UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE INFORMÁTICA

LADEMIR SILVA RODRIGUES JÚNIOR

PROJECT SCHEDULING PROBLEM NO CONTEXTO DA MONTAGEM DAS ESTRUTURAS EM EVENTOS

VIÇOSA - MINAS GERAIS
SETEMBRO - 2024

LADEMIR SILVA RODRIGUES JÚNIOR

PROJECT SCHEDULING PROBLEM NO CONTEXTO DA MONTAGEM DAS ESTRUTURAS EM EVENTOS

Projeto de pesquisa apresentado ao curso de Ciência da Computação da Universidade Federal de Viçosa como requisito parcial da disciplina Projeto Final de Curso.

Orientador: André Gustavo dos Santos

VIÇOSA - MINAS GERAIS

SETEMBRO - 2024

Resumo

O *Project Scheduling Problem* (PSP) é um desafio de otimização que visa determinar um cronograma ideal para a execução de um conjunto de atividades em um projeto. Para ilustrar esse problema, podemos considerar uma extensão chamada *Resource-Constrained Project Scheduling Problem* (RCPSP), que envolve a gestão de um único projeto, com cada atividade possuindo apenas uma forma de execução. Este trabalho propõe abordar uma generalização mais complexa conhecida como *Multi-Mode Resource-Constrained Scheduling Problem* (MRCPSP), que envolve a gestão de um projeto em que cada atividade pode ser executada de várias maneiras diferentes. Portanto, é necessário lidar com a diversidade de abordagens para a execução de atividades, além de gerenciar a alocação de recursos renováveis e não-renováveis de maneira eficiente. No contexto da montagem de estruturas de eventos, essa generalização do problema se torna particularmente relevante, uma vez que eventos podem exigir diferentes tipos de montagens. Cada uma dessas montagens é representada como uma atividade dentro do MRCPSP, e os recursos, como trabalhadores, peças e tempo, são limitados e precisam ser alocados de forma eficiente no planejamento de montagem. Isso é fundamental para garantir a entrega dentro dos prazos estipulados e minimizar os custos para a empresa responsável pelas estruturas de eventos. Assim, esse trabalho irá abordar técnicas baseadas em meta-heurísticas para propor um algoritmo capaz de resolver esse tipo de questão eficientemente, além de garantir um tempo de processamento factível para obter uma solução.

Palavras-chave: Meta-heurísticas, Múltiplos projetos, Múltiplos modos de execução, PSP, Otimização Combinatória, Montagem de eventos
Área do conhecimento: Ciência da Computação [1.03.00.00-7]
Linha de pesquisa: Algoritmos e Otimização Combinatória [DPI-040]

Abstract

The *Project Scheduling Problem* (PSP) is an optimization challenge aimed at determining an ideal schedule for executing a set of tasks within a project. To illustrate this problem, we can consider an extension called the *Resource-Constrained Project Scheduling Problem* (RCPSP), which involves managing a single project, with each task having only one way of execution. This paper proposes a more complex generalization known as the Multi-Mode Resource-Constrained Scheduling Problem (MRCPSP), which involves the management of a project in which each activity can be executed in several different ways. Therefore, it is necessary to deal with the diversity of approaches to task execution, in addition to managing the allocation renewable and non-renewable resource in a efficient way. In the context of event structure assembly, this generalization of the problem becomes particularly relevant since events may require different types of assemblies. Each of these assemblies is represented as a task within the MRCPSP, and resources such as workers, parts and time are limited and need to be allocated efficiently in the assembly planning. This is crucial to ensure delivery within the specified deadlines and minimize costs for the company responsible for event structures. Thus, this work will address metaheuristic-based techniques to propose an algorithm capable of efficiently solving this type of issue, while also ensuring a feasible processing time

Keywords: Metaheuristics, Multi-Project, Multi-Modes, PSP, Combinatorial Optimization, Event setup
Area of knowledge: Computer Science
Line of research: Algorithms and Combinatorial Optimization

Sumário

Lista de ilustrações

1 Introdução

O problema de programação de projetos desempenha um papel fundamental no contexto da montagem de estruturas para eventos, como feiras, shows, conferências e festivais. A eficiente organização e coordenação de todas as atividades envolvidas, desde alocação de equipes para uma estrutura específica até à disposição dos membros em cada unidade de montagem na construção dessas estruturas são de grande importância para garantir que a montagem aconteça dentro do prazo e, consequentemente, o evento ocorra sem problemas.

O *Project Scheduling Problem* (PSP) nesse contexto refere-se à atividade de definir o cronograma ideal para todas as etapas, desde o planejamento inicial até a desmontagem final. Isso inclui alocar recursos, determinar as dependências entre atividades e otimizar a utilização do tempo e dos recursos disponíveis. Encontrar a programação ideal é essencial para minimizar custos, evitar atrasos e garantir que a estrutura esteja pronta para receber os participantes do evento no prazo estipulado.

O PSP é um problema bastante conhecido na área de computação e pertence à classe NP-Difícil (BLAZEWICZ; LENSTRA; KAN, 1983), ou seja, não há algoritmo conhecido para resolver esse problema em tempo polinomial de maneira exata. O âmbito deste problema é vasto e não se limita apenas à montagem de eventos. Ele também se estende a projetos em diversas áreas, como a construção civil, onde o PSP pode ser aplicado para planejar e agendar todas as etapas de um projeto de construção, desde a preparação do terreno até a entrega final. Isso implica que qualquer projeto que envolva um conjunto de atividades a serem realizadas, com cada atividade tendo relações de precedência e requerendo um tempo específico para execução, pode ser generalizado e enquadrado nesse problema. O PSP é uma ferramenta valiosa para otimizar a gestão de projetos em uma ampla variedade de contextos.

Dessa forma, há certas variações do PSP para que esse problema se assemelhe a problemas do mundo real. Dentre elas, as que serão abordadas neste trabalho serão:

- RCPSP (*Resource-Constrained Project Scheduling Problem*): Cada etapa do projeto deve ser executada de acordo com um método específico, utilizando os recursos disponíveis. As atividades relacionadas ao projeto não têm variações na forma de execução, seguindo um único procedimento predefinido, mas os recursos limitam a execução simultânea de várias etapas.
- MRCPSP (*Multi-Mode Resource-Constrained Project Scheduling Problem*): Cada atividade tem diversas opções de execução, sendo possível designá-la para apenas

uma delas. Essas diferentes abordagens apresentam variações em termos de duração e exigências de recursos para a realização da atividade.

Vale ressaltar que, como o RCPSP é um problema que pertence à classe NP-Difícil e como o MRCPSP é apenas uma generalização do mesmo problema, consequentemente esta variação também pertencem a mesma classe de problemas.

Em resumo, o *Project Scheduling Problem* (PSP) e suas generalizações desempenham um papel fundamental na otimização de projetos em diferentes setores. Apesar de serem desafios computacionais complexos, sua resolução é essencial para a gestão eficiente de recursos e prazos. Este trabalho explorará essas variações em detalhes, abordando suas características distintas e estratégias de resolução e tendo um foco no contexto de montagem de estruturas para eventos.

2 Objetivos

2.1 Objetivo geral

O objetivo geral deste trabalho é propor uma heurística que seja capaz de obter resultados notáveis para o problema do PSP e suas variações, as quais foram apresentadas na seção anterior e são abordadas em diversos trabalhos acadêmicos. Desta maneira, será possível analisar um problema amplamente discutido na literatura e contribuir para a montagem de eventos, otimizando o uso de recursos e reduzindo o tempo de execução.

2.2 Objetivos específicos

Para que o objetivo geral deste trabalho seja atingido, os seguintes objetivos específicos devem ser atingidos:

- Desenvolver métodos baseados em meta-heurísticas para cada variação do problema PSP apresentada;
- Realizar uma análise comparativa entre as meta-heurísticas utilizadas, de modo a escolher a melhor para cada variação;
- Desenvolver um gerador de instâncias mais genéricas para o problema em questão;
- Aplicar a heurística desenvolvida a casos de estudo realísticos de montagem de eventos para verificar sua aplicabilidade e eficácia prática.

3 Revisão bibliográfica

Como esse trabalho tem a intenção de obter um método para obter soluções do *Multi-Mode Resource-Constrained Scheduling Problem* (MRCPSP), estudar variações mais complexas também é interessante para obter noções de caminhos que podem ser tomados, uma variação é a de múltiplos projetos e múltiplos modos (MRCMPSP). Essa generalização do PSP foi descrita na competição MISTA2013 (WAUTERS et al., 2016). Essa conferência resultou na publicação de diversas instâncias, um validador de soluções, um gerador de instâncias e também os melhores resultados dos times participantes.

Nessa mesma publicação, é exposto que foram utilizadas diversas técnicas baseadas em meta-heurísticas para obter as soluções - tanto *Single-Solution Based*, que são baseadas em vizinhança, quanto *Population Based*, que são baseadas em população. As soluções propostas pelas 9 melhores equipes estão descritos no artigo citado anteriormente.

A dissertação com título *Heurísticas Baseadas em Programação Inteira para o Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos* (SOARES, 2013) apresenta uma heurística fundamentada em programação inteira. Os autores dessa publicação foram os únicos dentre os vencedores da MISTA2013 que empregaram esse método de programação.

Além disso, há trabalho de conclusão de curso que também aborda o MMRCPSP intitulado *LAHC Aplicado ao Problema de Escalonamento de Múltiplos Projetos com Múltiplos Modos e Restrições de Recursos* (BALTAR, 2017) - orientado pela mesma autora da dissertação citada no parágrafo anterior. Eles propõem o uso do algoritmo *Late Acceptance Hill Climbing* (LAHC) para resolver esse problema. Essa técnica é uma adaptação do algoritmo que adiciona uma etapa de aceitação tardia, que permite explorar soluções que pioram o objetivo inicial, mas que podem ser promissoras.

Antes da definição da generalização mais abrangente do PSP, há muitos trabalhos que abordam as generalizações que foram usadas como base para criarem o MRCMPSP. O livro intitulado *Population-based approaches to the resource-constrained and discrete-continuous scheduling* (RATAJCZAK-ROPEL; SKAKOVSKI, 2018) trata de técnicas baseadas em população para solucionar o caso do RCPSP - a generalização mais simples do PSP - e o MRCPSP - extensão com vários projetos simultâneos e cada atividade com apenas um modo de execução.

Nesse viés, técnicas de algoritmo genético também foram aboradas na literatura. Especificamente para o RCMPSP, o trabalho com título de *A genetic algorithm for the resource constrained multi-project scheduling problem* apresentou uma solução que utiliza operadores genéticos, como mutação e cruzamento, para gerar novas soluções (GONÇAL-

VES; MENDES; RESENDE, 2008). Foram utilizadas instâncias criadas pelos próprios autores cujo processo de criação faz o uso de escolhas aleatórias de instâncias do tipo (*single-mode single-project*) - da PSPLIB (KOLISCH; SPRECHER, 1997) - que possuem 120 atividades para construir instâncias do tipo (*single-mode multi-project*).

Uma outra abordagem para obter uma solução da variação que contém múltiplos modos em cada atividade (MRCPSP) pode ser vista em (HARTMANN, 2001). Nesse trabalho é explicitado uma maneira interessante de avaliar uma solução, uma vez que dentro de uma população de soluções, existirão tanto indivíduos com solução factível quanto infactível. Tal maneira de avaliação considera o *makespan* como valor de avaliação para soluções viáveis e o tempo máximo de cada atividade somada a quantidade de recurso que é extrapolada como valor de avaliação para as infactíveis. Dessa maneira, é possível identificar as melhores soluções dentro das infactíveis mas qualquer factível será melhor que estas.

O trabalho citado no parágrafo anterior utiliza uma maneira de reduzir o espaço de busca de uma solução da variação com múltiplos modos (MRCPSP). Essa técnica é descrita em (SPRECHER; HARTMANN; DREXL, 1997) a qual consiste em realizar um pré-processamento para tentar desconsiderar algum modo e/ou algum recurso não renovável. Modos em que, por si só, já ultrapassam algum limite são desconsiderados e recursos não-renováveis em que a quantidade máxima utilizada deste recurso não ultrapassa o limite são considerados redundantes, uma vez que essa restrição não será desrespeitada independente das modos atribuídos às atividades.

Duas técnicas de construção de uma solução inicial bastantes difundidas na literatura relacionada ao PSP são descritas em (KOLISCH, 1996). A primeira abordagem consiste em uma construção de um cronograma de maneira serial, ou seja, possui uma sequência de execução de atividades vazia inicialmente e passa a alocar as atividades uma a uma considerando alguma regra de prioridade. No trabalho foi citada a regra de menor nó final num diagrama AOA (Activity-on-arrow) com desempate sendo a quantidade total de recurso utilizado pela atividade. A segunda é uma construção paralela em que aloca as atividades a partir da unidade de tempo atual, isto é, é alocado o máximo de atividades possível no tempo t desde que não desrespeite nenhuma restrição. A regra de prioridade citada para este método foi o de horário de término mais tardio da atividade.

Vale destacar a existência de uma biblioteca dedicada para o problema do PSP e suas extensões, a *Project Scheduling Problem Library* (PSPLIB) (KOLISCH; SPRECHER, 1997). É oferecido um conjunto abrangente de exemplos que abarcam as diferentes extensões citadas anteriormente. Essa coleção inclui uma ampla gama de instâncias com tamanhos variados que têm servido como modelos para a criação de instâncias subsequentes, inclusive aquelas empregadas na MISTA2013. Vale ressaltar que essas instâncias têm sido amplamente utilizadas na literatura acadêmica como em ((WAUTERS et al., 2016),

(CHEN et al., 2022), (BEŞIKCI; BILGE; ULUSOY, 2015), (RATAJCZAK-ROPEL; SKAKOVSKI, 2018), (MELO; QUEIROZ, 2021), dentre outros diversos trabalhos). Além disso, a PSPLIB também disponibiliza um gerador de instâncias que possibilita a criação de cenários inéditos para pesquisa e experimentação (KOLISCH; SPRECHER; DREXL, 1992).

Em trabalhos anteriores, como os de (BALTAR, 2017), (SOARES, 2013) e (KOLISCH; SPRECHER, 1997), destaca-se a utilização do Método do Caminho Crítico, conhecido como CPM (*Critical Path Method*). Este método tem a capacidade de determinar o tempo mínimo necessário para a conclusão de um projeto, levando em consideração exclusivamente as relações de precedência entre as atividades. Essa análise é conduzida por meio de uma ordenação topológica das atividades, classificando-as como críticas - aquelas cuja alteração de início impacta diretamente o término do projeto - ou não-críticas - atividades em que alterações não afetam o prazo final.

Adicionalmente, o Método do Caminho Crítico também fornece quatro características fundamentais para cada atividade: EST (*Earliest Start Time*), LST (*Latest Start Time*), EFT (*Earliest Finish Time*) e LFT (*Latest Finish Time*). Essas quatro variáveis permitem determinar o intervalo mais cedo que uma atividade pode iniciar e o mais tardar que pode ser concluída. Essas informações são essenciais para a gestão eficaz do projeto, proporcionando uma visão abrangente dos prazos e das possíveis folgas em cada atividade.

Por fim, também existe uma variação do Problema de Sequenciamento de Projetos com Restrições de Recursos e Precedência (MRCPSP) que incorpora a noção de precedência generalizada entre as atividades de um projeto. Em 1999, De Reyck e Herroelen apresentaram o algoritmo heurístico MMRCPS-GPR, que visa resolver esse desafio complexo (REYCK; HERROELEN, 1999). Este algoritmo heurístico se baseia na metaheurística *Tabu Search* e emprega uma variedade de técnicas de busca local para aprimorar a qualidade da solução.

4 O problema

4.1 Descrição

O problema estudado nesse trabalho - o Problema do Escalonamento de Projeto ou PSP (*Project Scheduling Problem*) - possui algumas variações conhecidas no campo da Ciência da Computação que vão desde lidar com um único projeto contendo várias atividades com apenas uma maneira de serem realizadas (*RCPSP*) até lidar com vários projetos e várias atividades com diversos modos de execução (*MRCMPSP*). As situações que podem ser generalizados para serem resolvidas como um PSP surgem quando há a necessidade de otimizar uma alocação de recursos para completar atividades dentro de um projeto atendendo às restrições específicas com o fito de diminuir o tempo total gasto para realizar todas atividades.

A figura 4.1 representa de maneira ilustrativa as distintas variações do problema, oferecendo uma visão abrangente de suas derivações. Para atender a situações realistas, esse trabalho irá abordar 4 extensões do PSP mas apenas as que são alvo deste trabalho serão descritas nas próximas seções.

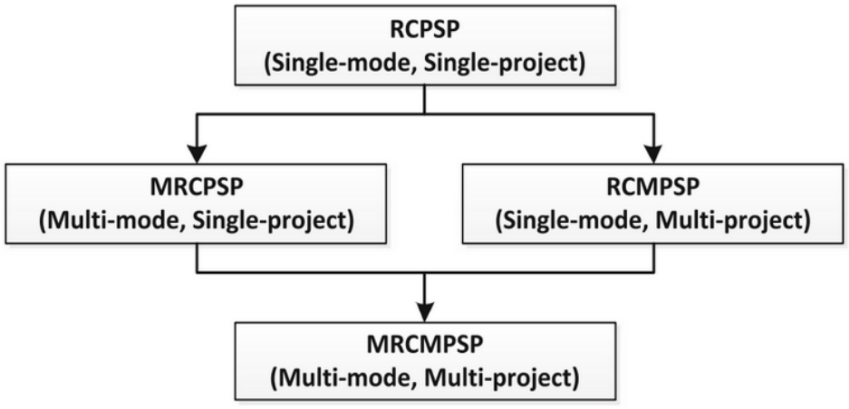

Figura 4.1 – Variações do PSP
Fonte: WAUTERS, Tony et al. The multi-mode resource-constrained multi-project scheduling problem: The MISTA 2013 challenge. (WAUTERS et al., 2016)

4.2 RCPSP (*Resource Constrained Project Scheduling Problem*)

Essa extensão compreende a mais simples das que serão abordadas nesse trabalho, em que há apenas um modo de execução de cada atividade e há apenas um projeto.

Existem duas restrições, são elas a relação de precedência entre as atividades - ou seja, há atividades que precisam acontecer após a realização de outras específicas - e a limitação de uso de recurso simultâneo. Essas restrições são detalhadas a seguir.

4.2.1 Tipos de restrições

- **Relação de precedência** refere-se à ordem ou prioridade estabelecida entre atividades, indicando qual deve ser executada antes de outra.
- **Uso de recurso** indica o limite de uso de cada recurso disponível no projeto, seja ele renovável (limite de uso simultâneo) ou não-renovável (limitado a uma quantidade para toda execução do projeto).

4.2.2 Recursos

Ademais, ainda é importante ter uma definição do que seja um recurso, quais seus tipos e características particulares.

- **Renováveis**: As quantidades desse recurso são renovadas a cada unidade de tempo, tendo seu limite apenas a quantidade de uso simultâneo.
- **Não-renováveis**: Há uma quantidade limitada de recurso que pode ser utilizada durante todo o projeto, portanto seu limite é apenas sua quantidade.

4.2.3 Definição

Nesta subseção, o problema será definido formalmente.

- **Projeto**: Conjunto J de n atividades para serem realizadas com o tempo máximo de T unidades de tempo.
- **Atividade**: Uma atividade i a ser realizada e que possui um tempo de duração d_i.
  - **Antecessores**: Cada atividade possui a_i antecessores e que pode somente ser iniciada quando todas atividades a_i forem concluídas.
  - **Não-preempção**: Atividades não podem ser interrompidas, portanto uma vez iniciadas elas continuam em execução até sua conclusão.
- **Origem e Sumidouro**: São duas atividades de controle. Possuem duração igual a 0 e a Origem não possui nenhum antecessor e Sumidouro não possui nenhum sucessor. Portanto, cada projeto possui $n + 2$ atividades para incluir essas duas atividades artificiais. Vale ressaltar que o tempo de conclusão do Sumidouro é o tempo total gasto para realizar todas atividades do projeto. Assim, o nó Origem

passa a ser o antecessor das atividades que inicialmente não tinham antecessores, e de maneira análoga, o nó Sumidouro torna-se o sucessor das atividades que não possuíam inicialmente sucessores.

- **Recursos**: Cada projeto possui um conjunto R de recursos renováveis e um conjunto N indicando a quantidade de cada recurso não-renovável.

4.2.4 Modelagem Matemática

A modelagem matemática do RCPSP com o tempo discretizado já foi explicitada em diversos trabalhos por se tratar de um problema conhecido no campo da pesquisa operacional e otimização, entretanto também será explicitado neste trabalho nas subseções abaixo. As modelagens apresentadas foram inspiradas nas presentes em (KOLISCH; SPRECHER, 1997) e (PRITSKER; WAITERS; WOLFE, 1969).

4.2.4.1 Constantes

- J: Conjunto de atividades.
- S: Quantidade total de atividades.
- T: Tempo máximo para conclusão de todas atividades presentes em J, que é a soma de todas as durações de execução das atividades presentes em J.
- D: Conjunto de duração de todas atividades.
- d_i: Tempo de duração da atividade i.
- a_i: Conjunto de antecessores da atividade i.
- R_k: Quantidade máxima de recurso renovável do recurso k.
- N_k: Quantidade do recurso não-renovável k.
- r_{ik}: Quantidade de recurso renovável k utilizado pela atividade i.
- n_{ik}: Quantidade de recurso não-renovável k utilizado pela atividade i

4.2.4.2 Variáveis

- $x_{it} \in \{0, 1\}$: indica se a atividade i está em execução no tempo t.
- $y_{it} \in \{0, 1\}$: indica se a atividade i inicia sua execução no tempo t.
- $c_{kt} \in \{0, R_k\}$: quantidade do recurso renovável k sendo utilizado no tempo t.
- $v_{kt} \in \{0, N_k\}$: quantidade do recurso não-renovável k sendo utilizado no tempo t.
- $y_{St} \in \{0, 1\}$: indica o tempo de conclusão da atividade tida como sumidouro.

4.2.4.3 Função objetivo

$$\min m_{max} = \sum_{t=0}^{T} y_{St} \cdot t \tag{4.1}$$

4.2.4.4 Restrições

$$\sum_{t=0}^{T} y_{it} = 1 \qquad \forall i \in J \tag{4.2}$$

$$y_{00} = 1 \tag{4.3}$$

$$\sum_{t=0}^{T} (t \cdot y_{it} + d_i) \leq \sum_{t=0}^{T} t \cdot y_{jt} \qquad i \in a_j, \forall j \in J \tag{4.4}$$

$$\sum_{t=0}^{T} x_{it} = d_i \qquad \forall i \in J \tag{4.5}$$

$$\sum_{t=0}^{T} \sum_{i \in J} r_{ik} \cdot x_{it} \leq R_k \qquad \forall i \in J, \forall k \in R \tag{4.6}$$

$$\sum_{t=0}^{T} n_{ik} \cdot x_{it} \leq N_k \qquad \forall i \in J, \forall k \in N \tag{4.7}$$

A função objetivo (4.1) foi construída de maneira a obter o melhor cronograma possível para o projeto, ou seja, com o fito de minimizar o *makespan* que é o tempo de conclusão da atividade tida como sumidouro. A restrição (4.2) garante que todas as atividades foram executadas apenas uma vez e também - consequentemente - indica que a atividade J_i foi executada sem preempção uma vez que a soma da quantidade de vezes em que foi iniciada é igual a 1. A restrição (4.3) garante que a atividade que indica o início do projeto comece no tempo 0. Para garantir a relação de precedência entre as atividades foi construída a restrição (4.4) que garante que o tempo de início de uma atividade somada à sua duração seja menor que o tempo de início da atividade sucessora a esta. A restrição (4.5) torna obrigatório que o somatórios das variáveis que indicam quando uma atividade i está em execução na unidade de tempo seja igual à duração da atividade em questão, o que garante que a atividade constará como em execução durante todo seu tempo de processamento. As restrições (4.6) e (4.7) indicam que as quantidades de recursos não podem ser extrapoladas de seus limites, tanto a quantidade de uso simultâneo de recursos renováveis quanto a quantidade total de uso de um recurso não-renovável.

4.3 MMRCPSP (*Multi-Mode Resource Constrained Project Scheduling Problem*)

Outra extensão do PSP que será abordada neste trabalho será a MMRCPSP, em que se adiciona uma dificuldade a mais na sua resolução e que, entretanto, tem uma proximidade maior com situações do mundo real do que a versão abordada na seção anterior. Como dito em seu título, esta extensão permite que cada atividade possua vários modos de execução. Podemos pensar - no contexto de montagem de estruturas metálicas em eventos - que um modo de execução de uma atividade específica utilize 5 funcionários e dure 3 horas, entretanto se for utilizado um conjunto de 8 funcionários a atividade possa ser concluída em 1 hora. Para provar a complexidade do problema, caso cada atividade possua apenas um modo de execução, o problema se torna o RCPSP que já é NP-Difícil (BLAZEWICZ; LENSTRA; KAN, 1983).

Algumas alterações foram implementadas em relação à extensão anterior devido à mudança nos modos de execução. Essas modificações serão abordadas em seguida.

4.3.1 Tipos de restrições

- **Unicidade de modo de execução** indica que cada atividade só pode ser executada de apenas um modo, mesmo tendo m modos de execução.

4.3.2 Definição

A parte da definição do MMRCPSP se extende à do RCPSP em alguns pontos - serão explicitadas apenas o que é diferente - que estão denotados abaixo:

- **Atividade**: Uma atividade i a ser realizada, possui m_i modos de execução.
  - **Modo de execução**: Indica uma maneira de executar a atividade i que faz o uso de uma quantidade específica de recursos e tem um determinado tempo de duração, sendo que ambos podem ser diferentes entre cada modo.

4.3.3 Modelagem Matemática

De forma semelhante à seção anterior, será explicitado abaixo apenas o que difere em relação à modelagem da extensão do PSP abordada anteriormente.

4.3.3.1 Constantes

- T: Tempo máximo para conclusão de todas atividades presentes em J, que é a soma de todas as durações de execução das atividades presentes em J, porém leva apenas

em consideração a maior duração entre os modos de execução.

- M_i: Conjunto de modos de execução da atividade i; no caso das atividades artificiais Origem e Sumidouro, há apenas um modo.
- d_{im}: Tempo de duração da atividade i sendo executada no modo m.
- r_{ikm}: Quantidade de recurso renovável k utilizado pela atividade i quando executada no modo m.
- n_{ikm}: Quantidade de recurso não-renovável k utilizado pela atividade i quando executada no modo m.

4.3.3.2 Variáveis

- $x_{itm} \in \{0, 1\}$: Indica se a atividade i está em execução no modo m no tempo t.
- $y_{itm} \in \{0, 1\}$: Indica se a atividade i inicia sua execução no modo m no tempo t.
- $y_{Stm} \in \{0, 1\}$: Indica o tempo de conclusão da atividade tida como sumidouro no modo m.

4.3.3.3 Função objetivo

$$\min m_{max} = \sum_{t=0}^{T} t \cdot y_{St0} \tag{4.8}$$

4.3.3.4 Restrições

$$\sum_{m \in M_i} \sum_{t=0}^{T} y_{itm} = 1 \qquad \forall i \in J \tag{4.9}$$

$$\sum_{m \in M_i} \sum_{t=0}^{T} (t \cdot y_{itm} + d_{im}) \leq \sum_{m \in M_j} \sum_{t=0}^{T} t \cdot y_{jtm} \qquad \forall j \in J, i \in a_j \tag{4.10}$$

$$\sum_{m \in M_i} \sum_{t=0}^{T} \sum_{i \in J} r_{ikm} \cdot x_{itm} \leq R_k \qquad \forall k \in R \tag{4.11}$$

$$\sum_{m \in M_i} \sum_{t=0}^{T} n_{ikm} \cdot y_{itm} \leq N_k \qquad \forall k \in N \tag{4.12}$$

$$\sum_{t=0}^{T} x_{itm} = \sum_{t=0}^{T} y_{itm} \cdot d_{im} \qquad \forall i \in J, \forall m \in M_i \tag{4.13}$$

As mudanças se referem a adicionar múltiplos modos de execução em cada atividade. Diante disso, a função objetivo (4.8) segue com o mesmo fito da função objetivo

do RCPSP (4.1), o de minimizar o *makespan* do projeto. A restrição (4.9) foi construída para garantir que uma atividade é executada apenas uma vez e de apenas uma maneira. A restrição (4.10) é uma extensão da (4.4) para poder abranger a duração de múltiplos modo de execução e lidar da maneira correta a relação de precedência entre atividades. As restrições (4.11) e (4.12) foram modificadas de suas versões *single-mode* para poder lidar com os múltiplos modos de execução de uma atividade, uma vez que cada modo possui uma quantidade de recurso diferente a ser utilizado. A última restrição (4.13) garante que tanto x_{itm} quanto y_{itm} estarão valendo 1 para o mesmo modo de execução , uma vez que a primeira indica se atividade i está em execução no tempo t no modo m, consequentemente seu somatório resulta em d_{im}.

4.4 Criação de instâncias

Embora a PSPLIB forneça um gerador de instâncias, este não recebeu atualizações para o sistemas operacionais atuais e, portanto, não está sendo mais utilizado. Com isso, um novo gerador de instâncias foi criado baseado no processo descrito em (KOLISCH; SPRECHER; DREXL, 1992) com o objetivo de ter casos de testes específicos para as heurísticas propostas.

As instâncias do MRCPSP presentes na PSPLIB possuem todas as mesmas quantidades de modos diferentes para cada atividade. Isso é um fato que as tornam menos realísticas quanto se trata do contexto de montagem de estruturas de eventos. Portanto, um dos objetivos principais para fazer um gerador de instâncias é que as atividades não possuam necessariamente a mesma quantidade de modos.

Outro fator importante é a quantidade de recursos diferentes. As instâncias presentes na biblioteca possuem, em sua maioria, 2 recursos renováveis e 2 recursos não-renováveis, o que contribui para um padrão na literatura mas, em contrapartida, diminui a similaridade com situações reais.

Um ponto importante desse gerador de instâncias é que será utilizado uma seed para os números pseudo-aleatórios escolhidos, para que seja facilitada a replicação das instâncias.

4.4.1 Etapas de criação

Antes da criação das tarefas, suas relações de precedência, seus modos e suas quantidade de uso de cada tipo de recurso, é necessário definir algumas constantes inicialmente.

- J: Quantidade de atividades.
- D_{min}: Fator mínimo da duração de uma atividade

- D_{max}: Fator máximo da duração de uma atividade
- R_{min}: Quantidade mínima de recursos diferentes
- R_{max}: Quantidade máxima de recursos diferentes
- K_{min}: Fator mínimo da demanda de recurso
- K_{max}: Fator máximo da demanda de recurso
- M_{min}: Quantidade mínima de modos por tarefa
- M_{max}: Quantidade máxima de modos por tarefa
- S_{max}: Quantidade máxima de sucessores por atividade
- $RF \in [0, 1]$: Fator que influencia na quantidade de tipos diferentes de recursos utilizados por um modo

Para fins de notação, é interessante definir algumas funções:

- $rand[a, b]$: Produz um número inteiro aleatório x tal que $x \in [a, b]$
- $round(a)$: Resulta na aproximação de a para o inteiro mais próximo.

A partir da definição das constantes, podemos definir a quantidade de recursos renováveis e não-renováveis que o projeto terá a partir das quantidades mínima e máxima de recursos diferentes. Dessa maneira, obtemos:

- r: $rand[R_{min}, R_{max}]$.
- n: $rand[R_{min}, R_{max}]$.

Sendo r e n referentes às quantidades de recurso renovável e não-renovável, respectivamente.

4.4.1.1 Geração das atividades

Definidas as constantes e as quantidade de recursos, agora é necessário a geração das atividades propriamente ditas.

É importante mencionar que para manter a similaridade com situações reais, pode-se observar que, no algoritmo 1, há uma proporcionalidade inversa entre a duração de um modo e a quantidade de recurso utilizado. Dessa forma, quanto menor o tempo de duração, maior será a quantidade de recurso utilizado pelo mesmo.

Algoritmo 1 Geração de atividades

```
1: procedure GERARATIVIDADES
2:     atividades ← lista vazia de tamanho J + 2
3:     atividades[0] ← atividade artificial de origem
4:     atividades[J + 1] ← atividade artificial de sumidouro
5:     for i ← 1 to J do
6:         qtdeModos ← rand[M_min, M_max]
7:         modos ← lista vazia de tamanho qtdeModos
8:         for j ← 0 to qtdeModos do
9:             duracao ← rand[D_min, D_max]
10:            fatorUso ← duracao / D_max
11:            usoRecursoRenovavel ← lista vazia
12:            for k ← 0 to r do
13:                recurso ← 0
14:                if rand[0, 1] < RF then
15:                    recurso ← rand[K_min, K_max] / fatorUso
16:                end if
17:                usoRecursoRenovavel ∪ recurso
18:            end for
19:            usoRecursoNaoRenovavel ← lista vazia
20:            for k ← 0 to n do
21:                recurso ← 0
22:                if rand[0, 1] < RF then
23:                    recurso ← rand[K_min, K_max] / fatorUso
24:                end if
25:                usoRecursoNaoRenovavel ∪ recurso
26:            end for
27:            modo ← (j, duracao, usoRecursoRenovavel, usoRecursoNaoRenovavel)
28:            modos ∪ modo
29:        end for
30:        atividade[i] ← (i, modos)
31:    end for
32: end procedure
```

J	D_{min}	D_{max}	R_{min}	R_{max}	K_{min}	K_{max}	M_{min}	M_{max}	S_{max}	RF
10	10	120	2	5	10	100	1	4	3	0.5

Tabela 4.1 – Parâmetros utilizados para as atividades da figura 4.2

Para fins de exemplificar o processo descrito acima, iremos utilizar os valores das contantes definidos na tabela 4.1 para criar um conjunto de atividades.

Na figura 4.2 podemos observar um exemplo de resultado obtido através do processo descrito anteriormente. Portanto, a segunda etapa do processo da criação de uma instância está concluído.

Tarefa	Modo	Duracao	RR1	RR2	RR3	RR4	RR5	RN1	RN2	RN3
0	0	0	0	0	0	0	0	0	0	0
1	0	96	80	92	15	0	0	18	0	112
1	1	112	75	100	81	0	0	0	0	98
1	2	87	93	0	0	0	0	0	74	0
2	0	52	0	166	126	0	0	221	90	0
2	1	68	169	0	0	0	79	70	0	0
2	2	36	0	0	263	246	0	290	36	0
2	3	34	0	0	0	0	162	0	0	218
3	0	90	0	133	0	0	110	0	0	0
4	0	40	0	0	0	0	0	0	33	207
5	0	87	34	0	0	0	0	0	68	75
5	1	37	0	55	0	262	223	0	155	0
6	0	97	0	19	0	120	0	61	115	30
6	1	100	110	30	0	39	69	0	0	18
6	2	63	76	127	0	137	0	188	36	43
6	3	86	97	0	0	0	0	50	30	131
7	0	33	72	356	83	0	254	43	116	305
8	0	69	0	40	0	113	0	0	0	43
8	1	76	66	48	0	15	0	94	48	26
8	2	44	0	0	128	0	0	253	43	147
8	3	73	134	57	116	0	0	0	128	101
9	0	44	201	114	90	27	0	0	0	0
9	1	84	0	105	0	0	0	0	138	137
9	2	55	200	63	128	0	30	109	0	58
9	3	65	0	0	0	46	0	112	0	0
10	0	106	0	0	100	0	0	60	37	0
11	0	0	0	0	0	0	0	0	0	0

Figura 4.2 – Atividades geradas como exemplo pelo processo acima com seed 23485729.
Fonte: próprio autor

4.4.1.2 Geração de relações de precedência

Com as atividades criadas, seus modos, durações e quantidade de recurso solicitada, ainda é necessário construir as relações de precedência entre as mesmas para obter a rede do projeto. O próximo passo é definir as relações de precedência entre as atividades.

Para isso, vamos definir algumas constantes que serão necessárias para criação das dependências.

- S_{1min}: Quantidade mínima de atividades iniciais
- S_{1max}: Quantidade máxima de atividades iniciais
- F_{Jmin}: Quantidade mínima de atividades finais
- F_{Jmax}: Quantidade máxima de atividades finais
- S_a: Conjunto de atividades iniciais de tamanho $rand[S_{1min}, S_{1max}]$
- F_a: Conjunto de atividades finais de tamanho $rand[F_{Jmin}, F_{Jmax}]$

Também é necessário citar algumas definições utilizadas em (KOLISCH; SPRECHER; DREXL, 1992).

- Aresta redundante: Aresta (i, j) em que há arestas (i_0, i_1),..., (i_{s-1}, i_s) com $i_0 = i$ e $i_s = j$ com $s \geq 2$.
- Complexidade da rede (c): A média de arestas não redundantes por vértice.
- A_{min}: Quantidade mínima de arestas não redundantes.
- A_{max}: Quantidade máxima de arestas não redundantes.

Aresta redundante também pode ser entendida como: se há outro caminho partindo do vértice i até o vértice j, então a aresta (i, j) é redundante. Para um grafo com $n \geq 6$ vértices, temos que $A_{min} = n - 1$ e que $A_{max} = n - 2 + (\frac{n-2}{2})^2$ para n sendo par e $A_{max} = n - 2 + (\frac{n-1}{2})(\frac{n-3}{2})$ para n sendo ímpar.

O conceito de complexidade, introduzido por (PASCOE, 1966), é uma medida em que mede o quão densa é a rede. É mostrado em (KOLISCH; SPRECHER; DREXL, 1992) que, para um número alto de complexidade, menos difícil de resolver a instância se torna. Portanto, é interessante que, para instâncias mais realísticas, esse número não seja alto.

Com as constantes definidas, os passos para criação das relações de precedência serão descritos:

Passo 1

São definido o conjunto de atividades iniciais e finais, ou seja, as atividades que serão as sucessores da atividade artificial de origem e predecessoras da atividade artificial sumidouro, respectivamente.

Passo 2

Para cada uma das atividades tidas como não iniciais, ou seja, atividades do conjunto $A = \{x | x \in J - S_a\}$, é escolhido um predecessor, de maneira aleatória, do conjunto de não finais, ou seja, $A' = \{x | x \in J - F_a\}$. São escolhidos apenas para essas atividades porque as inicias tem como predecessor apenas a atividade inicial artificial.

Passo 3

Para cada atividade que não possui um sucessor, é atribuído um sucessor aleatório. Note que nesse passo e no anterior, nenhuma aresta redundante é adicionada.

Passo 4

São atribuídos arcos de maneira aleatória, contanto que não sejam escolhidos mais predecessores para as atividades iniciais nem para a atividade sumidouro. De mesma maneira não podem ser escolhidos mais sucessores para as atividades finais. Esse processo se permanece até a complexidade desejada ou a quantidade de sucessores máxima por atividade ser atingida.

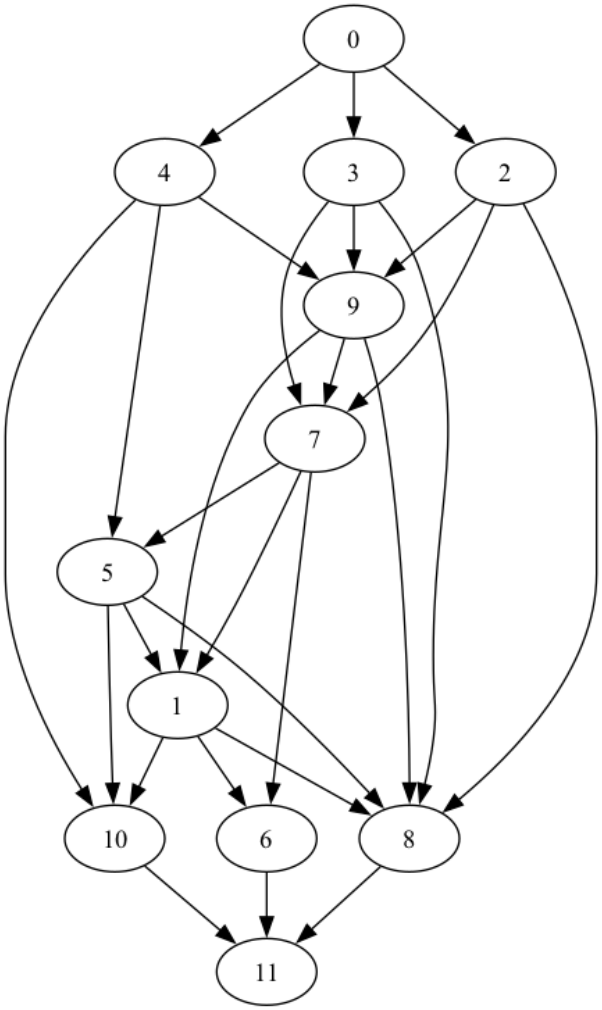

Figura 4.3 – Grafo de precedência de atividades com seed 23485729.
Fonte: próprio autor

Em cada passo desse processo, é necessário em que não sejam atribuídos arestas que criem ciclos, uma vez que o grafo de atividades são DAGs.

Na figura 4.3, é possível observar as relações de precedência criadas a partir do processo descrito. Pode-se observar que a quantidade de sucessores máximo foi atingido.

4.4.1.3 Geração das quantidades de recursos disponíveis

Por fim, a última etapa na criação de uma instância é a geração das disponibilidades de recursos renováveis e não renováveis.

Para isso, mais uma definição será utilizada para essa etapa.

- $RS \in [0, 1]$: Fator para quantidade de recurso.
- Q_r^{min}: Quantidade mínima requisitada pelo recurso r.

- Q_r^{max}: Quantidade máxima requisitada pelo recurso r.
- Q_r: Quantidade disponível do recurso r.

Dessa maneira, vamos usar RS como um fator de escala para definir a quantidade de disponibilidade de recurso, de maneira em que $Q_r = Q_r^{min} + RS(Q_r^{max} - Q_r^{min})$. Entretanto há maneiras diferentes de calcular as quantidade mínimas e máximas de cada tipo de recurso.

Recursos não renováveis

$$Q_r^{min} = \sum_{j=1}^{J} \min_{m=1...M_j} \{n_{jrm}\} \qquad (4.14) \qquad\qquad Q_r^{max} = \sum_{j=1}^{J} \max_{m=1...M_j} \{n_{jrm}\} \qquad (4.15)$$

Para calcular Q_r dos recursos não renováveis temos a equação mostrada anteriormente mas também é necessário garantir que essa quantidade seja um número inteiro, portanto, temos $Q_r = Q_r^{min} + round(RS(Q_r^{max} - Q_r^{min}))$ com $r \in N$.

Recursos renováveis

$$Q_r^{min} = \max_{j=1}^{J} \{min_{m=1...M_j} \{r_{jrm}\}\} \qquad (4.16)$$

O valor de Q_r^{min} é o mínimo necessário para que possamos executar qualquer tarefa em qualquer modo individualmente, por isso que é necessário obter o máximo dentre as quantidades mínimas de cada modo.

O valor de Q_r^{max} é obtido através da pico máximo da requisição do recurso r. Isto é obtido através a obtenção de um cronograma de atividades considerando apenas os modos que requisitam a quantidade máxima do recurso r. Para obter esse planejamento artificial, é necessário atender a restrição das relações de precedência, desconsiderando o limite da quantidade de recurso - uma vez que é este que estamos buscando.

Dessa maneira, as etapas para obtenção desse cronograma temporário são:

Passo 1

Obter a ordenação topológica dessas atividades.

Passo 2

Através do algoritmo 2 para cada recurso renovável, é possível obter a quantidade de requisição do recurso r a cada unidade de tempo. Dessa maneira, Q_r^{max} é o máximo dentre as quantidades de requisições do recurso r por tempo.

Com isso, podemos calcular a quantidade de disponibilidade do recurso r uma vez que obtemos as quantidades máximas e mínimas. Temos que $Q_r = Q_r^{min} + RS(Q_r^{max} - Q_r^{min})$ com $r \in R$.

Algoritmo 2 Cálculo de utilização de recursos renováveis

```
1: procedure UTILIZACAORECURSO(ordenação topológica)
2:     utilizacao_recurso ← Lista vazia de tamanho T
3:     for cada atividade em ordenacao_topologica do
4:         qtde_recurso ← Quantidade maximal do recurso r
5:         tempo_inicio ← Maior tempo de término das atividades antecessoras
6:         for t ← tempo_inicio to tempo_inicio+duracao do modo do
7:             utilizacao_de_recurso[t] ← utilizacao_de_recurso[t] + qtde_recurso
8:         end for
9:     end for
10: end procedure
```

4.4.1.4 Finalização

Com isso, temos a instância pronta, uma vez que todos os dados necessários foram obtidos através das etapas descritas anteriormente.

Vale ressaltar que com $RS = 1$ temos instâncias mais fáceis de serem resolvidas, uma vez que todos os recursos não renováveis serão redundantes e as quantidades limite dos renováveis nunca serão ultrapassadas. Conforme é mostrado em (KOLISCH; SPRECHER; DREXL, 1992), quanto mais RS se aproxima de 1, mais rápida a instância é resolvida.

Portanto, um valor intermediário é interessante para obter instâncias mais similares com a realidade.

5 Metodologia

Neste capítulo serão abordados a maneira como a solução é representada dentro da nossa modelagem, como é sua construção inicial e métodos de busca que foram aplicados a esse problema.

5.1 Representação da solução

A solução é representada como um array de números inteiros em que a posição i indica o tempo em que a atividade i foi iniciada. Dessa maneira, tal estrutura pode ser exemplificada abaixo.

$$\text{Sol} = [0, 4, 0, 0, 10, 23, 4, 4, 13, 6, 12, \dots] \tag{5.1}$$

Portanto, pode-se observar que a atividade 0 (Origem) tem seu início no tempo 0, a atividade 1 tem seu início no tempo 4 e assim por diante. Vale ressaltar que várias atividades podem iniciar ao mesmo tempo desde que não violem as restrições.

Para as variações que envolvem múltiplos modos de execução, para complementar a solução, há um segundo array o qual indica o modo m que está sendo executado pela atividade i, que pode ser exemplificado por 5.2.

$$\text{Sol}_m = [0, 2, 0, 3, 3, 2, 0, 1, 2, 1, 0, \dots] \tag{5.2}$$

5.2 Avaliação da solução

Cada elemento do espaço de busca é associado a um valor que indica sua qualidade, esse valor permite que possam ser feitas comparações entre soluções para saber o quanto uma é melhor que outra. Para obter esse valor é criada uma **função de avaliação** que analisa uma solução e faz o mapeamento com esse valor que ela representa.

Para esse problema, a função de avaliação é bastante simples pelo fato de existir a atividade artificial tida como sumidouro que marca o término de todo o projeto. Diante disso, para obter um valor associado a solução, basta buscar esse valor

$$\text{Sol} = [\dots, 56, 58, 50, 17, 35, 50, 36, 61, 53, 63] \tag{5.3}$$

Nesse exemplo, pode-se observar que a avaliação dessa solução exemplificada acima tem o valor de 63.

5.3 Construção inicial

Embora não seja uma garantia da heurística convergir rapidamente, é interessante ter uma solução inicial válida para facilitar o processo de busca dentro da vizinhança em questão. Vale lembrar que, para ser uma solução válida, é preciso atender às restrições tanto de ordem de precedência quanto de limitação de uso de recursos.

5.3.1 Instâncias de modo único

As atividades podem ser representadas por um DAG (Directed Acyclic Graph), possibilitando a modelagem das relações de precedência de maneira eficiente para consultas rápidas e a obtenção de uma ordenação topológica entre as atividades. A Figura 5.1 apresenta o DAG da instância `j301_1.sm` da PSPLIB (KOLISCH; SPRECHER, 1997), composta por 30 atividades, cada uma com apenas um modo de execução. Os nós 0 e 31 representam as atividades artificiais Origem e Sumidouro, respectivamente.

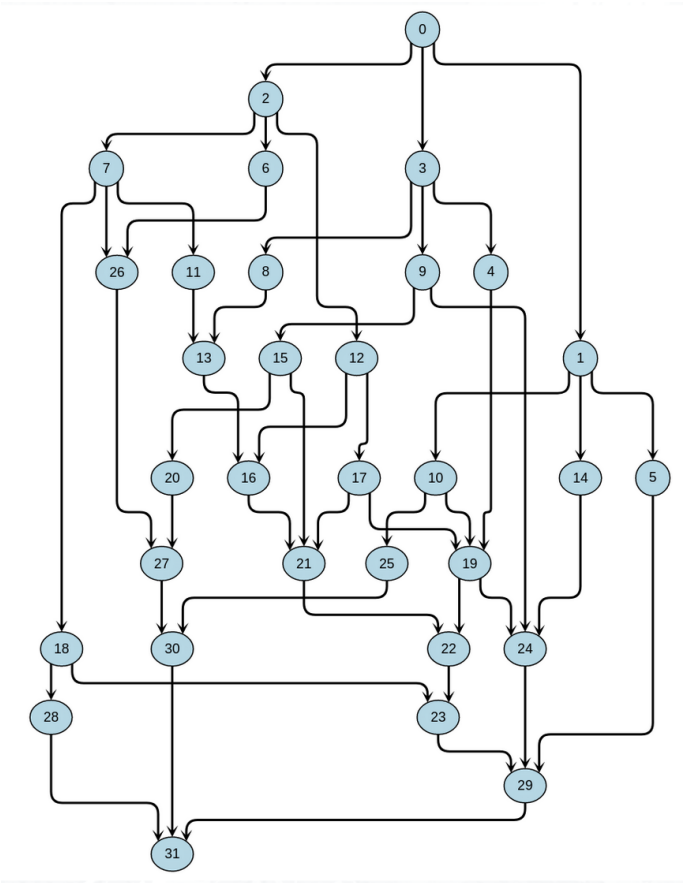

Figura 5.1 – Exemplo da instância j301_1.sm
Fonte: próprio autor

A partir do grafo presente na figura 5.1, é obtida uma ordenação topológica das atividades para auxiliar a construção. Agora é necessário alocar as atividades na ordem obtida pelo processo anterior, contudo é preciso respeitar a restrição de uso de recurso simultâneo para que a solução seja factível.

Seguindo o explicitado, uma atividade é adicionada na solução quando ela está liberada para começar - quando todas atividades precedentes estiverem sido concluídas e há quantidade de recursos suficientes para que seja iniciada e finalizada.

$$\begin{aligned}\text{Sol} = [&0, 0, 18, 0, 6, 8, 22, 22, 8, 6, 8, 31, 22, 33, 8, 16, 43, 29, 31, 43, 26, 49, 56,\\ &58, 50, 17, 35, 50, 36, 61, 53, 63] \quad (5.4)\end{aligned}$$

Seguindo o processo que foi descrito anteriormente, a solução inicial obtida para a instância `j301_1.sm` da PSPLIB (KOLISCH; SPRECHER, 1997) é a representada na sequência (5.4), que pode ser melhor observada no diagrama de Gantt ilustrado na figura 5.2.

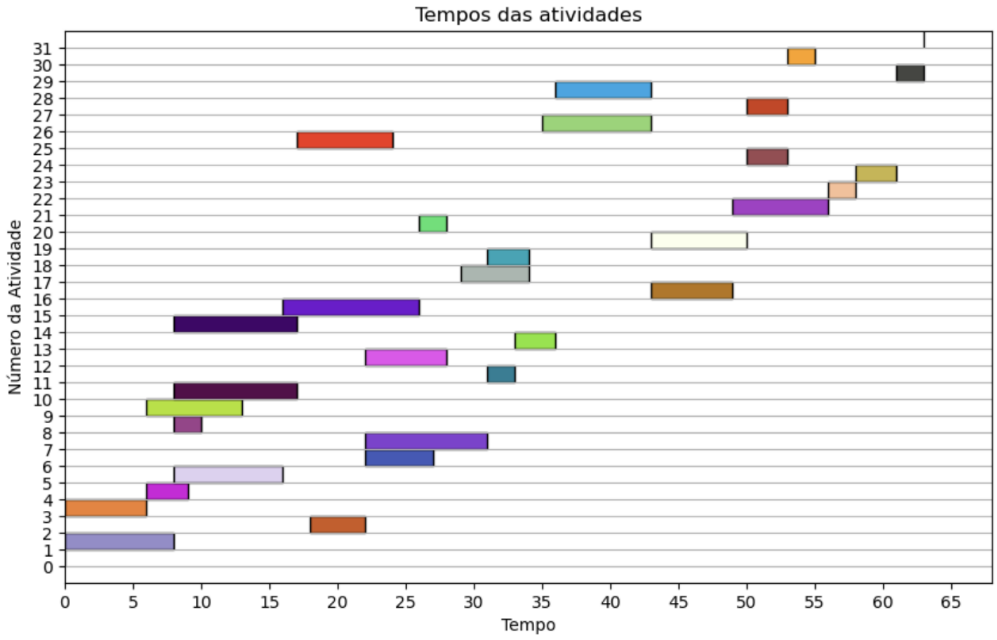

Figura 5.2 – Solução inicial obtida para a instância j301_1.sm
Fonte: próprio autor

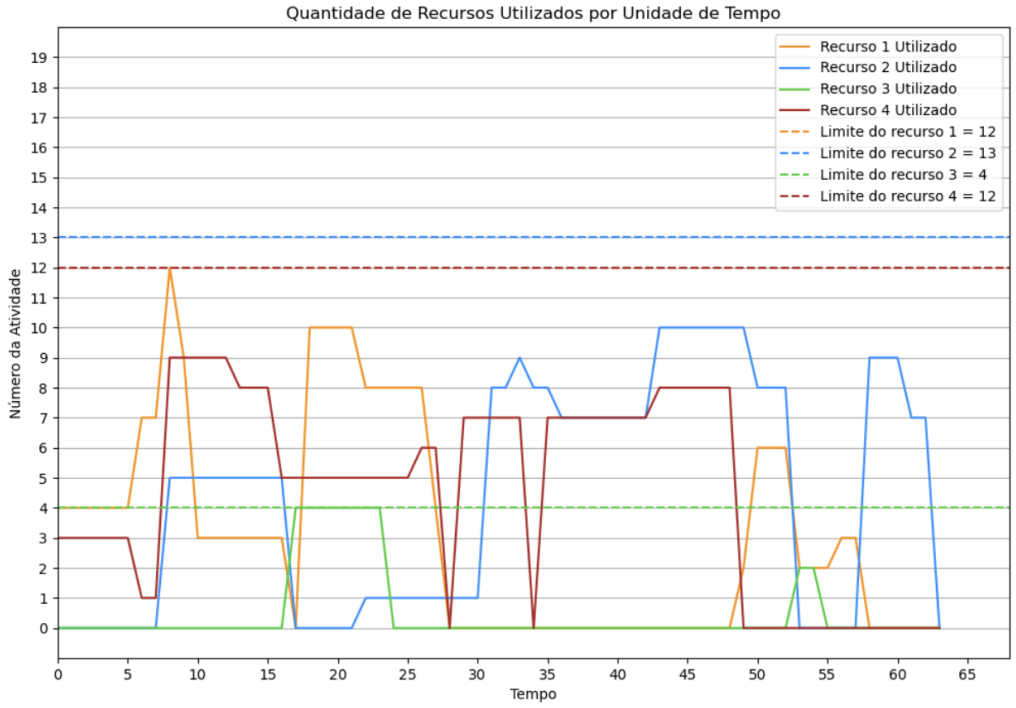

Figura 5.3 – Uso de recurso da solução inicial obtida para a instância j301_1.sm
Fonte: próprio autor

Na figura 5.2 é possível observar que há a ocorrência de atividades ao mesmo tempo, ou seja, foi possível alocar atividades simultaneamente de maneira que tanto a quantidade de uso de recurso quanto as relações de precedência foram respeitadas. Na figura 5.3 pode-se observar em detalhes como foi a quantidade de uso dos recursos durante toda a extensão da solução; Pontos interessantes desse gráfico são onde a quantidade de recurso utilizado chega ao máximo possível, como aconteceu com o recurso do tipo 1 (laranja) e o recurso do tipo 3 (verde), respectivamente nos tempos 8 e 17-23.

5.3.2 Instâncias de múltiplos modos

Nesse tipo de instância, há uma complicação maior para se criar a solução inicial, devido a cada atividade possuir várias formas de serem performadas. Como mostrado em (KOLISCH; DREXL, 1997), apenas encontrar uma solução factível para esta variação do problema é NP-completo quando há pelo menos 2 recursos não renováveis pois o autor demonstra que cada instância do problema *knapsack* pode ser polinomialmente transformado em uma instância do MRCPSP.

Dessa maneira, é interessante tentar reduzir o espaço de busca de instâncias que possuem esse tipo de característica. Um procedimento de redução pode ser visto em

(SPRECHER; HARTMANN; DREXL, 1997) para acelerar um algoritmo de *branch-and-bound* para o MRCPSP.

5.3.2.1 Pré-processamento

Para realizar essas operações, (SPRECHER; HARTMANN; DREXL, 1997) define alguns conceitos que são:

- Q_{jr}^{min}: Quantidade mínima em que a atividade j requisita r.
- Q_{jr}^{max}: Quantidade máxima em que a atividade j requisita r.
- Modo não-executável m_j: $\exists r \in R$ tal que $r_{ir} > R_r$
  ou $\sum_{i=1, j \neq i}^{J} Q_{ir}^{min} + n_{jrm_j} > N_r$
- Modo ineficiente m_j: Caso haja outro modo m_j^* que tenha duração menor ou igual a m_j e que as quantidade requisitadas de todos recursos sejam menores ou iguais.
- Recurso redundante $r \in N$: Um recurso r é tido como redundante quando $\sum_{j=1}^{J} Q_{jr}^{max} \leq N_r$.

Uma outra maneira de entender a definição de um modo não-executável: m_j é aquele em que, por si só, já ultrapasse o limite de algum recurso renovável ou que a soma minimal de um recurso não renovável r somada a quantidade requisitada de r por m_j ultrapasse N_r.

Com isso, as passos para realizar o pré-processamento são:

Passo 1

Remover todos os modos não executáveis, analisando tanto a questão dos recursos renováveis quanto a dos não renováveis.

Passo 2

Remover todos os recursos não renováveis redundantes.

Passo 3

Remover todos os modos ineficientes.

Passo 4

Se algum modo for removido no passo 3, voltar ao passo 2.

Pode-se observar que remoções feitas em qualquer passo do pré-processamento não alteram o melhor makespan possível para a instância, apenas reduz o espaço de busca. Com isso, a formação de uma solução inicial viável se torna menos complexa.

5.3.2.2 Criação do cronograma inicial

Com isso, agora é necessário a construção de fato de um cronograma inicial. Utilizando conceitos e técnicas abordadas em (HARTMANN, 2001) e em (KOLISCH, 1996), podemos fazer uma construção da solução inicial de maneira serial, ou seja, a partir de cada tarefa, verificar em qual unidade de tempo pode ser alocada.

Vale ressaltar que a construção de uma solução inicial factível para o MRCPSP que contenha mais de 2 recursos não renováveis é um problema NP-completo. Portanto, a primeira solução precisa atender apenas as restrições de precedência e quantidade de recursos renováveis.

Assim, os passos para criação de um cronograma inicial são:

Passo 1

Para a atividade j, atribuir um modo m aleatório tal que $m \in M_j$.

Passo 2

Verificar se a restrição referente às quantidades de recursos não renováveis estão sendo respeitadas, caso não, realiza uma busca local simples.

Escolhe aleatoriamente uma atividade $j \in J$ que tenha mais de um modo disponível. Escolhe aleatoriamente um outro modo diferente para j e que seja diferente do modo atual. Esse processo é repetido uma quantidade de vezes determinado como constante, no caso deste trabalho, o número máximo de tentativas foi de 5 vezes. Aceita esse novo modo caso seja melhor ou igual ao anterior. Esse passo também foi descrita em (HARTMANN, 2001).

Passo 3

Construir o cronograma de maneira serial, como descrito em (KOLISCH, 1996). Embora esse método ter sido proposto para a variação RCPSP, como temos um conjunto de modos pré-definidos, é possível apenas fixar os modos temporariamente e construir como se fosse uma instância de modos únicos.

Com os modos fixos, uma atividade é adicionada ao cronograma desde que não viole nem as restrições de precedência nem os limites de uso de recurso renovável. Além de que a probabilidade de uma atividade ser escolhida é baseada no tamanho da requisição pelos recursos renováveis, ou seja, as atividades que façam o uso da menor quantidade de recurso são escolhidas primeiro.

Após a execução desses 3 passos, obtemos uma solução inicial que necessariamente respeita as relações de precedência de limites de recurso renovável, mas pode ou não respeitar os dos não renováveis, como podemos observar na figura 5.4 em que o recurso 3, que é não renovável, tem seu uso ultrapassado. Semelhante a figura 5.2, também podemos observar as atividades que acontecem simultaneamente para a instância de múltiplos na

figura 5.5.

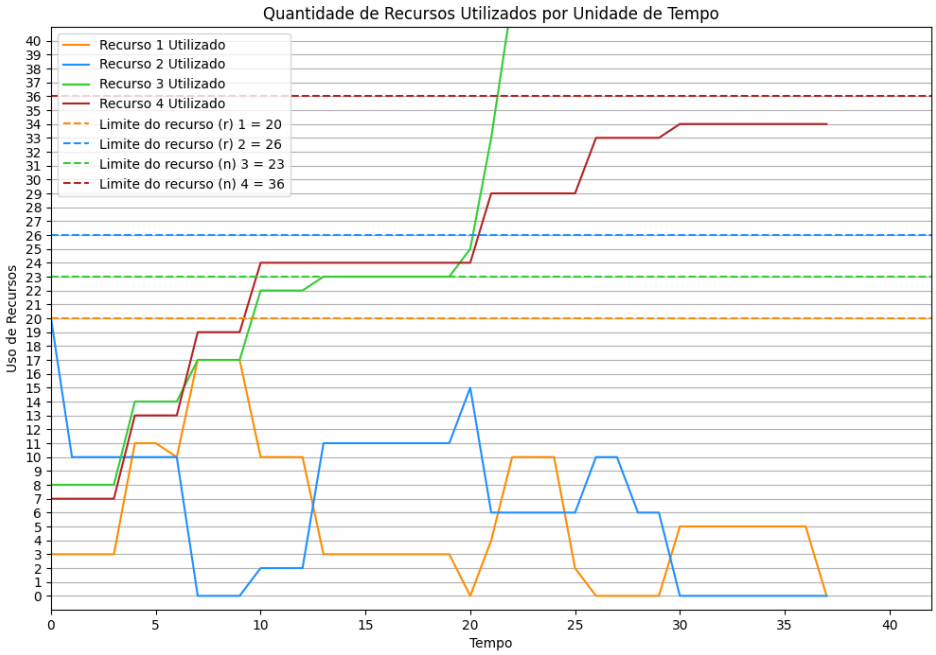

Figura 5.4 – Uso de recurso da solução inicial obtida para a instância c154_3.mm
Fonte: próprio autor

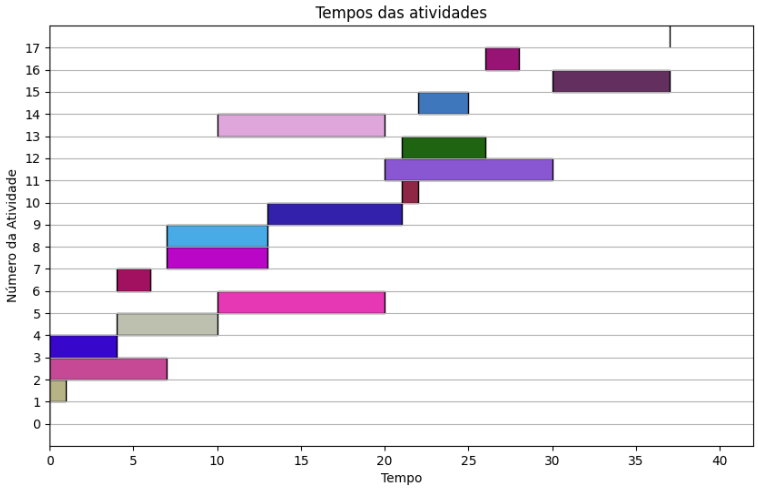

Figura 5.5 – Solução inicial obtida para a instância c154_3.mm
Fonte: próprio autor

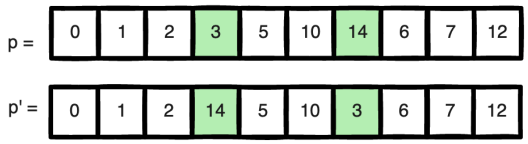

Figura 5.6 – Exemplo de uma troca simples
Fonte: próprio autor

5.4 Estruturas de vizinhanças

Para que seja possível explorar soluções vizinhas à atual, é preciso ter uma ou mais estruturas de vizinhanças. Assim, as que foram utilizadas neste trabalho serão descritas abaixo. Vale ressaltar que há vizinhanças que são aplicáveis em todas variações trabalhadas, enquanto outras se aplicam apenas em algumas delas.

5.4.1 Troca simples

Essa vizinhança busca realizar trocas entre duas atividades em uma sequência de execução de atividades P. As atividades J_i e J_j - selecionadas aleatoriamente - são trocadas na ordem, e uma nova sequência P' é obtida. A solução é então reconstruída de maneira a atender às restrições do problema em questão. Essa reconstrução acontece a partir da alocação das atividades presentes na nova sequência P' obtida, onde cada atividade será alocada seguindo essa ordem porém somente quando for possível alocá-la, ou seja, quando todas atividades antecessoras forem concluídas e quando sua execução não tenha a extrapolação da quantidade de uso de recurso renovável como consequência. A figura 5.6 exemplifica como é o funcionamento dessa vizinhança.

5.4.2 Inversão de subsequência

Essa configuração de vizinhança altera mais a solução quando comparada à anterior, uma vez que não apenas a ordem de execução de duas atividades são trocadas, mas também todas as atividades entre elas também são - algo semelhante à vizinhança *2-opt* que é bastante conhecida para o Problema do Caixeiro Viajante. São escolhidos dois números aleatoriamente, o primeiro é um número aleatório $i \in [1,$ quantidade de atividades $-1]$, o segundo é também um numero aleatório $j \in [i+1,$ quantidade de atividades $-1]$. Assim formando uma subsequência H de P que inicia na posição i da sequência e termina na posição j.

A partir de agora, todas as atividades da subsequência serão invertidas. A inversão se dá de maneira que a posição i de H será trocada com a posição j, $i+1$ com $j-1$ e assim por diante, levando em conta que i inicia com valor 0 - indicando a primeira posição - e terminaria esse processo gerando uma nova sequência P'. Vale ressaltar que após esse

processo, há a reconstrução da solução semelhante à que é feita na vizinhança de troca simples.

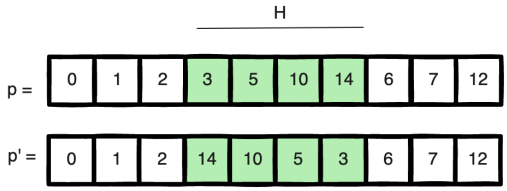

Figura 5.7 – Exemplo de inversão de subsequência
Fonte: próprio autor

5.4.3 Troca de modos

Essa estrutura de vizinhança se aplica apenas às variações que possuem múltiplos modos. Possui um funcionamento simples de apenas um movimento, é escolhido um número aleatório $i \in [1, \text{quantidade de atividades} - 1]$ e outro número aleatório $k \in M_i - \{\text{Sol}_{mi}\}$. A partir disso, é feita a troca de $\text{Sol}_{mi} = k$. Entretanto essa vizinhança possui uma particularidade em relação às apresentadas anteriormente, ela pode gerar soluções inválidas por extrapolar a quantidade de uso de recurso não-renovável, portanto é feita uma verificação sobre este ponto e, caso extrapole, é selecionado outro modo, caso todos outros modos também extrapolem o limite, é retornado a solução sem alterações.

Vale ressaltar que a troca de modos pode acontecer em 1 ou mais atividades por iteração, visando assim ter uma maior abrangência dentro desse espaço de busca em questão.

A figura 5.8 ilustra essa vizinhança.

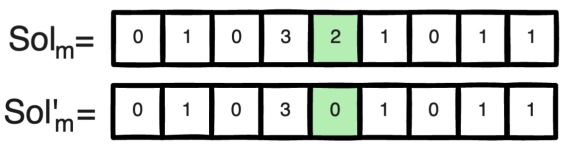

Figura 5.8 – Exemplo de Troca de modo
Fonte: próprio autor

5.5 Heurísticas

5.5.1 Hill Climbing

O algoritmo do *Hill Climbing* foi escolhido para ser o primeiro para realizar buscas locais dentro de alguma estrutura de vizinhança pelo fato de sua implementação ser simples e eficiente para caminhar pelo espaço de busca.

Algoritmo 3 Hill Climbing

```
1:  procedure HC(solucao)
2:      best ← solucao
3:      atual ← solucao
4:      qtdeIteracoesSemMelhora ← 0
5:      for i ← 0 to qtdeIteracoresMax do
6:          vizinho ← gerarVizinho(atual)      ▷ Gera vizinho com a vizinhança escolhida
7:          if vizinho.makespan() < current.makespan() then          ▷ Atualiza sol. atual
8:              atual ← vizinho
9:              qtdeIteracoesSemMelhora ← 0
10:             if vizinho.makespan() < best.makespan() then      ▷ Atualiza melhor sol.
11:                 best ← vizinho
12:             end if
13:         else
14:             qtdeIteracoesSemMelhora ← qtdeIteracoesSemMelhora + 1
15:             if qtdeIteracoesSemMelhora = qtdeIteracoesMaxSemMelhora then
16:                 break                                  ▷ Interrompe se tiver estagnado
17:             end if
18:         end if
19:     end for
20:     solucao ← best                                  ▷ Atribui melhor solução encontrada
21: end procedure
```

O algoritmo 3 demonstra - poupando de maiores detalhes - como funciona o *Hill Climbing* implementando. É interessante citar que não é necessário verificar a solução vizinha é factível, uma vez que o processo de vizinhança garante que a nova solução gerada não viola às restrições do problema. Dessa maneira, apenas o espaço de soluções viáveis é explorado.

5.5.2 Viabilizador de soluções

Essa heurística é específica para o caso da variação que possui múltiplos modos, uma vez que esta pode incluir soluções não viáveis. Diante disso, é necessário uma maneira de busca apenas entre os diferentes modos e não na sua sequência de atividades, já que o conjunto de modos selecionados pode desrespeitar a quantidade limite de recursos não renováveis.

Como não há uma garantia de que a solução seja viabilizada no processo, há a tentativa de pelo menos reduzir o quanto esta ultrapassa o limite de recurso.

Esse viabilizador foi feito para ser parte das próximas heurísticas e não uma como um todo.

Algoritmo 4 Viabilizador

```
1:  procedure ENABLER(solução, n)
2:      itMaxSemMelhora ← 10
3:      itSemMelhora ← 0
4:      atividadesAltas ← n atividades mais custosas da solução
5:      while itSemMelhora < itMaxSemMelhora do
6:          for atividade in atividadesAltas do
7:              solAtual ← solução
8:              solAtual ← troca o modo da atividade atual aleatoriamente
9:              if solAtual viável then
10:                 solução ← solAtual
11:                 return
12:             else
13:                 if fitness(solAtual) < fitness(solução) then
14:                     solução ← solAtual
15:                 else
16:                     itSemMelhora ← itSemMelhora + 1
17:                 end if
18:             end if
19:         end for
20:     end while
21: end procedure
```

5.5.3 Simulated Annealing

Outro método de exploração do espaço de busca que é bastante conhecido na literatura é o *Simulated Annealing* (KIRKPATRICK; JR; VECCHI, 1983). Esse algoritmo busca simular o recozimento de metais, processo este que precisa aquecer o material até um certo estado e depois resfriar de maneira lenta até que atinja um estado de equilíbrio.

Essa meta-heurística é bastante interessante porque - diferentemente da citada anteriormente - também são aceitas soluções que possuem um valor de avaliação pior do que a atual. Dessa maneira, é possível realizar uma busca ainda mais abrangente dentro da estrutura de vizinhança que estiver sendo utilizada em questão.

O Algoritmo 5 foi projetado para realizar 100 iterações em cada temperatura. Essa abordagem visa realizar uma busca mais abrangente em cada nível de temperatura, em contraste com uma única iteração, proporcionando uma exploração mais significativa do espaço de busca.

Algoritmo 5 Simulated Annealing

```
1: procedure SA(solucao, temperaturaInicial, taxaResfriamento, temperaturaMinima)
2:     best ← solucao
3:     atual ← solucao
4:     temperaturaAtual ← temperaturaInicial
5:     while temperaturaAtual > temperaturaMinima do
6:         for i ← 0 to 100 do                          ▷ Iterações por temperatura
7:             vizinho ← gerarVizinho(atual)   ▷ Gera vizinho com vizinhança escolhida
8:             if vizinho not enabled then              ▷ Apenas para múltiplos modos
9:                 enabler(vizinho)
10:            end if
11:            delta ← vizinho.makespan() − atual.makespan()
12:            if delta < 0 then                              ▷ Atualiza sol. atual
13:                atual ← vizinho
14:                if vizinho.makespan() < best.makespan() then
15:                    best ← vizinho                        ▷ Atualiza melhor sol.
16:                end if
17:            else
18:                prob ← e^(−delta / temperaturaAtual)
19:                if random ∈ [0, 1] < prob then
20:                    atual ← vizinho ▷ Atualiza sol. atual com uma certa probabilidade
21:                end if
22:            end if
23:        end for                                  ▷ Diminui temperatura atual
24:        temperaturaAtual ← temperaturaAtual · taxaResfriamento
25:    end while
26:    solucao ← best                          ▷ Atribui melhor solução encontrada
27: end procedure
```

5.5.4 Algoritmo genético

É interessante também ter abordagens a partir de heurísticas de população, uma vez que assim, pode-se obter um conjunto de soluções boas ao final do algoritmo e não apenas um como acontece nas abordagens de solução única. O algoritmo genético que será proposto abaixo teve bastante influência do que foi definido em (HARTMANN, 2001).

Assim, para o algoritmo genético é preciso definir alguns elementos e operações que são específicos dessa técnicas.

5.5.4.1 Constantes

Primeiro, é importante definir as constantes que irão reger o processamento do algoritmo, são elas:

- P: O tamanho da população.
- G: Quantidade de gerações.

Temos que P representa a quantidade de indivíduos que estarão presentes numas mesma geração e G representa a quantidade de vezes em que o processo de seleção, crossover e mutação ocorrerão.

5.5.4.2 Indivíduo

Um indivíduo é como uma solução é representada dentro do contexto desse tipo de técnica. Embora já tinha sido definido uma maneira de representação em 5.1, esta definição é mais adequada nessa situação.

Diante disso, um indivíduo é um par de listas, a primeira contendo uma lista de atividades $j_1 \dots j_J$ em que a precedência é respeitada e a segunda lista $\theta(j) = [m_1, \dots, m_J]$ é referente aos modos das atividades, de maneira em que o modo m_j é o designado para a atividade j_i.

Lista de atividades: [0, 3, 6, 2, 8, 7, 1, 9, 12, 10, 16, 14, 4, 5, 13, 11, 15, 17]

Modos: [0, 1, 2, 2, 1, 2, 0, 2, 2, 1, 0, 2, 0, 2, 0, 2, 0, 0]

Cronograma: [0, 0, 0, 0, 4, 10, 4, 7, 7, 13, 21, 20, 21, 10, 22, 30, 26, 37]

Figura 5.9 – Exemplo de indivíduo da instância com genótipo e fenótipo c154_3.mm

Além disso, cada indivíduo, semelhante ao que acontece na biologia, possui seu genótipo e seu fenótipo. No contexto desse algoritmo, um exemplo de genótipo é o exemplificado na figura 5.9, em que são os dados que serão passados para os próximos indivíduos através das gerações.

O fenótipo é o cronograma de atividades, em que é obtido através do processo de alocação serial das atividades (KOLISCH, 1996), semelhante ao processo da construção inicial, porém em vez de utilizar um conjunto de modos aleatório, utiliza o conjunto do indivíduo como também sua lista de atividades.

5.5.4.3 Função de avaliação

Embora já tenha sido definida uma forma de avaliar a solução na seção 5.2, esta só faz sentido para soluções factíveis. Entretanto, num contexto de múltiplas soluções é interessante que existam algumas infactíveis porque através delas pode ser possível alcançar indivíduos perto do ótimo.

Uma forma interessante de avaliar uma solução foi proposta em (HARTMANN, 2001). Para descrevê-la, é preciso definir alguns conceitos:

- L_r: Denota a quantidade em que a requisição do recurso r foi ultrapassada com $r \in N$.
- L: Denota a soma das quantidades ultrapassadas dos recursos não renováveis.
- M_i^{max}: Representa o *makespan* do indivíduo i.

Vale lembrar que T representa o tempo máximo para conclusão das atividades, como mostrado em 4.3.3.1.

Assim, temos que:

$$L_r = \sum_{j=1}^{J} n_{jr\theta(j)} - N_r \qquad (5.5)$$

$$L = \sum_{\substack{r \in N \\ L_r > 0}} L_r \qquad (5.6)$$

Com isso, a função de avaliação do indivíduo é $F(i) = M_i^{max}$ se a solução for factíveis e $F(i) = T + L$. Dessa maneira, a avaliação fará sentido tanto para soluções factíveis quanto para não factíveis, uma vez que as válidas sempre terão uma avaliação menor quando comparada as não válidas. Além disso, as infactíveis são punidas pela quantidade de recurso não renovável ultrapassado.

5.5.4.4 População inicial

Para gerar os indivíduos primitivos do algoritmo, apenas repetimos o processo descrito em 5.3.2.2 até que a população seja de tamanho P.

5.5.4.5 Crossover

A operação de crossover nos algoritmos genéticos serve para combinar indivíduos e, normalmente, gerar mais dois indivíduos como resultado dessa combinação, para facilitar o entendimento, vamos chamar os indivíduos geradores de "pai"e "mãe"e os gerados de "filho"e "filha". Dessa maneira, essa operação neste trabalho foi inspirada em (HARTMANN, 2001) e se dá da seguinte forma:

Primeiramente são escolhidos dois indivíduos aleatoriamente, uma mãe e um pai. Depois dois número aleatórios $1 \leq r1, r2 \leq J$ são escolhidos.

Com isso, a geração dos filhos é feita de maneira com que a filha herde a lista de atividades da mãe até a posição $r1$ e as atividades que não foram escolhidas, são obtidas na ordem em que aparecem na lista do pai. O mesmo processo é feito com filho porém herda as atividades até a posição $r1$ do pai e segue o mesmo procedimento com a mãe.

Em seguida, a filha herda a lista de modos da mãe até a posição $r2$ e o restante do pai. Semelhante ao que acontece com a lista de atividades, esse processo para o filho é o mesmo porém utilizando o pai primeiramente.

Assim, é calculado o fenótipo de cada um dos filhos e, posteriormente, são adicionados à população.

5.5.4.6 Seleção

Esse tipo de operação é utilizado para selecionar os indivíduos de uma população com base em algum critério. O utilizado neste trabalho foi o valor de avaliação com indivíduo.

Dessa maneira, nesse processo é, primeiramente, feita uma ordenação na população com base nesse valor e, posteriormente, é retirada a segunda metade do conjunto de indivíduos.

5.5.4.7 Mutação

A última operação a ser definida é a mutação. Esse tipo de operador atua sobre cada indivíduo com alguma probabilidade p_{mut}. Neste caso, a mutação pode alterar tanto a lista de atividades quanto a lista de modos.

Cada atividade $j_i \in J$ pode ser trocada com j_{i+1} com probabilidade p_{mut} desde que não resulte em uma lista de atividade que viole as relações de precedência. Em relação aos modos, cada um pode ser trocado, também com probabilidade p_{mut}, por outro $m_j \in M_j \ \ \& \notin \theta(j)$.

5.5.4.8 Busca local

Com uma tentativa de tornar o melhor indivíduo da população mais eficiente, também há um passo adicional de busca local após a operação de seleção. Essa etapa consiste na execução do algoritmo 5 no indivíduo com menor valor na função de avaliação.

Na busca em questão são realizadas apenas 1 iteração por temperatura, contrastando com as 100 feito está descrito na subseção 5.5.3.

5.5.4.9 Processamento do algoritmo

Com todos elementos e operadores definidos, sua sequência de execução se dá como mostrado no algoritmo 6.

Algoritmo 6 Algoritmo Genético

```
1: procedure GA(P, G)
2:     POP ← Gera população de tamanho P
3:     melhor ← Melhor indivíduo da população inicial
4:     for i ← 0 to G do
5:         seleção()
6:         melhor ← Melhor indivíduo de POP
7:         busca_local(melhor)
8:         crossover()
9:         mutação()
10:    end for
11: end procedure
```

6 Resultados

Nesse capítulo são apresentados os resultados sobre os algoritmos descritos nos capítulos anteriores. Os experimentos foram realizados em um computador com processador Apple M1 com 3.1GHz e 8GB de memória RAM. As heurísticas foram implementadas em C++17 e testadas no MACOS Sonoma 14.6.1.

Para os experimentos, foram selecionadas instâncias da PSPLIB e criadas a partir do gerador da seção 4.4 com diferentes quantidades de atividades. A nomenclatura dos grupos de instâncias segue o formato: I_J, onde I identifica o grupo, e J indica o número de atividades em cada instância.

A tabela 6.1 mostra os melhores resultados encontrados a partir da metodologia descrita anteriormente, bem como comparações entre três heurísticas: simulated annealing, algoritmo genético e algoritmo genético com etapa de busca local. Para alguns grupos de instâncias, mais de 50% dos melhores resultados conhecidos foram encontrados. Vale salientar que, especificamente para o grupo C15, todos os resultados ótimos foram encontrados pela abordagem proposta.

Cada heurística foi aplicada a cada grupo de instâncias em 10 execuções independentes, a fim de garantir a robustez dos resultados e minimizar o impacto dos fatores aleatórios das heurísticas.

Tabela 6.1 – Melhores resultados

	J10	**J12**	**J14**	**C15**	**J16**	**J18**
Resultados ótimos	# 349	# 350	# 286	# 551	# 223	#193
% em relação ao total	65,1%	63,9%	51,9%	100,0%	40,5%	34,9%
Média desvio padrão	0,212	0,135	0,324	0,659	0,463	0,558
Maior desvio padrão	2,098	1,643	2,449	3,972	2,049	3,674
Média desvio médio	0,160	0,107	0,254	0,493	0,356	0,430
Média coeficiente de variação	1,151	0,490	0,998	2,388	1,309	1,480
Média desvio percentual	3,7%	3,2%	4,3	0,00%	5,5%	6,7%
Maior diferença	10	9	9	0	10	19
Maior diferença percentual	23,8%	21,4%	23,1%	0,00%	25,6%	39,6%
Total de instâncias	536	547	551	551	550	552

As tabelas 6.2 e 6.3 apresentam as comparações entre as abordagens discutidas anteriormente e foram executadas para todo o grupo de instância em questão. Observa-se que a heurística mais robusta foi o algoritmo genético com busca local, conforme indicado pelo baixo valor médio do desvio-padrão. Além disso, o baixo valor médio do desvio absoluto reforça a consistência da solução, demonstrando apenas uma pequena discrepância em relação ao melhor makespan conhecido. Esse comportamento foi ainda mais evidente no grupo C15, onde todos os resultados ótimos conhecidos foram obtidos. Um outro ponto

interessante é que o SA, dentro das execuções que foram feitas neste estudo, não conseguiu encontrar uma solução viável para todas as instâncias presentes no grupo C15.

As tabelas 6.4 e 6.5 apresentam uma comparação do tempo de execução de cada abordagem. Pode-se notar que, embora o Simulated Annealing (SA) seja a técnica mais rápida, ele não seria a escolha ideal para eventos de grande porte devido à significativa diferença percentual observada nas Tabelas 6.2 e 6.2 quando comparado às outras heurísticas. Além disso, vale destacar que a inclusão da busca local no Algoritmo Genético (AG) foi bastante eficaz, pois aumentou a robustez e a eficiência da técnica sem impactar significativamente o tempo médio de execução.

Tabela 6.2 – Comparação entre heurísticas para J10 - makespan

	SA	**GA**	**GA + Busca**
Resultados ótimos	# 164	# 335	# **349**
% em relação ao total	30,6%	62,5%	**65,1%**
Média desvio padrão	1,31	1,12	**0,21**
Média desvio médio	1,00	0,84	**0,16**
Média coeficiente de variação	5,92	5,68	**1,15**
Média desvio percentual	9,9%	4%	**3,7%**
Maior diferença	11	10	10
Maior diferença percentual	44%	23,8%	23,8%
Maior desvio padrão	5,7	9	**2,1**
% Viabilidade	100%	100%	100%

Tabela 6.3 – Comparação entre heurísticas para C15 - makespan

	SA	**GA**	**GA + Busca**
Resultados ótimos	# 27	# 243	# **551**
% em relação ao total	4%	44,1%	**100%**
Média desvio padrão	2,20	0,66	**0,65**
Média desvio médio	1,67	0,50	**0,49**
Média coeficiente de variação	2,41	5,68	**2,38**
Média desvio percentual	24,7%	6,3%	**0,0%**
Maior diferença	28	13	**0**
Maior diferença percentual	59,4%	34,2%	**0,0%**
Maior desvio padrão	13,1	4,9	**3,9**
% Viabilidade	98,7%	100%	100%

Tabela 6.4 – Comparação entre heurísticas para J10 - Tempo de execução

	SA	**GA**	**GA + Busca**
Tempo mínimo	0,025s	0,7s	0,806s
Tempo máximo	0,124s	1,968s	2,981s
Média de tempo	0,030s	0,854s	1,009s
Desvio padrão	0,005	0,174	0,212
Coeficiente de variação	16,67	20,37	21,01

Tabela 6.5 – Comparação entre heurísticas para C15 - Tempo de execução

	SA	**GA**	**GA + Busca**
Tempo mínimo	0,032s	1,780s	1,775s
Tempo máximo	0,113s	4,067s	4,045s
Média de tempo	0,040s	2,021s	2,022s
Desvio padrão	0,007	0,171	0,171
Coeficiente de variação	17,14	8,46	8,46

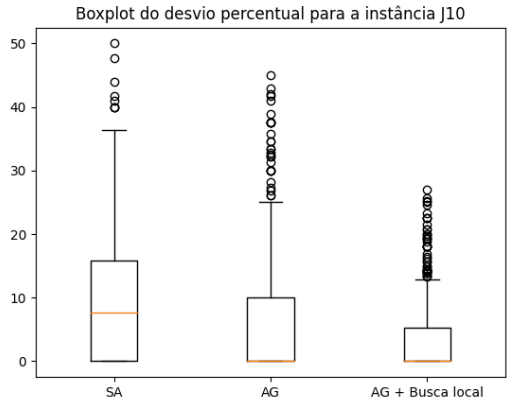

Figura 6.1 – Gráfico comparativo entre heurísticas para instâncias J10

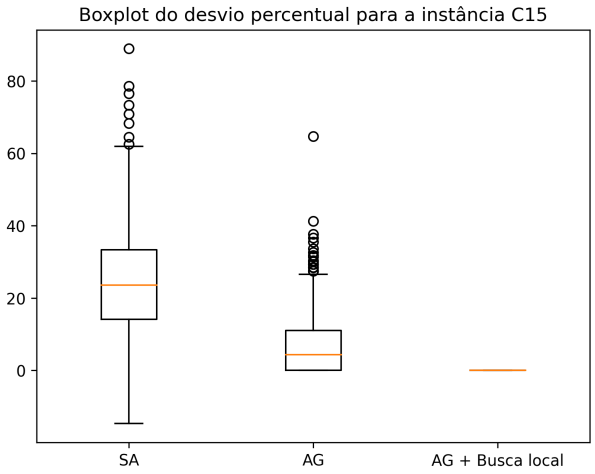

Figura 6.2 – Gráfico comparativo entre heurísticas para instâncias C15

A análise da Figura 6.1 revela pontos interessantes sobre o desempenho das heurísticas. Embora o AG e o AG + Busca Local apresentem medianas semelhantes, sugerindo resultados médios equivalentes, a busca local demonstra ser uma adição valiosa ao algoritmo genético. Isso se evidencia pela redução significativa na dispersão dos desvios, indicando uma maior consistência nos resultados. Com isso, para o grupo J10, o SA obteve o pior desempenho, seguido pelo AG, enquanto o AG + Busca Local se destacou como a melhor opção.

A Figura 6.2 corrobora as tendências observadas na Figura 6.1, com as heurísticas

mantendo a mesma hierarquia de desempenho. Entretanto ainda há alguns pontos que são interessantes de serem analisados como: SA ter valores negativos indica que, para algumas instâncias do grupo, não foi encontrada uma solução válida; A mediana do AG é maior do que zero, portanto, encontrou a solução ótima para menos de 50% do grupo C15.

Tabela 6.6 – Parâmetros para instâncias geradas

Parâmetro	valor
D_{min}	10
D_{max}	30
R_{min}	2
R_{max}	5
K_{min}	10
K_{max}	100
M_{min}	1
M_{max}	4
S_{max}	3
RF	0.6
RS	0.2
S_{1min}	3
S_{1max}	3
F_{Jmin}	3
F_{Jmin}	3
J	10/12/14

Tabela 6.7 – Comparação entre makespan das instâncias criadas

Grupo de instâncias	**L10**			**L12**			**L14**		
Heurística	**SA**	**GA**	**GA+Busca**	**SA**	**GA**	**GA+Busca**	**SA**	**GA**	**GA+Busca**
Maior desvio padrão	5.07	5.36	0	23.10	1.78	1.78	33.74	3.04	2.23
Média desvio padrão	1.39	0.32	0	9.23	0.37	0.17	14.93	0.35	0.42
Maior coeficiente de variação	3,98	3,79	0,00	11,67	1,18	1,21	13,29	1,66	1,14
Média coeficiente de variação	1,05	0,23	0,00	4,69	0,24	0,12	6,44	0,20	0,19

As instâncias utilizadas nas comparações feitas nas tabelas 6.7 foras geradas utilizando os parâmetros presentes na tabela 6.6. Apenas há a variação da quantidade de atividades em que J indica a quantidade de atividades que L_J possui. Os resultados encontrados nessa etapa reforçam a ideia de que a etapa de busca local adicionada ao AG obteve bons resultados, uma vez que foi a heurística que apresentou menor variabilidade. Também teve sempre a menor média de makespan quando comparada a outras técnicas, como pode ser observado na tabela 6.8.

Por fim, os resultados obtidos indicam que a combinação do algoritmo genético com a busca local (AG + Busca Local) pode ser uma estratégia eficaz para encontrar soluções de alta qualidade para os problemas analisados. A menor variabilidade e os melhores resultados médios obtidos com o AG + Busca Local sugerem que a busca local contribui para refinar as soluções encontradas pelo algoritmo genético. No entanto, é crucial

Tabela 6.8 – Médias entre makespans

L10			L12			L14		
SA	**GA**	**GA+Busca**	**SA**	**GA**	**GA+Busca**	**SA**	**GA**	**GA+Busca**
161,8	160	**160**	197,8	**165**	**165**	192	**158**	**158**
128,8	128	**128**	148,8	**127**	**127**	230,6	**190**	**190**
105	**103**	**103**	207,6	**182**	**182**	203,2	156,2	**154,2**
158	**157**	**157**	219,6	**177**	**177**	253	176,4	**176**
113	113	113	164	130	**129,6**	253,8	**209**	**209**
157	157	157	213,4	**146**	**146**	282,2	**256**	**256**
111,6	**103**	**103**	198,2	**151**	**151**	233,2	**204**	**204**
134,4	**134**	**134**	200,2	170	**169**	208,8	**158**	**158**
140,4	**140**	**140**	225	**216**	**216**	228,4	181,2	**180,4**
138	138	138	212	**195**	**195**	156,6	**123**	**123**
144,4	141,4	**139**	194,6	**169**	**169**	244,8	212	**211**
146,6	146,4	**146**	175,4	147	**146**	244	**211**	**211**
148,8	**148**	**148**	226	**208**	**208**	231,4	183,4	**181,4**
131,4	126,4	**126**	197,6	**156**	**156**	243,6	**201**	**201**
116,8	**113**	**113**	208,6	**170**	**170**	245,2	216	**209**

ressaltar que as conclusões obtidas são baseadas em um conjunto limitado de experimentos. Para generalizar os resultados e confirmar a superioridade do AG + Busca Local, são necessários estudos mais abrangentes, envolvendo um maior número de instâncias e configurações. Além disso, a comparação com outras metaheurísticas é fundamental para avaliar o desempenho relativo do AG + Busca Local.

7 Conclusão

Neste trabalho, foi estudada uma abordagem para otimizar o processo de montagem de estruturas para eventos, modelado como um problema de escalonamento, conhecido como Project Scheduling Problem (PSP). Duas variações do PSP foram abordadas: o Resource-Constrained PSP (RCPSP), onde as atividades têm um único modo de execução, e o Multi-Mode RCPSP (MRCPSP), onde as atividades podem ser realizadas em diferentes modos. A primeira variação foi incluída apenas para fins introdutórios.

Inicialmente, as instâncias utilizadas nos experimentos foram extraídas da PSPLIB, uma biblioteca de referência com diversas instâncias de PSP. Contudo, para refletir melhor as especificidades do cenário de montagem de estruturas para eventos — como a variação no número de modos por atividade e uma maior diversidade nos tipos de recursos — foi desenvolvido um gerador de instâncias personalizado. Esse gerador, baseado em seeds, permite a replicação exata das instâncias desde que os mesmos parâmetros sejam utilizados.

Para avaliar o desempenho, foram escolhidas as metaheurísticas de Simulated Annealing e Algoritmo Genético (AG), este último tanto em sua forma pura quanto combinado com busca local. A análise foi realizada sobre instâncias da PSPLIB e aquelas geradas pelo gerador desenvolvido. Os resultados mostraram que, embora o AG com busca local tenha apresentado o melhor desempenho geral, ainda é necessário um estudo mais detalhado em grupos de instâncias com diferentes características. Isso se deve ao fato de que, em um conjunto específico de instâncias, foram encontradas soluções ótimas em 100% dos casos, enquanto em outros grupos o desempenho foi inferior, com menos de 50% das soluções ótimas sendo alcançadas.

Para futuras pesquisas, recomenda-se o estudo mais aprofundado da heurística que melhor se destacou nesse trabalho para avaliar sua performance em diferentes cenários bem como aplicar outras metaheurísticas nesse contexto. Isso pode contribuir para o desenvolvimento de métodos mais robustos e versáteis, capazes de lidar com uma gama mais ampla de cenários práticos.

Referências

BALTAR, D. D. **LAHC aplicado ao problema de escalonamento de múltiplos projetos com múltiplos modos e restrições de recursos**. 53 p. Monografia — Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade, 2017. Graduação em Engenharia de Computação.

BEŞIKCI, U.; BILGE, Ü.; ULUSOY, G. Multi-mode resource constrained multi-project scheduling and resource portfolio problem. **European Journal of Operational Research**, Elsevier, v. 240, n. 1, p. 22–31, 2015.

BLAZEWICZ, J.; LENSTRA, J. K.; KAN, A. R. Scheduling subject to resource constraints: classification and complexity. **Discrete applied mathematics**, Elsevier, v. 5, n. 1, p. 11–24, 1983.

CHEN, J. C. et al. Applying hybrid genetic algorithm to multi-mode resource constrained multi-project scheduling problems. **Journal of the Chinese Institute of Engineers**, Taylor & Francis, v. 45, n. 1, p. 42–53, 2022.

GONÇALVES, J. F.; MENDES, J. J.; RESENDE, M. G. A genetic algorithm for the resource constrained multi-project scheduling problem. **European journal of operational research**, Elsevier, v. 189, n. 3, p. 1171–1190, 2008.

HARTMANN, S. Project scheduling with multiple modes: A genetic algorithm. **Annals of Operations Research**, Springer, v. 102, p. 111–135, 2001.

KIRKPATRICK, S.; JR, C. D. G.; VECCHI, M. P. Optimization by simulated annealing. **science**, American association for the advancement of 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**, Elsevier, v. 90, n. 2, p. 320–333, 1996.

KOLISCH, R.; DREXL, A. Local search for nonpreemptive multi-mode resource-constrained project scheduling. **IIE transactions**, Taylor & Francis, v. 29, n. 11, p. 987–999, 1997.

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

KOLISCH, R.; SPRECHER, A.; DREXL, A. **Characterization and generation of a general class of resource-constrained project scheduling problems: Easy and hard instances**. [S.l.], 1992.

MELO, L. V. de; QUEIROZ, T. A. de. Integer linear programming formulations for the rcpsp considering multi-skill, multi-mode, and minimum and maximum time lags. **IEEE Latin America Transactions**, IEEE, v. 19, n. 01, p. 5–16, 2021.

PASCOE, T. L. Allocation of resources cpm. **Revue Francaise de Recherche Operationnele**, SOC FRANCAISE DE RECHERCHE OPERATIONNELLE PARIS, FRANCE, v. 10, n. 38, p. 31–31, 1966.

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.

RATAJCZAK-ROPEL, E.; SKAKOVSKI, A. **Population-based approaches to the resource-constrained and discrete-continuous scheduling**. [S.l.]: Springer, 2018.

REYCK, B. D.; HERROELEN, W. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. **European Journal of Operational Research**, Elsevier, v. 119, n. 2, p. 538–556, 1999.

SOARES, J. A. **Heurísticas baseadas em programação inteira para o problema de escalonamento de múltiplos projetos com múltiplos modos e Restrições de recursos**. 113 p. Dissertação (Dissertação) — Universidade Federal de Ouro Preto, Ouro Preto, 2013. Mestrado em Ciência da Computação.

SPRECHER, A.; HARTMANN, S.; DREXL, A. An exact algorithm for project scheduling with multiple modes. **Operations-Research-Spektrum**, Springer, v. 19, p. 195–203, 1997.

WAUTERS, T. et al. The multi-mode resource-constrained multi-project scheduling problem: The mista 2013 challenge. **Journal of Scheduling**, Springer, v. 19, p. 271–283, 2016.