Matheus da Silva Rocha

Algoritmo Híbrido GRASP e Simulated Annealing para o Problema de Escalonamento de Projetos com Restrição de Recursos

ALEGRE - ES

2023

Matheus da Silva Rocha

Algoritmo Híbrido GRASP e Simulated Annealing para o Problema de Escalonamento de Projetos com Restrição 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 Ciência da Computação.

Universidade Federal do Espírito Santo

Centro de Ciências Exatas, Naturais e da Saúde

Departamento de Computação

Orientador: Prof. Dr. Edmar Hell Kampke

ALEGRE - ES

2023

MATHEUS DA SILVA ROCHA

ALGORITMO HÍBRIDO GRASP E SIMULATED ANNEALING PARA O PROBLEMA DE ESCALONAMENTO DE PROJETOS COM RESTRIÇÃO 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 Ciência da Computação.

Aprovado em 14 de julho de 2023.

COMISSÃO EXAMINADORA

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

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

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

Digitalizado com CamScanner

Agradecimentos

A Deus, primeiramente, por guiar e me amparar em todos os momentos vividos durante a graduação.

A toda minha família, especialmente meus pais Sérgio e Janaína, minha irmã Tainá, por enfrentarem tantas dificuldades e alegrias nesses anos de graduação. À minha avó Mirtes, que sempre foi uma mãezona para minha irmã e eu. Ao meu avô Pedro, que já não se faz mais presente, mas permanece vivo em meu coração como exemplo de vida. Também ao meu filho Pedro, que foi uma homenagem ao meu avô, por me fazer amadurecer e crescer tanto em tão pouco tempo.

A todos os professores do DCOMP, pelos ensinamentos transmitidos. Ao professor Edmar, pela orientação neste trabalho de conclusão de curso. E ao professor Geraldo, pela ajuda no decorrer do trabalho, com ideias e dicas.

A minha namorada Débora, por todo companheirismo, carinho e todos os momentos vividos, que parecem uma vida inteira.

A todas as amizades construídas na faculdade e meus amigos de república, peladas, festas e momentos que nunca esquecerei.

A todos, muito obrigado.

Resumo

Este trabalho apresenta uma solução baseada no algoritmo híbrido GRASP com *Simulated Annealing* para resolver o Problema de Escalonamento de Projetos com Restrição de Recursos (*Resource Constrained Project Scheduling Problem* - RCPSP). O método proposto utiliza o GRASP que iterativamente constrói uma solução e envia para o *Simulated Annealing* responsável por realizar a busca local. Os testes foram executados nas instâncias da PSBLIB (KOLISCH; SPRECHER, 1997) e os resultados computacionais validam a capacidade do método proposto em fornecer soluções viáveis para o RCPSP.

Palavras-chave: Escalonamento de Projetos, Restrição de Recursos, GRASP, *Simulated Annealing*.

Abstract

This work presents a solution based on the hybrid algorithm GRASP with Simulated Annealing to solve the Project Scheduling Problem with Resource Constraints - RCPSP. The proposed method uses GRASP that iteratively builds a solution and sends it to Simulated Annealing responsible for performing the local search. The tests were performed on instances of the PSBLIB (KOLISCH; SPRECHER, 1997) and the computational results validate the ability of the proposed method to provide viable solutions for the RCPSP.

Keywords: Project Scheduling, Resource Constraints, GRASP, Simulated Annealing.

Lista de Figuras

Figura 1 – Classificação dos Problemas de Escalonamento de Projetos. 13

Figura 2 – Lista de tarefas LS ordenada. 21
Figura 3 – Escolha aleatória da tarefa ρ. 21
Figura 4 – Definição da tarefa ϕ e do *range*. 21
Figura 5 – Escolha aleatória da tarefa μ dentro do intervalo $(\rho, \phi]$. 21
Figura 6 – Tarefas selecionadas para serem alteradas (ρ e μ). 22
Figura 7 – *Shift* de tarefas em LS. 22
Figura 8 – Reinserção da tarefa ρ em LS. 22
Figura 9 – Lista LS após a realização do movimento. 22

Figura 10 – Exemplo de instância com 10 tarefas. 25

Lista de Tabelas

Tabela 1 – Valores dos Parâmetros do GRASP. 26
Tabela 2 – Valores dos Parâmetros do SA. 27
Tabela 3 – Ambiente Computacional. 27
Tabela 4 – Resultados da Execução. 28
Tabela 5 – Comparação de Resultados. 29

Lista de Algoritmos

1 Algoritmo GRASP Proposto por Feo e Resende (1989) 17

2 Algoritmo *SA* Adaptado de Rocha (2013) 19

Lista de Abreviaturas e Siglas

CCENS	Centro de Ciências Exatas, Naturais e da Saúde
DCOMP	Departamento de Computação
FO	Função Objetivo
GRASP	*Greedy Randomized Adaptive Search Procedure*
LC	Lista de Candidatos
LRC	Lista Restrita de Candidatos
RCPSP	*Resource Constrained Project Scheduling Problem*
SA	*Simulated Annealing*
UFES	Universidade Federal do Espírito Santo

Sumário

1 Introdução **11**
1.1 O Problema e sua Importância 11
1.2 Objetivos 12
1.2.1 Objetivo Geral 12
1.2.2 Objetivos Específicos 12

2 Revisão de Literatura **13**
2.1 Meta-heurísticas para Problemas de Escalonamento 14

3 Metodologia **15**
3.1 Modelo Matemático do Problema 15
3.2 GRASP 16
3.2.1 Construção da Solução Inicial 17
3.2.1.1 Algoritmo de Construção da Solução Inicial 18
3.2.2 Busca local 18
3.2.2.1 *Simulated Annealing* 19
3.2.2.2 Geração de Soluções Vizinhas 20
3.2.2.3 Método Atualizar Solução 22

4 Resultados Obtidos **24**
4.1 Experimentos Computacionais 24
4.1.1 Instâncias 24
4.1.2 Escolha de Parâmetros 25
4.1.3 Detalhes de Implementação e Execução de Testes 27
4.2 Resultados Computacionais 27
4.3 Análise e Discussão 29

5 Conclusão **31**

Referências **32**

1 Introdução

Mudanças macroeconômicas tornaram a gestão de projetos cada vez mais necessária para o aumento da produtividade das empresas, visto que o mercado aspira por eficiência e agilidade. Por consequência, o campo da pesquisa de otimização visa reduzir tempo e gastos na produção, mantendo a qualidade do produto final. Além disso, por meio de uma melhor gestão, não há apenas benefícios econômicos, mas também ambientais, já que se gerenciado de forma eficaz, espera-se que o uso dos recursos diminuam (MELO, 2018).

Segundo Almeida, Correia e Gama (2016), para que se obtenha a otimização dos projetos é imprescindível a criação de um cronograma visando respeitar as restrições e possíveis regras de precedência das tarefas, a fim de aumentar o desempenho preestabelecido na gestão de projetos. Ademais, o conjunto de métodos científicos da área de pesquisa operacional pode auxiliar na tomada de decisões e na solução do problema para elaboração do cronograma de execução de tarefas em máquinas de uma indústria de forma otimizada.

O problema supracitado é NP-difícil *(Non-Deterministic Polynomial time)*. Isso significa que embora nunca tenha sido provado a não existência de um algoritmo eficiente, o mesmo ainda não foi encontrado. Nesse caso, é comum usar heurísticas e métodos de aproximação, cujo objetivo é encontrar uma solução aproximada com baixo tempo de processamento. Temos como exemplo uma fábrica de carros, que tem seus processos dinâmicos e que não podem parar, ou seja, necessita-se de uma resposta rápida, caso seja indispensável um recálculo durante o processo.

1.1 O Problema e sua Importância

O Problema de Escalonamento de Projetos com Restrição de Recursos *(Resource Constrained Project Scheduling Problem - RCPSP)* é constituído por um conjunto de tarefas a serem processadas com um número limitado de recursos. As tarefas competem pela utilização dos recursos, que podem ser distintos, tais como mão-de-obra, capital, ferramentas e outros. Logo, tem-se como objetivo escalonar as tarefas de modo a definir os tempos de início e os recursos utilizados, a fim de otimizar o tempo total de execução das tarefas, acelerando assim o fim do projeto.

Segundo Goldratt (1996), é possível identificar as características que motivam a solução do RCPSP em uma organização, por meio das seguintes situações:

1. Atraso na entrega parcial e final: inaptidão na entrega de um projeto, ou parte do

mesmo;

2. Dificuldade em atingir objetivos simples: incapacidade da equipe de seguir o planejamento do projeto, sem a necessidade de horas extras;

3. Sobrecarga da equipe: reflexo de uma má aplicação do tempo disponível criando picos de carga de trabalho, ou muitos problemas a serem resolvidos;

4. Acumulação de tarefas críticas: é o fenômeno percebido pela equipe de que caso não se entregue as tarefas atuais, isso ocasionará um atraso no tempo final do projeto.

De acordo com Kolisch (1996) o RCPSP é uma generalização do problema de *job-shop* estático e, portanto, pertence à classe de problemas NP-completo, que é uma intercessão dos problemas NP e os NP-difíceis. Ao relaxar as restrições de recursos, o RCPSP reduz-se ao Método de Caminho Crítico (*Critical Path Method - CPM*), que pode ser resolvido por recursão em tempo polinomial. Por conta da inerente intratabilidade do RCPSP, uma infinidade de abordagens exatas e heurísticas foram propostas na literatura.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é aplicar a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), com adição do *Simulated Annealing* na realização da busca local, para resolução do RCPSP.

1.2.2 Objetivos Específicos

a) Estudar os dados necessários para resolver o RCPSP.

b) Implementar a meta-heurística GRASP com *Simulated Annealing* para resolver o RCPSP.

c) Realizar testes de validação e calibragem de parâmetros no algoritmo desenvolvido.

d) Efetuar correções no algoritmo.

e) Executar a aplicação desenvolvida utilizando as instâncias propostas na literatura.

f) Avaliar os resultados obtidos e compará-los com os apresentados na literatura.

2 Revisão de Literatura

Habibi, Barzinpour e Sadjadi (2018) desenvolveram uma importante obra para o RCPSP, a qual teve como foco realizar um *review* dos desenvolvimentos para RCPSP. Este é um importante instrumento guiador das pesquisas do problema. Foram mapeados mais de 200 artigos, desde o ano de 1980, possibilitando a classificação e mensuração de diferentes tipos de RCPSP em diversas categorias e subcategorias, conforme apresentado na Figura 1.

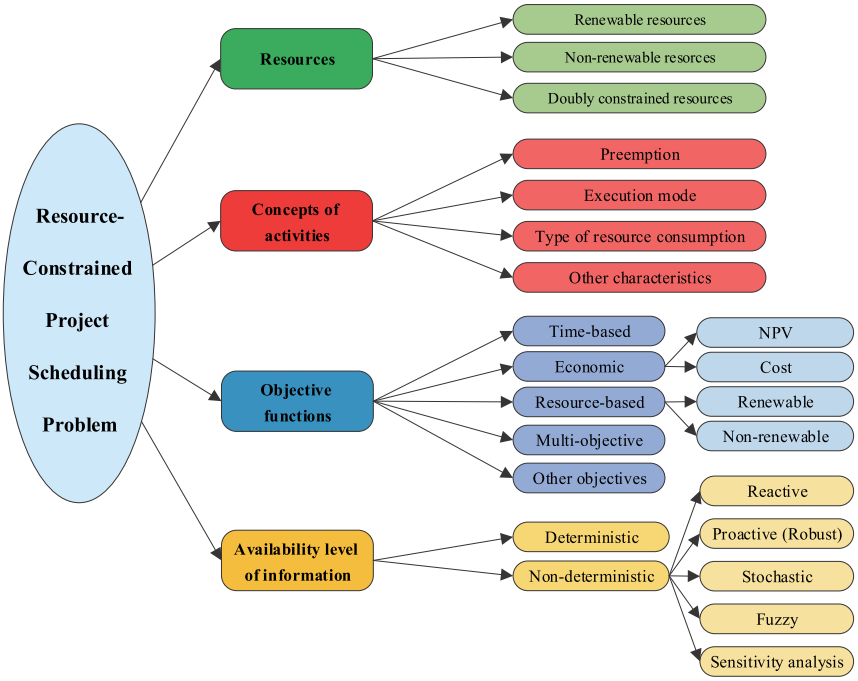

Figura 1 – Classificação dos Problemas de Escalonamento de Projetos.

Fonte: (HABIBI; BARZINPOUR; SADJADI, 2018).

A partir das contribuições de Habibi, Barzinpour e Sadjadi (2018) ficou notório a importância dos Problemas de Escalonamento de Projetos de uma maneira geral, em especial, da subcategoria focada nos recursos renováveis. Nesta subcategoria há particularidades como o fato dos recursos serem liberados quando a tarefa termina de ser processada, como exemplo: uso de máquinas, mão-de-obra, local de trabalho compartilhado, etc. Portanto, é fundamental para garantir a sustentabilidade e a conservação dos recursos naturais. Ele ajuda a gerenciar os recursos de maneira equilibrada e eficiente, minimizando o uso excessivo ou o desperdício, e contribui para minimizar os impactos ambientais negativos.

Dentre os artigos citados por Habibi, Barzinpour e Sadjadi (2018), foi possível notar a semelhança entre o problema apresentado por Kadam e Kadam (2014) com o apresentado neste trabalho. Os autores propuseram uma solução utilizando a meta-heurística Algoritimo Genético *(Genetic Algoritm - GA)* que foi executada nas instâncias da biblioteca PSBLIB (KOLISCH; SPRECHER, 1997). O GA mostrou-se capaz de diminuir consideravelmente o valor da Função Objetivo (FO) para se atingir soluções ótimas-locais em instâncias de grande porte, obtendo o melhor valor de desvio padrão para essas instâncias em comparação aos demais trabalhos.

2.1 Meta-heurísticas para Problemas de Escalonamento

Jain, Bilolikar e Joshi (2016), utilizaram as meta-heurísticas *Variable Neighborhood Search* (VNS) e GA, para resolução do RCPSP, de maneira análoga a qual é abordada neste trabalho. Os resultados são medianos quando comparados aos já encontrados na literatura. Mesmo assim é uma abordagem interessante devido sua fácil implementação.

Damodaran e Vélez-Gallego (2012) implementaram uma heurística baseada no *Simulated Annealing* (SA), seus resultados são praticamente equivalentes ao anterior implementado por Damodaran, Vélez-Gallego e Maya (2011) utilizando o GRASP.

Há por outro lado trabalhos mais recentes como o de Goncharov (2022), em que é apresentado um algoritmo de busca local, onde é encontrado melhores resultados para algumas instâncias contidas na biblioteca PSBLIB (KOLISCH; SPRECHER, 1997). Posteriormente, em 2023, tem-se que Goncharov (2023) apresenta uma implementação do algoritmo genético para o problema, que de maneira gulosa prioriza o grau de criticidade dos recursos, mecanismo esse derivado da solução de um problema relaxado com restrição de recursos acumulativos, este último também atingindo bons resultados frente aos apresentados na literatura.

De acordo com Liu et al. (2023), diversos tipos de métodos foram usados para resolver o RCPSP, entre eles as metas-heurísticas híbridas. Nesse grupo, o trabalho de Elsayed et al. (2017) se destaca por apresentar um framework com diversos operadores para algoritmos genéticos e algoritmos de evolução diferencial. O método proposto por Elsayed et al. (2017) apresenta uma das melhores performances para o RCPSP.

3 Metodologia

Neste trabalho foi implementada a meta-heurística GRASP, juntamente com o SA para refinar a busca local. As instâncias utilizadas nos testes são as comumente usadas em outros trabalhos da literatura e disponibilizadas na biblioteca PSPLIB (KOLISCH; SPRECHER, 1997).

Na Seção 3.1 é apresentado o modelo matemático do problema abordado, e na Seção 3.2 a meta-heurística GRASP implementada é descrita em detalhes.

3.1 Modelo Matemático do Problema

Como a abordagem deste trabalho tem cunho de pesquisa, será explorado o primeiro modelo do RCPSP com foco nas restrições de recurso e precedência. Este modelo é baseado em programação linear inteira mista na qual inclui as seguintes variáveis:

- m é não-negativa e representa o *makespan*, em outras palavras, o tempo de conclusão da última tarefa.
- n representa o número total de tarefas.
- T é a soma do tempo de processamento de todas as tarefas.
- d_i é o tempo de processamento da tarefa i.
- p refere-se aos possíveis tempo de início da tarefa a ser processada, que é pertencente ao conjunto P. Dado o fato do tempo ser unitário e inteiro, para a tarefa i, o valor de $p \in [0, T–d_i]$.
- x_{ip} é binária e indica com o valor 1 se a tarefa i começa seu processamento no tempo p, e valor 0 caso contrário.
- b representa um recurso.
- r representa a quantidade de recursos disponíveis.
- R_b representa a disponibilidade do recurso b.
- C_{ib} é a quantidade de recurso b que a tarefa i necessita para ser executada.
- *pred* é o conjunto de pares ordenados (i, j), contendo duas tarefas i e j. Nesse caso, a precedência representa que a tarefa i só pode começar depois que a j terminar.

Dessa forma, o modelo matemático proposto por Melo (2018), é descrito da seguinte forma:

$$\text{Minimizar} \quad m \tag{3.1}$$

Sujeito a:

$$\sum_{p \in P} x_{ip} = 1, \quad i = 0, 1, ..., n+1 \tag{3.2}$$

$$\sum_{i=0}^{n+1} \sum_{p \in \{t-d_i+1 \leq p \leq t\}}^{n+1} C_{ib} x_{ip} \leq R_b, \quad \forall t \in P, b = 1, 2, ..., r \tag{3.3}$$

$$x_{it} \leq \sum_{q \in P, q \geq t+d_i} x_{jq}, \quad \forall (i,j) \in pred, \quad \forall t \in P \tag{3.4}$$

$$m \geq \sum_{p \in P} x_{ip}(p + d_i), \quad i = 0, 1, ..., n+1 \tag{3.5}$$

A função objetivo (3.1) busca minimizar o *makespan*. As restrições em (3.2) asseguram que cada atividade deve ser iniciada em apenas um momento específico, levando em consideração que algumas atividades podem iniciar no mesmo instante. As restrições em (3.3) garantem que o consumo de recursos das atividades iniciadas em um determinado momento estejam dentro da disponibilidade de cada recurso. As restrições (3.4) asseguram que as relações de precedência entre as atividades sejam cumpridas, ou seja, que uma atividade só pode começar após o término de todas as suas atividades predecessoras. As restrições em (3.5) calculam o *makespan*, que deve ser no mínimo igual ao momento de término da última atividade processada (MELO, 2018).

3.2 GRASP

A meta-heurística GRASP foi proposta por Feo e Resende (1995), e já se provou no tempo, sendo frequentemente usada em problemas de roteamento, empacotamento e agendamento. No Algoritmo 1 é mostrado o pseudocódigo genérico da meta-heurística.

Algoritmo 1: Algoritmo GRASP Proposto por Feo e Resende (1989)

Entrada: Max_{Iter}, α
Saída: Solução S_{Melhor}

1 **início**
2 $\quad FO_{Melhor} \leftarrow \infty$;
3 $\quad$ **para** $i \leftarrow 1$ **até** Max_{Iter} **faça**
4 $\quad\quad S_{Inicial} \leftarrow GeraSolucaoInicial(\alpha)$;
5 $\quad\quad S_{Atual} \leftarrow BuscaLocal(S_{Inicial})$;
6 $\quad\quad$ **se** $f(S_{Atual}) < FO_{Melhor}$ **então**
7 $\quad\quad\quad S_{Melhor} \leftarrow S_{Atual}$;
8 $\quad\quad\quad FO_{Melhor} \leftarrow f(S_{Atual})$;
9 $\quad\quad$ **fim se**
10 $\quad$ **fim para**
11 **fim**

Basicamente o algoritmo GRASP consiste em duas fases: construção da solução inicial (linha 4) e refinamento da solução (linha 5), também conhecida como busca local.

No Algoritmo 1, existem dois parâmetros principais: o número máximo de iterações (Max_{Iter}) e o fator de aleatoriedade (α). Olhando mais minuciosamente, o algoritmo enquanto não atingir o Max_{Iter}, repetidamente constrói uma solução $S_{Inicial}$ (linha 4) e esta solução é melhorada por um procedimento de busca local (linha 5). A melhor solução encontrada até o momento (linhas 6 a 8) é sempre armazenada.

3.2.1 Construção da Solução Inicial

A construção da solução inicial combina elementos de busca aleatória e busca gulosa. Uma das características do GRASP é a utilização de dois critérios para a construção de uma solução inicial: Lista de Candidatos (LC) e Lista Restrita de Candidatos (LRC). O fator de aleatoriedade (α) é o parâmetro de mediação entre LC e LRC. A partir disso, tem-se as seguintes definições:

- LC: é uma lista com todos os elementos (candidatos) a serem inseridos na solução do problema. LC é mantida sempre ordenada por algum critério, geralmente relacionado ao impacto na qualidade da solução que está sendo construída.

- LRC: é uma lista formada por parte dos elementos mais bem avaliados da LC, de acordo com o critério usado na ordenação. Em outras palavras, a LRC é uma sublista formada com os elementos iniciais da LC.

- α: é um fator de ponderação que determina o tamanho da LRC e, consequentemente, quais candidatos serão considerados na seleção. Quando $\alpha = 0$, apenas o primeiro

elemento da LC irá fazer parte da LRC, em contrapartida, quando $\alpha = 1$, todos os candidatos da LC farão parte da LRC.

3.2.1.1 Algoritmo de Construção da Solução Inicial

Neste trabalho, para a construção de uma solução inicial, foi desenvolvido um método que iterativamente adiciona uma tarefa à solução $S_{Inicial}$, que inicialmente está vazia. Após a inserção de todas as tarefas, o método de construção finaliza retornando $S_{Inicial}$.

O primeiro passo do método é criar uma lista L com todas as n tarefas. A partir disso, em cada iteração do método a lista L é ordenada de forma não decrescente pelo valor λ_i de cada tarefa $i \in L$.

O valor de λ_i representa o instante de tempo mínimo de início do processamento da tarefa i, considerando as suas tarefas predecessoras. Em outras palavras, λ_i será determinado de acordo com o valor do maior instante de tempo de término do processamento das tarefas predecessoras de i. Como exemplo, considere que a tarefa 3 possui como predecessoras as tarefas 1 e 2. Considerando ainda que as tarefas 1 e 2 já foram inseridas em $S_{Inicial}$, e que os processamentos dessas tarefas se encerram, respectivamente, nos instantes 17 e 22, então o valor de λ_3 será 22.

Caso alguma tarefa predecessora de i não esteja inserida em $S_{Inicial}$, então o valor de λ_i será penalizado com um valor alto (Ψ) acrescido do tempo de processamento da tarefa i (d_i). Mais uma vez, para exemplificar essa situação, considere agora que a tarefa 3 possui as mesmas tarefas predecessoras 1 e 2 do exemplo anterior. No entanto, suponha que a tarefa 1 já foi inserida em $S_{Inicial}$ e que a tarefa 2 ainda não foi inserida em $S_{Inicial}$. Nesse caso o valor de $\lambda_3 = \Psi + d_3$.

Em cada iteração do método, após o cálculo de λ_i e da ordenação de L, é escolhida aleatoriamente uma tarefa t dentre as $\alpha \times |L|$ tarefas iniciais de L. Nesse momento t é removida de L e inserida na solução $S_{Inicial}$, sendo que o instante de início do processamento da tarefa t será λ_t, caso as tarefas predecessoras de t já estejam inseridas em $S_{Inicial}$, e caso contrário, será o maior instante de tempo de término das tarefas já inseridas em $S_{Inicial}$ até aquele momento.

Como dito anteriormente, após a inserção de todas as tarefas em $S_{Inicial}$ ($L = \emptyset$), o método termina e a solução $S_{Inicial}$ construída é retornada.

3.2.2 Busca local

A estratégia de busca local deste trabalho baseia-se na meta-heurística *Simulated Annealing* mostrada na Subseção 3.2.2.1.

3.2.2.1 *Simulated Annealing*

Annealing é um processo técnico que visa alcançar uma estrutura homogênea em um metal através do aquecimento a uma temperatura elevada, seguido de um resfriamento lento e controlado (HAESER; RUGGIERO, 2008).

O *Simulated Annealing* foi proposto por Kirkpatrick, Gelatt e Vecchi (1983), tem como propósito ser um algoritmo que busca seguir as técnicas do processo de *annealing*, e é utilizado de maneira geral na solução de problemas de otimização combinatória. No Algoritmo 2 é mostrado o pseudocódigo genérico da meta-heurística SA, com os seguintes parâmetros: Solução inicial ($S_{Inicial}$), Temperatura inicial ($T_{inicial}$), Temperatura final (T_{final}), Taxa de resfriamento (β) e Número de soluções vizinhas (N_v) que são geradas a cada iteração.

Algoritmo 2: Algoritmo *SA* Adaptado de Rocha (2013)

Entrada: Solução $S_{Inicial}$, $T_{inicial}$, T_{final}, β, N_v
Saída: Solução S_{Melhor}

```
1  início
2      T ← T_inicial;
3      S_Melhor ← S_Inicial;
4      S_Atual ← S_Inicial;
5      enquanto T > T_final faça
6          para i ← 1 até N_v faça
7              S_Vizinho ← GeraVizinho(S_Atual);
8              Δf ← f(S_Vizinho) − f(S_Atual);
9              se Δf < 0 então
10                 S_Atual ← S_Vizinho;
11                 se f(S_Vizinho) < f(S_Melhor) então
12                     S_Melhor ← S_Vizinho;
13                 fim se
14             senão
15                 Gera um número aleatório ρ ∈ [0, 1];
16                 se ρ < e^(−Δf/T) então
17                     S_Atual ← S_Vizinho;
18                 fim se
19             fim se
20         fim para
21         T ← T * β;
22     fim enquanto
23 fim
```

O algoritmo inicia com $T_{inicial}$ alta e é gradualmente resfriado até alcançar a temperatura final (T_{final}). A cada iteração, ou seja, para cada temperatura T, N_v soluções vizinhas são geradas. Se a solução vizinha gerada apresentar uma FO melhor do que a solução atual, essa é atualizada. No entanto, se a solução vizinha apresentar uma FO pior, ela ainda pode ser aceita com uma probabilidade calculada como $e^{-\Delta f/T}$, em que Δf representa a diferença entre as FO's da solução vizinha e da solução atual, e T é a temperatura atual. Quanto maior a diferença Δf e menor a temperatura T, menor será a chance de aceitar uma solução vizinha.

No início, quando a temperatura está alta, o algoritmo tende a aceitar uma grande variedade de soluções. À medida que a temperatura diminui, poucas soluções piores são aceitas, o que intensifica a busca em uma determinada vizinhança. A cada iteração, a temperatura T é multiplicada pelo valor β, onde β é uma taxa com valor entre 0 e 1. O algoritmo termina quando a temperatura T for menor que T_{final}.

3.2.2.2 Geração de Soluções Vizinhas

A geração de vizinhos é uma parte fundamental do algoritmo SA. Nesta seção, é apresentada a implementação do método que gera uma solução vizinha ($S_{Vizinho}$) a partir da solução S_{Atual}.

Portanto, para gerar uma solução vizinha, inicialmente se recebe uma lista de tarefas LS. A lista LS é então ordenada pelo tempo de início de processamento das tarefas em S_{Atual}. Em seguida, uma tarefa ρ é escolhida aleatoriamente entre as n tarefas de LS.

Após selecionar a tarefa ρ, é definido o intervalo (*range*) no qual a tarefa pode ser movida para a direita dentro de LS. Em outras palavras, esse intervalo define os possíveis locais de troca para a tarefa ρ. Para a definição do intervalo, é realizada uma busca a partir de ρ, com intuito de encontrar ϕ, a última tarefa não sucessora de ρ.

Em seguida, um local de troca é escolhido aleatoriamente no intervalo definido entre a tarefa ρ e a tarefa ϕ na lista LS. Isso significa que uma tarefa μ em LS será substituída pela tarefa ρ nesse local de troca escolhido.

Para realizar essa substituição, inicialmente a tarefa ρ é removida de LS, em seguida é feito um movimento de *shift* em LS da posição ocupada anteriormente pela tarefa ρ até o local de troca (posição ocupada pela tarefa μ). Então, a tarefa ρ é inserida na posição em que a tarefa μ estava antes do movimento de *shift*.

Por fim, após a realização do movimento descrito acima, a lista LS é submetida a um método de atualização dos tempos de início de processamento das tarefas, levando em consideração agora a restrição de recursos, uma vez que a restrição de precedência é observada durante o movimento. O método de atualização dos tempos de processamento retorna a solução $S_{Vizinho}$, e é apresentado na próxima subseção.

Essa abordagem de geração de vizinhos permite explorar diferentes soluções na busca por melhores configurações. Cada vizinho gerado representa uma solução potencialmente melhor ou pior em relação à S_{Atual}.

Temos abaixo um exemplo com intuito de mostrar com mais clareza o processo de geração de soluções vizinhas.

Na Figura 2, é mostrado um exemplo da lista LS, com 10 tarefas, ordenada pelo tempo de início de processamento em S_{Atual}.

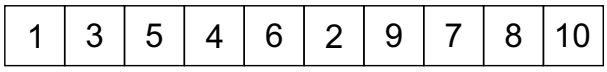

Figura 2 – Lista de tarefas LS ordenada.

O primeiro passo, é selecionar uma tarefa de maneira aleatória em LS. Neste exemplo é selecionado a tarefa 3 (ρ), como mostra a Figura 3.

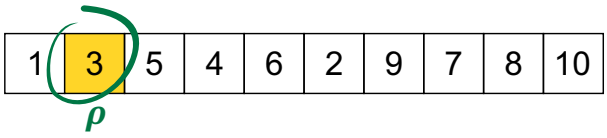

Figura 3 – Escolha aleatória da tarefa ρ.

Considerando que as tarefas 9 e 8 (destacas em vermelho na Figura 4) são sucessoras da tarefa 3 (ρ), ocorre a definição automática da tarefa 2 (última tarefa após ρ que não é sucessora de ρ) como a tarefa ϕ. Portanto, temos o *range* definido como sendo o intervalo: $(\rho, \phi]$. Este passo é importante para que não seja violada a regra de precedência das tarefas em LS, mantendo assim a solução vizinha viável.

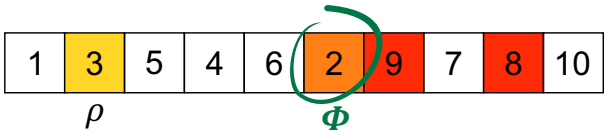

Figura 4 – Definição da tarefa ϕ e do *range*.

Neste momento é selecionada uma tarefa de maneira aleatória (μ) pertencente ao intervalo (*range*) definido acima. Considere como exemplo, a escolha da tarefa 4 na Figura 5 (destaque verde).

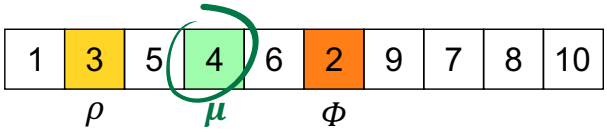

Figura 5 – Escolha aleatória da tarefa μ dentro do intervalo $(\rho, \phi]$.

Portanto, como mostrado na Figura 6, as tarefas selecionadas para serem alteradas são ρ e μ, que limitam o movimento a ser executado.

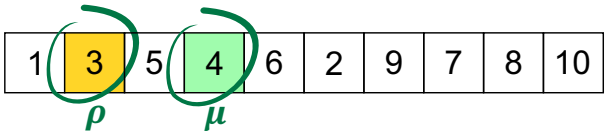

Figura 6 – Tarefas selecionadas para serem alteradas (ρ e μ).

Para a realização do movimento, conforme dito anteriormente, inicialmente a tarefa ρ é retirada de LS, para em seguida ser realizado um movimento de *shift*, que é responsável por realocar as tarefas em LS. Na Figura 7 é mostrado que as tarefas 5 e 4 são movidas para a esquerda.

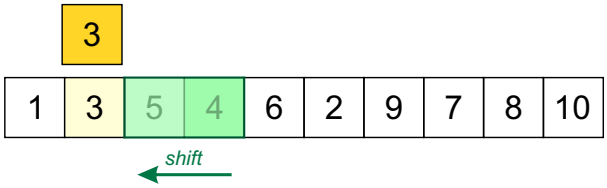

Figura 7 – *Shift* de tarefas em LS.

Após o *shift*, a tarefa 3 (ρ) é inserida novamente em LS, mas agora na posição onde anteriormente estava a tarefa 4 (μ), como pode ser visto na Figura 8.

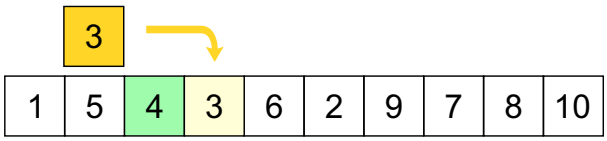

Figura 8 – Reinserção da tarefa ρ em LS.

Por fim, é apresentada na Figura 9 a lista LS após a realização do movimento. Esta lista representa a ordem de início de processamento das tarefas em $S_{Vizinho}$.

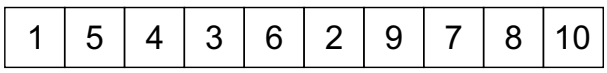

Figura 9 – Lista LS após a realização do movimento.

3.2.2.3 Método Atualizar Solução

Consiste basicamente em um método responsável por receber uma sequência de tarefas, e realizar o ajuste do tempo de início de processamento de cada tarefa, de acordo com recursos e precedência.

O método recebe a lista de tarefas LS. O método sequencialmente calcula o menor tempo possível de início de processamento de cada tarefa $i \in LS$, considerando as tarefas anteriores a tarefa i em LS e os recursos necessários para o processamento de i. Ao final, o tempo de início da tarefa i é atualizado na solução S.

Dessa forma, a solução S se torna viável, uma vez que a lista LS já respeitava a ordem de precedência das tarefas, e a partir de agora atende a restrição de utilização dos recursos.

4 Resultados Obtidos

Neste capítulo são detalhados nas Seções 4.1, 4.2 e 4.3, respectivamente, os experimentos computacionais para validar o método proposto, os resultados obtidos, e sua análise e discussão.

4.1 Experimentos Computacionais

Para validar a qualidade das soluções obtidas pelo método proposto, experimentos computacionais foram realizados. Nas seções seguintes são detalhados alguns aspectos importantes desta etapa, como as instâncias utilizadas e a definição dos parâmetros do GRASP+SA.

4.1.1 Instâncias

As instâncias utilizadas para realização dos experimentos computacionais são muito conhecidas na literatura, pois são utilizadas em diversos trabalhos, como Kadam e Kadam (2014), Jain, Bilolikar e Joshi (2016) e Goncharov (2023).

As instâncias da biblioteca PSPLIB são divididas em quatro categorias (j30, j60, j90 e j120). As categorias representam o tamanho (número de tarefas n) das instâncias, ou seja, as instâncias da categoria j30 possuem 30 tarefas, as instâncias da categoria j60 possuem 60 tarefas, e assim sucessivamente.

A partir disso, foram escolhidas seis instâncias de cada categoria, para que houvesse um espaço amostral conciso, contendo proporcionalmente a mesma quantidade de tarefas, e instâncias pequenas, médias e grandes. Dessa forma, o objetivo era validar o método proposto neste trabalho, em diversos cenários.

A Figura 10 apresenta o exemplo de uma instância hipotética j10.

```
******************************************************************************
file with basedata            : j10.bas
initial value random generator: -
******************************************************************************
projects                      :  1
jobs (incl. supersource/sink ):  12
horizon                       :  35
RESOURCES
  - renewable                 :  2   R
  - nonrenewable              :  0   N
  - doubly constrained        :  0   D
******************************************************************************
PROJECT INFORMATION:
pronr.  #jobs rel.date duedate tardcost  MPM-Time
    1     10      0      38       26       38
******************************************************************************
PRECEDENCE RELATIONS:
jobnr.    #modes  #successors   successors
   1        1          3          2   3   4
   2        1          2          5   6
   3        1          2          10  11
   4        1          1          9
   5        1          2          7   8
   6        1          2          10  11
   7        1          2          9   10
   8        1          1          9
   9        1          1          12
  10        1          1          12
  11        1          1          12
  12        1          0
******************************************************************************
REQUESTS/DURATIONS:
jobnr. mode duration  R 1  R 2
------------------------------------------------------------------------------
  1      1     0       0   0
  2      1     3       5   1
  3      1     2       0   4
  4      1     5       1   4
  5      1     4       1   3
  6      1     2       3   2
  7      1     3       3   1
  8      1     4       2   4
  9      1     2       4   0
 10      1     4       5   2
 11      1     6       2   5
 12      1     0       0   0
******************************************************************************
RESOURCEAVAILABILITIES:
  R 1  R 2
   6   8
******************************************************************************
```

Figura 10 – Exemplo de instância com 10 tarefas.

Onde os números assinalados representam respectivamente:

1. Número de recursos;
2. Número de tarefas;
3. Lista de sucessores de cada tarefa;
4. Tempo de processamento de cada tarefa;
5. Quantidade de cada recurso utilizado por cada tarefa;
6. Quantidade de cada recurso disponíveis;

4.1.2 Escolha de Parâmetros

Como proposto por Feo e Resende (1995), por padrão o GRASP recebe dois parâmetros, são eles, α e Max_{Iter}. Porém, para os testes realizados, o número de Max_{Iter} foi subs-

tituído pelo parâmetro Γ, que representa o tempo limite de execução do algoritmo. Neste caso, Γ foi definido de maneira que o mesmo possa se adaptar aos diferentes tamanhos de instâncias (n), que variam entre 30, 60, 90 e 120 tarefas.

O valor de α foi definido executando o algoritmo da heurística construtiva do GRASP, para os valores do conjunto: {`0, 0.2, 0.5, 0.8, 1.0`}, cinco vezes para cada instância e com cada valor de α por 300 segundos cada execução.

Como os melhores resultados de FO foram para 0.8 e 1.0. Foi realizado novamente, um novo teste para os valores do conjunto: {`0.8, 0.85, 0.9, 0.95, 1.0`}.

Logo, como os menores valores de FO foram majoritariamente obtidos com $\alpha = 0.85$, esse valor foi o escolhido para ser usado nos experimentos computacionais.

A Tabela 1 resume os parâmetros do GRASP definidos para a realização dos experimentos computacionais.

Tabela 1 – Valores dos Parâmetros do GRASP.

Parâmetro	Valor
α	0.85
Γ	$6 \times n$ seg

Já para definir os parâmetros do SA, foram realizados testes com propósito de encontrar a melhor combinação de parâmetros. Os conjuntos com valores usados nesses testes são apresentados abaixo:

- Temperatura inicial ($T_{inicial}$): {10, 50, 100, 200}
- Temperatura final (T_{final}): {0.01}
- Taxa de resfriamento (β): {0.1, 0.995, 0.999, 0.9}
- Número soluções vizinhas (N_v): {10, 100, 300, 500}

Porém, ao realizar as execuções foi percebido que havia um grande problema ao inserir parâmetros com escopo grande para instâncias pequenas, e o contrário, parâmetros com valores pequenos para instâncias grandes.

Dessa maneira foi introduzido a estratégia de parâmetros adaptativos (APT), proposta por Wang, Zhang e Zhou (2021). A Tabela 2, mostra os parâmetros definidos. Vale ressaltar que o parâmetro N_v foi definido em função do tamanho da instância, assim como, a temperatura inicial ($T_{inicial}$).

Tabela 2 – Valores dos Parâmetros do SA.

Parâmetro	Valor
$T_{inicial}$	$0.01 \times \mathtt{n}$
T_{final}	0.01
β	0.995000
N_v	$1.2 \times \mathtt{n}$

4.1.3 Detalhes de Implementação e Execução de Testes

O algoritmo foi desenvolvido na linguagem de programação C++, compilado com GCC *Compiler* 9.4.0, e os testes executados em um computador com as configurações apresentadas na Tabela 3.

Tabela 3 – Ambiente Computacional.

Item	Valor
Processador	Ryzen 5 3.6GHz
Memória ram	16GB
Sistema operacional	Ubuntu 20.04

Para cada uma das instâncias, foram realizadas cinco execuções, sendo que o tempo máximo de execução foi definido por Γ, que conforme dito anteriormente, representa para as instâncias de tamanho 30, 60, 90 e 120, respectivamente, os valores 180, 360, 540 e 720 segundos.

Todas as instâncias usadas nos experimentos computacionais deste trabalho estão disponíveis para *download* no *website* da biblioteca PSPLIB. Para as instâncias da categoria j30 o *website* disponibiliza também o valor da FO da solução ótima, enquanto que para as outras categorias o melhor valor obtido a partir de métodos heurísticos também é apresentado.

O código-fonte do método proposto neste trabalho e demais arquivos se encontram disponíveis em: https://github.com/matheuzsr/rcpsp.

4.2 Resultados Computacionais

Nesta seção, são mostrados os resultados computacionais obtidos a partir dos detalhes de execução descritos acima.

Na Tabela 4 pode-se observar os resultados para cada categoria de instância, $j30$, $j60$, $j90$ e $j120$. Para cada uma delas se tem o valor da menor FO obtida dentre as cinco execuções (f_{best}), a média (f_{avg}) e o desvio padrão (f_{sd}). Além disso, também é exibido na coluna ω o tempo gasto (em segundos) para que a solução com valor f_{best} fosse gerada.

Os resultados apresentados na Tabela 4 demonstram a eficácia do método proposto em diferentes instâncias. Na maioria das instâncias os resultados se mostraram consistentes. Isso pode ser observado através dos valores de f_{sd} próximos a zero. Essa consistência indica que a heurística mostrou-se robusta, produzindo soluções repetíveis e confiáveis para essas instâncias.

Ademais, é interessante notar que a instância $j1201_1$ apresentou um desvio padrão de $f_{sd} = 6,5$. Esse resultado indica uma maior variabilidade nos resultados encontrados para essa instância em particular, sugerindo que suas características podem ter impactado a eficiência do GRASP+SA, ou seja, como todas as instâncias na categoria $j120$ possuem a mesma quantidade de tarefas (120) e a mesma quantidade de recursos (4), observou-se mais atentamente a instância $j1201_1$ e foi possível notar que a quantidade de cada um dos recursos, é menor do que a encontrada nas demais instâncias da mesma categoria, o que pode ter impactado nessa variação dos resultados.

Tabela 4 – Resultados da Execução.

	Instâncias	f_{best}	f_{avg}	f_{sd}	$\omega(seg.)$
j30	j301_1	43	43	0,0	2,270633
	j301_2	47	47,6	0,5	0,887432
	j3024_9	43	43	0,0	0,000410
	j3024_10	53	53	0,0	0,000406
	j3048_9	59	59	0,0	0,000413
	j3048_10	54	54	0,0	0,000416
j60	j601_1	77	79,2	1,8	0,001436
	j601_2	72	75,2	2,8	4,063890
	j6024_9	80	80	0,0	0,001456
	j6024_10	66	66	0,0	0,001452
	j6048_9	82	82	0,0	0,001453
	j6048_10	70	70	0,0	0,001439
j90	j901_1	80	84	2,6	271,029068
	j901_2	94	95,8	1,3	194,870640
	j9024_9	80	80	0,0	0,003572
	j9024_10	89	89	0,0	0,003570
	j9048_9	89	89	0,0	0,003604
	j9048_10	93	93	0,0	0,003586
j120	j1201_1	127	135,6	6,5	0,007173
	j1201_2	131	139,2	5,2	675,747958
	j12030_9	101	102,2	1,3	698,537857
	j12030_10	91	92,4	1,1	711,374463
	j12060_9	101	102,4	0,9	585,594623
	j12060_10	99	100,6	1,1	538,571620

4.3 Análise e Discussão

Na Tabela 5 há uma comparação dos resultados obtidos pelo GRASP+SA, com os contidos na PSBLIB, que mapeia os melhores resultados da literatura.

Para cada instância é apresentado na coluna f^* o valor da melhor solução disponibilizada na literatura. Na coluna seguinte (f_{best}) é apresentado o menor valor de FO obtido pelo método proposto neste trabalho, onde os valores que estão em **negrito** representam que o método GRASP+SA obteve resultados idênticos aos resultados apresentados pela biblioteca PSPLIB. Por fim, na última coluna é apresentado o valor de Δ, que representa a diferença percentual entre f_{best} e f^*, e que é calculado pela fórmula abaixo:

$$\Delta = \frac{f_{best} - f^*}{f_{best}} \times 100$$

Tabela 5 – Comparação de Resultados.

	Instâncias	f^*	f_{best}	$\Delta(\%)$
j30	j301_1	43	**43**	0,00
	j301_2	47	**47**	0,00
	j3024_9	43	**43**	0,00
	j3024_10	53	**53**	0,00
	j3048_9	59	**59**	0,00
	j3048_10	54	**54**	0,00
j60	j601_1	77	**77**	0,00
	j601_2	68	72	5,56
	j6024_9	80	**80**	0,00
	j6024_10	66	**66**	0,00
	j6048_9	82	**82**	0,00
	j6048_10	70	**70**	0,00
j90	j901_1	73	80	8,75
	j901_2	92	94	2,17
	j9024_9	80	**80**	0,00
	j9024_10	89	**89**	0,00
	j9048_9	89	**89**	0,00
	j9048_10	93	**93**	0,00
j120	j1201_1	105	127	17,32
	j1201_2	109	131	16,79
	j12030_9	93	101	7,92
	j12030_10	86	91	5,49
	j12060_9	101	**101**	0,00
	j12060_10	89	99	10,10

Pode-se verificar que para o conjuntos $j30$ e $j60$, em apenas um caso, na instância $j601_2$ o algoritmo não chegou no ótimo conhecido disponível na PSBLIB, apresentando assim para esses conjuntos 91.6% de assertividade.

Para o conjunto de instâncias j90, o algoritmo não foi capaz de alcançar os ótimos conhecidos para as instâncias $j901_1$ e $j901_2$, mas obteve uma diferença percentual (Δ) baixa. Nas demais instâncias do mesmo conjunto ele chegou no ótimo conhecido em todas, com aproximadamente 3 milissegundos, conforme pode ser observado na Tabela 4.

Para as instâncias j120, a maior encontrada na PSBLIB, o algoritmo não performou tão bem quanto nos demais conjuntos, encontrando o ótimo conhecido em apenas uma das instâncias, e em um tempo relativamente alto. Nas demais instâncias a diferença percentual ficou um pouco elevada, principalmente para as instâncias $j1201_1$ e $j1201_2$ que apresentaram Δ de $17,32\%$ e $16,79\%$, respectivamente.

Portanto, pode-se dizer que para resolução do RCPSP o algoritmo proposto apresentou resultados consistentes e satisfatórios para pequenas e médias instâncias. Já para instâncias de grande porte não obteve o mesmo êxito, sendo necessários estudos futuros visando o aprimoramento da busca local.

5 Conclusão

Neste trabalho é tratado o Problema de Programação de Projetos com Restrições de Recursos (RCPSP). Foi proposto um método que usa a meta-heurística GRASP, sendo o *Simulated Annealing* usado na etapa de busca local. O método proposto gerou resultados bons e consistentes, em tempos consideravelmente baixos, levando em conta a disponibilidade limitada de recursos e regras de precedência das tarefas.

Os resultados obtidos mostraram também a eficácia dessa abordagem para diferentes instâncias do problema. Porém, os resultados mostram que o método tem mais eficácia para instâncias de pequeno e médio porte apresentando uma eficiência de 83% em atingir os resultados apresentados na literatura para as instâncias das categorias $j30$, $j60$ e $j90$.

Como trabalho futuro, recomenda-se explorar outro método de busca local, como por exemplo o que é proposto por Goncharov (2022), com intuito de encontrar soluções ainda melhores, principalmente para as instâncias de grande porte.

Referências

ALMEIDA, B. F.; CORREIA, I.; GAMA, F. S. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, v. 57, p. 91–103, 2016.

DAMODARAN, P.; VÉLEZ-GALLEGO, M. C. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications*, v. 39, n. 1, p. 1451–1458, 2012.

DAMODARAN, P.; VÉLEZ-GALLEGO, M. C.; MAYA, J. A grasp approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, v. 22, n. 5, p. 767–777, 2011.

ELSAYED, S. et al. Consolidated optimization algorithm for resource-constrained project scheduling problems. *Information Sciences*, v. 418-419, p. 346–362, 2017.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, v. 8, n. 2, p. 67–71, 1989.

FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of global optimization*, v. 6, n. 2, p. 109–133, 1995.

GOLDRATT, E. M. *Corrente crítica*. São Paulo-SP: Editora Nobel, 1996.

GONCHAROV, E. N. A local search algorithm for the resource-constrained project scheduling problem. *Journal of Applied and Industrial Mathematics*, v. 16, n. 4, p. 672–683, 2022.

GONCHAROV, E. N. An improved genetic algorithm for the resource-constrained project scheduling problem. In: OLENEV, N. et al. (Ed.). *International Conference on Optimization and Applications*. Cham-China: Springer Nature Switzerland, 2023. p. 35–47.

HABIBI, F.; BARZINPOUR, F.; SADJADI, S. Resource-constrained project scheduling problem: review of past and recent developments. *Journal of Project Management*, v. 3, p. 55–88, 2018.

HAESER, G.; RUGGIERO, M. G. Aspectos teóricos de simulated annealing e um algoritmo duas fases em otimização global. *Trends in Computational and Applied Mathematics*, v. 9, n. 3, p. 395–404, 2008.

JAIN, K.; BILOLIKAR, V.; JOSHI, K. A VNS-GA-based hybrid metaheuristics for resource constrained project scheduling problem. *International Journal of Operational Research*, v. 27, p. 437, 2016.

KADAM, S. U.; KADAM, N. S. Solving resource-constrained project scheduling problem by genetic algorithm. In: *2014 2nd International Conference on Business and Information Management (ICBIM)*. Durgapur–India: IEEE, 2014. p. 159–164.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.

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

KOLISCH, R.; SPRECHER, A. PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. *European journal of operational research*, v. 96, n. 1, p. 205–216, 1997.

LIU, Y. et al. A late-mover genetic algorithm for resource-constrained project-scheduling problems. *Information Sciences*, v. 642, p. 119164, 2023.

MELO, L. V. *Modelos de programação linear inteira para variantes do problema de programação de projetos com restrição de recursos*. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Goiás, 2018.

ROCHA, W. S. *Algoritmo GRASP para o Problema de Tabela-horário de Universidades*. Dissertação (Mestrado em Informática) — Universidade Federal do Espírito Santo, 2013.

WANG, F.; ZHANG, H.; ZHOU, A. A particle swarm optimization algorithm for mixed-variable optimization problems. *Swarm and Evolutionary Computation*, v. 60, p. 100808, 2021.