UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO

LUCAS DOS SANTOS CARVALHO

META-HEURÍSTICA HÍBRIDA *CLUSTERING SEARCH* COM *ITERATED LOCAL SEARCH* PARA O PROBLEMA DE ESCALONAMENTO DE PROJETOS COM RESTRIÇÕES DE RECURSOS

ALEGRE - ES
DEZEMBRO 2024

LUCAS DOS SANTOS CARVALHO

META-HEURÍSTICA HÍBRIDA*CLUSTERING SEARCH* COM *ITERATED LOCAL SEARCH* PARA O PROBLEMA DE ESCALONAMENTO DE PROJETOS COM RESTRIÇÕES DE RECURSOS

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Geraldo Regis Mauri

ALEGRE - ES

2024

Lucas dos Santos Carvalho

Meta-Heurística Híbrida Clustering Search com Iterated Local Search para o Problema de Escalonamento de Projetos com Restrições de Recursos

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em 18 de dezembro de 2024.

COMISSÃO EXAMINADORA

Prof. Dr. Geraldo Regis Mauri
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Dayan de Castro Bissoli
Universidade Federal do Espírito Santo

Prof. Dr. Edmar Hell Kampke
Universidade Federal do Espírito Santo

RESUMO

O escalonamento de projetos é um problema essencial para empresas de produção sob encomenda (*make-to-order*), sendo o *Resource-Constrained Project Scheduling Problem* (RCPSP) uma de suas principais formulações. O RCPSP consiste em uma generalização do *Job Shop Scheduling Problem*. O RCPSP consiste em escalonar atividades de um ou mais projetos, respeitando restrições de recursos renováveis, com o objetivo de minimizar a duração total do projeto (*makespan*). Neste trabalho, utiliza-se a meta-heurística *Clustering Search* (CS) para resolver a formulação clássica do RCPSP. A CS é combinada à estratégia *Iterated Local Search* (ILS) como meta-heurística de busca, que difere de abordagens anteriores, como a *Simulated Annealing* (SA), utilizadas no RCPSP. Os experimentos computacionais demonstraram que a CS alcança soluções ótimas ou próximas das melhores conhecidas com desvios médios inferiores a 11%, superando resultados reportados na literatura. Esses resultados indicam que a CS possui potencial para aplicação em cenários práticos do RCPSP, devido à sua eficácia e eficiência.

Palavras-chave: *Resource Constrained Project Scheduling Problem; Clustering Search*; *Iterated Local Search*.

LISTA DE FIGURAS

Figura 1 Exemplo de gráfico de solução do RCPSP. 11

Figura 2 Exemplo de RCPSP. 12

Figura 3 Ilustração da utilização dos recursos. 13

Figura 4 Estruturas de dados para representar uma solução. 27

Figura 5 Deslocamento das atividades. 29

Figura 6 Os componentes do CS. 32

LISTA DE TABELAS

Tabela 1 Resultados da busca na plataforma *Google Scholar* utilizando as palavras-chave em inglês. 16

Tabela 2 Resultados da busca na plataforma *Google Scholar* utilizando as palavras-chave em português. 16

Tabela 3 Resultados da busca na plataforma *ScienceDirect* utilizando as palavras-chave em inglês. 17

Tabela 4 Resultados da busca na plataforma *ScienceDirect* utilizando as palavras-chave em português. 18

Tabela 5 Resultados da busca na plataforma *SpringerLink* utilizando as palavras-chave em inglês. 18

Tabela 6 Resultados da busca na plataforma *SpringerLink* utilizando as palavras-chave em português. 19

Tabela 7 Resultados da busca na plataforma *IEEEXplore* utilizando as palavras-chave em inglês. 19

Tabela 8 Resultados da busca na plataforma *IEEEXplore* utilizando as palavras-chave em português. 20

Tabela 9 Principais estratégias híbridas para resolução do RCPSP. 25

Tabela 10 Parâmetros do CS+ILS. 42

Tabela 11 Resultados dos testes para busca local na meta-heurística de busca ILS. 44

Tabela 12 Resultados comparativos entre PSPLIB e CS+ILS. 46

Tabela 13 Resultados comparativos entre PSPLIB e Bissoli e Mauri (2024). . . . 46

Tabela 14 Resultados comparativos entre CS+ILS e Bissoli e Mauri (2024). . . . 46

LISTA DE SIGLAS

AM	*Analyser Module*
CS	*Clustering Search*
IC	*Iterative Clustering Component*
ILS	*Iterated Local Search*
LS	*Local Search Heuristic*
PSPLIB	*Project Scheduling Problem Library*
RCPSP	*Resource-Constrained Project Scheduling Problem*
SA	*Simulated Annealing*
SM	*Search Metaheuristic*

SUMÁRIO

1.1 O problema e sua importância 11

1.2 Objetivos 13

1.2.1 Objetivo geral 13

1.2.2 Objetivos específicos 13

1.3 Estrutura do Trabalho 14

2 Revisão de Literatura 15

2.1 Trabalhos relacionados ao RCPSP 20

2.2 Trabalhos relacionados ao CS 22

2.3 O estado da arte do RCPSP 24

3 Metodologia 26

3.1 Modelagem do RCPSP 26

3.1.1 Estruturas de dados para representação da solução 26

3.1.2 Solução inicial 27

3.1.3 Estrutura de vizinhança 28

3.2 *Iterated Local Search* 29

3.2.1 Solução Inicial 30

3.2.2 Perturbação 30

3.2.3 Critério de Aceitação 30

3.3 *Clustering Search* 31

3.3.1 Meta-Heurística de Busca 32

3.3.2 Componente de Agrupamento Iterativo 34

3.3.3 Módulo Analisador 35

3.3.4 Heurística de Busca Local 35

4 Experimentos Computacionais 41

4.1 Configuração dos Experimentos 41

4.2 Busca Local 42

4.3 Resultados 45

5 Considerações Finais 49

Referências 50

1 INTRODUÇÃO

O escalonamento de projetos é um fator importante para empresas de produção sob encomenda (*make-to-order*), onde as capacidades são reduzidas para lidar com os conceitos de gerenciamento enxuto (BRUCKER et al., 1999).

O problema de escalonamento de projetos com restrições de recursos, conhecido como *Resource-Constrained Project Scheduling Problem* (RCPSP), consiste em organizar as atividades de um ou mais projetos, do início ao fim, sem atrasos, utilizando recursos renováveis e buscando minimizar a duração total do projeto. Esse problema é denotado como sendo uma generalização do *Job Shop Scheduling Problem* (HERROELEN; DEMEULEMEESTER; REYCK, 1999). Segundo Kolisch e Hartmann (1999), o objetivo do RCPSP é garantir a precedência e viabilidade das atividades para que sejam concluídas no prazo previsto, maximizando a eficiência e minimizando o tempo total do projeto.

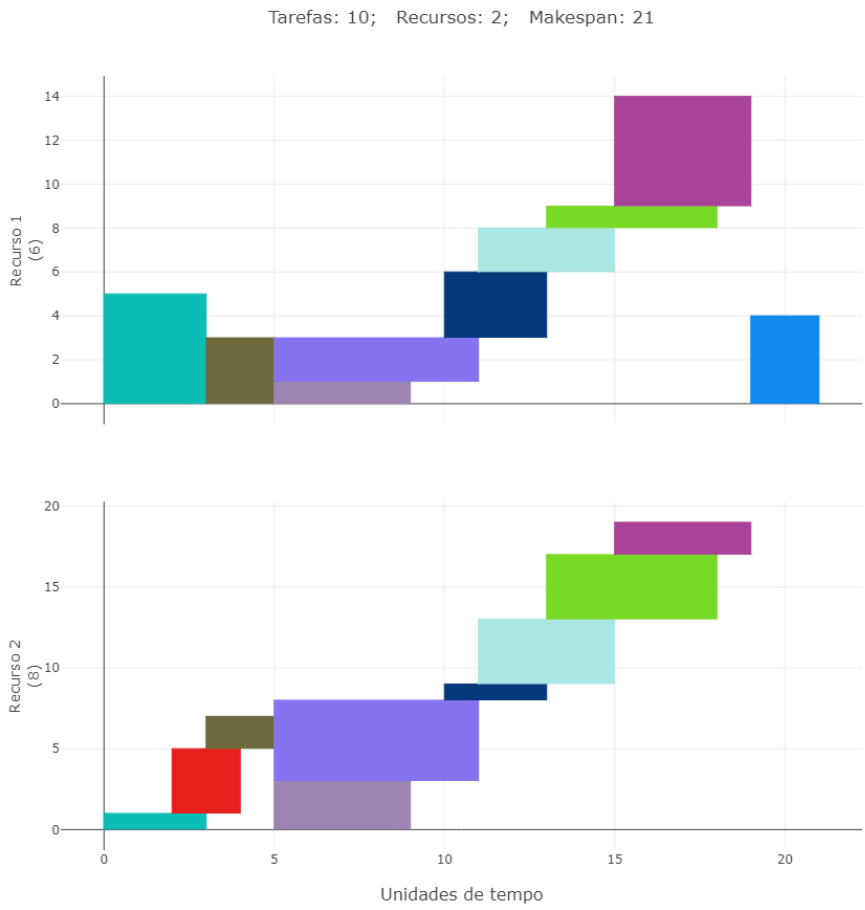
Neste trabalho, apresenta-se a aplicação de uma meta-heurística híbrida para a resolução do RCPSP. A escolha da meta-heurística *Clustering Search* (CS) baseia-se em seu bom desempenho em outros problemas abordados na literatura, como o *Berth Allocation Problem* (OLIVEIRA; MAURI; LORENA, 2012), o *Point-Feature Cartographic Label Placement* (RABELLO et al., 2014) e o *Bi-Objective Flexible Job Shop Scheduling Problem* (ALTOÉ et al., 2018). Embora o CS já tenha sido aplicado com sucesso em outros contextos, sua utilização no RCPSP foi pouco explorada. Este trabalho, além de investigar essa aplicação, compara os resultados obtidos com o estudo de Bissoli e Mauri (2024), que também emprega o CS para o RCPSP, mas em conjunto com a meta-heurística de busca *Simulated Annealing* (SA). Neste estudo, optou-se pela combinação do CS com *Iterated Local Search* (ILS), uma abordagem ainda não aplicada à resolução desse problema.

1.1 O problema e sua importância

O RCPSP é relevante para profissionais da área de gestão de projetos e planejamento de operações, abrangendo uma ampla gama de aplicações que incluem desde *softwares* de gerenciamento de projetos até sistemas de planejamento e escalonamento de produção (PELLERIN; PERRIER; BERTHAUT, 2020).

A Figura 1 ilustra uma representação gráfica das tarefas de uma solução ótima para uma instância composta por dez tarefas (em cores distintas) e dois recursos, com capacidades de seis e oito unidades. A figura contém dois gráficos, nos quais cada barra representa uma tarefa e indica a utilização dos recursos ao longo do tempo. Os gráficos ilustram como as tarefas compartilham os recursos disponíveis, respeitando as limitações previamente estabelecidas. O *makespan* é representado no eixo x, simbolizando o tempo.

Figura 1: Exemplo de gráfico de solução do RCPSP.

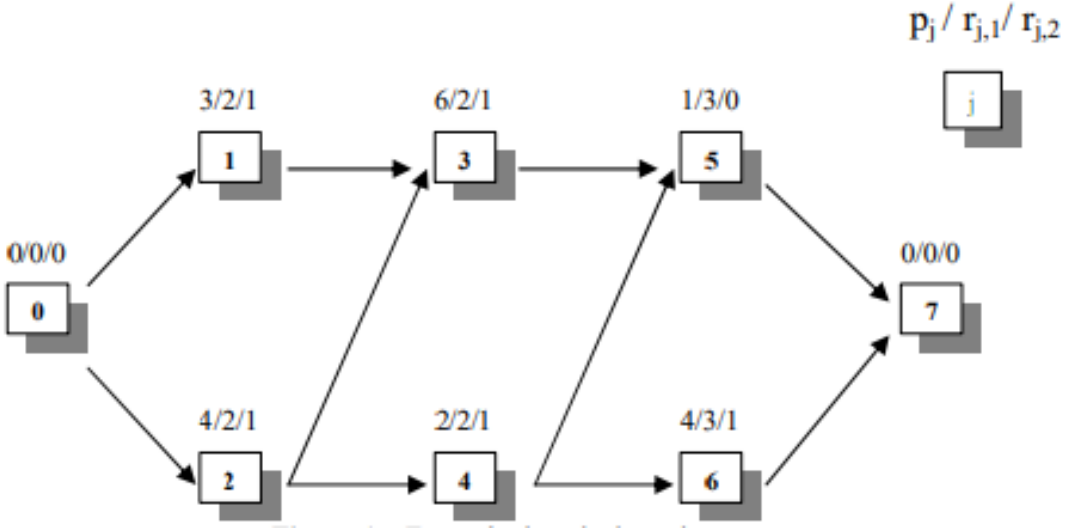


Fonte: O autor.

Como ilustrado na Figura 1, o RCPSP consiste em encontrar uma escala de atividades a fim de minimizar a duração dos projetos (*makespan*). O *makespan* é definido como a diferença entre o início e o final dos projetos, sendo restringido por cada atividade que é executada uma única vez. Cada atividade inicia-se apenas se todas as suas atividades predecessoras estiverem completas e se houver disponibilidade do recurso renovável necessário (PELLERIN; PERRIER; BERTHAUT, 2020).

Para ilustrar a precedência das tarefas no RCPSP, a Figura 2 apresenta um diagrama sequencial de atividades referente a um exemplo com dois recursos e oito atividades, incluindo as tarefas inicial e final. No diagrama, p_j indica a duração de cada etapa, enquanto $r_{j.1}$ e $r_{j.2}$ representam a utilização dos respectivos recursos. As relações de precedência entre as tarefas são indicadas por setas, e os valores numéricos acima dos retângulos que simbolizam as atividades expressam a duração e o consumo de recursos. Observa-se que a tarefa três tem como predecessoras as tarefas um e dois.

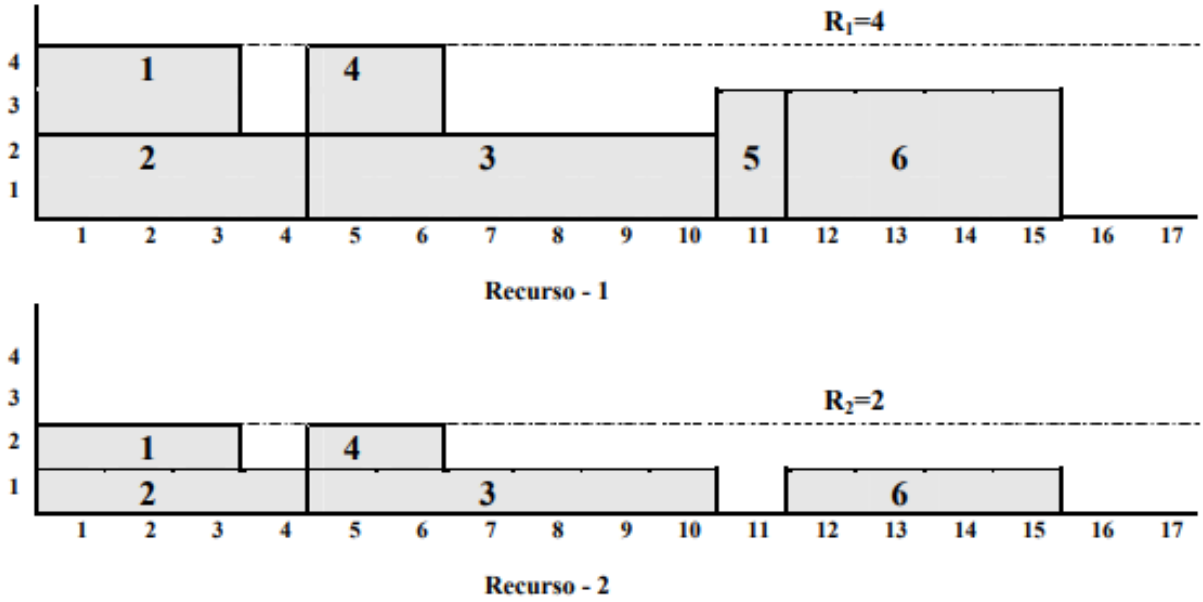
Figura 2: Exemplo de RCPSP.



Fonte: Mendes e Gonçalves (2004).

Uma outra representação do mesmo exemplo é apresentada na Figura 3, que ilustra a utilização dos recursos $r_{j.1}$ e $r_{j.2}$ ao longo do tempo por meio de dois gráficos. Nessa figura, observa-se a alocação dos recursos pelas tarefas, conforme descrito na Figura 2. Nota-se que, embora haja recursos disponíveis ao término da tarefa um, a tarefa três não se inicia até a conclusão da tarefa dois, em conformidade com a relação de precedência mencionada anteriormente.

Figura 3: Ilustração da utilização dos recursos.



Fonte: Mendes e Gonçalves (2004).

1.2 Objetivos

Nesta seção, são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo geral

Aplicar a meta-heurística CS com a meta-heurística ILS como geradora de soluções para a resolução do RCPSP, visando avaliar seu desempenho em comparação com estudos existentes na literatura.

1.2.2 Objetivos específicos

1. Implementar a meta-heurística CS em conjunto da meta-heurística ILS, representada como CS+ILS, para resolver o RCPSP;

2. Realizar experimentos computacionais utilizando as instâncias de teste disponibilizadas pela biblioteca PSPLIB[1];

[1] https://www.om-db.wi.tum.de/psplib/main.html

3. Comparar os resultados obtidos com o estudo de Bissoli e Mauri (2024), que utiliza a meta-heurística CS com a meta-heurística de busca SA, destacando as diferenças e potenciais vantagens das abordagens.

1.3 Estrutura do Trabalho

O Capítulo 2 deste trabalho apresenta uma revisão das principais obras da literatura, incluindo estudos com abordagens similares ao RCPSP, utilizando ou não o CS. O Capítulo 3 detalha a metodologia adotada, e o Capítulo 4 expõe os resultados obtidos. As considerações finais são discutidas no Capítulo 5, seguidas pelas referências bibliográficas.

2 REVISÃO DE LITERATURA

Este capítulo apresenta outros trabalhos que foram utilizados como base para o desenvolvimento deste.

A busca de trabalhos correlatos ao RCPSP foi realizada nas plataformas *Google Scholar*[1], *ScienceDirect*[2], *IEEEXplore*[3] e *SpringerLink*[4], utilizando como parâmetros de busca as palavras-chave: *"Clustering Search"*, *"Resource Constrained Project Scheduling Problem"* e "RCPSP"; e em português: "Problema de Escalonamento de Projetos com Restrições de Recursos" e "Meta-heurística RCPSP". A busca de cada palavra-chave foi realizada de maneira independente, de modo a atrelar a palavra-chave ao seu respectivo número de resultados.

Para a seleção dos trabalhos, utilizou-se uma análise baseada inicialmente na leitura do título de cada um dos resultados da busca, verificando a existência de relação entre o mesmo e o assunto tratado nesta pesquisa. Para os trabalhos selecionados, foi realizada a leitura do seu resumo.

As Tabelas 1 e 2 apresentam o resultado da análise das buscas realizadas na plataforma *Google Scholar*. Deve-se ressaltar que esta plataforma limita o número de resultados entre 10 e 50 links por página, o que reduz o número total de resultados acessíveis a menos de 1000 por pesquisa. Logo, não foi possível analisar todos os artigos das palavras-chave que tiveram o resultado maior do que 1000.

[1] https://scholar.google.com/
[2] https://www.sciencedirect.com/
[3] https://ieeexplore.ieee.org/
[4] https://link.springer.com/

Tabela 1: Resultados da busca na plataforma *Google Scholar* utilizando as palavras-chave em inglês.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Clustering Search"	4.280	1
"Resource Constrained Project Scheduling Problem"	469.000	1
"RCPSP"	7.310	0
Total	**480.590**	**2**

Fonte: O autor.

Devido à quantia elevada de resultados, os artigos passíveis de análise foram tidos como os disponíveis nas primeiras páginas de busca, como dito anteriormente, e dos 480.590 resultados apresentados na Tabela 1, quatro artigos haviam sido selecionados, porém dois foram descartados pela impossibilidade de acesso, mesmo utilizando as credenciais institucionais da UFES; portanto, apenas dois foram selecionados.

Como pode ser observado na Tabela 2, dois dos 11.901 resultados obtidos foram selecionados por estarem diretamente relacionados com o tema tratado nesta pesquisa, enquanto os outros resultados encontrados nas primeiras páginas de pesquisa não satisfaziam as condições para seleção e, na melhor das hipóteses, tangenciam o assunto.

Tabela 2: Resultados da busca na plataforma *Google Scholar* utilizando as palavras-chave em português.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Problema de Escalonamento de Projetos com Restrições de Recursos"	11.800	1
"Meta-heurística RCPSP"	101	1
Total	**11.901**	**2**

Fonte: O autor.

As Tabelas 3 e 4 apresentam o resultado da análise das buscas realizadas na plataforma *ScienceDirect*.

A Tabela 3 também apresenta uma quantidade elevada de resultados, totalizando 18.837. Na primeira etapa de seleção, foram escolhidos cinco trabalhos, enquanto nas etapas subsequentes (segunda e terceira) nenhum resultado foi incluído. Contudo, um dos trabalhos inicialmente selecionados foi descartado, resultando na escolha final de quatro trabalhos, todos eles referentes a outros problemas que utilizam CS como parte de suas soluções.

Tabela 3: Resultados da busca na plataforma *ScienceDirect* utilizando as palavras-chave em inglês.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Clustering Search"	759	4
"Resource Constrained Project Scheduling Problem"	17.372	0
"RCPSP"	706	0
Total	**18.837**	**4**

Fonte: O autor.

A Tabela 4 não apresenta nenhum resultado devido à plataforma *ScienceDirect* favorecer a publicação de artigos na língua inglesa e, dessa maneira, nenhum artigo foi selecionado.

Tabela 4: Resultados da busca na plataforma *ScienceDirect* utilizando as palavras-chave em português.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Problema de Escalonamento de Projetos com Restrições de Recursos"	0	0
"Meta-heurística RCPSP"	0	0
Total	**0**	**0**

Fonte: O autor.

As Tabelas 5 e 6 apresentam o resultado da análise das buscas realizadas na plataforma *SpringerLink*.

A Tabela 5 possui uma quantidade elevada de resultados, totalizando 48.919. Na primeira palavra-chave, foram selecionados três trabalhos, porém estes mesmos não puderam ser acessados, mesmo com as credenciais institucionais. Na segunda e terceira etapas, foram selecionados apenas dois resultados, um em cada etapa. Portanto, dos cinco resultados escolhidos, apenas dois trabalhos foram obtidos em sua integridade e, assim sendo, selecionados.

Tabela 5: Resultados da busca na plataforma *SpringerLink* utilizando as palavras-chave em inglês.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Clustering Search"	1.146	0
"Resource Constrained Project Scheduling Problem"	46.727	1
"RCPSP"	1.046	1
Total	**48.919**	**2**

Fonte: O autor.

A Tabela 6 também não apresenta nenhum resultado devido à plataforma *SpringerLink*

favorecer a publicação de artigos na língua inglesa e, dessa maneira, nenhum artigo foi selecionado.

Tabela 6: Resultados da busca na plataforma *SpringerLink* utilizando as palavras-chave em português.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Problema de Escalonamento de Projetos com Restrições de Recursos"	0	0
"Meta-heurística RCPSP"	0	0
Total	**0**	**0**

Fonte: O autor.

As Tabelas 7 e 8 apresentam o resultado da análise das buscas realizadas na plataforma *IEEEXplore*.

A Tabela 7 possui 1.398 resultados. Apesar da quantia elevada de resultados, apenas dois preencheram satisfatoriamente as condições e, portanto, foram selecionados.

Tabela 7: Resultados da busca na plataforma *IEEEXplore* utilizando as palavras-chave em inglês.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Clustering Search"	885	1
"Resource Constrained Project Scheduling Problem"	367	0
"RCPSP"	146	1
Total	**1.398**	**2**

Fonte: O autor.

A Tabela 8 não apresenta nenhum resultado para artigos em língua portuguesa devido ao problema abordado anteriormente e, dessa maneira, nenhum artigo foi selecionado.

Tabela 8: Resultados da busca na plataforma *IEEEXplore* utilizando as palavras-chave em português.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
"Problema de Escalonamento de Projetos com Restrições de Recursos"	0	0
"Meta-heurística RCPSP"	0	0
Total	**0**	**0**

Fonte: O autor.

Após a análise de todos os resultados, 12 trabalhos relacionados ao tema desta pesquisa foram selecionados, sendo: cinco sobre o RCPSP, seis sobre o CS e um de revisão de literatura. Além disso, outros oito trabalhos foram identificados por meio de referência cruzada, sendo: cinco relacionados ao RCPSP, um ao CS e dois de revisão de literatura. Ressalta-se, ainda, um último trabalho que aborda tanto o RCPSP quanto o CS, totalizando 21 trabalhos. Estes são apresentados em detalhes a seguir.

2.1 Trabalhos relacionados ao RCPSP

Pritsker, Waiters e Wolfe (1969) apresentam um algoritmo para a resolução da variante de múltiplos projetos do RCPSP que utiliza uma solução orientada a programação linear 0-1 (zero-um). O algoritmo foi capaz de acomodar uma grande gama de condições e se provou ser mais eficiente que outros algoritmos, tidos até 1968, em termos de número de variáveis e constantes.

Ikonomou, Galletly e Daniel (1998) apresentam um algoritmo para a resolução do RCPSP utilizando uma heurística adaptativa chamada *Tabu Search*. Os autores submeteram o algoritmo a instâncias baseadas em problemas reais obtidos no projeto europeu PAMIPS e obtiveram resultados demonstrando que, para instâncias de pequena e média escala, é possível obter resultados viáveis; porém, para uma escala maior, o *Tabu Search* não se mostrou capaz de produzir uma solução viável.

Lorenzoni, Albuquerque e Nascimento (2001) apresentam uma implementação da

meta-heurística SA para a resolução do *Multi-Mode Resource-Constrained Project Scheduling Problem* em conjunto com uma nova forma de representação do escalonamento em árvores. Os resultados obtidos através de 15 instâncias padrão apresentaram bons resultados na busca de soluções viáveis.

Mendes e Gonçalves (2004) apresentam uma abordagem que combina um *Genetic Algorithm* com um esquema de geração de soluções para a resolução de RCPSP. Os testes realizados nas instâncias de conjuntos `j30`, `j60` e `j120` demonstraram que esta abordagem obteve excelentes resultados.

Zhang, Xu e Peng (2008) propõem um algoritmo baseado no *Genetic Algorithm* para a resolução de RCPSP. Os autores apresentam uma nova permutação de esquema de codificação baseada em prioridade, combinando os esquemas de codificação por permutação com a codificação baseada em prioridade. Os testes foram realizados com instâncias disponíveis na PSPLIB e demonstram, quando comparados com outros algoritmos, que esse método é efetivo para a resolução do RCPSP.

Cravo (2009) apresenta uma implementação da meta-heurística *Greedy Randomized Adaptive Search Procedure* para a resolução do *Multi-Mode Resource-Constrained Project Scheduling Problem*, uma extensão do RCPSP. Os resultados obtidos para a solução das instâncias apresentadas na PSPLIB demonstram que o *Greedy Randomized Adaptive Search Procedure* consegue ser um método competitivo e, em um estudo de caso específico, apresentou bom desempenho quando comparado com resultados da literatura.

Muller (2009) apresenta um algoritmo de *Adaptive Large Neighborhood Search*, unificando diversas técnicas dentro deste *framework*, para a resolução de RCPSP. Os resultados computacionais nas instâncias de conjuntos `j30`, `j60` e `j120` demonstram a competitividade com outros algoritmos estado-da-arte.

Muller (2011) novamente apresenta um algoritmo de *Adaptive Large Neighborhood Search*, porém para a resolução de *Multi-Mode Resource-Constrained Project Scheduling Problem*. O algoritmo proposto apresenta três novos *bounds* específicos para o multimodo. Seus testes foram executados em instâncias disponibilizadas na PSPLIB. Os resultados dos testes demonstraram que esse algoritmo foi competitivo, mas incapaz de superar os melhores algoritmos.

Azevedo, Pessoa e Torres (2012) apresentam um modelo de *Genetic Algorithm* para solução de um RCPSP, representado por um estudo de caso de uma empresa de

petróleo e gás. Esse algoritmo utilizava uma representação cromossômica com decodificação por *Schedule Generation Schemes* paralela e de mutação com cruzamentos parametrizados seguidos de mutação aleatória. Como resultados, os autores concluíram a impossibilidade de se manter o escopo do projeto original, uma vez que este possuía uma expectativa contratual de um ano, enquanto sua expectativa real girava em torno de dois anos.

Crawford et al. (2015) propõe uma abordagem baseada no algoritmo *Artificial Bee Colony* para a resolução do RCPSP. Os autores apresentam o modelo matemático e descrevem o comportamento do *Artificial Bee Colony*, acreditando ser possível que este algoritmo seja efetivo em situações competitivas.

2.2 Trabalhos relacionados ao CS

Chaves e Lorena (2010) apresentam um algoritmo para resolver o *Capacitated Centered Clustering Problem* utilizando a meta-heurística CS com um algoritmo SA para a geração de soluções. Os autores compararam os testes com um algoritmo SA sem o CS e com um *Variable Neighborhood Search* proposto na literatura. O CS foi melhor que ambos em quase todas as instâncias, produzindo diversas novas melhores soluções.

Oliveira, Mauri e Lorena (2012) apresentam um algoritmo para a resolução do *Berth Allocation Problem* utilizando CS com o uso de SA para a geração de soluções. Os autores demonstram que o CS é uma alternativa útil para encontrar boas soluções através de testes comparativos com outros métodos propostos na literatura. Quando comparado com outras abordagens recentes na literatura, os resultados do algoritmo foram superiores.

Nagano, Silva e Lorena (2012) aplicam uma meta-heurística híbrida *Genetic Algorithm*-CS em um algoritmo chamado de *Evolutionary Clustering Search* para solucionar o *no-wait flow shop problem with set-up times*. Os autores, através dos testes experimentais, chegam as conclusões de que o CS é superior ao melhor algoritmo conhecido na literatura (ILS), que o *Evolutionary Clustering Search* obteve uma performance melhor que o melhor método para instâncias de larga escala, e que novas melhores soluções foram obtidas pelo CS, totalizando 1441 instâncias.

Rabello et al. (2014) apresentam um algoritmo para a resolução do *Point-Feature Car-*

tographic Label Placement utilizando CS com SA para gerar soluções. Os experimentos computacionais realizados em instâncias de até 750 pontos resultaram em soluções ótimas e os testes realizados em instâncias de até 13.206 pontos obtiveram resultados melhores que outros encontrados na literatura.

Altoé et al. (2018) apresentam algoritmos para a resolução do *Flexible Job Shop Scheduling Problem* e para o *Bi-Objective Flexible Job Shop Scheduling Problem*. Esses problemas são extensões do *Job Shop Scheduling Problem*. Os autores propõem uma solução baseada na meta-heurística híbrida CS, usando SA como método gerador de soluções. Os resultados mostram que o algoritmo foi capaz de encontrar a melhor solução para 6 dos 10 problemas clássicos considerados para o *Flexible Job Shop Scheduling Problem*, e para o *Bi-Objective Flexible Job Shop Scheduling Problem* o algoritmo foi capaz de encontrar 94 soluções não-dominadas em um conjunto de 20 instâncias propostas.

González et al. (2019) propõem duas abordagens para resolver *Traffic Counting Location Problem*, um algoritmo *Branch-and-Cut* e uma heurística CS. O algoritmo *Branch-and-Cut* foi capaz de obter resultados ótimos dentro de 7200 segundos, além de ser capaz de retornar soluções de alta qualidade nos casos onde não se obteve o resultado ótimo. O CS implementado foi comparado com um *Genetic Algorithm* estado-da-arte e, após os testes, se demonstrou superior em todos os aspectos. A comparação do *Branch-and-Cut* com o CS demonstrou que o segundo foi capaz de encontrar soluções melhores em menos tempo para 11 instâncias, demonstrando que o mesmo tinha redefinido o estado-da-arte para este problema.

Abreu et al. (2020) apresentam um modelo matemático e propõem um algoritmo para o *Network Sensor Location Problem* usando um algoritmo *Branch-and-Cut* e a heurística CS para encontrar soluções para o problema. Para avaliar a eficácia do algoritmo, testes computacionais foram realizados em instâncias reais correspondendo a estados brasileiros. Os resultados do *Branch-and-Cut* demonstraram que as soluções viáveis foram encontradas em 22,22% dos casos e boas soluções foram encontradas para 47,01%, enquanto os resultados do CS demonstraram que soluções ótimas foram encontradas para 18,38% dos casos e boas soluções em 92,31%. Quando comparados, o CS encontrou soluções melhores que o *Branch-and-Cut* para 82,48% dos cenários.

2.3 O estado da arte do RCPSP

De acordo com Kolisch e Hartmann (1999), até 1999 as estratégias mais adotadas na resolução do RCPSP consistiam em SA, *Tabu Search* e *Genetic Algorithm*. Porém, Brucker et al. (1999), ainda em 1999, demonstram em suas pesquisas que o foco das pesquisas se deslocou para heurísticas mais elaboradas, como *Truncated Branch-and-Bound*, heurísticas baseadas em *integer programming*, conceitos de *disjunctive arc*, análise local baseada em constantes, técnicas de amostragem e técnicas de busca local.

Porém, de acordo com Pellerin, Perrier e Berthaut (2020), pode-se observar uma mudança no cenário dos principais métodos usados baseando-se no fato de que os melhores métodos exatos são capazes de resolver instâncias de até 60 atividades. Desta maneira, o cenário das estratégias mais recentes se vê dominado por meta-heurísticas, híbridas ou não.

Também de acordo com Pellerin, Perrier e Berthaut (2020), meta-heurísticas se provaram excelentes para resolver o RCPSP. SA, *Tabu Search*, *Artificial Immune*, *Bee Colony*, *Genetic Algorithm*, *Particle Swarm*, *Scatter Search* e *Ant Colony*, estão, dentre outras, mais comumente listadas em pesquisas recentes. Ainda segundo os autores, em anos recentes, um número crescente de meta-heurísticas híbridas vêm sendo usadas para a resolução de problemas de otimização. Os autores identificaram ao menos 27 tipos de meta-heurísticas híbridas que foram desenvolvidas para a resolução do RCPSP. Estes resultados são exibidos na Tabela 9.

Dentre as abordagens híbridas recentes, destaca-se o estudo de Bissoli e Mauri (2024), que propõe o uso da meta-heurística CS em conjunto com o SA para resolver o RCPSP. O método combina a técnica de busca em *clusters* com procedimentos de busca local, demonstrando eficácia na obtenção de soluções de alta qualidade, frequentemente próximas das melhores conhecidas na literatura. Os autores observaram baixo desvio entre execuções e tempos de processamento reduzidos, reforçando o potencial da CS como uma alternativa prática para cenários reais.

A capacidade da CS de obter soluções robustas a posiciona como uma estratégia promissora para problemas de otimização complexos, consolidando a tendência de uso de meta-heurísticas híbridas na resolução do RCPSP. Essa eficácia, somada à ausência de estudos que integram CS com ILS, justifica a escolha dessa abordagem para aplicação neste contexto.

Tabela 9: Principais estratégias híbridas para resolução do RCPSP.

Classe de Meta-Heurística	**Tipo de Meta-Heurística**
Meta-Heurística de Busca Local	*Adaptive Large Neighborhood Search* (ALNS) *Filter-and-Fan* (FAF) *Greedy Randomized Adaptive Search Procedure* (GRASP) *Large Neighborhood Search* (LNS) *Local Search* (LS) *Nested Partitions* (NP) *Simulated Annealing* (SA) *Tabu Search* (TS)
Baseada em População	*Ant Colony Optimization* (ACO) *Artificial Immune* (AI) *Bee Colony Optimization* (BCO) *Differential Evolution Algorithm* (DEA) *Electromagnetism* (EM) *Estimation of Distribution* (EOD) *Evolutionary Algorithm* (EA) *Evolutionary Programming* (EP) *Firefly Algorithm* (FA) *Generic Algorithm* (GA) *Memetic Algorithm* (MA) *Particle Swarm Optimization* (PSO) *Population Learning* (PL) *Shuffle Frog-Leaping* (SFL) *Scatter Search* (SS) *Termite Colony Optimization* (TCO)
Meta-heurística de Aprendizado	*Neural Network* (NN)

Fonte: Pellerin, Perrier e Berthaut (2020).

3 METODOLOGIA

A metodologia deste trabalho se baseia na utilização da meta-heurística híbrida CS para a resolução do RCPSP. A meta-heurística ILS é utilizada como meta-heurística geradora de soluções para o CS.

Apresenta-se a seguir o detalhamento dos recursos utilizados para implementação da meta-heurística CS proposta neste trabalho para solucionar o RCPSP.

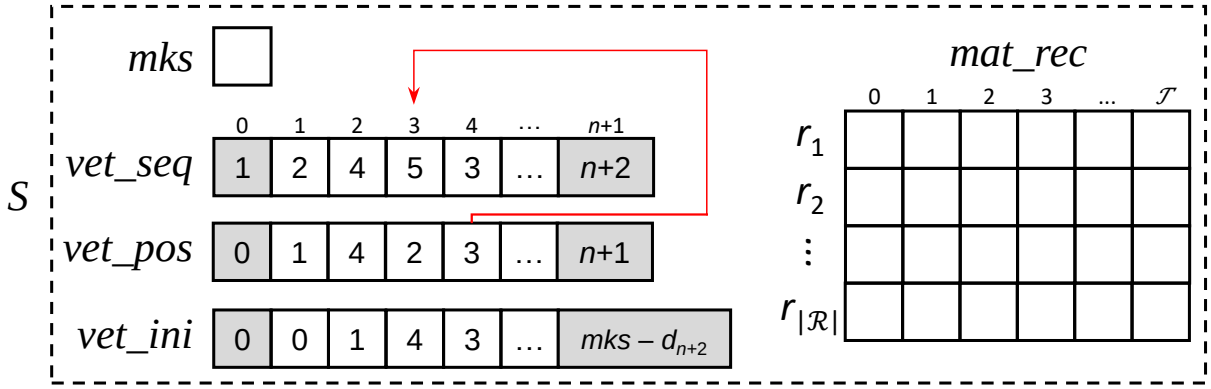
3.1 Modelagem do RCPSP

3.1.1 Estruturas de dados para representação da solução

Para a representação de uma solução S do RCPSP, são utilizadas as estruturas de dados descritas por Bissoli e Mauri (2024), conforme demonstrado na Figura 4. A solução é armazenada em vetores de dimensão $n + 2$, onde n representa o número total de atividades, e uma matriz de dimensão $|R| \times T$, onde $|R|$ é o número de recursos disponíveis e T é o horizonte de planejamento, ou seja, o número total de períodos considerados.

Além disso, a solução inclui duas tarefas fictícias, sendo a tarefa 1 a tarefa inicial e a tarefa $n + 2$ a tarefa final, que são utilizadas para facilitar a modelagem do problema. Para calcular o tempo total necessário para concluir todas as atividades, uma variável chamada *mks* armazena o valor do *makespan* da solução.

Figura 4: Estruturas de dados para representar uma solução.



Fonte: Bissoli e Mauri (2024).

- **vet_seq**: armazena a sequência das atividades.
- **vet_pos**: contém a posição de cada atividade em **vet_seq**.
- **vet_ini**: registra o instante de início de cada atividade.
- **mat_rec**: registra o consumo de recursos ao longo do tempo.

Essas estruturas são fundamentais para definir e manipular a sequência e o tempo de início das atividades, mantendo a viabilidade da solução em relação às restrições de precedência e disponibilidade de recursos.

3.1.2 Solução inicial

A solução inicial é gerada conforme o Algoritmo 1 de Bissoli e Mauri (2024), que é caracterizado como um algoritmo construtivo guloso. O algoritmo organiza as atividades de acordo com as relações de precedência, em sequência, calculando os instantes mínimos de início com base na duração das predecessoras. A sequência inicial é então registrada em **vet_seq** e **vet_pos**.

1. A sequência de atividades é determinada considerando as precedências.
2. Os instantes de início são atualizados conforme o Algoritmo 3, garantindo o uso dos recursos disponíveis e a não violação de precedências.

Algoritmo 1: CRIAR_SOLUCAO( )

```
1  S.vet_seq ← Ordenar(𝒥, a_j);
2  S.vet_pos[0] ← 0;
3  S.vet_ini[0] ← 0;
4  para (k ← 1; k < n + 2; k + 1) faça
5      i ← S.vet_seq[k − 1];
6      j ← S.vet_seq[k];
7      S.vet_pos[j − 1] ← k;
8      S.vet_ini[j − 1] ← S.vet_ini[i − 1] + d_i;
9  fim
10 S ← DEFINIR_INICIOS(S, 1);
11 retorna (S);
```

Algoritmo 2: GERAR_VIZINHA(S)

```
1  S′ ← S;
2  repita
3      j ← Aleatório(j ∈ 𝒥 | j ≠ 1 e j ≠ n + 2);
4      k ← max {S′.vet_pos[i − 1] ∀ i | (i, j) ∈ 𝒫};
5      w ← min {S′.vet_pos[i − 1] ∀ i | (j, i) ∈ 𝒫};
6      p_ds ← Aleatório[k + 1, w − 1];
7  até (p_ds ≠ S′.vet_pos[j − 1]);
8  p_or ← S′.vet_pos[j − 1];
9  se (p_ds < p_or) então
10     para (k ← p_or; k > p_ds; k − 1) faça
11         S′.vet_seq[k] ← S′.vet_seq[k − 1];
12         S′.vet_pos[S′.vet_seq[k]] ← k;
13     fim
14 senão
15     para (k ← p_or; k < p_ds; k + 1) faça
16         S′.vet_seq[k] ← S′.vet_seq[k + 1];
17         S′.vet_pos[S′.vet_seq[k]] ← k;
18     fim
19 fim
20 S′.vet_seq[p_ds] ← j;
21 S′.vet_pos[j − 1] ← p_ds;
22 S′ ← DEFINIR_INICIOS(S′, min {p_or, p_ds});
23 retorna (S′);
```

Algoritmo 3: DEFINIR_INICIOS(S, pos)

```
1  Zerar(S.mat_req);
2  S.mks ← 0;
3  para (k ← 1; k < pos; k + 1) faça
4      j ← S.vet_seq[k];
5      w ← S.vet_ini[j − 1];
6      para (r ∈ ℛ) faça
7          para (t ← w; t < w + d_j; t + 1) faça
8              S.mat_req[r][t] ← S.mat_req[r][t] + q_rj;
9          fim
10     fim
11     S.mks ← max {S.mks, w + d_j};
12 fim
13 para (k ← pos; k < n + 1; k + 1) faça
14     j ← S.vet_seq[k];
15     w ← max {S.vet_ini[i − 1] + d_i ∀ i | (i, j) ∈ 𝒫};
16     enquanto (verdadeiro) faça
17         para (r ∈ ℛ) faça
18             para (t ← w; t < w + d_j; t + 1) faça
19                 se (S.mat_req[r][t] + q_rj > c_r)
                   então
20                     w ← w + 1;
21                     ir para linha 16;
22                 fim
23             fim
24         fim
25         ir para linha 27;
26     fim
27     S.vet_ini[j − 1] ← w;
28     para (r ∈ ℛ) faça
29         para (t ← w; t < w + d_j; t + 1) faça
30             S.mat_req[r][t] ← S.mat_req[r][t] + q_rj;
31         fim
32     fim
33     S.mks ← max {S.mks, w + d_j};
34 fim
35 S.vet_ini[n + 1] ← S.mks;
36 S.mks ← S.mks + d_{n+2};
37 retorna (S);
```

3.1.3 Estrutura de vizinhança

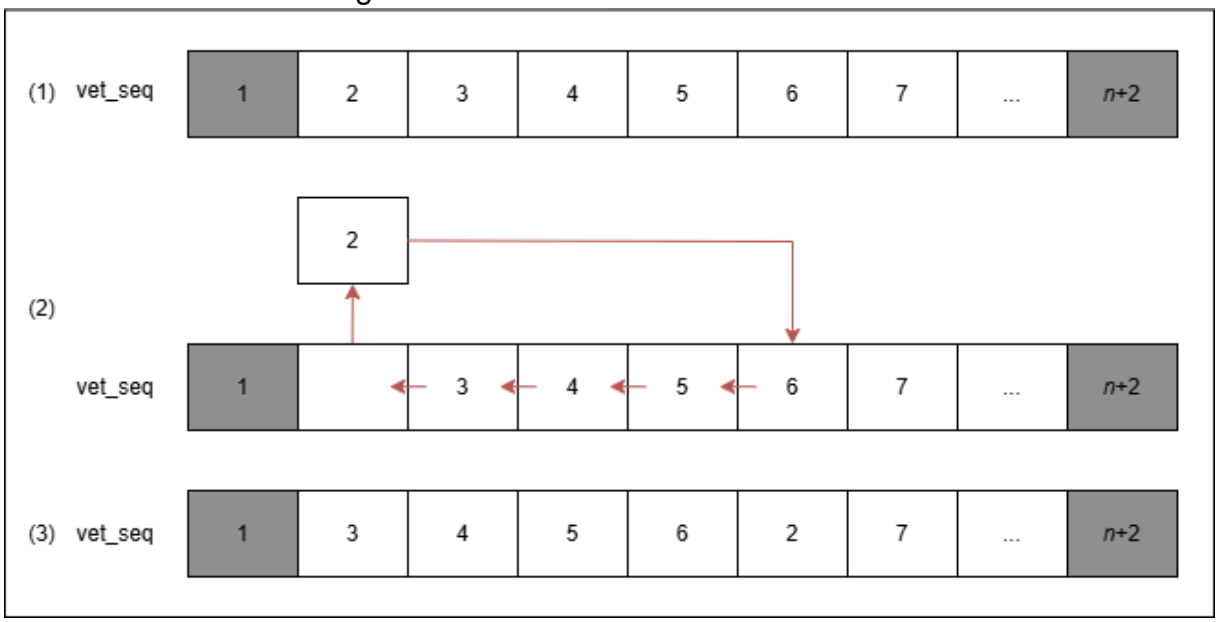
A estrutura de vizinhança segue a abordagem do Algoritmo 2 de Bissoli e Mauri (2024), que descreve a maneira como são realizadas as perturbações na solução. Uma atividade é escolhida aleatoriamente, e sua posição é ajustada entre sua última predecessora e sua primeira sucessora, garantindo que a sequência de atividades permaneça viável.

- **Deslocamento das atividades**: Uma atividade e uma posição alvo são selecionadas aleatoriamente. A atividade é removida do vetor, e as atividades entre a posição da atividade removida e a posição alvo são deslocadas, deixando a posição alvo vaga. Em seguida, a atividade selecionada é reinserida na posição alvo agora desocupada. Esse deslocamento é ilustrado na Figura 5.

- **Atualização dos instantes de início**: Após a movimentação, o Algoritmo 3 re-

calcula os instantes de início das atividades, assegurando o uso otimizado dos recursos disponíveis.

Figura 5: Deslocamento das atividades.



Fonte: O autor.

3.2 *Iterated Local Search*

A essência do *Iterated Local Search* pode ser resumida da seguinte forma: ela constrói iterativamente uma sequência de soluções geradas por uma heurística embutida, resultando em soluções significativamente melhores do que aquelas obtidas por tentativas aleatórias repetidas da mesma heurística (LOURENÇO; MARTIN; STÜTZLE, 2010).

Dois pontos principais definem um algoritmo como um ILS: (i) deve existir uma única cadeia de soluções sendo seguida, o que exclui algoritmos baseados em populações; (ii) a busca por melhores soluções ocorre em um espaço reduzido, definido pela saída de uma heurística de busca local, considerada como uma "caixa preta"(LOURENÇO; MARTIN; STÜTZLE, 2010).

3.2.1 Solução Inicial

Escolhas padrão para a solução inicial incluem uma solução aleatória ou uma solução gerada por uma heurística de construção gulosa. Soluções iniciais gulosas apresentam duas vantagens principais em relação às soluções iniciais aleatórias: (i) quando combinadas com busca local, soluções gulosas frequentemente resultam em soluções de melhor qualidade; (ii) uma busca local a partir de soluções gulosas requer, em média, menos passos de melhoria, o que reduz o tempo de processamento necessário para a busca local (LOURENÇO; MARTIN; STÜTZLE, 2010).

3.2.2 Perturbação

O principal problema da melhoria iterativa é que ela frequentemente fica presa em ótimos locais significativamente inferiores ao ótimo global. Assim como no SA, o ILS escapa desses ótimos locais aplicando perturbações ao mínimo local atual (LOURENÇO; MARTIN; STÜTZLE, 2010).

De maneira geral, a busca local não deve ser capaz de reverter a perturbação; caso contrário, o algoritmo retornará ao ótimo local já visitado. Um movimento aleatório em um vizinhança de ordem superior à utilizada pelo algoritmo de busca local frequentemente atinge esse objetivo e resulta em um algoritmo satisfatório. Resultados ainda melhores podem ser alcançados se as perturbações considerarem as propriedades do problema e forem bem ajustadas ao algoritmo de busca local (LOURENÇO; MARTIN; STÜTZLE, 2010).

3.2.3 Critério de Aceitação

O ILS realiza uma caminhada aleatória no espaço dos mínimos locais. O mecanismo de perturbação, combinado com a busca local, define as transições possíveis entre a solução atual e uma solução “vizinha”. O critério de aceitação determina então se essa solução será aceita ou não como a nova solução atual (LOURENÇO; MARTIN; STÜTZLE, 2010).

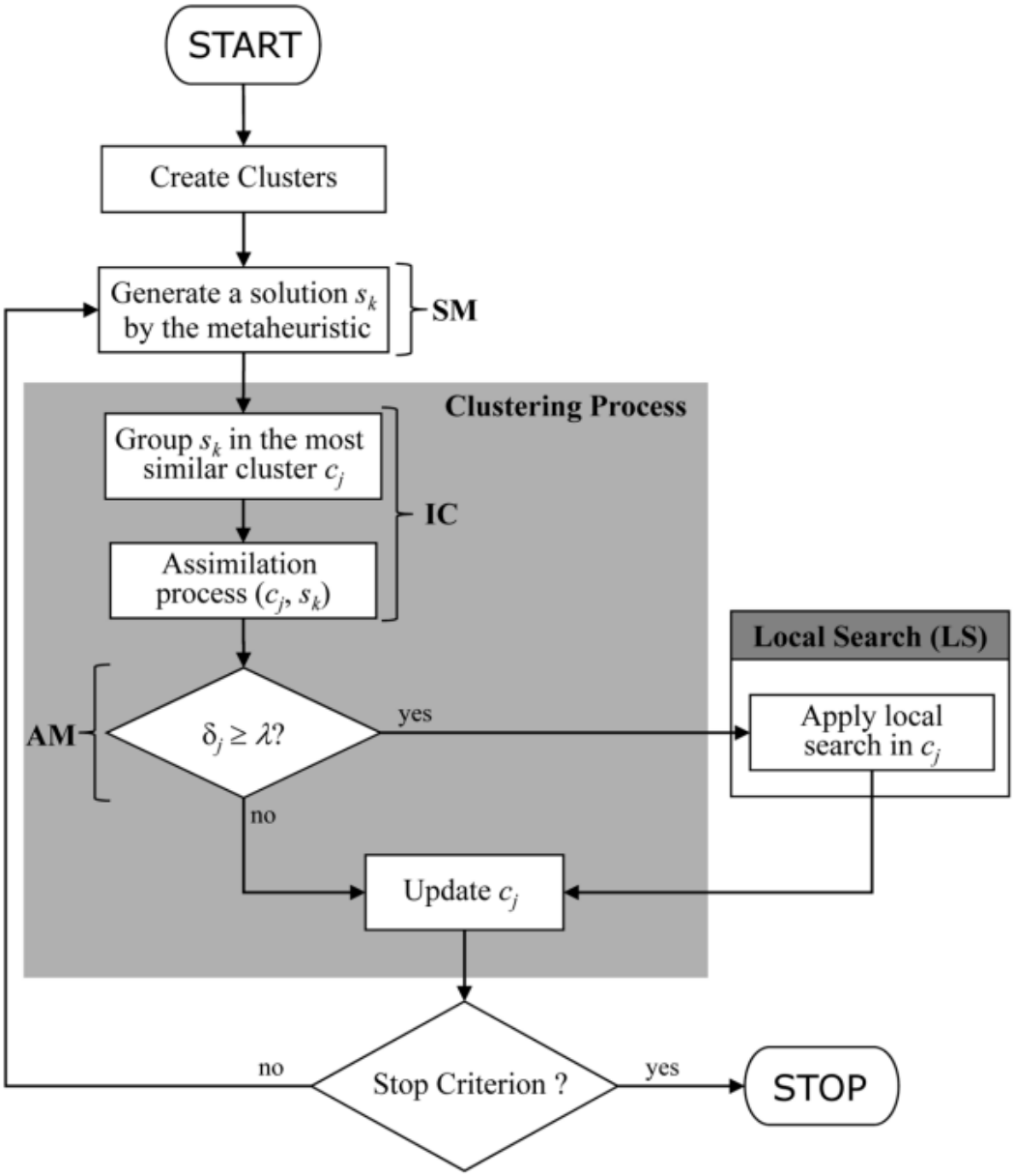
3.3 *Clustering Search*

Segundo Oliveira, Chaves e Lorena (2013), a meta-heurística CS implementa *clustering* para detectar áreas promissoras no espaço de busca. Um *cluster* é definido por um centro que é, geralmente, inicializado de maneira aleatória e, posteriormente, tende a progredir para pontos promissores no espaço de busca. O número de *clusters* pode ser fixo a priori ou determinado dinamicamente de acordo com amplitude da busca nas áreas que serão exploradas pela meta-heurística. Nos casos em que é determinado dinamicamente, os *clusters* podem ser criados de uma maneira que todas as soluções candidatas são cobertas por pelo menos um *cluster*.

Ainda segundo Oliveira, Chaves e Lorena (2013), a sua cobertura é definida por uma distância métrica que computa a similaridade entre uma solução e o centro do *cluster*, e deve considerar a natureza do problema. Um *cluster* também possui densidade, chamada de volume, que é uma indicação do nível de atividade dentro do respectivo *cluster*.

O CS pode ser dividido conceitualmente em quatro partes independentes: a Meta-Heurística de Busca (*Search Metaheuristic* - SM), o Componente de Agrupamento Iterativo (*Iterative Clustering Component* - IC), o Módulo Analisador (*Analyser Module* - AM), e a Heurística de Busca Local (*Local Search Heuristic* - LS), como ilustrado na Figura 6. Tais partes são descritas a seguir (OLIVEIRA; CHAVES; LORENA, 2013).

Figura 6: Os componentes do CS.



Fonte: Oliveira, Chaves e Lorena (2013).

3.3.1 Meta-Heurística de Busca

O componente SM pode ser implementado por qualquer algoritmo de otimização que consiga gerar soluções diversificadas no espaço de busca. Neste trabalho, optou-se pelo uso do ILS, visando compará-lo ao estudo de Bissoli e Mauri (2024), que utiliza o SA. O SM deve atuar como um gerador completo de soluções, explorando o espaço por meio da manipulação de soluções de acordo com suas estratégias específicas de

busca (OLIVEIRA; CHAVES; LORENA, 2013).

Segundo Lourenço, Martin e Stützle (2010), o ILS possui muitas das características desejáveis em uma meta-heurística: é simples, fácil de implementar, robusto e altamente eficaz. A ideia essencial do ILS está em focar a busca não no espaço completo de soluções, mas em um subespaço menor definido pelas soluções localmente ótimas para um determinado mecanismo de otimização. Essas características, aliadas à facilidade de implementação e ao potencial para explorar subespaços menores, como os *clusters* gerados pelo CS, justificam sua escolha como meta-heurística de busca.

O pseudocódigo do CS, adaptado para utilizar o ILS como meta-heurística geradora de soluções, é apresentado no Algoritmo 4. Nele, os seguintes parâmetros são empregados:

γ**:** número máximo de *clusters*;

λ**:** volume máximo dos *clusters*;

r_{max}**:** limite do índice de ineficácia dos *clusters*;

$Time_{max}$**:** limite de tempo de execução da CS.

Algoritmo 4: $\mathrm{CS} + \mathrm{ILS}(\gamma, \lambda, r_{max}, Time_{max}, S_0)$

```
1  clt ← 1; r_i ← 0; v_i ← 0(1 ≤ i ≤ γ); S ← S_0;
2  S ← PRIMEIRA_MELHORA(S);
3  enquanto Time < Time_max faça
4      S' ← S;
5      r ← Aleatrio(3, .., 0.4 ∗ n);
       /* O valor de r será escolhido aleatoriamente entre 3 e 40% do número total de tarefas */
6      para i ← 0 até r faça
7          S' ← GERAR_VIZINHA(S');
8      fim
9      S' ← PRIMEIRA_MELHORA(S');
10     se S'.mks < S.mks então
11         S ← S';
12     fim
13     se clt < γ então
14         clt ← clt + 1; v_clt ← v_clt + 1; c_clt ← S';
15     senão
16         i ← min_{i∈{1,...,γ}} {H_i(S')}; v_i ← v_i + 1;
17         c_i ← Melhor(S', c_i);
18         se v_i = λ então
19             v_i ← 1;
20             se r_i = r_max então
21                 r_i ← 0; c_i ← GERAR_VIZINHA(c_i);
22             senão
23                 c_i ← PRIMEIRA_MELHORA(c_i);
24                 se c_i foi melhorado pela busca local então
25                     r_i ← 0;
26                 senão
27                     r_i ← r_i + 1;
28                 fim
29             fim
30             S ← Melhor(S, c_i);
31         fim
32     fim
33 fim
34 retorna S
```

3.3.2 Componente de Agrupamento Iterativo

Este componente possui a finalidade de agrupar soluções similares em grupos, mantendo um centro de *cluster* representativo. Uma distância métrica deve ser definida, a priori, permitindo se mensurar a similaridade para o processo de *clustering* (OLIVEIRA; CHAVES; LORENA, 2013).

A semelhança entre duas soluções é medida pela distância de Hamming (H) (HAMMING, 1950), conforme definida por Bissoli e Mauri (2024). Essa distância corresponde ao número de posições nas quais as entradas correspondentes diferem. No

contexto do RCPSP, foi considerado o número de diferenças no vetor **vet_seq** das soluções comparadas (linha 16 do Algoritmo 4).

3.3.3 Módulo Analisador

O componente AM examina cada *cluster*, em intervalos regulares, indicando um provável *cluster* promissor. Ele conta o número de soluções geradas pelo SM e agrupadas dentro do *cluster*, e, sempre que o volume deste *cluster* alcança um limiar λ, o mesmo precisará ser melhor investigado (OLIVEIRA; CHAVES; LORENA, 2013).

3.3.4 Heurística de Busca Local

Por último, o componente LS é um módulo de busca interna que provê a exploração de uma suposta área promissora, enquadrada em um *cluster* (OLIVEIRA; CHAVES; LORENA, 2013). As estratégias de busca local utilizadas são descritas as seguir:

O algoritmo de Primeira Melhora (Algoritmo 5) é uma abordagem de busca local que explora as soluções vizinhas da solução atual em uma ordem qualquer, aceitando a primeira solução encontrada que melhora a qualidade da solução atual. Este método é eficiente, pois evita explorar desnecessariamente todas as vizinhanças quando uma melhora é identificada rapidamente. Contudo, a qualidade da solução final pode depender da ordem em que as vizinhanças são exploradas.

Algoritmo 5: PRIMEIRA_MELHORA(S)

```
1  lista ← 𝒥 − {1, n + 2};
2  l ← 0;
3  enquanto (l < n) faça
4      pos ← l+ Aleatório(n − l);
5      j ← S.vet_seq[pos];
6      k ← max {S.vet_pos[i − 1] ∀ i | (i, j) ∈ 𝒫};
7      w ← min {S.vet_pos[i − 1] ∀ i | (j, i) ∈ 𝒫};
8      S′ ← S;
9      para q ← k até w; q ≠ S.vet_pos[j − 1] faça
10         S′ ← S;
11         se (q < S′.vet_pos[j − 1]) então
12             para (o ← S′.vet_pos[j − 1]; o > q; o − 1) faça
13                 S′.vet_seq[o] ← S′.vet_seq[o − 1];
14                 S′.vet_pos[S′.vet_seq[o]] ← o;
15             fim
16         senão
17             para (o ← S′.vet_pos[j − 1]; o < q; o + 1) faça
18                 S′.vet_seq[o] ← S′.vet_seq[o + 1];
19                 S′.vet_pos[S′.vet_seq[o]] ← o;
20             fim
21         fim
22         S′.vet_seq[q] ← j;
23         S′.vet_pos[j − 1] ← q;
24         S′ ← DEFINIR_INICIOS(S′, min {S′.vet_pos[j − 1], q});
25         se S′.mks < S.mks então
26             S ← S′;
27             ir para linha 2;
28         fim
29     fim
30     aux ← lista[l];
31     lista[l] ← lista[pos];
32     lista[pos] ← aux;
33     l ← l + 1;
34 fim
35 retorna (S);
```

O algoritmo de Melhor Melhora (Algoritmo 6) examina todas as soluções vizinhas da solução atual e seleciona a melhor delas, ou seja, aquela que proporciona a maior melhora na função objetivo. Embora tenda a encontrar soluções de maior qualidade em comparação à Primeira Melhora, esse método pode ser mais custoso em termos computacionais, especialmente quando o número de vizinhos é grande, devido à necessidade de avaliar todas as opções antes de decidir.

Algoritmo 6: MELHOR_MELHORA(S)

```
1  enquanto verdadeiro faça
2      best_mks ← S.mks;
3      para l ← 1 até n − 1 faça
4          pos ← i;
5          j ← S.vet_seq[pos];
6          k ← max {S.vet_pos[i − 1] ∀ i | (i, j) ∈ P};
7          w ← min {S.vet_pos[i − 1] ∀ i | (j, i) ∈ P};
8          S′ ← S;
9          para q ← k até w; q ≠ S.vet_pos[j − 1] faça
10             S′ ← S;
11             se (q < S′.vet_pos[j − 1]) então
12                 para (o ← S′.vet_pos[j − 1]; o > q; o − 1) faça
13                     S′.vet_seq[o] ← S′.vet_seq[o − 1];
14                     S′.vet_pos[S′.vet_seq[o]] ← o;
15                 fim
16             senão
17                 para (o ← S′.vet_pos[j − 1]; o < q; o + 1) faça
18                     S′.vet_seq[o] ← S′.vet_seq[o + 1];
19                     S′.vet_pos[S′.vet_seq[o]] ← o;
20                 fim
21             fim
22             S′.vet_seq[q] ← j;
23             S′.vet_pos[j − 1] ← q;
24             S′ ← DEFINIR_INICIOS(S′, min {S′.vet_pos[j − 1], q});
25             se S′.mks < best_mks então
26                 best_pos ← q;
27                 best_job ← j;
28                 best_mks ← S′.mks;
29             fim
30         fim
31     fim
32     se best_mks ≠ S.mks então
33         se (best_pos < S.vet_pos[best_job − 1]) então
34             para (o ← S.vet_pos[best_job − 1]; o > best_pos; o − 1) faça
35                 S.vet_seq[o] ← S.vet_seq[o − 1];
36                 S.vet_pos[S.vet_seq[o]] ← o;
37             fim
38         senão
39             para (o ← S.vet_pos[best_job − 1]; o < best_pos; o + 1) faça
40                 S.vet_seq[o] ← S′.vet_seq[o + 1];
41                 S.vet_pos[S.vet_seq[o]] ← o;
42             fim
43         fim
44         S.vet_seq[best_pos] ← best_job;
45         S.vet_pos[best_job − 1] ← best_pos;
46         S ← DEFINIR_INICIOS(S, min {S.vet_pos[best_job − 1], best_pos});
47     senão
48         interromper;
49     fim
50 fim
51 retorna (S);
```

A Busca Aleatória (Algoritmo 7) explora o espaço de soluções de maneira não sistemática, selecionando vizinhos de forma completamente aleatória, sem considerar previamente a qualidade das soluções geradas. Embora este método tenha baixo custo computacional e possa ser útil para escapar de ótimos locais em problemas complexos, ele geralmente apresenta um desempenho inferior em termos de convergência para soluções de alta qualidade, devido à ausência de um critério estruturado para guiar a busca.

Algoritmo 7: $\mathrm{BUSCA_ALEATORIA}(S, n)$

```
1  enquanto verdadeiro faça
2      flag ← 1;
3      para i ← 0 até n − 1 faça
4          S' ← S;
5          GERAR_VIZINHO(S');
6          se S'.mks < S.mks então
7              S ← S';
8              flag ← 0;
9          fim
10     fim
11     se flag = 1 então
12         interromper;
13     fim
14 fim
15 retorna S;
```

Essas estratégias foram implementadas em um algoritmo de ILS a fim de se realizar testes comparativos de efetividade. Os Algoritmos 8, 9, 10 e 11, apresentados a seguir, demonstram essas implementações:

Algoritmo 8: $\mathrm{F}(Time_{max}, S)$

```
1  S' ← S;
2  S' ← PRIMEIRA_MELHORA(S');
3  Time ← 0;
4  enquanto Time ≤ Time_max faça
5      S'' ← S';
6      S'' ← GERAR_VIZINHA(S'');
7      S'' ← PRIMEIRA_MELHORA(S'');
8      se S''.mks < S'.mks então
9          S' ← S'';
10         se S'.mks < S.mks então
11             S ← S';
12         fim
13     fim
14 fim
15 retorna (S);
```

Algoritmo 9: $\mathrm{F} + \mathrm{B}(Time_{max}, S)$

```
1  S' ← S;
2  S' ← PRIMEIRA_MELHORA(S');
3  Time ← 0;
4  enquanto Time ≤ Time_max faça
5      S'' ← S';
6      S'' ← GERAR_VIZINHA(S'');
7      r ← Aleatrio(1, ..., 2);
8      se r = 1 então
9          S'' ← PRIMEIRA_MELHORA(S'');
10     senão
11         S'' ← MELHOR_MELHORA(S'');
12     fim
13     se S''.mks < S'.mks então
14         S' ← S'';
15         se S'.mks < S.mks então
16             S ← S';
17         fim
18     fim
19 fim
20 retorna (S);
```

Algoritmo 10: $\mathrm{F} + \mathrm{B} + \mathrm{R}(Time_{max}, S)$

```
1  S' ← S;
2  S' ← PRIMEIRA_MELHORA(S');
3  Time ← 0;
4  enquanto Time ≤ Time_max faça
5      S'' ← S';
6      S'' ← GERAR_VIZINHA(S'');
7      r ← Aleatrio(1, ..., 3);
8      se r = 1 então
9          S'' ← PRIMEIRA_MELHORA(S'');
10     fim
11     senão se r = 2 então
12         S'' ← MELHOR_MELHORA(S'');
13     fim
14     senão
15         S'' ← BUSCA_ALEATORIA(S'');
16     fim
17     se S''.mks < S'.mks então
18         S' ← S'';
19         se S'.mks < S.mks então
20             S ← S';
21         fim
22     fim
23 fim
24 retorna (S);
```

Algoritmo 11: $\mathrm{RP}(0, \mathrm{X}) + \mathrm{F}(Time_{max}, S)$

```
1  S' ← S;
2  S' ← PRIMEIRA_MELHORA(S');
3  Time ← 0;
4  enquanto Time ≤ Time_max faça
5      S'' ← S';
6      r ← Aleatrio(3, .., 0.X ∗ n);
7      para i ← 0 até r faça
8          S'' ← GERAR_VIZINHA(S'');
9      fim
10     S'' ← PRIMEIRA_MELHORA(S'');
11     se S''.mks < S'.mks então
12         S' ← S'';
13         se S'.mks < S.mks então
14             S ← S';
15         fim
16     fim
17 fim
18 retorna (S);
```

Por fim, a busca local utilizada na CS+ILS (linha 23 do Algoritmo 4), conforme descrito por Bissoli e Mauri (2024), foi a de primeira melhora (Algoritmo 5).

4 EXPERIMENTOS COMPUTACIONAIS

Para avaliar a qualidade das soluções obtidas pelo algoritmo proposto para o RCPSP, foram realizados experimentos computacionais utilizando as instâncias disponibilizadas na biblioteca PSPLIB, nas categorias `j30`, `j60`, `j90` e `j120`. Aspectos importantes dessa etapa incluíram a configuração do ambiente experimental e a definição dos parâmetros e instâncias utilizadas.

4.1 Configuração dos Experimentos

A configuração dos experimentos neste trabalho segue a metodologia descrita por Bissoli e Mauri (2024), utilizando a linguagem de programação C++ e o compilador Microsoft Visual Studio Community 2022 (64 bits). Assim como no trabalho original, os experimentos foram realizados em um computador com processador Intel i7 3.2GHz, 64GB de memória RAM e sistema operacional Windows 10, replicando a infraestrutura utilizada para permitir comparações diretas entre os resultados.

Para avaliar o desempenho da CS+ILS, foram selecionadas instâncias da PSPLIB. A PSPLIB oferece 2040 instâncias de problemas com diferentes números de atividades, incluindo conjuntos de 30, 60, 90 e 120 atividades ($n = \{30, 60, 90, 120\}$), sendo 600 instâncias com 120 atividades e 480 instâncias para os demais conjuntos. Inspirados pelo estudo de Bissoli e Mauri (2024), as instâncias com solução ótima não comprovadas, listadas nos arquivos $jXlb.sm$ ($X = n$) da PSPLIB, foram inicialmente priorizadas por sua complexidade elevada. Já algumas instâncias que não estavam listadas nos arquivos *jXlb.sm* foram desconsideradas, resultando em um conjunto de 28 instâncias com 60 atividades, 70 com 90 atividades e 97 com 120 atividades. Além disso, para testar o desempenho da CS em instâncias de menor complexidade, foram executadas as 480 instâncias com 30 atividades, cujas soluções ótimas já foram estabelecidas.

Cada instância foi executada cinco vezes, com um tempo limite de um minuto para

instâncias de 30 atividades e cinco minutos para as demais. A configuração dos parâmetros da CS seguiu as recomendações de Rabello et al. (2014), Bissoli, Zufferey e Amaral (2021) e Bissoli e Mauri (2024), além de ajustes empíricos sem calibração adicional, conforme a Tabela 10, que apresenta os valores de cada parâmetro.

Tabela 10: Parâmetros do CS+ILS.

Parâmetro	Valor selecionado
γ	10
λ	7
r_{max}	2

Fonte: O autor.

4.2 Busca Local

A escolha do tipo de busca local, bem como a estratégia por trás das múltiplas perturbações (geração consecutiva de vizinhos) na solução, foi baseada em uma série de testes para determinar qual abordagem seria mais eficaz. Durante os testes, foram considerados diferentes cenários de busca local, os quais foram executados aleatoriamente, com a mesma probabilidade atribuída a cada um. Os cenários de teste foram definidos da seguinte forma:

- $\boldsymbol{F}$ = Primeira Melhora (Algoritmo 8)
- $\boldsymbol{F} + \boldsymbol{B}$ = Primeira ou Melhor Melhora (Algoritmo 9)
- $\boldsymbol{F} + \boldsymbol{B} + \boldsymbol{R}$ = Primeira, Melhor Melhora ou Busca Aleatória (Algoritmo 10)

Nos outros cenários (Algoritmo 11), foi utilizada a Primeira Melhora após uma sequência de perturbações (geração de vizinhos), que ocorre um número de vezes igual a um valor aleatório entre 3 e $X \times$ número de tarefas, onde X está especificado no nome do cenário (por exemplo, $RP(0, X) + F$). Os cenários testados foram:

- $RP(0, 1) + F$
- $RP(0, 2) + F$
- $RP(0, 3) + F$

- $RP(0,35) + F$
- $RP(0,4) + F$
- $RP(0,45) + F$
- $RP(0,5) + F$

Os testes mencionados foram realizados em um código de ILS, que pode ser observado nos Algoritmos 8, 9, 10 e 11. Esses testes foram conduzidos utilizando quatro instâncias do problema: `j301_1`, `j601_1`, `j901_1` e `j1201_1`. Para cada instância, foram realizadas cinco execuções para cada caso de teste, com um tempo máximo de 1 minuto para a instância `j301_1` e 5 minutos para as outras instâncias. Os resultados dessas execuções foram utilizados para avaliar as estratégias de busca local e auxiliar na escolha das melhores abordagens, que posteriormente seriam integradas ao algoritmo de CS.

A Tabela 11 resume os resultados obtidos durante os testes do RCPSP para diferentes instâncias e métodos. Cada linha representa um cenário específico, e os valores apresentados destacam a performance do algoritmo em termos de *makespan*, quantidade de soluções avaliadas e tempos de execução. Esses cenários foram empregados para descobrir qual das três abordagens é mais eficiente. As suas colunas são organizadas da seguinte maneira:

- **INST/MTD (Instância e Método):** Esta coluna combina o nome da instância utilizada nos testes e os cenários de teste empregados.
- **MKS (*Makespan*):** Refere-se a média dos tempos totais necessários para concluir todas as atividades do projeto nas soluções encontradas.
- **SOL_BST (Melhor Solução):** Indica o número médio de soluções até encontrar a melhor, ordenado conforme os passos do algoritmo.
- **SOL_TOT (Soluções Totais):** Representa a média dos números totais de soluções geradas pelo algoritmo durante toda a execução dos testes.
- **T_BST (Tempo para Melhor Solução):** Mostra a média do tempo necessária para encontrar a melhor solução durante as execuções do algoritmo.
- **T_TOT (Tempo Total):** Apresenta o tempo total de execução do algoritmo, também em segundos.

Tabela 11: Resultados dos testes para busca local na meta-heurística de busca ILS.

INST/MTD	MKS	SOL_BST	SOL_TOT	T_BST	T_TOT
j301_1					
F	**43***	15,4	76321	0,0136	60
F+B	**43***	33,6	58581,8	0,0354	60,0002
F+B+R	**43***	22	**94639,8**	0,0162	60
RP(0,2)+F	**43***	**9,4**	63625,2	**0,0114**	60,0004
j601_1					
F	**77***	35,2	72893,8	0,1456	300,0026
F+B	**77***	36,8	64630,4	0,1694	300,0016
F+B+R	**77***	55,8	**87890,6**	0,2074	300,0004
RP(0,2)+F	**77***	**8,2**	57122,6	**0,0432**	300,002
j901_1					
F	77,4	2648,4	16638,2	49,929	300,0124
F+B	77,6	1618	15212,2	31,8758	300,017
F+B+R	76,6	7051,2	**23046**	91,6058	300,0144
RP(0,1)+F	76,8	2507,4	15609	48,1174	300,0134
RP(0,2)+F	76	**1080,2**	10897,8	**29,6654**	300,0204
RP(0,3)+F	**74,6**	4345,6	11756,6	112,6708	300,0148
RP(0,35)+F	**74,6**	3243,4	13640,2	76,2248	300,009
RP(0,4)+F	75,2	2799,4	13571,6	61,9232	300,0128
RP(0,45)+F	75	2092,6	13474,6	47,1446	300,0164
RP(0,5)+F	75,6	3714,6	13146,8	85,4228	300,0066
j1201_1					
F	112,2	2794,2	8311,8	102,4254	300,0132
F+B	113	1714	6003,4	84,3268	300,04
F+B+R	113,4	2596,4	**9031**	83,3832	300,033
RP(0,1)+F	111,2	2430	6990,2	101,3108	300,036
RP(0,2)+F	109,8	1436,6	4545	89,3332	300,0398
RP(0,3)+F	110	884,6	5647,6	45,8212	300,0246
RP(0,35)+F	109,6	1501,2	5723,2	78,2168	300,0332
RP(0,4)+F	110,4	**684,6**	5707,2	**35,2776**	300,027
RP(0,45)+F	**108,8**	1611,6	5480,8	87,5068	300,0228
RP(0,5)+F	**108,8**	1667,2	5402,6	93,1954	300,0358

**makespan* ótimo (provado).

Fonte: O autor.

De acordo com a Tabela 11, foi observado que a abordagem **F** (Primeira Melhora) se destacou, sendo a mais eficaz nas execuções realizadas. Também foi possível concluir que o valor $X = 0.4$ se destacou por apresentar uma média de boas soluções, além de exigir menos iterações para alcançar as melhores soluções. Esse cenário foi, portanto, o escolhido para ser integrado ao CS no código final, como exemplificado no Algoritmo 4.

4.3 Resultados

As tabelas a seguir apresentam os resultados comparativos entre os valores fornecidos pela PSPLIB e os obtidos pela CS+ILS, bem como os resultados de Bissoli e Mauri (2024).

Na primeira coluna da Tabela 12, estão listados os conjuntos de instâncias analisadas. Em seguida, são apresentados os valores fornecidos pela PSPLIB. Para a CS+ILS, são exibidos os valores médios dos *makespans* obtidos entre as instâncias de cada conjunto (AVG). Além disso, são incluídos o melhor *makespan* encontrado (BST) e o desvio percentual (DEV), calculado como $(\text{AVG} - \text{BST})/\text{BST} \times 100$.

De forma semelhante a outros trabalhos recentes da literatura (PELLERIN; PERRIER; BERTHAUT, 2020; BISSOLI; MAURI, 2024), foram calculados os desvios médios das melhores soluções obtidas pela CS em relação aos melhores limitantes inferiores conhecidos (*DevLB*). Adicionalmente, foram calculados os desvios médios em relação aos limitantes superiores (*DevUB*), ou seja, os menores valores de *makespan* conhecidos.

Esses desvios foram obtidos de acordo com as expressões apresentadas a seguir. Nessas equações, calcula-se inicialmente o desvio para cada instância e, posteriormente, é realizada a média em relação ao número total de instâncias em cada conjunto.

$$\text{DevLB (\%)} = \text{Média}\left(\frac{\text{BST} - \text{LB}}{\text{LB}}\right) \times 100 \tag{6}$$

$$\text{DevUB (\%)} = \text{Média}\left(\frac{\text{BST} - \text{UB}}{\text{UB}}\right) \times 100 \tag{6}$$

Tabela 12: Resultados comparativos entre PSPLIB e CS+ILS.

INST	PSPLIB		CS+ILS					CS+ILS × PSPLIB	
SET	UB	LB	BST	AVG	DEV (%)	#Soluções	Tempo (seg)	DevLB (%)	DevUB (%)
j30	58,99*	58,99*	58,99	59,00	0,01	120311,61	0,35	0	0
j60	109,04	102,46	110,50	111,48	0,88	13751481,24	73,64	7,86	1
j90	125,51	118,29	129,80	131,07	0,99	14884675,22	101,19	9,67	3,45
j120	131,25	124,69	138,42	139,97	1,11	14597603,56	134,97	10,55	5,45

*média do *makespan* ótimo (provado).

Fonte: O autor.

Os resultados do código de Bissoli e Mauri (2024) seguem o mesmo formato, dispostos na Tabela 13, com valores médios para o *makespan*, o *makespan* médio e o desvio percentual (DEV). Além disso, são apresentados a quantidade total de soluções geradas e o tempo médio de execução.

Tabela 13: Resultados comparativos entre PSPLIB e Bissoli e Mauri (2024).

INST	PSPLIB		Bissoli e Mauri					Bissoli e Mauri × PSPLIB	
SET	UB	LB	BST	AVG	DEV (%)	#Soluções	Tempo (seg)	DevLB (%)	DevUB (%)
j30	58,99*	58,99*	58,99	58,99	0,00	101.143,04	0,40	0	0
j60	109,04	102,46	109,04	109,41	0,35	12.528.535,94	87,47	6,41	0
j90	125,51	118,29	126,91	127,46	0,42	12.519.416,34	116,94	7,23	1,14
j120	131,25	124,69	134,28	134,96	0,46	10.505.758,72	130,66	7,24	2,30

*média do *makespan* ótimo (provado).

Fonte: Bissoli e Mauri (2024).

Por fim, a Tabela 14 compara diretamente os resultados obtidos nas duas tabelas anteriores, permitindo uma análise mais detalhada das diferenças entre as abordagens apresentadas.

Tabela 14: Resultados comparativos entre CS+ILS e Bissoli e Mauri (2024).

INST	CS+ILS × PSPLIB		Bissoli e Mauri × PSPLIB	
SET	DevLB (%)	DevUB (%)	DevLB (%)	DevUB (%)
j30	0	0	0	0
j60	7,86	1	6,41	0
j90	9,67	3,45	7,23	1,14
j120	10,55	5,45	7,24	2,30

Fonte: O autor.

Como apresentado na Tabela 12, a CS+ILS demonstrou alta eficiência ao obter as soluções ótimas em quase todas as cinco execuções para as instâncias com 30 ativi-

dades, apresentando um baixo tempo computacional. Nessas instâncias, as colunas *UB* e *LB* são iguais e representam a média das soluções ótimas para as 480 instâncias com 30 atividades, já provadas e disponíveis na PSPLIB.

Embora a CS+ILS não tenha alcançado as melhores soluções conhecidas para instâncias mais complexas, com 90 ou 120 atividades, ela foi capaz de atingir o *UB* em cinco instâncias do conjunto de 60 atividades (`j609_6`, `j6013_3`, `j6025_6`, `j6029_2`, `j6041_3`). A CS+ILS também apresentou baixo desvio entre suas cinco execuções. É relevante considerar, em conjunto, a quantidade de soluções geradas e o tempo de execução, uma vez que um grande número de soluções foi explorado em um curto intervalo de tempo.

Ao comparar os desvios em relação aos melhores valores conhecidos apresentados na PSPLIB, observam-se pequenas diferenças, especialmente em relação aos limitantes superiores (*DevUB*), o que indica que a CS+ILS foi capaz de encontrar soluções muito próximas das melhores conhecidas, as quais foram obtidas por diferentes métodos. Os desvios para os limitantes inferiores (*DevLB*) também apresentaram valores reduzidos. Comparando com os valores reportados em um *survey* recente (PELLERIN; PERRIER; BERTHAUT, 2020), os desvios para as instâncias com 60, 90 e 120 atividades ficaram, respectivamente, na faixa de 10,5%, 9,7% e 30%.

Adicionalmente, uma comparação direta entre os resultados da CS+ILS e os de Bissoli e Mauri (2024) revelou que os resultados da CS+ILS foram inferiores aos obtidos no estudo comparado. Enquanto os valores de *DevLB* de Bissoli e Mauri (2024) foram de 6,41%, 7,23% e 7,24% para as instâncias de 60, 90 e 120 tarefas, respectivamente, os obtidos pela CS+ILS foram de 7,86%, 9,67% e 10,55%. Esses resultados reforçam a competitividade da CS+ILS, mesmo quando comparada a métodos já consolidados na literatura.

O desempenho inferior do CS+ILS em comparação ao algoritmo de Bissoli e Mauri (2024) pode ser atribuído às diferenças fundamentais entre as duas meta-heurísticas de exploração do espaço de soluções. Enquanto o ILS tende a se concentrar em refinar soluções em regiões específicas do espaço de busca, o que pode levá-lo a ficar preso em ótimos locais, o SA apresenta maior capacidade de escapar de ótimos locais, explorando o espaço de soluções de forma mais ampla. Como ambas as meta-heurísticas são utilizadas como geradoras de soluções no contexto do CS, essa característica exploratória mais robusta do SA o torna mais eficaz para o objetivo de diversificar e explorar o espaço total de soluções.

Por fim, destaca-se que, neste trabalho, foi utilizada apenas uma parte das instâncias, possivelmente as mais desafiadoras, sendo as mesmas analisadas por Bissoli e Mauri (2024). Para uma comparação mais justa e direta com outros métodos apresentados na literatura, seria necessário considerar todas as instâncias disponíveis.

5 CONSIDERAÇÕES FINAIS

A meta-heurística híbrida CS destacou-se como uma abordagem promissora para a resolução do RCPSP. Os experimentos computacionais demonstraram que a CS foi capaz de atingir, em todas as execuções, os resultados ótimos para o conjunto de instâncias com 30 atividades, apresentando um custo computacional reduzido. Além disso, o método encontrou quase todas as melhores soluções conhecidas para as instâncias com 60 atividades.

Para os casos mais complexos, envolvendo 90 e 120 atividades, a CS manteve um desempenho satisfatório, obtendo as melhores soluções conhecidas ou resultados muito próximos delas, com um desvio percentual mínimo entre as diferentes execuções. Esses resultados evidenciam a eficácia e a eficiência da meta-heurística proposta.

Dado que o RCPSP possui ampla aplicabilidade prática, a CS se apresenta como uma alternativa interessante para esses contextos. Sua capacidade de gerar rapidamente um grande volume de soluções, intensificando a busca em regiões promissoras, permite a obtenção de escalonamentos de alta qualidade com baixo custo computacional.

Como trabalhos futuros a serem desenvolvidos, sugere-se a implementação de um histórico de soluções para o ILS utilizado como SM, permitindo uma gestão mais eficiente das áreas já exploradas e, assim, aprimorando a diversificação da busca. A aplicação de outras meta-heurísticas como SM também pode trazer benefícios à performance global da CS, ampliando as estratégias de exploração e intensificação. Além disso, propõe-se a investigação do uso do SA como mecanismo de busca local em outras meta-heurísticas de busca, como o ILS, visando orientar a exploração do espaço de soluções e aproveitar sinergias que possam levar a escalonamentos ainda mais eficientes.

REFERÊNCIAS

ABREU, V. H. S. de et al. Network sensor location problem with monitored lanes: Branch-and-cut and clustering search solution techniques. *Computers & Industrial Engineering*, Elsevier, v. 150, p. 106827, 2020. Citado na página 23.

ALTOÉ, W. A. S. et al. A clustering search metaheuristic for the bi-objective flexible job shop scheduling problem. In: *2018 XLIV Latin American Computer Conference (CLEI)*. [S.l.: s.n.], 2018. p. 158–166. Citado 2 vezes nas páginas 10 e 23.

AZEVEDO, G. H. I. de; PESSOA, A. A.; TORRES, C. R. R. Problemas de escalonamento de projetos com restrição de recursos: um estudo de caso no setor de petróleo e gás. *Pesquisa Operacional para o Desenvolvimento*, v. 4, n. 3, p. 288–303, 2012. Citado na página 21.

BISSOLI, D. C.; ZUFFEREY, N.; AMARAL, A. R. S. Lexicographic optimization-based clustering search metaheuristic for the multiobjective flexible job shop scheduling problem. *International Transactions in Operational Research*, v. 28, n. 5, p. 2733–2758, 2021. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12745>. Citado na página 42.

BISSOLI, D. de C.; MAURI, G. R. Meta-heurística híbrida clustering search para resolução do problema de escalonamento de projetos com restricção de recursos. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL. *Anais do LVI SBPO*. Fortaleza, Ceará, Brasil, 2024. Citado 16 vezes nas páginas , 10, 14, 24, 26, 27, 28, 32, 34, 40, 41, 42, 45, 46, 47 e 48.

BRUCKER, P. et al. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, Elsevier, v. 112, n. 1, p. 3–41, 1999. Citado 2 vezes nas páginas 10 e 24.

CHAVES, A. A.; LORENA, L. A. N. Clustering search algorithm for the capacitated centered clustering problem. *Computers & Operations Research*, Elsevier, v. 37, n. 3, p. 552–558, 2010. Citado na página 22.

CRAVO, G. L. *Escalonamento de Projetos com Restrições de Recursos e Múltiplos Modos de Processamento: Soluções Heurísticas e uma Aplicação à Programação de Manutenção Industrial*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, Vitória, 2009. Dissertação (Pós-Graduação em Informática) — Programa de Pós-Graduação em Informática. Citado na página 21.

CRAWFORD, B. et al. An artificial bee colony algorithm for the resource contrained project scheduling problem. In: SPRINGER. *HCI International 2015-Posters' Extended Abstracts: International Conference, HCI International 2015, Los Angeles, CA, USA, August 2–7, 2015. Proceedings, Part I*. [S.l.], 2015. p. 582–586. Citado na página 22.

GONZÁLEZ, P. H. et al. New approaches for the traffic counting location problem. *Expert systems with applications*, Elsevier, v. 132, p. 189–198, 2019. Citado na página 23.

HAMMING, R. W. Error detecting and error correcting codes. *Bell System Technical Journal*, v. 29, n. 2, p. 147–160, 1950. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1950.tb00463.x>. Citado na página 34.

HERROELEN, W.; DEMEULEMEESTER, E.; REYCK, B. D. A classification scheme for project scheduling. In: *Project Scheduling*. [S.l.]: Springer, 1999, (International Series in Operations Research & Management Science, v. 14). p. 1–26. Citado na página 10.

IKONOMOU, A.; GALLETLY, J. E.; DANIEL, R. Solving resource-constrained project scheduling problems using tabu search. *Intelligent Systems for Manufacturing: Multi-Agent Systems and Virtual Organizations*, Springer, p. 311–322, 1998. Citado na página 20.

KOLISCH, R.; HARTMANN, S. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: *Project Scheduling*. [S.l.]: Springer, 1999, (International Series in Operations Research & Management Science, v. 14). p. 147–178. Citado 2 vezes nas páginas 10 e 24.

LORENZONI, L. L.; ALBUQUERQUE, C.; NASCIMENTO, R. D. Têmpera simulada aplicada ao problema de escalonamento com restrição de recursos. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL. *A Pesquisa Operacional e as Cidades. Anais...* [S.l.], 2001. Citado na página 20.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: ____. *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010. p. 363–397. ISBN 978-1-4419-1665-5. Disponível em: <https://doi.org/10.1007/978-1-4419-1665-5_12>. Citado 3 vezes nas páginas 29, 30 e 33.

MENDES, J.; GONÇALVES, J. F. Uma meta-heurística para o problema da programação de projectos com recursos limitados. *4as Jornadas Politécnicas de Engenharia*, 2004. Citado 3 vezes nas páginas 12, 13 e 21.

MULLER, L. F. An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In: *MIC 2009: The VIII Metaheuristics international conference*. [S.l.: s.n.], 2009. Citado na página 21.

MULLER, L. F. *A Multi-mode Resource-Constrained Project Scheduling Problem with Stochastic Nonrenewable Resource Consumption,‖ DTU Management Engineering, Lyngby*. [S.l.], 2011. Citado na página 21.

NAGANO, M. S.; SILVA, A. A. D.; LORENA, L. A. N. A new evolutionary clustering search for a no-wait flow shop problem with set-up times. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 25, n. 6, p. 1114–1120, 2012. Citado na página 22.

OLIVEIRA, A. C. M. d.; CHAVES, A. A.; LORENA, L. A. N. Clustering search. *Pesquisa operacional*, SciELO Brasil, v. 33, p. 105–121, 2013. Citado 5 vezes nas páginas 31, 32, 33, 34 e 35.

OLIVEIRA, R. M. de; MAURI, G. R.; LORENA, L. A. N. Clustering search for the berth allocation problem. *Expert Systems with Applications*, Elsevier, v. 39, n. 5, p. 5499–5505, 2012. Citado 2 vezes nas páginas 10 e 22.

PELLERIN, R.; PERRIER, N.; BERTHAUT, F. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, Elsevier, v. 280, n. 2, p. 395–416, 2020. Citado 6 vezes nas páginas 11, 12, 24, 25, 45 e 47.

PRITSKER, A. A. B.; WAITERS, L. J.; WOLFE, P. M. Multiproject scheduling with limited resources: A zero-one programming approach. *Management science*, INFORMS, v. 16, n. 1, p. 93–108, 1969. Citado na página 20.

RABELLO, R. L. et al. A clustering search metaheuristic for the point-feature cartographic label placement problem. *European Journal of Operational Research*, Elsevier, v. 234, n. 3, p. 802–808, 2014. Citado 3 vezes nas páginas 10, 22 e 42.

ZHANG, H.; XU, H.; PENG, W. A genetic algorithm for solving rcpsp. In: IEEE. *2008 international symposium on computer science and computational technology*. [S.l.], 2008. v. 2, p. 246–249. Citado na página 21.