, p. 1–75. Springer US, Boston, MA. URL `https://doi.org/10.1007/978-1-4615-4102-8_1`.

Glover, Fred. (2014). Exterior path relinking for zero-one optimization. *International Journal of Applied Metaheuristic Computing (IJAMC)*, v. 5, n. 3, p. 1–8.

Guo, Jiaqi; Long, Jiancheng; Szeto, WY; Tan, Weimin e Jian, Sisi. (2025). Vehicle routing in one-way carsharing service with ridesharing options: A variable neighborhood search algorithm. *Transportation Research Part C: Emerging Technologies*, v. 171, p. 104983.

Hansen, Pierre e Mladenović, Nenad. (2001). Variable neighborhood search: Principles and applications. *European journal of operational research*, v. 130, n. 3, p. 449–467.

Hansen, Pierre; Mladenović, Nenad; Brimberg, Jack e Pérez, José A. Moreno. (2019). *Variable Neighborhood Search*, p. 57–97. Springer International Publishing, Cham. ISBN 978-3-319-91086-4. doi: 10.1007/978-3-319-91086-4_3. URL `https://doi.org/10.1007/978-3-319-91086-4_3`.

Hansen, Pierre; Mladenović, Nenad; Todosijević, Raca e Hanafi, Saïd. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, v. 5, n. 3, p. 423–454.

Hartmann, Sönke e Briskorn, Dirk. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 207, n. 1, p. 1–14. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2009.11.005. URL `https://www.sciencedirect.com/science/article/pii/S0377221709008558`.

Hartmann, Sönke. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, v. 45, n. 7, p. 733–750.

Hartmann, Sönke. (2001). Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, v. 102, p. 111–135.

Hartmann, Sönke. (2014). Time-varying resource requirements and capacities. *Handbook on project management and scheduling vol. 1*, p. 163–176. Springer.

Hartmann, Sönke e Briskorn, Dirk. (2022). An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 297, p. 1–14.

Hill, Alessandro; Lalla-Ruiz, Eduardo; Voß, Stefan e Goycoolea, Marcos. (2019). A multi-mode resource-constrained project scheduling reformulation for the waterway ship scheduling problem. *Journal of scheduling*, v. 22, n. 2, p. 173–182.

Jarboui, Bassem; Damak, Najeh; Siarry, Patrick e Rebai, A. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, v. 195, n. 1, p. 299–308.

Jędrzejowicz, Piotr e Ratajczak-Ropel, Ewa. (2015). Reinforcement learning strategy for solving the mrcpsp by a team of agents. *International Conference on Intelligent Decision Technologies*, p. 537–548. Springer, (2015).

Józefowska, Joanna; Mika, Marek; Różycki, Rafał; Waligóra, Grzegorz e Węglarz, Jan. 02(2001). Simulated annealing for multimode resource-constrained project scheduling. *Annals of Operations Research - Annals OR*, v. 102, p. 137–155. doi: 10.1023/A:1010954031930.

Karakostas, Panagiotis e Sifaleras, Angelo. (2025). Learning-assisted improvements in adaptive variable neighborhood search. *Swarm and Evolutionary Computation*, v. 94, p. 101887.

Kellenbrink, Carolin e Helber, Stefan. (2015). Scheduling resource-constrained projects with a flexible project structure. *European journal of operational research*, v. 246, n. 2, p. 379–391.

Kelley, J. E. Jr. (1963). The critical-path method: Resources planning and scheduling. Muth, J. F. e Thompson, G. L., editors, *Industrial Scheduling*, p. 347–365. Prentice Hall, Englewood Cliffs.

Kelley Jr, James E e Walker, Morgan R. (1959). Critical-path planning and scheduling. *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, p. 160–173, (1959).

Khalili-Damghani, Kaveh; Tavana, Madjid; Abtahi, Amir-Reza e Santos Arteaga, Francisco J. (2015). Solving multi-mode time–cost–quality trade-off problems under generalized precedence relations. *Optimization Methods and Software*, v. 30, n. 5, p. 965–1001.

Klein, Robert. (2000). Project scheduling with time-varying resource constraints. *International Journal of Production Research*, v. 38, n. 16, p. 3937–3952.

Kolisch, Rainer. (1996). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, v. 90, n. 2, p. 320–333.

Kolisch, Rainer. (2013). *Project scheduling under resource constraints: efficient heuristics for several problem classes*. Springer Science & Business Media.

Kolisch, Rainer. (2015). *Shifts, Types, and Generation Schemes for Project Schedules*, p. 3–16. Springer International Publishing, Cham. ISBN 978-3-319-05443-8. doi: 10.1007/978-3-319-05443-8_1. URL `https://doi.org/10.1007/978-3-319-05443-8_1`.

Kolisch, Rainer; Sprecher, Arno e Drexl, Andreas. (1992). Characterization and generation of a general class of resource-constrained project scheduling problems: Easy and hard instances. Relatório técnico, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel.

Laguna, Manuel; Martí, Rafael; Martinez-Gavara, Anna; Perez-Peló, Sergio e Resende, Mauricio GC. (2023). 20 years of greedy randomized adaptive search procedures with path relinking. *arXiv e-prints*, v. n/a, p. arXiv–2312.

Laguna, Manuel e Marti, Rafael. (1999). Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, v. 11, n. 1, p. 44–52.

Laguna, Manuel; Martí, Rafael; Martínez-Gavara, Anna; Pérez-Peló, Sergio e Resende, Mauricio GC. (2025). Greedy randomized adaptive search procedures with path relinking. an analytical review of designs and implementations. *European Journal of Operational Research*, v. 327, n. 3, p. 717–734. doi: https://doi.org/10.1016/j.ejor.2025.02.022.

Lan, Shaowen; Lu, Yongliang e Fan, Wenjuan. (2025). An adaptive variable neighborhood search for the traveling salesman problem with job-times. *Journal of Heuristics*, v. 31, n. 2, p. 18.

Li, Heng e Zhang, Hong. (2013). Ant colony optimization-based multi-mode scheduling under renewable and nonrenewable resource constraints. *Automation in construction*, v. 35, p. 431–438.

Li, Ke Y e Willis, Robert J. (1992). An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, v. 56, n. 3, p. 370–379.

López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Cáceres, Leslie Pérez; Birattari, Mauro e Stützle, Thomas. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58.

Lova, Antonio; Tormos, Pilar e Barber, Federico. (2006). Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, v. 10, n. 30, p. 69–86.

Lova, Antonio; Tormos, Pilar; Cervantes, Mariamar e Barber, Federico. (2009). An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. *International Journal of Production Economics*, v. 117, p. 302–316.

Lozano-Osorio, Isaac; Oliva-García, Andrea e Sánchez-Oro, Jesús. (2023). Dynamic path relinking for the target set selection problem. *Knowledge-Based Systems*, v. 278, p. 110827.

Machado-Domínguez, Luis F; Paternina-Arboleda, Carlos D; Vélez, Jorge I e Barrios-Sarmiento, Agustin. (2021). A memetic algorithm to address the multi-node resource-constrained project scheduling problem. *Journal of Scheduling*, v. 24, n. 4, p. 413–429.

Martí, Rafael e Sandoya, Fernando. (2013). Grasp and path relinking for the equitable dispersion problem. *Computers & Operations Research*, v. 40, n. 12, p. 3091–3099.

Mika, Marek; Waligora, Grzegorz e Węglarz, Jan. (2008). Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. *European Journal of Operational Research*, v. 187, n. 3, p. 1238–1250.

Mika, Marek; Waligóra, Grzegorz e Weglarz, Jan. (2015). *Overview and State of the Art*, p. 445–490. Springer International Publishing, Cham. ISBN 978-3-319-05443-8. doi: 10.1007/978-3-319-05443-8_21. URL `https://doi.org/10.1007/978-3-319-05443-8_21`.

Mladenović, Nenad e Hansen, Pierre. (1997). Variable neighborhood search. *Computers & operations research*, v. 24, n. 11, p. 1097–1100.

Muritiba, Albert Einstein Fernandes; Rodrigues, Carlos Diego e daCosta, Francíio Araùjo. (2018). A path-relinking algorithm for the multi-mode resource-constrained project scheduling problem. *Computers & Operations Research*, v. 92, p. 145–154.

Nguyen, Su e Kachitvichyanukul, Voratas. (2012). An efficient differential evolution algorithm for multi-mode resource-constrained project scheduling problems. *International Journal of Operational Research*, v. 15, n. 4, p. 466–481.

Noori, Siamak; Taghizadeh, Kaveh e others,. (2018). Multi-mode resource constrained project scheduling problem: a survey of variants, extensions, and methods. *International Journal of Industrial Engineering & Production Research*, v. 29, n. 3, p. 293–320.

Okada, Ikutaro; Takahashi, Koji; Zhang, Wenqiang; Zhang, Xiaofu; Yang, Hongyu e Fujimura, Shigeru. (2014). A genetic algorithm with local search using activity list characteristics for solving resource-constrained project scheduling problem with

multiple modes. *IEEJ Transactions on Electrical and Electronic Engineering*, v. 9, n. 2, p. 190–199.

Patterson, James H; Słowiński, R; Talbot, FB e Węglarz, J. (1989). An algorithm for a general class of precedence and resource constrained scheduling problems. *Advances in project scheduling*, p. 3–28. Elsevier.

Peidro, David; Martin, Xabier A; Panadero, Javier e Juan, Angel A. (2024). Solving the uncapacitated facility location problem under uncertainty: a hybrid tabu search with path-relinking simheuristic approach. *Applied Intelligence*, v. 54, n. 7, p. 5617–5638.

Pérez-Peló, Sergio; Sánchez-Oro, Jesús e Duarte, Abraham. (2020). Finding weaknesses in networks using greedy randomized adaptive search procedure and path relinking. *Expert Systems*, v. 37, n. 6, p. e12540.

Peteghem, Vincent Van e Vanhoucke, Mario. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, v. 201, n. 2, p. 409–418. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2009.03.034. URL `https://www.sciencedirect.com/science/article/pii/S037722170900191X`.

Pritsker, A Alan B; Waiters, Lawrence J e Wolfe, Philip M. (1969). Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, v. 16, n. 1, p. 93–108.

Ramos, Alfredo; Miranda-Gonzalez, Pablo; Olivares-Benitez, Elias e Mendoza, Abraham. 03(2025). Generalized benders decomposition-based matheuristics for the multi-mode resource-constrained project scheduling problem. *Optimization and Engineering*, v. n/a, n. n/a, p. 1–45. doi: 10.1007/s11081-025-09964-1.

Ramos, Alfredo S; Olivares-Benitez, Elias e Miranda-Gonzalez, Pablo A. (2022). Multi-start iterated local search metaheuristic for the multi-mode resource-constrained project scheduling problem. *Expert Systems with Applications*, v. 39.

Ranjbar, Mohammad; De Reyck, Bert e Kianfar, Fereydoon. (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, v. 193, n. 1, p. 35–48.

Resende, Mauricio GC e Ribeiro, Celso C. (2003). Grasp and path-relinking: Recent advances and applications. *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, volume 6, p. 1–6. pp. T6-1–T6-6, (2003).

Resende, Mauricio GC e Ribeiro, Celso C. (2005). Grasp with path-relinking: Recent advances and applications. *Metaheuristics: progress as real problem solvers*, v. n/a, n. n/a, p. 29–63.

Resende, Mauricio GC e Ribeiro, Celso C. (2016). *Optimization by GRASP*. Springer.

Resende, Mauricio GC e Werneck, Renato F. (2004). A hybrid heuristic for the p-median problem. *Journal of heuristics*, v. 10, n. 1, p. 59–88.

Ribeiro, Celso C.; Urrutia, Sebastián e deWerra, Dominique. (2023). *Metaheuristics and Local Search*, p. 57–98. Springer International Publishing, Cham. ISBN 978-3-031-37283-4. doi: 10.1007/978-3-031-37283-4_3. URL https://doi.org/10.1007/978-3-031-37283-4_3.

Rodriguez, Francisco J.; Glover, Fred; García-Martínez, Carlos; Martí, Rafael e Lozano, Manuel. (2017). Grasp with exterior path-relinking and restricted local search for the multidimensional two-way number partitioning problem. *Computers & Operations Research*, v. 78, p. 243–254.

Sánchez-Oro, Jesús; López-Sánchez, Ana D; Martínez-Gavara, Anna; Hernández-Díaz, Alfredo G e Duarte, Abraham. (2021). A hybrid strategic oscillation with path relinking algorithm for the multiobjective k-balanced center location problem. *Mathematics*, v. 9, n. 8, p. 853.

Schnell, Alexander e Hartl, Richard F. (2017). On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. *Operations Research Perspectives*, v. 4, p. 1–11.

Schrage, Linus. (1970). Solving resource-constrained network problems by implicit enumeration—nonpreemptive case. *Operations Research*, v. 18, n. 2, p. 263–278.

Seeanner, Florian; Almada-Lobo, Bernardo e Meyr, Herbert. (2013). Combining the principles of variable neighborhood decomposition search and the fix&optimize heuristic to solve multi-level lot-sizing and scheduling problems. *Computers & Operations Research*, v. 40, n. 1, p. 303–317.

Sleptchenko, Andrei; Sifaleras, Angelo e Hansen, Pierre, editors. *Variable Neighborhood Search - 9th International Conference, ICVNS 2022, Abu Dhabi, United Arab Emirates, October 25-28, 2022, Revised Selected Papers*, volume 13863 of *Lecture Notes in Computer Science*, (2023). Springer. ISBN 978-3-031-34500-5. doi: 10.1007/978-3-031-34500-5. URL https://doi.org/10.1007/978-3-031-34500-5.

Soliman, Omar S. e Elgendi, Elshimaa. 02(2014). A hybrid estimation of distribution algorithm with random walk local search for multi-mode resource-constrained project scheduling problems. *International Journal of Computer Trends and Technology*, v. 8. doi: 10.14445/22312803/IJCTT-V8P111.

Souza, Marcone Jamilson Freitas. (2008). Inteligência computacional para otimização. *Notas de aula, Departamento de Computaçao, Universidade Federal de Ouro Preto, disponıvel em http://www. decom. ufop. br/prof/marcone/InteligenciaComputacional/InteligenciaComputacional. pdf*, v. 6.

Sprecher, Arno. (1994). *Resource-Constrained Project Scheduling: Exact Methods for the Multi-Mode Case*, volume 409 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin.

Sprecher, Arno; Hartmann, Sönke e Drexl, Andreas. (1997). An exact algorithm for project scheduling with multiple modes. *Operations-Research-Spektrum*, v. 19, n. 3, p. 195–203.

Sudarsan, Bharadwaj e Pugalendhi, Ganesh Kumar. (2025). Clustering using grasp and path relinking for the max k-cut problem. *Data Mining and Knowledge Discovery*, v. 39, n. 4, p. 28.

Talbot, F Brian. (1982). Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, v. 28, p. 1197–1210.

Tchao, Celso e Martins, Simone L. (2008). Hybrid heuristics for multi-mode resource-constrained project scheduling. Maniezzo, Vittorio; Battiti, Roberto e Watson, Jean-Paul, editors, *Learning and Intelligent Optimization*, p. 234–242, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN 978-3-540-92695-5.

Todosijević, Raca; Stančić, Olivera; Stanimirović, Zorica e Mišković, Stefan. (2025). General variable neighborhood search for the capacitated single allocation hub maximal covering problem: R. todosijević et al. *Top*, v. 33, n. 2, p. 262–303.

Tseng, Lin-Yu e Chen, Shih-Chieh. (2009). Two-phase genetic local search algorithm for the multimode resource-constrained project scheduling problem. *IEEE Transactions on Evolutionary Computation*, v. 13, n. 4, p. 848–857.

Uribe, Nicolás R; Herrán, Alberto e Colmenar, J Manuel. (2024). Path relinking strategies for the bi-objective double floor corridor allocation problem. *Knowledge-Based Systems*, v. 305, p. 112666.

Van Peteghem, Vincent e Vanhoucke, Mario. (2011). Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *Journal of Heuristics*, v. 17, n. 6, p. 705–728.

Van Peteghem, Vincent e Vanhoucke, Mario. (2014). 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.

Van Peteghem, Vincent e Vanhoucke, Mario. (2014). 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. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2013.10.012. URL `https://www.sciencedirect.com/science/article/pii/S0377221713008357`.

Vanhoucke, Mario e Coelho, José. (2018). A tool to test and validate algorithms for the resource-constrained project scheduling problem. *Computers & Industrial Engineering*, v. 118, p. 251–265.

Vanhoucke, Mario e Coelho, Jose. (2019). Resource-constrained project scheduling with activity splitting and setup times. *Computers & Operations Research*, v. 109, p. 230–249.

Vanhoucke, Mario; Coelho, José; Debels, Dieter; Maenhout, Broos e Tavares, Luís V. (2008). An evaluation of the adequacy of project network generators with systematically sampled networks. *European Journal of Operational Research*, v. 187, n. 2, p. 511–524.

Voudouris, Christos; Tsang, Edward P.K. e Alsheddy, Abdullah. (2010). *Guided Local Search*, p. 321–361. Springer US, Boston, MA. ISBN 978-1-4419-1665-5. doi: 10.1007/978-1-4419-1665-5_11. URL `https://doi.org/10.1007/978-1-4419-1665-5_11`.

Waligóra, Grzegorz. (2014). Discrete-continuous project scheduling with discounted cash inflows and various payment models—a review of recent results. *Annals of Operations Research*, v. 213, n. 1, p. 319–340.

Wang, Ling e Fang, Chen. (2012). An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. *Computers & Operations Research*, v. 39, n. 2, p. 449–460.

Watermeyer, Kai e Zimmermann, Jürgen. (2020). A branch-and-bound procedure for the resource-constrained project scheduling problem with partially renewable resources and general temporal constraints. *OR spectrum*, v. 42, n. 2, p. 427–460.

Wauters, Tony; Verbeeck, Katja; Berghe, G Vanden e De Causmaecker, Patrick. (2011). Learning agents for the multi-mode project scheduling problem. *Journal of the operational research society*, v. 62, n. 2, p. 281–290.

Wauters, Tony; Verbeeck, Katja; Vanden Berghe, Greet e De Causmaecker, Patrick. (2009). A multi-agent learning approach for the multi-mode resource-constrained project scheduling problem. *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, p. 1–8. International Foundation for Autonomous Agents and Multiagent Systems, (2009).

Xiao, Yiyong; Kaku, Ikou; Zhao, Qiuhong e Zhang, Renqian. (2011). A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems. *European Journal of Operational Research*, v. 214, n. 2, p. 223–231.

Zabihi, Sina; Kahag, Mahdi Rashidi; Maghsoudlou, Hamidreza e Afshar-Nadjafi, Behrouz. (2019). Multi-objective teaching-learning-based meta-heuristic algorithms to solve multi-skilled project scheduling problem. *Computers & Industrial Engineering*, v. 136, p. 195–211.

Zaman, Forhad; Elsayed, Saber; Sarker, Ruhul e Essam, Daryl. (2020). Hybrid evolutionary algorithm for large-scale project scheduling problems. *Computers & Industrial Engineering*, v. 146, p. 106567.

Zhang, Hong; Tam, CM e Li, Heng. (2006). Multimode project scheduling based on particle swarm optimization. *Computer-aided civil and infrastructure engineering*, v. 21, n. 2, p. 93–103.

Zhang, Lieping; Luo, Yingxiong e Zhang, Yu. (2015). Hybrid particle swarm and differential evolution algorithm for solving multimode resource-constrained project scheduling problem. *Journal of Control Science and Engineering*, v. 2015, n. 1, p. 923791.

Zimmerman, Donald W e Zumbo, Bruno D. (1993). Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks. *The Journal of Experimental Education*, v. 62, n. 1, p. 75–86.