CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Diretoria de Pesquisa e Pós-Graduação

Programa de Pós-Graduação em Modelagem Matemática e Computacional

Um GRASP Reativo para o Problema de Sequenciamento de Máquinas Paralelas não Relacionadas com Recurso Escasso Adicional

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

Aluno : Cesar Augusto Souza de Oliveira

Orientador : Prof. Dr. Sérgio Ricardo de Souza

Co-Orientador : Dr. Rodney Oliveira Marinho Diana

Belo Horizonte
Fevereiro de 2023

Oliveira, Cesar Augusto Souza de
O48g Um GRASP reativo para o problema de sequenciamento de máquinas paralelas não relacionadas com recurso escasso adicional / Cesar Augusto Souza de Oliveira. – 2023.
67 f.

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Sérgio Ricardo de Souza.
Coorientador: Rodney Oliveira Marinho Diana.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Sequenciamento de tarefas – Teses. 2. Alocação de recursos – Teses. 3. Processamento paralelo (Computadores) – Teses. 4. Makespan – Teses. 5. GRASP (Sistema operacional de computador) – Teses. 6. Metaheurística – Teses. I. Souza, Sérgio Ricardo de. II. Diana, Rodney Oliveira Marinho. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 519.6

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

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

"UM GRASP REATIVO PARA O PROBLEMA DE SEQUENCIAMENTO DE MÁQUINAS PARALELAS NÃO RELACIONADAS COM RECURSO ESCASSO ADICIONAL"

Dissertação de Mestrado apresentada por **Cesar Augusto Souza de Oliveira**, em 24 de fevereiro de 2023, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

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

Documento assinado digitalmente
gov.br RODNEY OLIVEIRA MARINHO DIANA
Data: 20/09/2023 12:25:18-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Rodney Oliveira Marinho Diana
Centro Federal de Educação Tecnológica de Minas Gerais

Documento assinado digitalmente
gov.br MARIA AMELIA LOPES SILVA
Data: 19/09/2023 08:33:50-0300
Verifique em https://validar.iti.gov.br

Profª. Drª. Maria Amélia Lopes Silva
Universidade Federal de Viçosa

Documento assinado digitalmente
gov.br GUSTAVO CAMPOS MENEZES
Data: 18/09/2023 10:18:33-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Gustavo Campos Menezes
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida a impressão,

Prof. Dr. Thiago de Souza Rodrigues
Coordenador do Programa de Pós-Graduação em
Modelagem Matemática e Computacional

Agradecimentos

Agradeço ao Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) e ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, pela oportunidade e apoio na realização deste curso de mestrado.

Agradeço ao meu orientador, Prof. Dr. Sérgio Ricardo de Souza, e ao coorientador, Dr. Rodney Oliveira Marinho Diana, por todo o tempo dedicado, conselhos, ensinamentos e principalmente muita paciência.

Agradeço a minha parceira Rovena Vieira pela motivação, apoio, paciência e compreensão nos momentos da minha ausência. Ainda assim não fui poupado do seu amor, carinho e um sorriso sempre no rosto.

Quero agradecer a minha mãe Filomena, pelo amor incondicional, preocupações e orações constantes. Por em muitos momentos sofrer mais do que eu mesmo durante momentos difíceis. A minha irmã Andreia que sempre deixou clara sua admiração e respeito por mim.

Agradeço aos meus filhos Vitória e Bernardo por me presentearem todos os dias com sorrisos alegres e muito carinho em forma de travessuras. Graças a vocês insisti na recusa de desistir.

Também gostaria de agradecer ao tio Altamiro e a tia Maria da Penha por me acolherem em Belo Horizonte durante os semestres de disciplinas. Me garantiram, banho quente, comida gostosa, cama aconchegante, e um carinho de verdadeiros pais. Sou profundamente grato a vocês.

Agradeço ao colega Anderson Vasconcelos pela parceria, caronas, boas conversas e troca de conhecimento desde o início dessa jornada.

Por fim, agradeço ao CEFET-MG Campus de Curvelo, por meio dos diretores e chefias que não mediram esforços para o meu sucesso durante essa trajetória.

"Mesmo desacreditado e ignorado por todos, não posso desistir, pois para mim, vencer é nunca desistir"

Albert Einstein

Resumo

Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas não relacionadas com recurso escasso adicional (*Unrelated Parallel Machine Scheduling Problem with Secondary Resources* – UPMR). Neste problema, um determinado conjunto de tarefas serão distribuídas em um conjunto de máquinas. O objetivo é minimizar o instante de conclusão da última tarefa (*makespan*). A cada instante de tempo existe uma quantidade máxima de recursos disponível para processar as tarefas do sequenciamento, cada tarefa demanda uma certa quantidade desse recurso ao ser processada. Caso o limite total de recurso seja ultrapassado, a solução se torna infactível. Para resolução do problema, propomos um algoritmo GRASP Reativo juntamente com um método de busca local VND. Um método guloso para o reparo de soluções infactíveis foi proposto e aplicado em conjunto com o GRASP Reativo. Um conjunto de instâncias da literatura é usado para a validação e comparação dos resultados. Análises realizadas com os resultados obtidos indicaram que a abordagem proposta obteve resultados satisfatórios em boa parte das instâncias em relação à abordagens existentes na literatura.

PALAVRAS-CHAVE: Sequenciamento de tarefas, Recurso, Máquinas Paralelas, *Makespan*, *GRASP* Reativo, Metaheuristica.

Abstract

This work addresses the Unrelated Parallel Machine Scheduling Problem with Secondary Resources - UPMR. In this problem, a certain set of tasks will be distributed across a set of machines. The main goal is to minimize the completion of time the last task – Makespan. At every instant of time there is a maximum amount of resources available to process sequencing tasks, each task demands a certain amount of this resource when being processed. If the total resource limit is exceeded, the solution becomes infeasible.To solve the problem, we propose a Reactive GRASP algorithm together with a VND local search method. A greedy method for repairing infeasible solutions was proposed and applied together with GRASP Reactive. A set of instances from the literature were used for validation and comparison of results. Analysis carried out with the results obtained indicated that the proposed approach obtained satisfactory results in most instances in relation to approaches existing in the literature.

Keywords: Scheduling, Resource, Parallel machines, Makespan, Reactive GRASP, Metaheuristica.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Este capítulo apresenta o contexto no qual o Problema de Sequenciamento de Máquinas Paralelas não Relacionadas com Recurso Escasso Adicional é proposto. Motivações para a realização da pesquisa são descritas na Seção 1.2. Limitações e diferenciais identificados no percorrer deste trabalho são apresentados na Seção 1.3. Por fim, a Seção 1.4 mostra a organização geral do texto como um todo.

1.1 Apresentação

A busca por maior ganho de produtividade e redução nos custos de produção tem acarretado em cada vez mais a substituição da mão de obra humana por processos automatizados executadas por máquinas, levando a incrementos na velocidade de trabalho, maior eficiência e volume das operações. Em diversos processos produtivos, grande parte do trabalho humano restante está associado à manutenção desses processos automatizados de produção. O crescimento significativo na velocidade dos processos produtivos, contudo, tem sido acompanhado pelo surgimento de gargalos associados, sobretudo, à necessidade de recursos externos como combustível, eletricidade, água, matéria prima, e a própria mão de obra humana mais especializada. Estes recursos normalmente são limitados e incorporam altos custos.

Para levar a um melhor manejo destes recursos escassos e valiosos, a tomada de decisão e a otimização de processos, especialmente em grandes ambientes de produção, são fatores muito relevantes para tornar manufaturas competitivas. Diante disso, é clara a importância de se desenvolver processos mais rápidos, inteligentes e sustentáveis para atender tais necessidades.

O problema abordado neste trabalho é conhecido como Problema de Sequenciamento de Máquinas Paralelas não Relacionadas com Recurso Escasso Adicional (*Unrelated Parallel Machine Scheduling Problem with Secondary Resources* – UPMR). Trata-se de uma variante do problema clássico de Sequenciamento de Máquinas Paralelas não Relacionadas (*Unrelated Parallel Machine Scheduling Problem* – UPM), na qual considera-se a existência de uma demanda adicional dos recursos envolvidos nos processos. Esta variante do UPM torna-o mais próximo da realidade de ambientes de manufatura, nos quais sempre haverá uma demanda de recursos externos envolvidos nos processos.

Nesse problema em especial, o montante de recursos consumidos por instante de processamento varia para cada tarefa e pode também variar entre máquinas. O

recurso é utilizado pela tarefa enquanto ela é processada, e liberado assim que esta tarefa tem seu processamento concluído. A cada instante de tempo, existe uma quantidade máxima de recursos renováveis disponíveis para as tarefas que estiverem simultaneamente sob processamento naquele instante. Como os recursos são renováveis, uma vez que uma tarefa finaliza o processamento, esta libera a quantidade de recursos consumida.

Considerando este ambiente, neste trabalho um sequenciamento consiste em obter uma solução factível (i.e., uma solução que respeite a quantidade máxima de recursos por instante de tempo) e minimizar o *makespan*. Denotado como C_{max}, o *makespan* representa o instante de tempo necessário para o processamento de todas as tarefas do sequenciamento. O objetivo é, portanto, processar todo o conjunto de tarefas no menor tempo possível.

Para as condições próprias do UPMR, aborda-se neste trabalho uma etapa em especial, que chamaremos de Reparo. Esta etapa é responsável por produzir soluções factíveis do ponto de vista dos recursos, ou seja, garantir que a quantidade utilizada de recursos escassos disponíveis não seja ultrapassada em nenhum momento do sequenciamento. O método de reparo proposto visa tornar factível a solução, de modo a impactar o mínimo possível o valor de C_{max} (*makespan*). O método de reparo proposto foi aplicado com a metaheurística *Greedy Randomized Adaptive Search Procedure* (*GRASP*) (Feo e Resende, 1995), em sua forma reativa.

1.2 Motivação

Os problemas de planejamento que consideram recurso adicional buscam refletir melhor a realidade dos ambientes de produção, principalmente aqueles relacionados a processos manufaturados em que há demanda por recursos limitados externos. Com isso, novos problemas surgem a cada momento, principalmente com a crescente complexidade dos processos automatizados impulsionados pelos avanços tecnológicos e uma necessidade cada vez maior de uso de recursos naturais de forma sustentável.

Esta dissertação aborda conceitos e propõe métodos que podem ser estendidos para outras aplicações além da estudada neste trabalho. O método proposto é implementado com o intuito de ser adaptável a diferentes metaheurísticas, sendo flexível e permitindo a sua incorporação em novos estudos, testes de adequação e combinação com diferentes técnicas.

1.3 Limitações e Diferencial

Neste trabalho apenas o objetivo de minimizar o *makespan* é estudado, utilizando, para tal, heurísticas e metaheurísticas clássicas. Embora exista a possibilidade de desenvolver configurações mais especializadas das técnicas escolhidas, a utilização de meios mais generalizados contribui para uma boa validação do método, em especial para o método de reparo de soluções infactíveis proposto. Na busca de validar um procedimento auxiliar, de aspecto secundário (busca local, reparo), é fundamental que o procedimento primário (metaheurística) seja o mais generalizado e padronizado possível.

As instâncias utilizadas se resumem em um único grupo utilizado em trabalhos relevantes da literatura. Os resultados de trabalhos da literatura foram usados como *benchmark* para avaliação dos resultados deste estudo, considerando-se, no entanto, as diferenças dos objetivos específicos de cada abordagem. A finalidade deste estudo se concentra em validar e avaliar a potencialidade do projeto de metaheurísticas como um todo, não se preocupando em avaliar o método de reparo especificamente, embora este seja uma contribuição importante do trabalho. As metaheurísticas avaliadas aqui podem apresentar resultados não competitivos quando comparado a abordagens metaheurísticas de estado da arte.

O diferencial deste trabalho se concentra em desenvolver operadores específicos para um método de reparo e avaliar resultados gerais. Estes operadores se concentram apenas em reparos referentes à soluções infactíveis em relação à restrição de recurso adicional. Uma característica de reparo proposta por meios desses operadores consiste em reparar uma solução infactível com o mínimo de piora possível do *makespan*. Outros operadores poderão ser igualmente propostos, com características ainda mais específicas ou mais adaptadas a um determinado contexto.

1.4 Organização do Texto

O restante do texto da dissertação está organizado segundo os comentários de cada capítulo a seguir. O Capítulo 2 apresenta o problema, a partir de uma visão mais detalhada e teórica, por meio de definições, modelo matemático e trabalhos correlatos da literatura. O Capítulo 3 apresenta o embasamento teórico utilizado para o desenvolvimento deste trabalho. O Capítulo 4 mostra, de forma detalhada, a principal contribuição do estudo, com os algoritmos propostos como abordagens específicas para resolução do problema. No Capítulo 5 são detalhados os experimentos computacionais realizados, juntamente com uma análise mais específica dos resultados obtidos. Por fim, no Capítulo 6 é feita uma análise geral e apresentadas as conclusões obtidas com a realização do trabalho.

Capítulo 2

Caracterização do Problema

Este Capítulo apresenta o problema abordado nesta dissertação. Está organizado da seguinte forma. A Seção 2.1 descreve uma visão geral do problema de sequenciamento de tarefas, com ênfase em problemas que utilizam restrições de recurso adicional. A Seção 2.2 apresenta a caracterização do problema proposto para o estudo, sua definição e formulação matemática, e, por fim, um exemplo.

2.1 Problema de Sequenciamento de Tarefas

Nesta seção, discutiremos as características gerais de problemas de sequenciamento de tarefas de uma forma generalizada, como notações, os ambientes de produção, o envolvimento de recurso adicional e critérios de desempenho.

2.1.1 Características Gerais

Em Graham et al. (1979) foi introduzida a notação $\alpha \mid \beta \mid \gamma$ para descrever as principais características de um problema de sequenciamento de tarefas, na forma:

- O campo α descreve as características do ambiente das máquinas;
- O campo β descreve as características do processamento das tarefas;
- o campo γ especifica a medida de desempenho pela qual uma solução é avaliada.

Como exemplo do uso dessa notação, considere um ambiente no qual é dado um conjunto $N = \{1, \ldots, n\}$ de tarefas em um determinado ambiente, composto por um conjunto $M = \{1, \ldots, m\}$ de máquinas. Um problema de sequenciamento consiste, então, em organizar as tarefas no conjunto de máquinas, de modo a minimizar ou maximizar um certo objetivo. Neste ambiente, a notação $Rm \mid s_{ijk} \mid C_{max}$ é utilizada para representar um problema de sequenciamento de máquinas paralelas não relacionadas (ou seja, $\alpha = Rm$), em que a transição entre duas tarefas j e k, adjacentes na sequência de tarefas atribuídas a uma máquina i, exige um tempo de preparação s_{ijk}, que depende da sequência e da máquina considerada (portanto, $\beta = s_{ijk}$), tendo, como objetivo, a minimização do máximo instante de conclusão das tarefas (assim, $\gamma = C_{max}$).

A seguir são apresentados tipos de problemas clássicos de sequenciamento de tarefas e suas notações, segundo Graham et al. (1979).

2.1.1.1 Ambientes de produção: Máquinas Paralelas

Em ambientes de sequenciamento de tarefas em máquinas paralelas, cada tarefa é executada uma única vez em uma única máquina dentre as disponíveis. As máquinas podem ser classificadas como idênticas, uniformes ou não-relacionadas. Em máquinas idênticas ($\alpha = Pm$), o tempo de processamento de uma tarefa j, representado por p_j, é o mesmo em qualquer máquina. Em máquinas uniformes ($\alpha = Qm$), o tempo de processamento é dado por $p_{ij} = q_i p_j$ para um dado fator de velocidade q_i de M_i ($i = 1, \ldots, m$). Ou seja, os tempos de processamento de cada tarefa são proporcionais à velocidade da máquina em que a tarefa for alocada. Por fim, para máquinas não-relacionadas ($\alpha = Rm$), os tempos de processamento p_{ij} de cada tarefa j variam de acordo com a máquina i na qual a tarefa é atribuída.

Outro ponto a ser observado é a elegibilidade de máquinas, podendo estar presente nos mais diversos ambientes de planejamento de produção. Isso significa que nem todas as tarefas podem ser processadas por todas as máquinas. Esta situação é representada por $\beta = M_{ij}$.

Também é possível que alguma máquina não esteja disponível e/ou que alguma tarefa não possa ser iniciada a partir do instante inicial do horizonte de planejamento. Uma máquina pode estar em utilização em um outro processo previamente planejado ou ainda estar em manutenção; sendo assim, só estará disponível para o sequenciamento a partir de um instante de tempo r_i. Estas são situações em que os instantes de liberação tanto de máquinas quanto de tarefas não coincidem com o início do horizonte de planejamento e são identificados por $\beta = r_i$.

2.1.1.2 Ambientes de produção: *Flowshop*, *Jobshop*, *Openshop*

Há ambientes de planejamento de produção, de ocorrência muito comum, em que o processamento de uma tarefa envolve a realização de diferentes operações, cada uma delas em uma determinada máquina (Vallada e Ruiz, 2011). Os ambientes em questão apresentam características específicas, como:

(i) tipo de fluxo das tarefas ao longo das máquinas;

(ii) relações de precedência entre as operações de cada tarefa;

(iii) numero de máquinas em cada estagio;

(iv) possibilidade (ou não) de esperar entre operações consecutivas de uma dada tarefa.

Em ambientes *Flowshop*, todas as tarefas são compostas por um mesmo número de operações e a sequência na qual as máquinas são requisitadas para processar as operações é a mesma para qualquer tarefa. O tempo para processamento da k-ésima operação da tarefa j é denotado por p_{jk}. Um caso particular é aquele em que $p_{jk} = 0$ para o conjunto de operações de algumas tarefas. Isso significa que algumas tarefas saltam alguns estágios. Uma referência a este caso particular é encontrada em Ruiz

et al. (2008), sendo denominado *Flowshop Flexível* ($\alpha = FFm$). Por outro lado, caso exista um conjunto de máquinas paralelas em um dado estágio, o problema passa a ser classificado como *Flowshop Híbrido* e sua representação é dada por $\alpha = HFm$.

Em ambientes *Jobshop* ($\alpha = Jm$), cada tarefa requisita as máquinas em sequências específicas para o processamento das operações. Também é possível que tarefas diferentes sejam compostas por diferentes números de operações.

Em um ambiente *Openshop* ($\alpha = Om$), as sequências de processamento das operações de cada tarefa não estão definidas *a priori*, como acontece nos ambientes *Flowshop* e *Jobshop*. Dadas algumas sequências possíveis para cada tarefa, a solução de um problema *Openshop* passa por estabelecer, para cada tarefa, a sequência de processamento a ser empregada.

Em quaisquer destes ambientes é possível a imposição de uma restrição segundo a qual não pode haver tempo de espera (*No-Wait*) na transição entre dois estágios subsequentes de uma tarefa. Esta restrição, representada por $\beta = nwt$, ocorre, segundo Pinedo e Hadavi (1992), com muita frequência em ambientes *Flowshop*. O processo de laminação de aço, em que se deve evitar ao máximo o resfriamento da chapa entre dois estágios consecutivos da laminação, é um exemplo prático de aplicação da restrição *No-wait*. Um exemplo na direção oposta é apresentado em Ruiz et al. (2008) e refere-se a uma indústria de produção de cerâmica em que, após passar por um forno para secagem, a peça deve aguardar um tempo para resfriamento, antes da aplicação do verniz.

2.1.2 Recurso Adicional

Segundo Edis et al. (2013), a maioria dos estudos de sequenciamento em máquinas paralelas consideram a máquina como único recurso. No entanto, na realidade dos ambientes de produção, as tarefas demandam recursos adicionais. Nessa seção são apresentadas classificações, notações e aspectos teóricos da restrição de recurso no sequenciamento de tarefas em máquinas paralelas.

Garey e Grehem (1973) introduz um elemento adicional de realismo no modelo de planejamento, postulando a existência de um conjunto de "recursos", na forma de um sistema composto de n processadores idênticos abstratos. A função deste sistema é processar um conjunto $N = \{1, \ldots, n\}$ de tarefas, nas quais o seu processamento é sujeito à disponibilidade de um conjunto de recursos $\mathbb{R} = \{R_1, \ldots, R_S\}$.

Segundo Błażewicz et al. (2007), recursos podem ser classificados como *tipo* e *categoria*. A classificação por tipo leva em consideração as funções cumpridas pelo recurso. Recursos de mesmo tipo são assumidos que cumprem as mesmas funções. A classificação por categoria é considerada a partir de dois pontos de vista. No primeiro ponto de vista, as categorias são diferenciadas pela limitação de recursos, sendo normalmente classificadas em três categorias definidas como:

(i) recursos renováveis: seu **uso total** é limitado e temporariamente disponível em cada instante de processamento, sendo recomposto automaticamente após a sua utilização;

(ii) recursos não-renováveis: neste caso, o **consumo total do recurso é limitado e**, caso o recurso seja consumido pelo processamento de uma tarefa, ele

não é recomposto após o processamento e não pode ser utilizado novamente (Słowiński, 1980);

(iii) recursos duplamente limitados, em que o uso e o consumo total de recursos são considerados.

No segundo ponto de vista, uma categoria de recurso é distinguida pela divisibilidade do recurso, sendo classificadas como *discreta* e *contínua*. O recurso discreto é distribuído por quantidades unitárias de um conjunto finito de elementos. O recurso contínuo, por outro lado, é alocado em uma certa quantidade por um intervalo de tempo, de forma arbitrária e sem um conhecimento prévio.

Retomando o esquema de notações introduzido anteriormente, focaremos a descrição de recurso adicional denotado pelo parâmetro $\beta_2 \in \{\emptyset, res\ \lambda\delta\rho\}$, em que:

- $\beta_2 = \emptyset$: sem restrição de recurso;
- $\beta_2 = res\ \lambda\delta\rho$: há restrição de recurso especificada.

A notação $\lambda\delta\rho \in \{\cdot, k\}$ representa, respectivamente, o número de tipos de recurso, limite de recurso e requisitos de recurso. Assim:

- Se λ é um inteiro positivo, então a quantidade de recursos é constante e igual a λ; se $\lambda = \cdot$, então a quantidade de recursos será fornecida na entrada;
- Se δ é um inteiro positivo, então a quantidade de recursos é constante e igual a δ; se $\delta = \cdot$, então todos os recursos serão fornecida na entrada;
- Se ρ é um inteiro positivo, então todas as demandas de recursos serão limitadas à ρ; se $\rho = \cdot$, então não haverá limites especificados.

Kellerer e Strusevich (2008) estende o esquema de classificação, usando *Bi, Int* e *Lin* para o campo β, para especificar casos em que o tempo de processamento é dependente do recurso:

- *Bi*: refere-se à alocação binária de recurso, ou seja, se a tarefa $i \in N$ não recebe o recurso, o tempo de processamento se mantém igual a p_i; caso contrário, o tempo de processamento se torna $p_i - \pi_i$;
- *Int*: representa a alocação inteira do recurso. Se a tarefa $i \in N$ recebe τ unidades de recurso, o tempo de processamento atual é igual a $p_{i\tau}$;
- *Lin*: reforça que o tempo de processamento atual depende linearmente do número de unidades de aceleração da alocação do recurso para a tarefa, ou seja, se à tarefa $i \in N$ são dadas τ unidades do recurso, logo, o tempo de processamento atual será igual a $p_{i\tau} = p_i - \tau \times \pi_i$.

2.1.3 Critérios de desempenho

Os critérios de desempenho são os objetivos que um modelo de otimização do problema busca maximizar ou minimizar. Na notação de (Graham et al., 1979), são representados pelo terceiro campo (γ).

Um problema de planejamento pode ser estudado sem necessariamente apresentar um critério de desempenho em sua definição. Neste caso, o campo γ seria representado por $\gamma = \emptyset$ como proposto em (Blazewicz, 1981).

O critério de desempenho mais estudado em ambientes de máquinas paralelas não relacionadas é a minimização do *makespan*, representado por $\gamma = C_{\max}$ (Fanjul-Peyro et al., 2017; Diana et al., 2021; Fanjul-Peyro, 2020; Ruiz e Andrés-Romano, 2011; Vallada e Ruiz, 2011; Vallada et al., 2019). Outra função objetivo muito estudada é a de minimização do tempo corrido total (*total flow time* – $\sum C_i$). Este critério busca minimizar a soma dos tempos finais de todas as máquinas, ou a média dos tempos finais, sendo mencionado em diversos trabalhos, como Błażewicz et al. (1987); Edis et al. (2008); Edis e Oguz (2011). Já a notação $L_{\max}$ representa o atraso máximo (*maximum lateness*), dado por $L_{\max} = \max_j \{C(T_j) - d(T_j)\}$, em que $d(T_j)$ (*due date*) representa a data de vencimento da tarefa j. Este critério busca minimizar o maior atraso $L_{\max}$ (Błazewicz et al., 1993; Blazewicz et al., 1983).

Outros critérios conhecidos dentro do ambiente de planejamento envolvendo recurso adicional passam por $\sum U_j$, em que U_j irá valer 1 se a tarefa j é atrasada, e valerá 0, caso contrário (Ruiz-Torres et al., 2007). Em critérios como $\sum w_j U_j$, em que w_j representa um peso atribuído à tarefa j, denotando a sua importância em relação às demais tarefas, o indicador de atraso U_j é relacionado com o peso da tarefa w_j, quando o peso é considerado no problema. Da mesma forma, os critérios $\sum T_j$ e $\sum w_j T_j$ representam, respectivamente, a soma dos atrasos da tarefa j e a soma dos atrasos de T_j relacionados aos pesos w_j atribuídos às tarefas atrasadas (Brucker e Krämer, 1996).

2.2 Problema abordado

A Seção 2.2.1 descreve as características principais do problema que é objeto de estudo desta dissertação. Em seguida, na Seção 2.2.2, é apresentado um exemplo do problema descrito na Seção 2.2.1. Por fim, na Seção 2.2.3, é apresentado um modelo matemático para o problema abordado.

2.2.1 Definição do Problema UPMR

Neste trabalho estudamos o problema UPMR (*Unrelated Parallel Machine Scheduling Problem with Additional Resources*). Este problema de sequenciamento tem, como entrada, um conjunto $M = \{1, 2, \ldots, m\}$ de máquinas e um conjunto $N = \{1, 2, \ldots, n\}$ de tarefas para serem atribuídas e sequenciadas nas máquinas presentes em M. Este sequenciamento é restrito por R_{max} unidades de recursos por unidade de tempo; cada tarefa j demanda $p_{ij} \in \mathbb{Z}^+$ unidades de tempo para ser processada na máquina i; e $r_{ij} \in \mathbb{Z}^+$ unidades de recurso necessários, por unidade de tempo, para a tarefa j ser finalizada na máquina i.

O objetivo é minimizar o *makespan* ($C_{\max}$) e satisfazer às seguintes restrições:

(a) uma mesma máquina não pode processar mais que uma tarefa ao mesmo tempo;

(b) cada tarefa precisa ser processada por exatamente uma máquina;

(c) o processamento de uma tarefa não pode ser interrompido antes que seja completado;

(d) em nenhum instante de tempo poderá ser utilizado mais do que R_{max} unidades de recurso.

A partir desta descrição, tem-se as seguintes notações:

- p_{ij}: tempo de processamento da tarefa j na máquina i;
- r_{ij}: recurso consumido pela tarefa j para o processamento na máquina i;
- R_{max}: recurso adicional máximo disponível por unidade de tempo;
- C_i: tempo de processamento final da máquina i;
- S: solução factível.

Assim, esta dissertação aborda o problema de sequenciamento de tarefas em máquina paralelas não relacionadas ($\alpha = R$), utilizando apenas um tipo de recurso ($\beta = res1 \quad \cdot \quad \cdot$), com o objetivo de minimizar o *makespan* ($\gamma = C_{max}$). Seguindo a notação de Graham et al. (1979) descrita nas seções acima, este problema pode ser definido como $Rm \mid res1 \quad \cdot \quad \cdot \mid C_{max}$.

2.2.2 Exemplo do Problema UPMR

Considere uma instância de UPMR com duas máquinas ($m = 2$), cinco tarefas ($n = 5$) e cinco unidades de recurso adicional ($R_{\max} = 5$), de acordo com as matrizes abaixo:

$$P = \begin{array}{c} \\ M_1 \\ M_2 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_4 & N_5 \end{array} \\ \begin{pmatrix} 1 & 2 & 2 & 2 & 1 \\ 2 & 1 & 2 & 3 & 1 \end{pmatrix} \end{array} \tag{2.1}$$

$$R = \begin{array}{c} \\ M_1 \\ M_2 \end{array} \begin{array}{c} \begin{array}{ccccc} N_1 & N_2 & N_3 & N_4 & N_5 \end{array} \\ \begin{pmatrix} 4 & 3 & 3 & 4 & 2 \\ 2 & 5 & 4 & 2 & 5 \end{pmatrix} \end{array} \tag{2.2}$$

A matriz 2.1, chamada P, apresenta os tempos de processamento p_{ij} de cada tarefa N_i em cada máquina M_j; a matriz 2.2, chamada R, indica a quantidade de recurso r_{ij} que cada tarefa N_i necessita durante seu processamento na máquina M_j.

A Figura 2.1 apresenta um sequenciamento de cinco tarefas em duas máquinas. Na máquina 1 (*Máq. 1*), estão sequenciadas as tarefas 1, com tempo de execução igual a 1 e que demanda 4 unidades de recurso, e, em seguida, a tarefa 4, com tempo de execução igual a 2 e que demanda 4 unidades de recurso. Na segunda

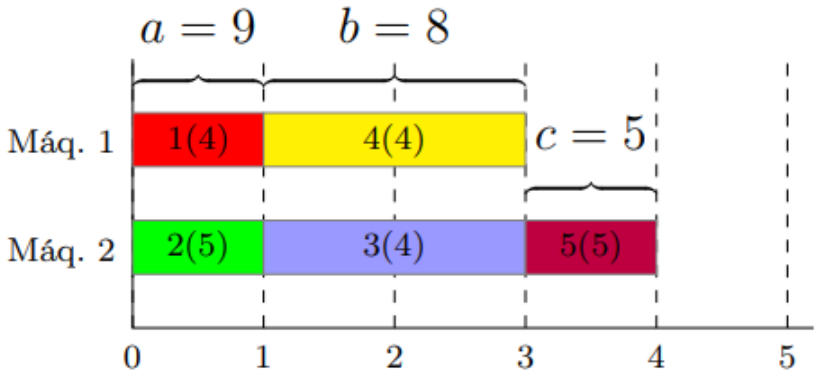


Figura 2.1: Sequenciamento.

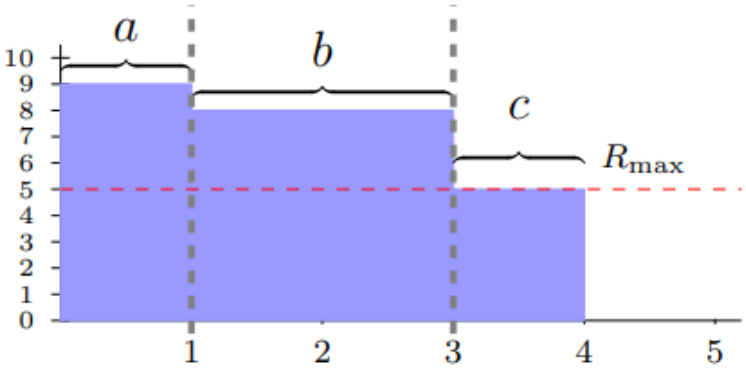


Figura 2.2: Consumo de Recurso.

máquina (*Máq. 2*), estão sequenciadas 3 tarefas ($\{2, 3, 5\}$), com respectivos tempos de execução iguais a $\{1, 2, 1\}$ e valores de recurso para cada uma das tarefas iguais a $\{5, 4, 5\}$. Ainda nessa figura estão representados os valores resultantes da soma total dos recursos para cada instante de tempo, ou seja, $a = 9$ unidades de recurso para o primeiro instante de tempo, $b = 8$ para o intervalo de tempo 1 até 3 e $c = 5$ do instante 3 até o instante 4. Na Figura 2.2, o gráfico representa o consumo total de recurso por unidade de tempo. As linhas pontilhadas na vertical separam as regiões representadas pelas somas de recursos a, b e c e a linha pontilhada horizontal indica o limite de recurso máximo para esta instância, dado por $R_{\max} = 5$. Quanto aos tempos finais de cada máquina, temos:

$$C_1 = p_{11} + p_{12} = 1 + 2 = 3$$

$$C_2 = p_{21} + p_{22} + p_{23} = 1 + 2 + 1 = 4$$

Portanto, $C_{\max} = \max\{C_1,\ C2\} = 4$. Toda a região acima da linha pontilhada na Figura 2.2 indica uma demanda de recurso indisponível, o que torna o sequenciamento infactível.

2.2.3 Formulação Matemática

O modelo matemático para este problema, introduzido por (Edis e Oguz, 2012a) e adaptado em (Fanjul-Peyro et al., 2017), é apresentado a seguir.

Sejam x_{ijk} as variáveis de decisão do problema, na forma:

- $x_{ijk} = 1$ se a tarefa j é sequenciada na máquina i e completa a execução no tempo k, e zero, caso contrário. Essa variável somente existe para $k \geq p_{ij}$.

O modelo matemático é dado, portanto, por:

$$\min \quad C_{\max} \tag{2.3}$$

$$\text{suj. a} \quad \sum_{i} \sum_{k \geq p_{ij}} k x_{ijk} \leq C_{\max}, \quad \forall j \tag{2.4}$$

$$\sum_{i} \sum_{k \geq p_{ij}} x_{ijk} = 1, \quad \forall j \tag{2.5}$$

$$\sum_{i} \sum_{s \in \{\max\{k, p_{ij}, \ldots, k+p_{ij}-1\}\}} x_{ijs} \leq 1, \quad \forall i, k \tag{2.6}$$

$$\sum_{i} \sum_{j} \sum_{s \in \{\max\{k, p_{ij}, \ldots, k+p_{ij}-1\}\}} r_{ij} x_{ijs} \leq R_{\max}, \quad \forall k \tag{2.7}$$

Neste modelo, $C_{\max}$ é o *makespan* e o objetivo a ser minimizado. A restrição (2.4) determina o *makespan*. A restrição (2.5) determina que cada tarefa deve ser atribuída a somente uma máquina e é executada apenas uma única vez. A restrição (2.6) certifica que a mesma máquina não processa mais que uma tarefa por vez. A restrição (2.7) determina que, em nenhum instante, mais do que $R_{\max}$ unidades de recursos podem ser utilizados.

2.3 Trabalhos relacionados

Esta seção apresenta um levantamento bibliográfico, mostrando os trabalhos relacionados ao tema abordado, especialmente trabalhos que tratam problemas com restrição de recursos adicionais.

Em Davis e Heidorn (1971) é apresentado um caso bastante elaborado da utilização de recursos em um ambiente de planejamento em redes. Este caso em particular oferece uma demanda chamada de *"multiresource"*, em que três diferentes tipos de recursos são requeridos por cada tarefa, sendo que a disponibilidade e a limitação de cada tipo de recurso variam entre si. Em Weglarz et al. (1977), o mesmo problema é solucionado por meio de uma método simplex adaptado. O problema inclui uma quantidade de tipos de recursos, um vetor com limites para cada tipo de recurso e uma matriz R com a quantidade de recurso por atividade. Ou seja, para cada atividade da rede, existe um tipo e quantidade de recurso, além do vetor que estipula o limite de cada recurso. O método simplex é também aplicado, com a subrotinas ARSME, dentre outras conhecidas durante o processo de busca, como GEN, KFIND e PRISET. Em Garey e Grehem (1973) é apresentada a preocupação de tratar problemas de planejamento considerando um conjunto de recursos limitados, trazendo mais realismo para o modelo abstrato padrão de um sistema de multiprocessamento. O trabalho aponta limites por meio de teoremas e provas, considerando diferentes conjuntos de recursos. Em (Garey e Johnson, 1975) o mesmo problema tem um estudo mais aprofundado, avaliando a complexidade computacional dos modelos de (Garey e Grehem, 1973). O estudo apresenta casos especiais nos quais é

possível executar algoritmos em tempo polinomial de ordem inferior. No entanto, todos os principais resultados mostram que a maioria dos casos, mesmo com um único recurso, são problemas das classe NP-completo e, portanto, tão difícil quanto o conhecido Problema do Caixeiro Viajante.

Em Błażewicz (1979) é proposto um algoritmo generalizado de complexidade $O(n^2)$, permitindo tempos de término não iguais para o problema de planejamento em máquinas idênticas com *dead-line* e tarefas com tempo de término e restrição de recurso. Embora o problema já fosse amplamente estudado, o autor introduz o trabalho com a utilização de recurso. Em Slowinski (1981) se menciona a preferência de tarefas em ambiente de planejamento com recurso adicional em máquinas independentes. Em Blazewicz et al. (1983) são apresentadas classificações e a complexidade de problemas de sequenciamento sujeitos à restrição de recursos. Em Błażewicz et al. (1986), o problema de sequenciamento de tarefas em dois processadores com *dead-lines* também é abordado com recurso adicional. O trabalho tem o aspecto teórico e aborda a complexidade do problema usando técnicas de redutibilidade do problema ONE-IN-THREE 3 SAT como provas.

Em Błażewicz et al. (1987) é apresentado um estudo sobre o problema de sequenciamento de tarefas em processadores paralelos com restrição de recurso e com o objetivo de minimizar o tempo corrido médio (*Mean Flow-Time*). Três variações de abordagem do problema são estudadas, sendo diferenciadas por número de processadores e utilização do recurso. São apresentadas provas para um algoritmo $O(n^3)$, capaz de obter um sequenciamento ótimo, e a prova de NP-Difícil para as duas últimas abordagens. Em Błażewicz et al. (2007) é dada uma apresentação didática do problema de planejamento, considerando os principais pontos e um capítulo é dedicado à restrição de recurso adicional. Em (Daniels et al., 1996), duas versões da alocação de recurso são introduzidas. A primeira, denominada estática, ocorre quando a decisão de alocação é feita e mantida durante todo o processo. A segunda, denominada dinâmica, ocorre quando cada cada recurso pode ser realocado no decorrer do processo. Para cada versão, é apresentada uma formulação matemática e um estudo de complexidade do problema, além de uma abordagem heurística e uma exata. Outros estudos abordando *scheduling* com recurso flexível (*flexible-resource*) são apresentados em Daniels et al. (1997) e Daniels et al. (1999).

Problemas de sequenciamento têm sido fortemente estudados nas últimas décadas, como podemos ver em Kravchenko e Werner (2011). Apesar disto, variantes envolvendo recursos adicionais ainda são pouco investigadas. Esta parte do levantamento da literatura apresentará trabalhos mais recentes que tratam problemas de sequenciamento com recurso adicional. Ruiz e Andrés-Romano (2011) propõem um problema de máquinas paralelas não relacionadas com tempos de processamento que depende da atribuição de recursos. O limite disponível de recursos adicionais não é considerado. Ruiz-Torres et al. (2007) estuda o problema de máquinas paralelas uniformes sem tempo de *setup* no qual o tempo de processamento das máquinas depende do recurso atribuído (também sem limites). Edis e Oguz (2012b) usam modelos de programação inteira para resolver o problema de máquinas paralelas com recurso flexível, nos quais a adição de recursos pode acelerar os tempos de processamento das máquinas. Edis e Oguz (2012a) propõem soluções para um problema real de restrição de recursos em máquinas paralelas usando modelos de programação inteira e programação de restrições. Edis et al. (2013) apresenta uma revisão de

literatura, com discussões sobre trabalhos publicados envolvendo o sequenciamento de tarefas em máquinas paralelas com recursos adicionais. O trabalho discute cinco categorias: (i) ambiente das máquinas; (ii) recurso adicional; (iii) função objetivo; (iv) complexidade dos problemas; e (v) métodos utilizados para a resolução. Em Bitar et al. (2016) é proposto um algoritmo memético para resolver um problema de máquinas paralelas não relacionadas com recurso auxiliar em uma oficina de fotolitografia de uma fábrica de semicondutores (problema também abordado em Cakici e Mason (2007)). Dois critérios de otimização foram considerados separadamente: (i) a minimização do tempo corrido médio e (ii) a maximização do número de produtos processados. Em Zheng e Wang (2016), um algoritmo da mosca da fruta (*Fruit Fly Optimization Algorithm*) adaptativo em dois estágios é aplicado na resolução do problema.

Considerando o ambiente UPMR abordado neste trabalho, Fanjul-Peyro et al. (2017) apresentam dois modelos matemáticos, um já existente na literatura e o segundo proposto pelos autores, baseado em uma formulação do problema *bin-packing*. Os modelos matemáticos são testados sobre um conjunto de instâncias de pequeno porte. Três metaheurísticas são apresentadas para resolver instâncias de médio porte. Cada uma das metaheurísticas utiliza internamente as estratégia dos MIPs propostos inicialmente no processo de busca, totalizando 6 diferentes aplicações de metaheurísticas. Os resultados de todos os testes são comparados entre si, mostrando a superioridade das estratégias metaheurísticas sobre os modelos matemáticos. Em Villa et al. (2018), novas metaheurísticas são propostas e testadas sobre as instâncias de Fanjul-Peyro et al. (2017) e incluindo um grupo de 600 instâncias de grande porte. Em Vallada et al. (2019), são apresentadas três metaheurísticas (i.e. *Scatter Search Algorithm*, *Enriched Scatter Search Algorithm* e *Enriched Iterated Greedy Algorithm*) comparadas aos métodos propostos por Fanjul-Peyro et al. (2017).

Outros trabalhos trazem diferentes abordagens para problemas variantes do UPMR, como em Fleszar e Hindi (2018) e Arbaoui e Yalaoui (2018), com o uso de programação de restrições. Fanjul-Peyro (2020) apresentam um modelo matemático e algoritmos exatos para o problema de máquinas paralelas não relacionadas com recursos adicionais sobre o processamento e *setup*. Yepes-Borrero et al. (2020) propõe um algoritmo GRASP para resolver o problema com tempos de *setup* e recursos adicionais limitados para o *setup*.

Capítulo 3

Fundamentação Teórica

Este Capítulo apresenta os fundamentos teóricos necessários para a elaboração desta dissertação. A Seção 3.1 introduz o conceito de estruturas de vizinhanças. A Subseção 3.1.1 apresenta a definição de movimentos e os tipos mais utilizados em problemas de sequenciamento em ambientes de máquinas paralelas não relacionadas. Nas Seções 3.2 e 3.3 são apresentados conceitos e os principais tipos de heurísticas, além de conceitos básicos de metaheurísticas, dando ênfase à metaheurística GRASP padrão e da sua variação reativa utilizada nesta dissertação.

3.1 Estruturas de vizinhança

Uma estrutura de vizinhança é definida como segue. Seja um espaço de busca S, uma solução $s \in S$ e uma função N que realiza uma modificação em s gerando outra solução $s' \in S$; então, a vizinhança de s é o subespaço V de todas as possíveis soluções $s' \in S$ resultantes da aplicação da função N. Portanto, $V = N(s)$, sendo $N(s) \subseteq S$. De um ponto de vista heurístico, a modificação realizada pela função N sobre s gerando s' é chamado de *movimento*, se referindo a uma modificação m que transforma s em s'. Essa operação é representada por $s' \leftarrow s \oplus m$.

3.1.1 Movimentos em problemas UPMR

Para o problema de máquinas paralelas não relacionadas com recurso adicional (UPMR), as estruturas de vizinhança encontradas na literatura normalmente são baseados em movimentos de "*troca*" e "*inserção*".

Na literatura de sequenciamento de tarefas, um movimento de troca consiste em, dadas duas tarefas j e k, tais que $j \neq k$, as duas tarefas têm suas posições trocadas entre si no sequenciamento.

Já um movimento de inserção consiste em, dada uma tarefa j, esta é removida da posição p que ocupa no sequenciamento e inserida na posição $p' \neq p$ do sequenciamento.

Note que ambos os tipos de movimentos podem envolver uma ou mais máquinas. Quando o movimento se dá em apenas uma máquina, este é considerado um movimento do tipo "interno"; já quando ocorre a movimentação de tarefas entre máquinas diferentes, é considerado um movimento do tipo "externo".

Em Yepes-Borrero et al. (2020) três estruturas de vizinhança são exploradas, na forma de estruturas de troca externa, inserção externa e troca interna, já que o problema abordado envolve precedência de tarefas por tempos de preparação. Em Fanjul-Peyro e Ruiz (2010) tem-se a utilização de um número reduzido de estruturas de vizinhanças, sendo apenas duas baseadas em troca e inserção. Uma busca local restrita também é utilizada, com o objetivo de melhorar soluções reduzindo tempo computacional. Este método de busca local é baseado em um *Restricted Local Search* (*RLS*), que utiliza a escolha de forma aleatória de uma tarefa de uma determinada máquina e a troca com outra tarefa de uma determinada máquina seleciona ou a inseri em uma posição de outra máquina.

Em Pinheiro et al. (2020) tem-se o caso do uso de até seis estruturas de vizinhança, envolvendo movimentos com grupos de tarefas (*Batch Insertion* e *Batch Swap*), além de estruturas já conhecidas, como *Job Insertion* e *Job Swap*. Este trabalho compara os resultados das meta-heurísticas *Simulated Annealing* (SA) e *Iterated Greedy* (IG), tendo como objetivo minimizar o tempo de atraso no sequenciamento de tarefas com famílias de *setups* e restrição de recursos. Em Villa et al. (2018), duas estruturas de vizinhança são consideradas, quais sejam, inserções externas e trocas externas. No entanto, são aplicadas de diferentes formas: considerando recurso e não considerando recurso. Em todos os casos, a busca local explora as estruturas primeiramente a partir da máquina que define o *makespan* e, em seguida, a partir de todas as máquinas, incluindo a máquina que define o *makespan*. Em Vallada et al. (2019) também são utilizadas duas estruturas de vizinhanças baseadas em inserção e troca, sem considerar recurso. Adicionalmente, é utilizada a heurística de busca local restrita (RLS), como em Fanjul-Peyro e Ruiz (2010).

Deve-se notar que o incremento no número de estruturas de vizinhança não necessariamente incrementa o desempenho dos algoritmos que utilizam as mesmas. Essa questão é dependente das características e peculiaridades de cada problema.

3.2 Heurísticas

Um problema de otimização mono-objetivo de minimização pode ser definido como: dado um espaço dimensional S, no qual cada ponto $s \in S$ representa uma solução do problema, e uma função $f : S \rightarrow \mathbb{R}$ que avalia o desempenho de s, associando, assim, a cada solução $s \in S$ um valor real $f(s)$. O objetivo é encontrar uma solução $s^* \in S$, de modo que $f(s^*) \leq f(s)\ \forall\ s \in S$. Quando o espaço S apresenta natureza combinatória, grande parte desses problemas são classificados como NP-difíceis, não havendo algoritmos para a resolução em tempo polinomial.

Dentre inúmeros exemplos, um bem conhecido é o Problema do Caixeiro Viajante (PCV). O problema é descrito por um conjunto de n cidades, de modo que o caixeiro viajante deve sair de uma cidade de origem, visitar cada uma das $n-1$ cidades restantes uma única vez e retornar à cidade de origem percorrendo a menor distância possível. Ou seja, percorrer a rota fechada de comprimento mínimo passando uma única vez por cada cidade.

Para analisarmos a real dimensão de um problema combinatório, assumiremos, para este exemplo do PCV, que a distância entre uma cidade i até outra j seja simétrica, ou seja, $d_{ij} = d_{ji}$. Assim, o número total de rotas possíveis é $(n-1)!/2$.

Para 20 cidades, $n = 20$ tem-se 6×10^{16} rotas possíveis. Um computador que avalia uma rota em cerca de 10^{-8} segundos gastaria cerca de 19 anos para encontrar a melhor rota.

Segundo Gaspar-Cunha et al. (2012), heurísticas são procedimentos que tratam problemas de otimização sem dispor de garantias teóricas da qualidade da solução nem de que a solução ótima exata seja obtida, e nem tampouco de garantias de que a solução obtida tenha algum tipo de proximidade em relação à solução ótima exata. Em outras palavras, na prática, é suficiente encontrar uma "boa" solução para o problema, ao invés do ótimo global, que só seria encontrado após um considerável esforço computacional.

Nessa dissertação serão abordadas heurísticas construtivas e de refinamento, além das metaheurísticas, mais inteligentes e flexíveis, tratadas na Seção 3.3.

3.2.1 Heurística Construtiva

Uma heurística construtiva tem, como objetivo, construir uma solução, elemento por elemento. O critério de escolha de cada elemento varia de acordo com uma função de avaliação $g(.)$ adotada, que, por sua vez, depende do problema abordado. Uma solução inicial pode ser construída de forma *gulosa*, de modo que cada elemento tem seu valor estimado e somente o "melhor" é inserido a cada passo. Como uma exemplo genérico, seja o pseudocódigo apresentado no Algoritmo 1.

A construção também pode ser realizada de forma *aleatória*, de maneira que os elementos são inseridos na solução sem um critério pré-estabelecido de escolha. Nesse caso, os elementos são "sorteados" de forma aleatória para fazerem parte da solução. Construções aleatorizadas são formulações comuns e que podem ser combinadas com um determinado método guloso. O principal exemplo de estruturas desse tipo é apresentado em Feo e Resende (1995) e consiste na fase construtiva da metaheurística GRASP, em que essa combinação entre aleatoriedade e construção gulosa pode ser controlada por um fator α.

Algoritmo 1: Pseudocódigo do algoritmo de Construção Gulosa

Entrada: $g\,(.)$
Saída: s
Inicialize o conjunto C de elementos candidatos;
$s \leftarrow \emptyset$;
início
  enquanto $(C \neq \emptyset)$ **faça**
    $g(t_{melhor}) = melhor\,\{g(t) \mid t \in C\}$;
    $s \leftarrow s \cup \{t_{melhor}\}$;
    Atualize o conjunto C de elementos candidatos;
  fim
  retorna s;
fim

3.2.2 Heurística de Refinamento

Partindo de uma solução s conhecida, uma heurística de refinamento consiste em explorar, por uma estrutura de vizinhança $N(s)$ definida, todos os vizinhos da solução s, buscando, segundo um critério de avaliação, soluções vizinhas que seja melhores que s. As heurísticas clássicas de refinamento são os métodos de Descida Completa *Best Improvement*, Primeira Melhora (*First Improvement*) e Descida Randômica (*Random Improvement*).

O método de subida/descida completa consiste em, dada uma solução s e uma estrutura de vizinhança $N(s)$, explora-se todos os vizinhos s' desta vizinhança e a solução s' que acarretar o melhor movimento de melhora é realizada; assim, s' se torna a nova solução s e uma nova iteração é iniciada. O método é executado até que não seja mais encontrada nenhuma solução s' que seja melhor que s e o método é finalizado, retornando um ótimo local de $N(s)$.

Em alguns casos, esse procedimento pode ser considerado caro computacionalmente. Nessa situação, a alternativa disponível é a descida através do primeiro movimento de melhora (*First Improvement*). Nesse método de descida/subida, os vizinhos da solução s são explorados um a um, até que a primeira melhora seja encontrada e a exploração de todos os vizinhos de s só ocorrerá no pior caso.

Outra maneira de aplicação para o método de descida/subida e que também não explora a vizinhança $N(s)$ completamente é o método de descida randômico. Nesse método, um vizinho s' de s é escolhido aleatoriamente. Se a solução s' retornada representa melhora em relação a s, a solução é aceita; caso contrário, uma nova solução vizinha de s é gerada randomicamente em $N(s)$. O método é executado até que um número predefinido de iterações do algoritmo, ou iterações sem melhora, seja alcançado.

3.2.2.1 *Variable Neighborhood Descent* - VND

A heurística *Variable Neighborhood Descent* - (VND), introduzida por Mladenović e Hansen (1997), é um método de busca local que realiza busca no espaço de soluções através de trocas sistemáticas das estruturas de vizinhança, retornando sempre para a primeira estrutura quando uma solução de melhora é encontrada.

Considerando um conjunto ordenado N, composto por k_{max} estruturas de vizinhança, deve-se escolher a primeira estrutura N_1 e explorá-la por meio de um método de descida. Assim que a primeira vizinhança tiver sido explorada, inicia-se a exploração da próxima vizinhança N_2. Caso haja melhora em s', esta se torna a nova solução s e o método reinicia a exploração da primeira vizinhança N_1; caso contrário, as demais estruturas serão exploradas na ordem definida, até que todas as k_{max} estruturas de vizinhança não apresentem melhora. O Algoritmo 2 apresenta um pseudocódigo do VND.

O algoritmo se baseia na ideia que, ao melhorar uma solução em uma estrutura de vizinhança, outras estruturas previamente exploradas podem apresentar novas melhoras a partir da nova solução de melhora encontrada.

Algoritmo 2: Pseudocódigo do algoritmo VND.

Entrada: $f(.), N(.), kmax, s$
Saída: s

```
início
    Seja r o número de estruturas diferentes de vizinhança;
    k ← 1; enquanto (k ≤ kmax) faça
        Encontre o melhor vizinho s' ∈ N^(k)(s);
        se (f(s') < f(s)) então
            s ← s';
            k ← 1;
        fim
        senão
            k ← k + 1;
        fim
    fim
    retorna s;
fim
```

3.3 Meta-heurísticas

Segundo Ribeiro (1996), as metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. Segundo os mesmos autores, as metaheurísticas se diferenciam das heurísticas convencionais por serem providas de mecanismos para tentar escapar de ótimos locais ainda distantes dos ótimos globais, também chamados de armadilhas de ótimos locais. As metaheurísticas destinadas a problemas de otimização combinatória podem ser classificadas em duas categorias, definidas de acordo com o princípio usado para exploração do espaço de soluções: metaheurísticas de busca local e metaheurísticas de busca populacional.

Nas metaheurísticas baseadas em busca local, a exploração do espaço de soluções é feita por estruturas de vizinhança, que são aplicadas a cada passo na solução corrente, gerando outra solução promissora em sua vizinhança. As metaheurísticas Busca Tabu (Glover, 1986), *Simulated Annealing* (Kirkpatrick et al., 1983), *Variable Neighborhood Search* (VNS) (Hansen et al., 1999) e *Iterated Local Search* (Lourenço et al., 2003) são exemplos dessa categoria.

Métodos baseados em busca populacional, por sua vez, consistem em manter um conjunto de boas soluções e combiná-las de forma a tentar produzir soluções ainda melhores. Algoritmos Genéticos (Holland, 1992), Meméticos (Moscato et al., 1989), Colônia de Formigas (Dorigo, 1992) (Dorigo et al., 1996) e PSO (*Particle Swarm Optimization*) (Eberhart e Kennedy, 1995) são exemplos de algoritmos dessa categoria.

3.3.1 GRASP

A metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) foi proposta em Feo e Resende (1995). A sua forma clássica consiste em um método iterativo de múltiplas partidas, na qual cada iteração consiste em uma fase de construção de uma solução viável, seguida de uma fase de refinamento ou busca local. Assim, a fase de busca local visa melhorar a qualidade das soluções a partir da solução construída na fase anterior. Por fim, a melhor solução encontrada de todas as iterações da metaheurística é retornada como resultado. Os únicos parâmetros a serem definidos na metaheurística GRASP são os parâmetros α, que calibra o quão guloso/aleatório será o processo de construção da solução, e o parâmetro critério de parada, que comumente é o número de iterações do algoritmo.

Algoritmo 3: Pseudocódigo do Algoritmo GRASP

```
   Entrada: iterMax, α
   Saída: s*
1  f(s*) ← +∞;
2  it ← 0;
3  início
4      enquanto (it < iterMax) faça
5          s' ← FaseConstrutiva(α);
6          s" ← BuscaLocal(s');
7          se (f(s") < f(s*)) então
8              s* ← s";
9          fim
10         it ← it + 1;
11     fim
12     retorna s*;
13 fim
```

O Algoritmo 3 apresenta um pseudocódigo genérico da metaheurística GRASP. Para valores pequenos de α, as soluções são construídas com um maior grau de "gulosidade". À medida que se aumenta o valor de α, se reduz o nível de "gulosidade" e aumenta-se o nível de aleatoriedade. Nota-se que a metaheurística GRASP iterativamente constrói uma solução s' (linha 5) e esta solução é submetida a uma tentativa de melhora por um procedimento de busca local (linha 6), dando origem à solução s''. Sempre é armazenada a melhor solução encontrada até o momento (linhas 7 e 8).

3.3.1.1 Fase construtiva

Na fase de construção, uma solução, não necessariamente factível, é iterativamente construída com um elemento por vez. Como é explicado em Feo e Resende (1995), a cada iteração do método de construção, a escolha do próximo elemento é determinada por uma ordenação da lista dos elementos candidatos, respeitando uma função gulosa. Essa função genericamente mede o benefício de selecionar cada elemento. Após a escolha de cada elemento, a lista de candidatos tem seus benefícios associados a cada candidato atualizada, o que caracteriza o procedimento como uma heurística adaptativa. O fator probabilístico do GRASP é caracterizado por uma escolha aleatória em uma lista restrita, composta pelos melhores candidatos e chamada de *Restricted Candidate List* (RCL). Essa característica do método de construção garante maior variabilidade nas soluções construídas, apesar de não garantir que uma determinada solução construída seja sempre a melhor possível em termos de beneficio.

Algoritmo 4: Pseudocódigo do Algoritmo da fase Construtiva GRASP

```
   Entrada: α
   Saída: s
1  início
2      Inicializa lista de candidatos LC;
3      s ← ∅;
4      enquanto (LC ≠ 0) faça
5          c^min ← min{c(e) | e ∈ LC};
6          c^max ← max{c(e) | e ∈ LC};
7          ConstrurLRC : LRC ← {e ∈ LC | c(e) ≤ c^min + α(c^max − c^min)};
8          e ← SelecionarAleatorio(LCR);
9          s ← s ∪ {e};
10         Atualiza lista de candidatos LC;
11         Reavalia o custo incremental c(e) para todo e ∈ LC
12     fim
13     retorna s;
14 fim
```

O Algoritmo 4 mostra a fase de construção do GRASP. A lista de candidatos (LC) e a solução a ser construída são inicializados nas linhas 2 e 3, respectivamente. O laço entre a linha 4 à linha 12 é repetido enquanto existir candidatos na lista restrita a serem inseridos na solução. Nas linhas 5 e 6 são obtidos os valores de custo mínimo e custo máximo associados aos candidatos da LC. Na linha 7 é construída a lista restrita de candidatos. Nas linhas 8 e 10, o elemento é escolhido aleatoriamente e integrado à solução. A lista de candidatos, então, é atualizada e os custos reavaliados nas linhas 10 e 11. Por fim, a solução s construída é retornada na linha 13.

3.3.2 *GRASP Reativo*

O procedimento GRASP Prais e Ribeiro (1999) levanta a discussão sobre a definição do parâmetro α em termos da qualidade e diversidade das soluções durante a fase de construção e os impactos deste parâmetro na eficiência do algoritmo. Prais e Ribeiro (2000) apresenta uma modificação do procedimento GRASP, chamada de GRASP Reativo, que propõe a construção da lista restrita de candidatos através do parâmetro α que se auto-ajusta com base em soluções previamente encontradas. Nesta proposta, não há um único valor de α pré-definido e, sim, um conjunto $\mathcal{A} = \{\alpha_1, \ldots, \alpha_m\}$, composto por m valores aceitáveis. A cada iteração da fase de construção, um valor de α é selecionado de forma aleatória em $\mathcal{A}$. Inicialmente todos os elementos de $\mathcal{A}$ têm chances equiprováveis de serem escolhidos. Toma-se p_i como probabilidade associada à escolha de α_i, para $i = 1, \ldots, m$ inicialmente como $p_i = 1/m$, $i = 1, \ldots, m$, o que corresponde a uma distribuição uniforme. O método, então, visa atualizar periodicamente a distribuição de probabilidade p_i, $i = 1, \ldots, m$ usando dados coletados durante as iterações já realizadas do procedimento. A atualização das probabilidade é feita a cada iteração, a partir do valor da eficiência da solução gerada pelo parâmetro α_i selecionado. Assim, cada solução construída a partir de um valor α_i, $i = 1, \ldots, m$ tem seu valor de função objetivo associado a α_i. Para a atualização das probabilidades p_i, é considerada a razão da melhor solução até então encontrada s^* pela média do custo de todas as soluções encontradas com $\alpha = \alpha_i$ durante a fase construtiva. A atualização dos valores de p_i é realizada a cada bloco de y iterações. Para isso, deve-se obter q_i, computando:

$$q_i = \left(\frac{f(s^*)}{A_i} \right)^{\delta}, \quad \forall i = 1, 2, \ldots, m \tag{3.1}$$

para $i = 1, \ldots, m$ e obter os valores de p_i, $i = 1, \ldots, m$, dado por

$$p_i = \frac{q_i}{(\sum_{j=1}^{m} q_j)}, \quad \forall i = 1, 2, \ldots, m \tag{3.2}$$

O Algoritmo 5 mostra as etapas do método padrão GRASP Reativo:

Algoritmo 5: Pseudocódigo do Algoritmo R-GRASP

```
   Entrada: iterGraspMax, y
   Saída: s*
1  início
2      f(s*) ← +∞;
3      iterGrasp ← 1;
4      Definir   A = {α_1, α_2, ..., α_m};
5      for i = 1 até m do
6          p_i ← 1/m; m_i ← 0;
7      end
8      enquanto (iterGrasp < iterMaxGrasp) faça
9          α ← Selecione α_i ∈ A com probabilidade p_i;
10         s_1 ← FaseConstrutiva(α);
11         s_2 ← BuscaLocal(s_1);
12         se (f(s_2) < f(s*)) então
13             s* ← s_2;
14         fim
15         se (it mod y = 0) então
16             Recalcular A_i, ∀i ∈ {1, 2, ..., m};
17             Recalcular q_i, ∀i ∈ {1, 2, ..., m};
18             Atualizar as probabilidades p_i, ∀i ∈ {1, 2, ..., m};
19         fim
20         iterGrasp ← iterGrasp + 1;
21     fim
22     retorna s*;
23 fim
```

As linhas de 1 a 7 inicializam as variáveis valor de função objetivo $f(s^*)$, contador de iterações $iterGrasp$, conjunto A de valores de α e o conjuntos de probabilidades com os valores *default* de p_i. O laço de repetição entre as linhas 8 e 21 se dá enquanto o critério de parada não for satisfeito. Nesse laço, tem-se, na linha 9, a escolha de um valor de α no conjunto de possíveis valores com a probabilidade p_i de ser escolhido. Na linha 10, a fase de construção retorna uma nova solução s_1 a partir do valor α escolhido na linha anterior. Na linha 11, uma nova solução s_2 é obtida a partir do procedimento de busca local aplicado sobre a solução recém construída s_1 da linha anterior. Nas linhas 12 e 13 ocorre a verificação de melhora e a substituição da melhor solução s^*, caso $f(s_2) < f(s^*)$. Antes do fim da iteração, o algoritmo faz a verificação se um intervalo k de iterações foi completo; sendo verdade, ocorre a atualização da distribuição de probabilidades para seleção dos valores de α. Para isso, são recalculados os valores de A_i na linha 16 e de q_i na linha 17; logo, é possível que as probabilidades p_i sejam atualizadas na linha 18. Na linha 22, após a condição de parada ser satisfeita, a melhor solução encontrada s^* é retornada.

A abordagem reativa utilizada sobre o método básico GRASP apresenta melhoras em termos de robustez e qualidade de soluções devido à maior diversificação e menor necessidade de especificações e calibragens para valores de α. Estes resultados são confirmados em aplicações do método em diversas áreas, como planejamento de redes de transmissão de energia (Binato e Oliveira, 2002), *job shop scheduling* (Binato et al., 2002), designação de redes de telefonia móvel (Gomes et al., 2001), desenvolvimento de rede rodoviária rural (Scaparra e Church, 2005), problema localidades capacitado (Delmaire et al., 1999), *strip-packing* (Alvarez-Valdés et al., 2008) e um *combined production-distribution problem* (Boudia et al., 2007).

Capítulo 4

Metodologias e Soluções Propostas

Este Capítulo aborda a implementação do método GRASP Reativo, proposta para resolução do problema de sequenciamento de tarefas com objetivo de minimização de *makespan* em ambientes de máquinas paralelas não relacionadas que utilizam recursos adicionais (UPMR). A Seção 4.1 apresenta um esquema de representação de uma solução para o problema tratado neste trabalho. A Seção 4.2 apresenta o procedimento GRASP Reativo para o UPMR. A Seção 4.3 descreve uma proposta de implementação do algoritmo VND como método de busca local. A Seção 4.4 detalha o funcionamento do método de reparo para tratar soluções infactíveis. É importante enfatizar que o método de reparo proposto neste Capítulo é uma contribuição desta dissertação, embora não haja uma avaliação específica deste método e sim um detalhamento da sua implementação. Assim, para um melhor entendimento do mesmo, este método é detalhado através de um exemplo, incluído na Seção 4.4.4.

4.1 Representação da Solução

Uma solução s é representada por um vetor de ponteiros com m posições, em que cada posição representa uma máquina. Cada posição do vetor de ponteiros aponta para um vetor de tarefas. As posições desse vetor representam posições das tarefas na ordem em que são processadas. A Figura 4.1 ilustra a representação de uma solução. Nesse exemplo, as tarefas $J1$ e $J4$ estão alocadas na máquina 1 (M_1) e as tarefas $J2$, $J3$ e $J5$ estão alocadas na máquina 2 (M_2).

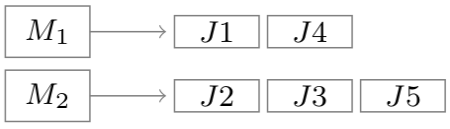


Figura 4.1: Exemplo da representação de uma solução.

4.2 GRASP Reativo para o UPMR

O algoritmo GRASP Reativo aqui proposto é baseado na estrutura básica desse método, apresentada na Seção 3.3.2. Os operadores utilizados em cada fase do

procedimento são detalhados nas seções a seguir.

4.2.1 Fase construtiva para solução do UPMR

O procedimento de construção GRASP utilizado neste trabalho consiste, a princípio, em atribuir as tarefas do conjunto N no conjunto M de máquinas. Uma solução é iniciada vazia; uma tarefa $j \in N$ é inserida em uma máquina $i \in M$. A seleção da tarefa a ser sequenciada e a máquina na qual está irá ser atribuída se dá pelo critério guloso, que seleciona a tarefa e a máquina que acarretará no menor incremento do *makespan* na iteração atual. Logo, a função gulosa é descrita por:

$$g(x) = \arg\min\{p_{ij} \mid j \in \bar{N}\}$$

sendo $\bar{N} = j \in (N - (N \cap s))$. Esse procedimento termina quando todas as tarefas do conjunto N sejam atribuídas às máquinas do conjunto M.

A etapa de construção de uma solução inicial do problema é feita elemento a elemento, sem se preocupar, durante essa fase, com a restrição de recurso. Soluções infactíveis por excesso de recurso serão certamente geradas na fase de construção; no entanto, a fase de reparo de soluções com este tipo de infactibilidade será tratada nas fases seguintes do algoritmo, detalhada na Seção 4.4. O Algoritmo 6 mostra as etapas da fase de construção de uma solução e é descrito a seguir.

Na linha 2, é feita a definição da lista de candidatos (LC), contendo todos os elementos $e_j \in LC$ que deverão fazer parte da solução. Na linha 3, a solução vazia s, que será construída. Os passos seguintes são internos ao laço de repetição, que inicia na linha 4 e termina na linha 9. A condição de parada é satisfeita quando todos os elementos do conjunto LC estiverem inseridos na solução s. Na linha 5 são selecionados os elementos para a lista restrita de candidatos (LRC). Esta lista é obtida por candidatos cujo custo associado $c(e)$ satisfaçam à expressão 4.1. Para isso, é necessário definir o parâmetro α, tal que:

$$LCR = \{e \in LC \mid c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\} \tag{4.1}$$

sendo $\alpha = 0$ para um algoritmo puramente guloso e $\alpha = 1$ corresponde ao algoritmo puramente aleatório. Na linha 6, um elemento e da LRC é escolhido aleatoriamente e adicionado à solução s na linha 7. Na linha 8, a LC é atualizada, com a remoção do elemento e inserido em s. Na linha 10, a solução s completa é retornada.

Algoritmo 6: Pseudocódigo do algoritmo da fase Construtiva R-GRASP

```
Entrada: α
Saída: s
1  início
2      Inicializa lista de candidatos LC;
3      s ← ∅;
4      enquanto (LC ≠ 0) faça
5          LRC ← ConstrurLRC(LC, α);
6          e ← SelecionarAleatorio(LCR);
7          s ← s ∪ {e};
8          Atualiza lista de candidatos LC;
9      fim
10     retorna s;
11 fim
```

Neste trabalho, 4 critérios foram utilizados para o cálculo do custo $c(e)$ de cada elemento da LC. Os critérios são descritos nas Seções 4.2.1.1, 4.2.1.2, 4.2.1.3 e 4.2.1.4, respectivamente. Usaremos a matriz de tempos de execução da expressão 4.2 para exemplificar o funcionamento de cada critério, quando aplicado à fase de construção.

$$p_{ij} = \begin{bmatrix} 1 & 2 & 4 & 3 & 3 & 1 \\ 3 & 1 & 4 & 4 & 1 & 2 \end{bmatrix} \tag{4.2}$$

4.2.1.1 Construção CT-1

A LCR na construção CT-1 será definida a partir da soma dos tempos de processamento de cada tarefa presente em LC. Para isto, cada tarefa que ainda não está presente na solução (candidato) terá seu custo guloso calculado pela soma dos tempos de processamento desta tarefa em cada máquina do sequenciamento, ou seja, $c(e_j) = \sum_{i=0}^{m} p_{ij}$. Com o custo guloso de cada tarefa e_j calculado, a lista restrita de candidatos é definida, em seguida, pela equação 4.1, sendo o valor de α definido pelo GRASP Reativo. Após, um elemento da LCR é escolhido aleatoriamente, com probabilidade uniforme, e adicionado na solução. A tarefa selecionada será inserida na máquina que tenha o menor tempo de processamento da tarefa.

	$J1$	$J2$	$J3$	$J4$	$J5$	$J6$
Máquina 1	1	2	4	3	3	1
Máquina 2	3	1	4	4	1	2
Soma	**4**	**3**	**8**	**7**	**4**	**3**

A Tabela 4.2.1.1 mostra o conjunto com o custo guloso de cada tarefa definido como $\{4, 3, 8, 7, 4, 3\}$. Considerando o valor de $\alpha = 0,6$ e aplicando na equação 4.1, tem-se $g = 3 + 0,6 * (8 - 3) = 6$. A LCR será composta por todas as tarefas para as quais as somas sejam menores ou iguais a 6; logo, $LCR = \{J1, J2, J5, J6\}$. Se a tarefa escolhida aleatoriamente em LRC for $J2$, a máquina escolhida será a máquina 2, que demanda menor tempo de execução da tarefa. O último passo da iteração é remover a tarefa $J2$ da lista de candidatos LC. Para a alocação do próximo elemento, parte-se do valor g, que continuará o mesmo anterior $g = 6$. Assim, se

a escolha aleatória da $LRC = \{J1, J5, J6\}$ for $J6$, ela será alocada na máquina 1, que demanda menor tempo de execução para $J6$. A essa altura, tem-se uma solução parcial s em que $Maq_1 = \{J6\}$ e $Maq_2 = \{J2\}$. Este procedimento será repetido, até que todas as tarefas em LC sejam alocadas na solução s.

4.2.1.2 Construção CT-2

A LCR será definida a partir do incremento do *completion time* acarretado pela inserção de cada tarefa presente em LC para uma máquina específica escolhida *a priori*. A definição da máquina se dá a partir do *completion time* de cada máquina do sequenciamento da solução parcialmente construída, ou é considerada a primeira máquina do sequenciamento no caso de uma solução ainda vazia. A máquina com menor *completion time* é selecionada. Após definida qual máquina i que receberá a próxima tarefa, o custo guloso ($c(e_j)$) de cada tarefa e_j candidata a entrar na solução é igual ao seu tempo de processamento na máquina i, ou seja $c(e_j) = p_{ij}$.

	~~J1~~	~~J2~~	~~J3~~	~~J4~~	J5	J6
Máquina 1	1	2	4	3	**3**	**1**
Máquina 2	3	1	4	4	1	2

Supondo, no exemplo, que já haja uma solução pré-construída, em que $Maq_1 = \{J1, J4\}$ e $Maq_2 = \{J2, J3\}$, tem-se $C_1 = 4$ e $C_2 = 5$, sendo C_1 e C_2, respectivamente, o *completion time* das máquinas Maq_1 e Maq_2. A máquina de menor tempo total até então é Maq_1. O valor GRASP g será calculado para as tarefas que ainda estão em $LC = \{J5, J6\}$, para seus respectivos custos gulosos na $\{3, 1\}$. Considerando o valor de $\alpha = 0,6$, tem-se $g = 1 + 0,6 * (3 - 1) = 2,2$. A tarefa selecionada para fazer parte da solução nesse caso é $J6$, com custo guloso $c(e) = 1$, que é menor que $g = 2,2$.

4.2.1.3 Construção CT-3

A LRC no procedimento de construção CT-3 será definida a partir de todos os tempos da matriz p_{ij} das tarefas presentes em LC. Considerando os tempos de processamento presentes no exemplo definido na expressão 4.2, tem-se que $c^{min} = 1$ e $c^{max} = 4$. Ao se calcular o primeiro valor grasp com $\alpha = 0,6$, obtém-se $g = 1 + 0,6 * (4 - 1) = 2,8$. Assim, a LRC é composta por todas as tarefas da LC em que pelo menos um dos tempos $p_{ij} \leq 2,8$.

Para este exemplo, a primeira LRC seria composta pelas tarefas $\{J1, J2, J5, J6\}$. Uma tarefa escolhida aleatoriamente na LRC só pode ser alocada na máquina que demandar um tempo de processamento menor ou igual ao valor grasp g calculado. No exemplo, se a tarefa escolhida for $J5$, ela só poderá ser alocada na Máquina 2. No caso em que a inserção da tarefa seleciona de LRC puder ser realizada em duas ou mais máquinas, será selecionada a máquina que demandar menor tempo de processamento da tarefa. O critério de menor *completion time* sempre será avaliado e aplicado quando conveniente para o processo de construção.

Ao continuarmos com o exemplo, o próximo valor grasp g continuará sendo $2,8$ e $LRC = \{J1, J2, J6\}$. Se a escolha aleatória for a tarefa $J6$, na qual mais de uma máquina demanda tempo menor ou igual a $2,8$, a máquina escolhida será a

de menor *completion time* após a inserção. Neste caso, a Máquina 1 demanda o menor tempo de processamento e também resulta o menor *completion time*. Assim, a solução parcial s estaria com $Maq_1 = \{J6\}$ e $Maq_2 = \{J5\}$. O procedimento se repete até que a solução s esteja completa com todos os elementos de LC.

4.2.1.4 Construção CT-4

A *LRC* na construção CT-4 será definida por uma combinação dos 2 primeiros tipos de construção citados nas Seções 4.2.1.1 e 4.2.1.2. Para isso, duas listas auxiliares são criadas, uma para cada tipo de construção. A etapa de escolha aleatória do elemento da *LRC* será feita separadamente, selecionando um elemento de cada lista auxiliar. Por fim, o elemento que puder ser melhor alocado de modo que resulte no melhor custo para a solução será escolhido. Neste tipo, o elemento escolhido poderá ser alocado na máquina que representar o menor custo para a solução, sem nenhuma restrição.

4.2.2 Alfas setorizados

Para o GRASP Reativo, é considerado uma faixa de valores para α, como explicado na Seção 4.2. Foram feitos testes para 3 diferentes expansões dos valores, considerando o intervalo entre os elementos do conjunto. De forma arbitrária, foram escolhido os conjuntos A_1, A_2 e A_3, dados conforme a Tabela 4.1.

Tabela 4.1: Expansões para conjuntos de valores α.

	$\times 1$	$\times 3$	$\times 4$	$\times 5$	$\times 6$	$\times 7$	$\times 8$	$\times 9$	$\times 10$
A_1	0.03	0.09	0.12	0.15	0.18	0.21	0.24	0.27	0.3
A_2	0.06	0.18	0.24	0.3	0.36	0.42	0.48	0.54	0.6
A_3	0.09	0.27	0.36	0.45	0.54	0.63	0.72	0.81	0.9

Os 3 conjuntos foram definidos de modo que o intervalo entre a total gulosidade, representada por $\alpha = 0$, e a total aleatoriedade, representada por $\alpha = 1$, fosse dividido em 3 partes. Na primeira parte, os valores iniciam em $0,03$ até $0,3$, representando $\frac{1}{3}$ do grupo. A segunda parte representa $\frac{2}{3}$ do conjunto, com o primeiro valor iniciando em $0,06$ até $0,6$. O terceiro conjunto representa a totalidade dos 9 elementos, iniciando de $0,09$ até $0,9$. O objetivo dessa setorização é investigar se valores de α mais agrupados ou menos agrupados podem influenciar na qualidade dos resultados.

4.3 Adaptação do VND para o UPMR

Para a fase de refinamento, foi utilizado um método que explora o espaço de soluções realizando trocas sistemáticas de estruturas de vizinhanças, baseado no algoritmo VND proposto em Mladenović e Hansen (1997), porém, com variações, como será descrito a seguir. O método não considera a restrição de recurso durante a busca, aceitando soluções de melhora em relação ao *makespan*, factíveis ou não. O procedimento, então, usa a forma clássica do método VND, cuja estrutura de

vizinhança é explicada na Seção 3.1. Todos as soluções de melhora obtidas através dos vizinhos de s durante a busca são inseridos em um conjunto elite, que será chamado de conjunto E. Ao final da busca, as soluções contidas em E passarão pelo procedimento de reparo, e a melhor solução reparada s_f^* será retornada.

O Algoritmo 7 descreve o método proposto. Neste, é recebida, como entrada, uma solução s recém construída. Para esta solução s, ainda não é exigido viabilidade, conforme a restrição de recurso. Também são entradas as estruturas de vizinhança utilizadas na busca, dadas pelo conjunto $N = \left\{N^{(1)}, N^{(2)}, \cdots, N^{(k)}\right\}$. O procedimento se dá buscando o melhor vizinho s' de uma vizinhança $N^{(k)}(s)$ (linha 17), sendo k o índice da estrutura de vizinhança buscada na vez. Inicia-se a busca pela primeira estrutura de vizinhança $k = 1$, presente em N. Após a busca do melhor vizinho em $N^{(k)}(s)$, se s' superar s em qualidade, segundo a função de avaliação f (linha 7), então s recebe s' e o conjunto E adiciona s' (linhas 8 e 10). Além disto, retoma-se a exploração da primeira vizinhança $k = 1$ a partir da nova solução s' encontrada (linha 9). Se nenhuma solução de melhora for encontrada durante a exploração da vizinhança $N^{(k)}$, a próxima estrutura de vizinhança $N^{(k+1)}$ será explorada até que k_{max} vizinhanças do conjunto N sejam exploradas. Após o algoritmo não apresentar melhora na exploração das estruturas de vizinhança, na linha 16 o método *Reparo* (4.4) é aplicado em todas as soluções do conjunto E, gerando o conjunto F de soluções factíveis. A melhor solução factível em F (linha 17) é retornada como melhor solução factível s_f^* (linha 18).

Algoritmo 7: Pseudocódigo do algoritmo de Busca Local

```
   Entrada: s, N(.), k_max
   Saída: s*_f                                   /* solução refinada e factível */
1  E ← ∅;                                        /* Inicializa conjunto elite */
2  k ← 1;
3  Remove todos os idle-times se houver em s;
4  início
5      enquanto (k ≤ k_max) faça
6          Encontre o melhor vizinho s' ∈ N^(k)(s);
7          se (f(s') < f(s)) então
8              s ← s';
9              k ← 1;
10             E ← E ∪ {s'};                     // Solução adicionada no conj.  E
11         fim
12         senão
13             k ← k + 1;
14         fim
15     fim
16     F = Reparo(E);                            // Aplica o reparo a cada solução E
17     s*_f = arg min {f(s') | s' ∈ F};
18     retorna s*_f; // Seleciona a melhor solução factível
19 fim
```

Neste trabalho, duas estruturas de vizinhança, definidas a seguir, são avaliadas.

Algoritmo 8: Pseudocódigo do algoritmo de exploração da vizinhança de inserções externas

```
   Entrada: s
   Saída: s
1  início
2      i ← MaiorCompletionTime(s);
3      Makespan ← Makespan(s);
4      MelhorInsercao ← ∅;
5      para cada tarefa j da máquina i faça
6          para cada máquina k ≠ i da solução s faça
7              MakespanInsercao ← AvaliaInsercao(k, j);
8              se MakespanInsercao ≤ Makespan então
9                  MelhorInsercao ← {k, j};
10                 Makespan ← MakespanInsercao;
11             fim
12         fim
13     fim
14 fim
15 retorna s;
```

4.3.1 Inserção Externa

A estrutura de vizinhança de Inserção Externa ($N^{(1)}$) usada neste trabalho é baseada em movimentos de inserção. Estes movimentos consistem na remoção de uma tarefa j alocada em uma máquina i e a reinserção desta tarefa em uma máquina w, sendo $i \neq w$. Este movimento, exemplificado na Figura 4.2, é explorado pela estrutura de vizinhança no Algoritmo 8, na seguinte forma: cada tarefa j alocada em uma máquina i, em que i é a máquina que define o *makespan*, i.e., $i|C_i = C_{max}$, é inserida ao final de todas máquinas $w \neq i$ restantes.

Figura 4.2: Exemplo de Inserção externa da tarefa J3, originalmente na máquina M_2, na máquina M_1.

4.3.2 Troca Externa

A estrutura de vizinhança de Troca Externa ($N^{(2)}$) é baseada em movimentos de troca mostrados na Figura 4.3. Estes movimentos são definidos como: dadas duas tarefas j e k atribuídas, respectivamente, nas máquinas i e w, sendo $i \neq w$, j tem sua posição trocada com k, ou seja, j passa a ser processada em w na posição de k; já k passa a ser processada em i na posição de j. Baseado neste movimento, a estrutura de troca explorada no Algoritmo 9 consiste em trocar cada tarefa j da

Algoritmo 9: Pseudocódigo do algoritmo de exploração da vizinhança de trocas externas

```
   Entrada: s
   Saída: s
 1 início
 2     i ← MaiorCompletionTime(s);
 3     Makespan ← Makespan(s);
 4     MelhorTroca ← ∅;
 5     para cada tarefa j da máquina i faça
 6         para cada máquina k ≠ i da solução s faça
 7             para cada tarefa w da máquina k faça
 8                 MakespanTroca ← AvaliaTroca(i, j, k, w);
 9                 se MakespanTroca ≤ Makespan então
10                     MelhorTroca ← {j, w};
11                     Makespan ← MakespanTroca;
12                 fim
13             fim
14         fim
15     fim
16 fim
17 retorna s;
```

máquina i, $i|C_i = C_{max}$ (máquina que define o *makespan*), com todas as tarefas k de todas as demais máquinas w, sendo $w \neq i$.

Figura 4.3: Exemplo de Troca externa entre as tarefas J1 e J5.

4.4 Proposta de Reparo

Esta seção apresenta o método de reparo de soluções infactíveis pela restrição de recurso. Se uma solução está com este tipo de infactibilidade, significa que, em pelo menos um instante de tempo, existe uma demanda de utilização de recurso maior que o limite máximo estabelecido arbitrariamente por meio da entrada do valor de R_{max}. Para resolver este problema, é necessário um procedimento de correção do excesso, que será denominado genericamente de **Reparo**.

O método de reparo para soluções infactíveis por excesso de recurso é ponto de maior contribuição neste trabalho. O objetivo é propor um método simples e que seja efetivo. A proposta se inicia com a forma de abordagem do problema, que consiste em tratar o problema de otimização separadamente da restrição de recurso. Embora o problema como um todo seja composto pelo resultado de todas as suas

restrições, pode-se definir sub-problemas. Nesse caso, o primeiro problema seria refinar uma solução sem considerar a restrição de recurso, definido como um sub-problema de otimização. O segundo sub-problema é mais simples do que o primeiro, pois não se trata de um problema de otimização, não é exigido, para este problema, a minimização de recurso, deseja-se apenas manter o uso de recurso total menor ou igual ao limite permitido pelo valor R_{max}, podendo ser solucionado de forma gulosa.

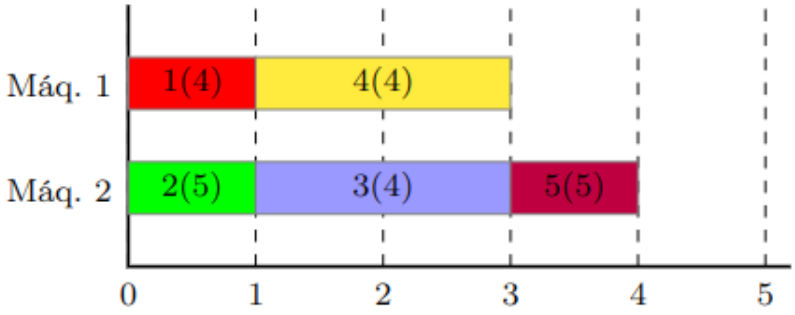
Ao tratar esta restrição do problema separadamente, alguns pontos são identificados. O primeiro deles se refere às modificações feitas em uma solução s infactível para se obter uma solução s'. Se s for uma solução previamente refinada (ótimo local), ao passar pelo procedimento de reparo as modificações necessárias para se obter uma solução reparada irão impactar diretamente o antigo valor de C_{max} e a piora na qualidade da solução, portanto, será iminente. Por outro lado, partindo de uma solução reparada e buscando refiná-la, a mesma lógica se aplica, ou seja, as modificações feitas pelo procedimento de busca local tornarão a solução corrente em algum momento infactível. Uma maneira de resolver este impasse é, portanto, partir de uma solução refinada infactível e, então, priorizar modificações que não piorem ou que piorem minimamente a qualidade dessa solução corrente. Desse modo, pode-se obter soluções de qualidade sem considerar a restrição de recurso primeiramente e, só então, repará-las, visando a menor piora possível.

4.4.1 Reparo por Cascata (CAS)

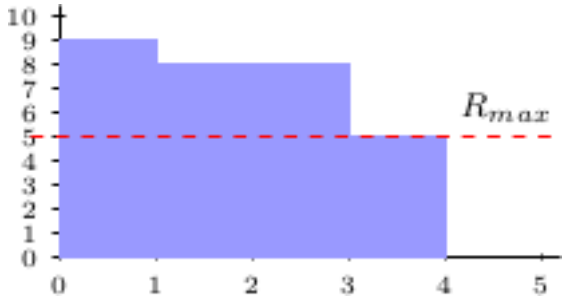
O método proposto neste trabalho foi nomeado como *Reparo por Cascata*, devido à característica que o procedimento utiliza para eleger as prioridades das etapas, iniciando da melhor etapa (mais alta), até passar para a etapa consecutivamente inferior (mais baixa), após esgotar a capacidade da etapa anterior. A última etapa, dentro da analogia, seria a etapa mais baixa, ou seja, com o menor benefício, já que a possibilidade de maiores benefícios já foram exploradas em etapas anteriores, mas sem sucesso até então. Basicamente, o método que será descrito busca fazer o reparo após classificar as opções disponíveis, priorizando sempre as opções capazes de retornarem o maior benefício, que, dentro do escopo do problema, se traduz por maior redução de recurso e menor aumento de *makespan*. A título de simplificação, o termo utilizado será apenas CAS para se referir ao método de reparo proposto.

O método CAS inicia avaliando a factibilidade de uma solução. Para isso, cada instante de tempo do sequenciamento tem sua soma de recursos simultâneos avaliados; se houver um excesso, é medido em quantas unidades de tempo esse mesmo excesso se mantém. Essa região contínua é identificada nesse texto como *plateau*. Um *plateau*, após identificado, deverá ser reduzido ou eliminado. Não havendo mais *plateaus* na solução, esta é considerada factível.

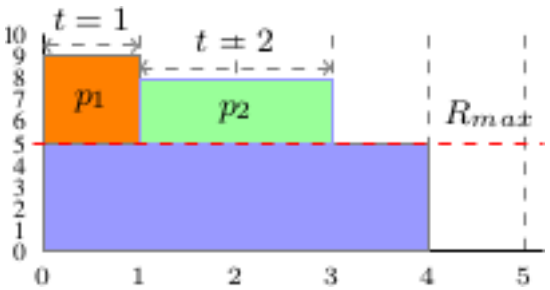
Considere o sequenciamento mostrado na Figura 4.4(a). Na Figura 4.4(b), toda a região acima da linha pontilhada representa o excesso de recursos. Já na Figura4.4(c), o primeiro retângulo indica o primeiro *plateau*, representado por p_1, que se estende pelo tempo $t = 1$, ultrapassando R_{max} em 4 unidades. O segundo *plateau*, dado por p_2, se estende por duas unidades de tempo $t = 2$, excedendo R_{max} em três unidades. O *plateau* p_2 é identificado juntamente com p_1 para fins didáticos, já que, na prática, apenas p_1 importa. O reparo de p_1 mudará o cenário, exigindo uma nova avaliação dos recursos para determinar um possível novo *plateau*. O procedimento



(a) Sequenciamento.



(b) Consumo de recurso.



(c) *Plateaus* identificados.

Figura 4.4: Exemplo de sequenciamento infactível.

é finalizado quando nenhum *plateau* for mais identificado, retornando uma solução considerada factível.

Uma vez determinado um *plateau*, o passo seguinte é identificar as tarefas envolvidas nessa região e compor o conjunto P_s. Para cada máquina, é selecionada a primeira tarefa que estiver alocada na região do *plateau* e a sua sucessora. No exemplo da Figura 4.4, as tarefas envolvidas em p_1 são $\{J1, J2\}$ e suas tarefas consecutivas são, respectivamente, $\{J4, J3\}$; logo, tem-se $P_s = \{J1, J4, J2, J3\}$. Este conjunto representa p_1.

O Algoritmo 10 apresenta o método de reparo. Este algoritmo recebe, como entrada, um sequenciamento infactível s, uma função $g(\cdot)$, que avalia e escolhe o melhor vizinho obtido das estruturas aplicadas (Seção 4.4.2) e um conjunto M, composto por k estruturas de movimentos de reparo. A saída é uma solução factível s. Na linha 1 ocorre a seleção das tarefas que definem o *plateau*. Na linha 3 é avaliado se existe região de *plateau*; caso exista, a solução é infactível e as estruturas de reparo M serão avaliadas sobre os elementos de P_s por meio da função $g(s')$. Neste caso, o

Algoritmo 10: Pseudocódigo do Algoritmo de Reparo CAS.

Entrada: $s, g(.), M(.)$
Saída: s
$P_s = \{x_{ij} \mid \sum_{i=1}^{m} r_{ij} > R_{max}\}$; `// Tarefas envolvidas no plateau`
início
  enquanto $(P_s \neq \emptyset)$ **faça**
    Busca melhor realocação $s^{'} \in M_k(s)$;
    se $(g(s') < g(s))$ **então**
      $s \leftarrow s'$;
    fim
    senão
      $s' \leftarrow IdleTimeInsert(s)$; `// Algoritmo 11`
    fim
    Atualiza (P_s); `// Reavalia plateau`
  fim
  retorna s;
fim

movimento que reduz ou elimina o *plateau* é aplicado no sequenciamento infactível s e, então, P_s é atualizado. Caso não seja identificado algum movimento que reduza o *plateau*, a última opção de tentativa para eliminação deste *plateau* será por meio da inserção de tempos ociosos, que chamaremos de *Idle Times* e será introduzido na Seção 4.4.3.

4.4.2 Estruturas de realocação de tarefas no *plateau*

Neste trabalho, propõe-se a utilização de três estruturas de realocação de tarefas envolvidas na região do *plateau*. Estas estruturas são:

(i) ***SwapIn***: dadas duas tarefas j e k, tal que $j, k \in P$ e alocadas na mesma máquina i, sendo k consecutiva a j, as tarefas j e k têm suas posições trocadas entre si. A estrutura é obtida após o movimento *SwapIn* ser aplicado em todas as máquinas em que há tarefas que pertencem a P;

(ii) ***SwapOut***: dadas duas tarefas j e k, alocadas respectivamente nas máquinas i e w, sendo $i \neq w$, as tarefas $j \in P$ e $k \in P$ têm suas posições trocadas entre si, preservando as respectivas posições. Assim, j é transferida para a máquina w na posição original de k; já k será realocada na máquina i na posição original de j. A estrutura é obtida após o movimento ser aplicado a cada tarefa pertencente ao conjunto P, ou seja, cada tarefa pertencente a P é trocada com todas as demais tarefas também pertencentes a P alocadas em outras máquinas;

(iii) ***InsertOut***: dada uma tarefa $j \in P$, alocada na máquinas i, e uma tarefa $k \in P$, alocada na máquina w, sendo $i \neq w$ e k necessariamente se iniciando no ponto de ruptura p (início do *plateau*), insere-se, então, a tarefa j na posição da tarefa k. A posição em que j ocupava na máquina i é assumida pela tarefa

consecutiva, se essa existir, e a tarefa k na máquina w se torna consecutiva à tarefa j. A estrutura é obtida após o movimento *InsertOut* ser aplicado em cada tarefa pertencente a P, ou seja, cada tarefa pertencente a P será inserida na posição anterior das demais tarefas de P.

4.4.3 Inserção de *Idle-Times*

Esta estratégia consiste em manter uma determinada máquina ociosa durante um período de tempo que seja suficiente para que uma determinada região se torne factível. A inserção de um *idle-time* em uma determinada máquina deve deslocar o instante de processamento das tarefas alocadas na máquina para o futuro, o que, obviamente, tem impacto direto no tempo de término de todas as tarefas subsequentes. Devido a isto, o procedimento de reparo considera a inserção de *idle-time* apenas quando todos os procedimentos de reparo por realocações não surtirem efeito.

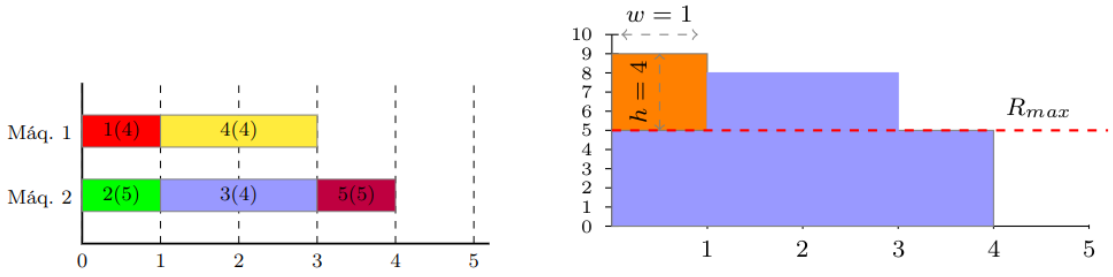
Algoritmo 11: Pseudocódigo do Algoritmo de Inserção de *Idle-Times*

Entrada: s, p
Saída: s
1 **início**
2 $P \leftarrow$ tarefas envolvidas em p;
3 $t \leftarrow$ Comprimento de p;
4 $N \leftarrow \emptyset$;
5 **for** $i = 0, \ldots, m$ **do**
6 $C'_i \leftarrow C_i + t$;
7 $N \leftarrow N \cup \{C'_i\}$;
8 **end**
9 $i' \leftarrow \arg\min \{C'_i \in N\}$;
10 Adiciona τ na maq i' na posição $j - 1$;
11 $s \leftarrow \tau_{i',j-1}$;
12 **retorna** s;
13 **fim**

O Algoritmo 11 mostra o funcionamento do método de inserção de *idle-times*. As entradas são a solução s (infactível) e p, que representa um conjunto de informações sobre o *plateau* de s. Na linha 2 são agrupadas as tarefas envolvidas no *plateau*; na linha 3 é obtido o comprimento do *plateau*, que será convertido em tempo ocioso τ. Em N, são colocados os tempos finais de todas as máquinas C_i adicionados a t. O menor tempo de término após a soma de t indica a posição da máquina que terá o *idle-time* τ incorporado à solução s.

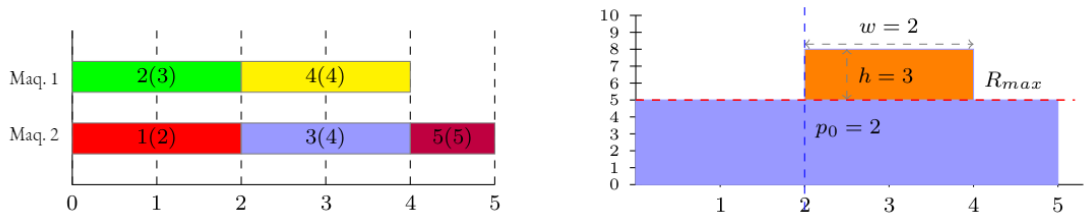
4.4.4 Exemplo da ação de reparo

Nas Figuras 4.5(a) e 4.5(b), tem-se um sequenciamento com recurso excedendo $R_{max} = 5$. O ponto inicial de ruptura é $t = 0$. As tarefas envolvidas nessa região serão identificadas. A cor laranja na Figura 4.5(b) indica a região do *plateau*, sendo h a altura, dada por $h = r - R_{max}$, e, no exemplo, $h = 4$. O valor w representa a largura do *plateau*, dado pelo tempo em que r se mantém inalterado. No exemplo, este valor é $w = 2$. O ponto inicial do *plateau* é $p_0 = 0$, no qual as tarefas envol-



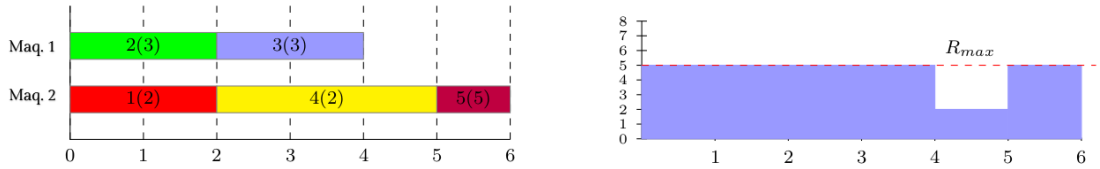
(a) Sequencia infactível. (b) *Plateau* identificado.

Figura 4.5: Ação de Reparo.



(a) Sequencia infactível. (b) *Plateau* identificado.

Figura 4.6: Ação de Reparo.



(a) Sequenciamento factível. (b) Recurso utilizado.

Figura 4.7: Ação de Reparo: final.

vidas são $\{1, 2\}$ e as tarefas consecutivas são, respectivamente, $\{4, 5\}$. O primeiro movimento faz a troca de cada tarefa envolvida no ponto de ruptura com sua tarefa subsequente (*SwapIn*), se o valor de recurso for menor que o recurso da tarefa anterior. No exemplo, isso é possível na tarefa 2, que demanda um recurso igual a 5 e sua consecutiva, a tarefa 3 com recurso igual a 4. A troca dessas tarefas reduziria em 1 a altura do *plateau*, mas sem reparar a região. Ainda assim, essa opção poderá ser útil, caso uma melhor não seja encontrada na avaliação dos demais movimentos. O segundo movimento é o de trocas extra-máquinas (*SwapOut*), que impactará recurso e também tempo de término das máquinas envolvidas. No exemplo, a troca será feita entre as tarefas 1 e 2. Esse movimento, como mostrado na Figura 4.6(b), é capaz de reparar o *plateau* atual, que agora atende à restrição. Assim, ambas as máquinas sofrem impacto no tempo de término e o valor do *makespan* aumenta de 4 para 5.

Na próxima iteração do algoritmo o novo *plateau* identificado agora tem $w = 2$ e $h = 3$, iniciando em $p_0 = 2$. O movimento escolhido também será *SwapOut* nas tarefas 4 e 3. O movimento resulta na solução factível mostrado nas Figuras 4.7(a) e 4.7(b).

Assim, o procedimento foi capaz de reparar a solução infactível, como mostrado

na Figura 4.7(b), de modo que o valor de $R_{max} = 5$ agora é respeitado. Não se pode afirmar que o sequenciamento, que agora é factível, é uma solução ótima nem de boa qualidade, mas, sim, que atende à restrição de utilização de recurso estabelecida por R_{max}.

4.5 Modelo de Reparo da literatura

O método de reparo utilizado em Villa et al. (2018) e em Vallada et al. (2019) tem como objetivo obter uma solução factível S de um sequenciamento infactível A. A ideia consiste em criar um novo conjunto de tarefas, denominado *Pending Jobs (PJ)*, e mover tarefas de A para PJ até que se obtenha um sequenciamento parcial factível. Assim, a partir de A, as tarefas são movidas para PJ priorizando o maior consumo de recurso. Estas tarefas em PJ serão reinseridas no sequenciamento A uma-a-uma, seguindo duas estratégias:

(i) alocar uma determinada tarefa escolhida no conjunto *Pending Jobs* no final da máquina em que a mesma estava originalmente;

(ii) alocar a tarefa no final da máquina de menor *completion time*.

A restrição de recurso deve ser satisfeita em ambos os casos. Após alocar a tarefa pendente no final da máquina escolhida, o método de reparo troca a sua posição com a tarefa anterior se duas condições forem satisfeitas:

(i) não houver tempo ocioso entre ambas tarefas; e

(ii) o consumo de recurso da tarefa anterior for menor que o consumo da tarefa pendente.

Esta troca será repetida se as novas posições também forem satisfeitas pela condição (ii). O procedimento é repetido até que todas as tarefas pendentes sejam reinseridas e resulte em uma solução factível.

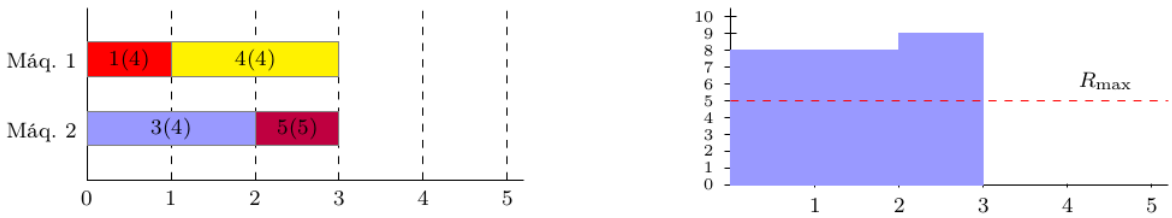


Figura 4.8: Sequenciamento parcial.

Partindo do sequenciamento inicial sugerido no Exemplo 2.2.2 e aplicando este método de reparo *Pending Job*, após identificar infactibilidade é iniciado o processo de desconstrução do sequenciamento, removendo as tarefas pela ordem de maiores valores de recurso até que o sequenciamento parcial se torne factível. A tarefa 2 será a primeira a ser removida e obtém-se o sequenciamento parcial mostrado na Figura 4.8.

Ao fim desta etapa de destruição, o sequenciamento conterá apenas a tarefa 1 na máquina Maq_1 e as demais tarefas compondo o conjunto $PJ = \{J2, J3, J4, J5\}$. Tem-se, portanto, um sequenciamento parcial factível, com um consumo total de recurso abaixo de $R_{max} = 5$.

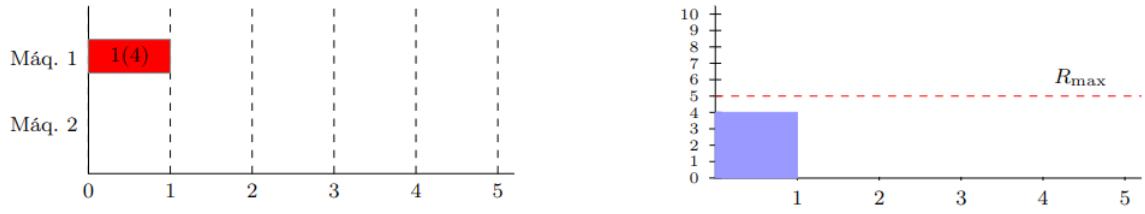


Figura 4.9: Sequenciamento parcial factível.

O procedimento de reconstrução se inicia reincorporando, na máquina de menor *Completion Time*, a mesma tarefa que ocupava a mesma posição antes do processo de destruição. No exemplo, se trata da tarefa $J2$; no entanto, esta tarefa não poderá ocupar esta posição, devido ao excesso de recurso, sendo, então, alocada na máquina seguinte de menor CT, ou seja, a máquina Maq_1. Como a tarefa $J2$ demanda recurso menor que a tarefa anterior $J1$, suas posições são trocadas e obtém-se o sequenciamento parcial factível da Figura 4.10.

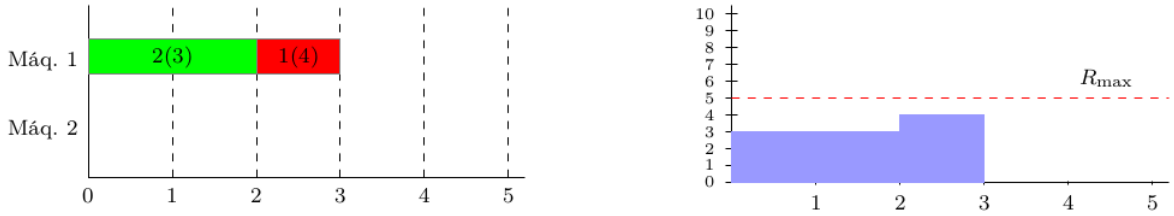


Figura 4.10: Sequenciamento parcial factível.

A tarefa seguinte $J3$ causará infactibilidade se for alocada na Maq_2. Então, também será alocada na Maq_1 e trocada com a tarefa $J1$ por ter valor menor de recurso. Encontra-se, então, o sequenciamento factível e parcial da Figura 4.11.

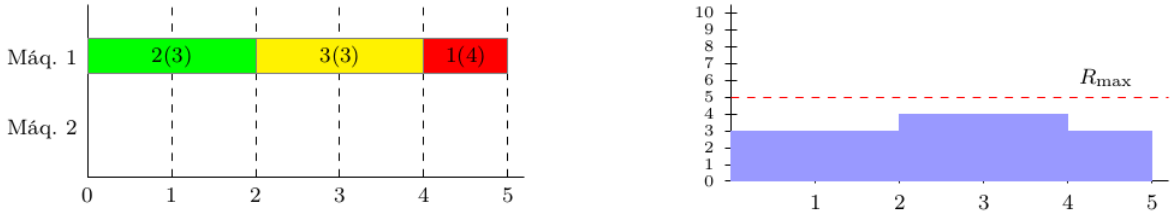


Figura 4.11: Sequenciamento parcial factível.

Por fim, determina-se uma solução factível e com todos as tarefas pendentes reinseridas. As tarefas foram realocadas a partir de um sequenciamento parcial factível, seguindo as regras definidas pelo método. A solução encontrada é mostrada na Figura 4.12.

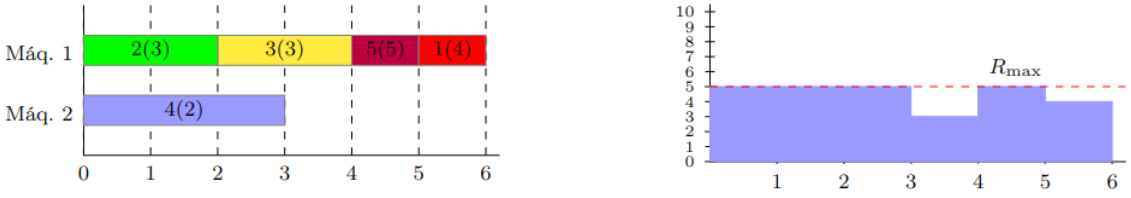


Figura 4.12: Solução factível.

A justificativa para propor um método diferente para reparo de soluções infactíveis é tentar um menor número de realocações e avaliá-las em regiões específicas do sequenciamento. Apenas a melhor realocação avaliada será aplicada. Características como essas não são exploradas no método da literatura descrito.

Capítulo 5

Experimentos Computacionais

Este Capítulo apresenta os resultados obtidos pelo algoritmo GRASP Reativo proposto no capítulo 4. A Seção 5.1 descreve características das instâncias que foram utilizadas e o procedimento de geração dos arquivos. A Seção 5.2 apresenta o ambiente computacional no qual foram realizados as execuções dos algoritmos e quais parâmetros foram empregados. A Seção 5.3 apresenta os critérios de parada do algoritmo e a medida de desvios para apresentação dos resultados. A Seção 5.4 mostra os resultados obtidos, comparando os tipos de construção GRASP propostos. A Seção 5.5 descreve os resultados referentes aos métodos de busca local *First Improvement* e *Best Improvement*. A Seção 5.6 apresenta os resultados referentes aos diferentes conjuntos de parâmetro α sugeridos para o R-GRASP. A Seção 5.7 apresenta uma comparação entre o método proposto e os resultados da literatura. Na Seção 5.8 é feito um resumo conclusivo, a partir dos resultados obtidos e apresentados no capítulo.

5.1 Características de Problemas-teste

Para avaliar o método proposto, foram realizados experimentos sobre o conjunto de instâncias introduzido por Fanjul-Peyro et al. (2017) e testados com a implementação de modelos nos quais os melhores resultados obtidos serão denominados como *Fanjul2017*. O mesmo conjunto de instâncias foi utilizado em aplicações de métodos como *Scatter Search Algorithm*, *Enriched Scatter Search Algorithm* e *Enriched Iterated Greedy Algorithm* em trabalhos como Villa et al. (2018); Vallada et al. (2019). Este conjunto é formado por 1500 instâncias, divididas em 3 grupos, denominados *Small*, com 450 instâncias, para $n = \{8, 12, 16\}$ e $m = \{2, 4, 6\}$; *Medium*, também com 450 instâncias, e $n = \{20, 25, 30\}$ e $m = \{10, 20, 30\}$; e *Large*, com 600 instâncias e $n = \{50, 150, 250, 350\}$ e $m = \{10, 20, 30\}$. Para cada combinação de número de tarefas e número de máquinas, há 5 conjuntos relacionados ao tempo de processamento e ao recurso. Quanto ao tempo de processamento, são gerados considerando a distribuição uniforme para os intervalos:

(i) p_{ij} $U\,(1, 100)$: distribuição mais comum em maquinas paralelas, em que $U\,(a, b)$ e inteiros aleatórios são uniformemente distribuídos entre a e b incluindo os extremos;

(ii) $p_{ij} = U(10, 100)$: muito comum trabalhos envolvendo problemas de sequenciamento em máquinas paralelas, como em Ghirardi e Potts (2005); Martello et al. (1997); Mokotoff e Jimeno (2002);

(iii) $p_{ij} = U(100, 200)$ introduzido por Fanjul-Peyro e Ruiz (2010): nivelado pela diferença relativa entre o mínimo e máximo tempo de processamento. Considerado como abordando problemas mais realísticos;

(iv) *Correlated Jobs*: para cada tarefa, é atribuído $p_j = U(1, 100)$ mais um tempo adicional $p'_{ij} = U(1, 20)$, obtendo $p_{ij} = p_j + p'_{ij}$;

(v) *Correlated Machines*: para cada máquina, é atribuído $p_i = U(1, 100)$ e o processamento da tarefa nessa máquina é acrescido de $p'_{ij} = U(1, 20)$, obtendo $p_{ij} = p_i + p'_{ij}$.

Dados estes cinco intervalos que controlam o tempo de processamento das tarefas, também são considerados o número de recursos consumidos para o processamento de cada tarefa em uma dada máquina. Assim, r_{ij} é gerado aleatoriamente de duas diferentes formas:

(i) $r_{ij} = U(1, 9)$;

(ii) Por intervalo: nesse caso, o valor de recursos r_{ij} demandado para cada par tarefa-máquina segue a seguinte fórmula:

$$r_{ij} = \left(\frac{p_{ij} - U_{min}}{d}\right) + R_{lo} + U(-3, 3)$$

em que:

$$d = \frac{U_{max} - U_{min}}{R_{up} - R_{lo} + 1}$$

e $U_{\max}$ e $U_{\min}$ são teoricamente o mínimo e máximo número de recurso demandado por cada tarefa-máquina do sequenciamento. Essa expressão é truncada para desempate no intervalo $[1, 9]$, se for necessário.

Por fim, o número máximo $R_{\max}$ de recurso disponível por instante de tempo é computado considerando $r_{\min} = 1$ e $r_{\max} = 9$ pela expressão a seguir:

$$R_{\max} = m \cdot \frac{r_{\max} + r_{\min}}{2} = m \cdot \frac{9 + 1}{2} = 5m$$

sendo m é o número de máquinas disponíveis. Os arquivos relacionados estão disponíveis em `http://soa.iti.es`.

5.2 Ambiente e Parâmetros

Todos os métodos foram executados em um computador Dell Optiplex 780, processador Intel(R) Core(TM)2 Duo 2.93GHz, arquitetura i386, 8 GBytes de memória RAM, rodando em um Sistema Operacional Debian GNU/Linux 9. A linguagem de programação usada foi Java 8, OpenJdk versão 1.8.0_222, IDE Eclipse versão 2020-06(4.16.0). Não é utilizado processamento paralelo neste trabalho.

O método VND implementado aplicou as heurísticas *First Improvement* e *Best Improvement* como algoritmos de busca local. Em ambas as heurísticas, a primeira vizinhança na ordem de execução é $N_s^2(s)$ (Troca Externa) e a segunda $N_s^1(s)$ (Inserção Externa). O algoritmo R-GRASP utilizou como padrão o conjunto de valores do parâmetro α na forma $\alpha \in \{0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9\}$. O parâmetro y de atualização da distribuição de probabilidade para escolha de um novo α foi utilizado como $y = 100$ para todas as execuções do algoritmo R-GRASP.

5.3 Critério de Parada e Medida de Desvios

Para o experimento, foram utilizados dois critérios de parada: (i) número máximo de iterações ($iter_{max}$); e (ii) número de iterações sem melhora ($iter_{sm}$). Para os grupos de instâncias *Small* e *Medium*, o número máximo de iterações utilizado foi $iter_{max} = 10000$. Para o grupo *Large*: $iter_{max} = 10000$ e $iter_{sm} = 500$. Cada grupo de instâncias teve cada instância executada 5 vezes.

Para cada comparação do algoritmo R-GRASP em relação aos algoritmos da referência, o valor *RPD* (*Relative Percent Deviation*, dado por:

$$RPD = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \cdot 100 \tag{5.1}$$

é calculado para cada instância, sendo $Best_{sol}$ o melhor resultado conhecido na literatura e Heu_{sol} a média de 5 execuções do algoritmo R-GRASP. A média dos tempos de cada execução acompanha os resultados obtidos em segundos.

5.4 Resultados: Construção R-GRASP

A etapa de construção de uma solução é aquela em que reside toda a ideia principal da meta-heurística *GRASP*. Essa seção apresenta os resultados obtidos por quatro tipos diferentes de construção. Os resultados serão analisados separadamente para os três grupos de instâncias, sendo que, para o grupo *Large*, foi utilizado um subgrupo de amostras contendo 10% do número instâncias totais do conjunto.

5.4.1 Resultados para as Instâncias do grupo *Small*

Para as instâncias do grupo *Small*, os quatro tipos de construção alcançam melhorias sem muitas diferenças entre os tipos. O Tipo CT-2 se destaca na Tabela 5.1 com as melhores médias e também com o maior tempo médio.

Os pontos de dificuldade também coincidem visualmente. Na Figura 5.1 é possível visualizar os pontos de coincidência e divergência entre os métodos em relação aos subgrupos de instâncias do grupo *Small*. As divergências são mais acentuadas nos subgrupos 8×2, 8×4. A construção do Tipo 3 apresenta o melhor desempenho para estes grupos. Nos demais subgrupos, os pontos coincidem, aproximadamente.

Para verificar se existe diferença estatística entre os tipos de construção, foi realizado um teste *Paired Wilcoxon signed-rank* (Wilcoxon, 1947). O teste pareado considera a hipótese $H_0 : m_1 - m_2 = 0$ (ou seja, não há diferença estatística entre os tipos de construção) e $H_1 : m_1 - m_2 \neq 0$ (há diferença estatística entre os tipos

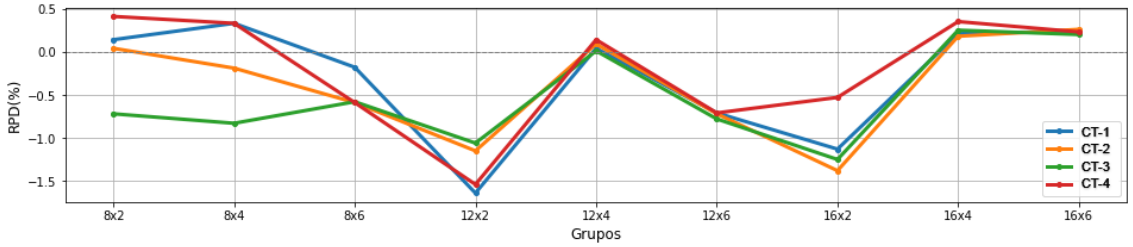


Figura 5.1: Tipos de Construção – Algoritmo *R-GRASP* Grupo *Small*

Tabela 5.1: Resultados para o conjunto de instâncias *Small*.

$n \times m$		CT-1			CT-2			CT-3			CT-4		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
8	2	**-0,61%**	0,04%	3,29	**-3,33%**	**-0,72%**	4,37	**-1,26%**	0,41%	3,91	**-1,08%**	0,14%	3,53
	4	**-0,26%**	**-0,19%**	3,25	**-1,23%**	**-0,83%**	5,29	0,16%	0,33%	5,09	0,26%	0,33%	4,35
	6	**-0,59%**	**-0,59%**	1,49	**-0,59%**	**-0,58%**	3,89	**-0,66%**	**-0,59%**	4,84	**-0,24%**	**-0,18%**	2,67
12	2	**-1,77%**	**-1,15%**	9,35	**-2,22%**	**-1,06%**	10,56	**-2,34%**	**-1,54%**	7,62	**-2,81%**	**-1,64%**	7,70
	4	**-0,11%**	0,09%	8,20	**-0,14%**	0,01%	12,47	0,02%	0,14%	10,83	**-0,17%**	0,04%	10,37
	6	**-0,85%**	**-0,72%**	6,17	**-0,88%**	**-0,78%**	11,70	**-0,83%**	**-0,71%**	11,57	**-0,83%**	**-0,71%**	8,98
16	2	**-1,85%**	**-1,38%**	21,25	**-1,68%**	**-1,25%**	21,03	**-1,56%**	**-0,53%**	13,58	**-2,69%**	**-1,13%**	14,79
	4	0,01%	0,18%	18,88	0,05%	0,25%	24,44	**-0,14%**	0,35%	18,49	**-0,21%**	0,21%	18,69
	6	0,22%	0,26%	13,75	0,19%	0,20%	21,59	0,20%	0,23%	20,36	0,20%	0,24%	16,82
Média		**-0,64%**	**-0,38%**	9,51	**-1,09%**	**-0,53%**	12,82	**-0,71%**	**-0,21%**	10,70	**-0,84%**	**-0,30%**	9,77

Tabela 5.2: p-valores entre os pares para o conjunto de instâncias *Small*.

	CT-4	CT-3	CT-2
CT-1	0.02575	0.05053	0.6474
CT-2	0.01001	0.01567	
CT-3	9,50E-04		

de construção). Os testes estatísticos são conduzidos no RStudio 2022.07.2+576 *"Spotted Wakerobin" Release for Ubuntu Bionic* e considera um nível de significância de 95% ($\alpha = 0,05$).

A Tabela 5.2 mostra os p-valores do teste para cada par de comparação entre os tipos. Os pares CT-1 vs. CT-3 e CT-1 vs. CT-2 não apresentaram evidências para rejeitar a hipótese nula H_0, com $p = 0,05032 > 0,05$ e $p = 0,6474 > 0,05$, respectivamente. Ou seja, não existe diferença estatisticamente significativa entre entre estes tipos. Já os demais pares são diferentes estatisticamente com nível de significâncias de 95% ($\alpha = 0.05$).

Os procedimentos de construção CT-1 e CT-3 do algoritmo *R-GRASP* se destacaram nos resultados do grupo *Small*, conforme mostrado no Gráfico 5.3, alcançando igualmente 15, 33% de vitórias, ou seja, soluções que superaram a literatura. Os tipos CT-2 e CT-4 alcançaram 13, 11% e 13, 78% respectivamente.

O Gráfico 5.3 mostra uma comparação entre os resultados que superaram ou coincidiram com a literatura. O destaque é para o *R-GRASP* CT-3, com 93, 56% das soluções que alcançaram ou superaram o resultado da literatura para cada instância do grupo *Small*. Em seguida, tem-se o algoritmo *R-GRASP* CT-2, com 92, 67%; *R-GRASP* CT-1, com 91, 56%; e, por fim, 89, 78% para o *R-GRASP* CT-4.

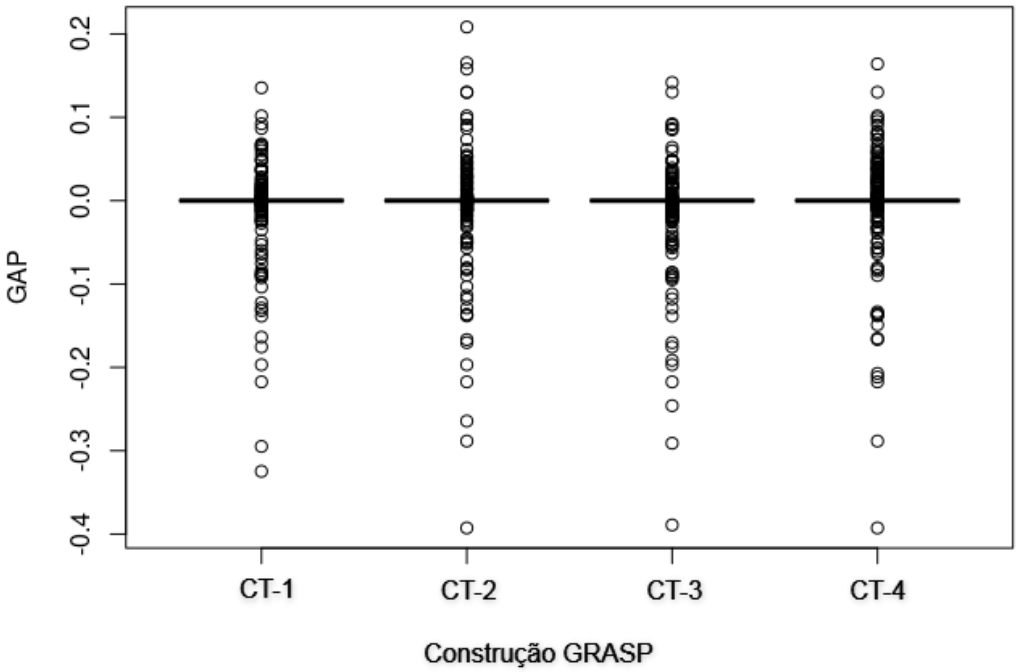


Figura 5.2: Tipos de Construção GRASP

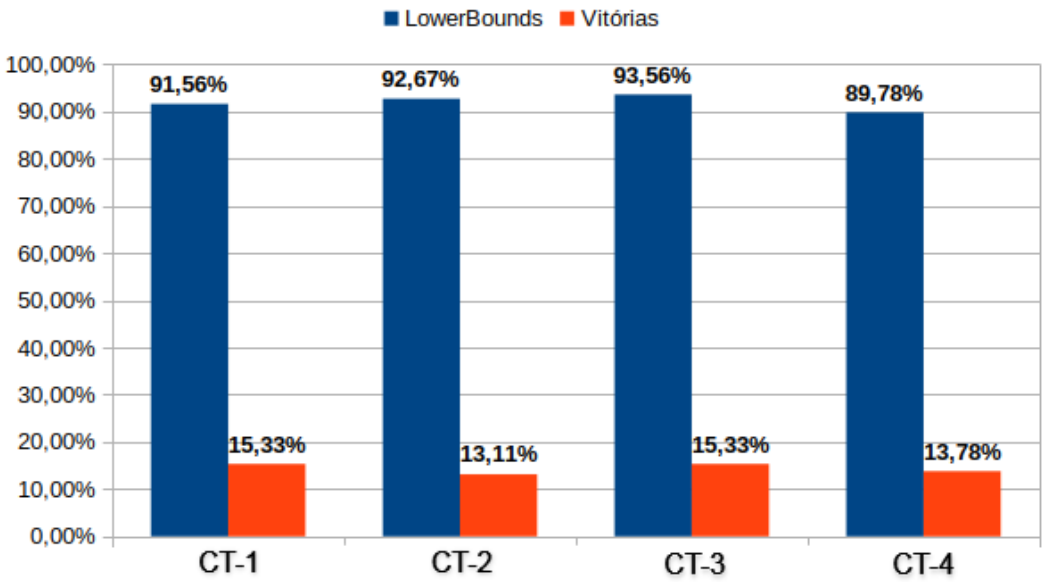


Figura 5.3: *Lower Bounds* e Vitórias do R-GRASP

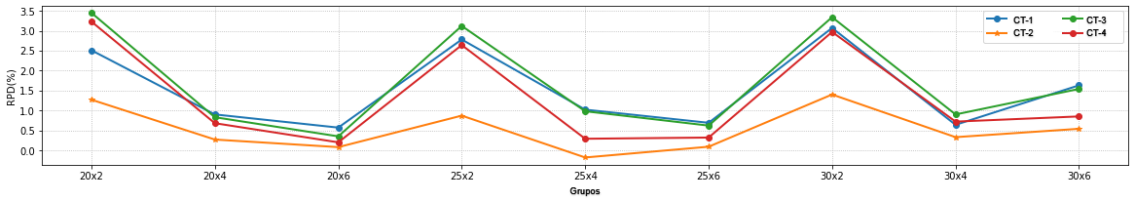


Figura 5.4: Tipos de Construção *R-GRASP* - Grupo *Medium*

Tabela 5.3: Resultados para o conjunto de instâncias *Medium*.

$n \times m$		CT-1			CT-2			CT-3			CT-4		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
20	2	1,43%	2,51%	1,07	0,02%	1,27%	54,68	2,34%	3,45%	1,32	1,83%	3,23%	4,25
	4	0,44%	0,90%	0,95	0,04%	0,27%	52,12	0,29%	0,83%	1,56	0,43%	0,68%	6,19
	6	0,28%	0,57%	0,57	0,03%	0,08%	31,56	0,19%	0,35%	1,21	0,13%	0,20%	6,2
25	2	1,85%	2,78%	2,22	**-0,17%**	0,87%	108,51	1,62%	3,12%	3,09	0,65%	2,64%	7,25
	4	0,36%	1,02%	2,24	**-0,38%**	**-0,18%**	85,81	0,38%	0,98%	3,26	**-0,09%**	0,29%	9,74
	6	0,22%	0,69%	1,38	**-0,10%**	0,09%	71	0,30%	0,62%	2,12	0,11%	0,32%	10,87
30	2	1,94%	3,07%	3,32	0,79%	1,40%	483,36	1,87%	3,34%	3,99	1,85%	2,97%	10,41
	4	0,23%	0,64%	2,69	0,10%	0,33%	111,6	0,50%	0,90%	3,83	0,51%	0,72%	12,81
	6	0,85%	1,63%	2,91	0,32%	0,54%	145,9	1,01%	1,54%	4,33	0,56%	0,85%	18,77
Média		0,84%	1,53%	1,93	0,07%	0,52%	127,17	0,94%	1,68%	2,75	0,66%	1,32%	9,61

Tabela 5.4: $p-$Valor para o conjunto de instâncias *Medium*.

	CT-4	CT-3	CT-2
CT-1	$1.07e-04$	0.03984	$< 2.2e-16$
CT-2	$< 2.2e-16$	$< 2.2e-16$	
CT-3	$1.27e-12$		

5.4.2 Resultados para as Instâncias do grupo *Medium*

Para as instâncias do grupo *Medium*, o algoritmo *R-GRASP* CT-2 apresenta destaque visual quando observado na Figura 5.4, uma vez que se mantém mais estável pelos subgrupos e alcança melhorias em relação à literatura nos grupos 25×2. O algoritmo *R-GRASP* CT-2 se manteve com os melhores valores médios em todos os subgrupos em relação aos demais tipos de construção.

Seguindo a mesma metodologia de comparação do grupo *small*, o teste *Paired Wilcoxon signed-rank* foi realizado para comparação dos tipos de construção um a um, sendo o resultado apresentado na Tabela 5.4.

Na Tabela 5.4 foram colocados os p-valores do teste para cada par de comparação entre os tipos. Todos os pares apresentaram evidências para rejeitar a hipótese nula H_0, ou seja, existe diferença estatisticamente significativa entre todos os tipos com nível de significâncias de 95% ($\alpha = 0,05$).

No Gráfico 5.6, o algoritmo *R-GRASP* CT-1 se destacou em resultados do grupo *Medium*, alcançando 20% de vitórias, superando os melhores resultados da literatura. Os tipo CT-2, CT-3 e CT-4 alcançaram $9,56\%$, 12% e $15,33\%$ de vitórias, respectivamente.

O Gráfico 5.6 mostra uma comparação entre os resultados que superaram ou coincidiram com a literatura. O destaque é para o algoritmo *R-GRASP* CT-1, com 79, 11% das soluções que alcançaram ou superaram o resultado da literatura para cada instância do grupo *Medium*. Em seguida, tem-se o algoritmo *R-GRASP* CT-2, com 61, 33%; *R-GRASP* CT-3, com 66, 89%; e, por fim, 72% para o *R-GRASP* CT-4.

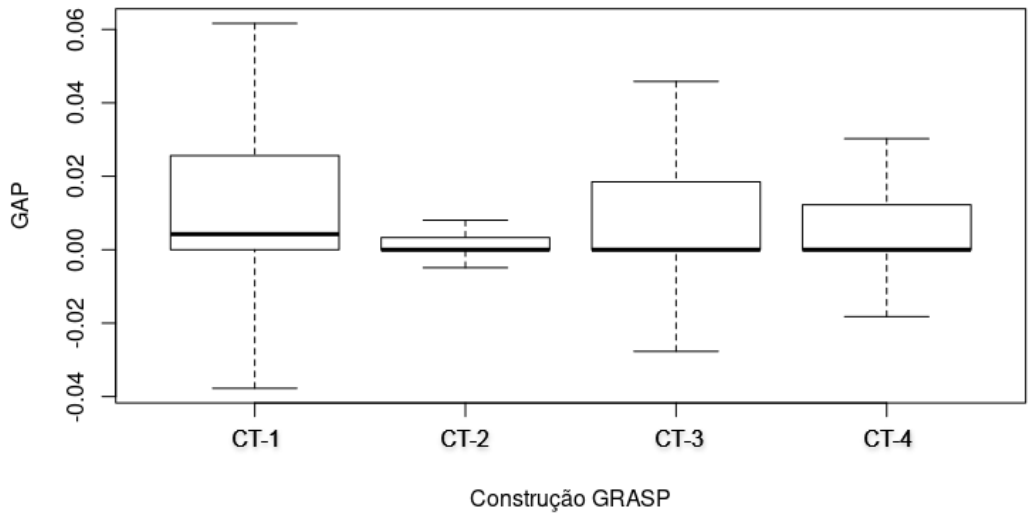


Figura 5.5: Tipos de Construção *R-GRASP* - Grupo *Medium*

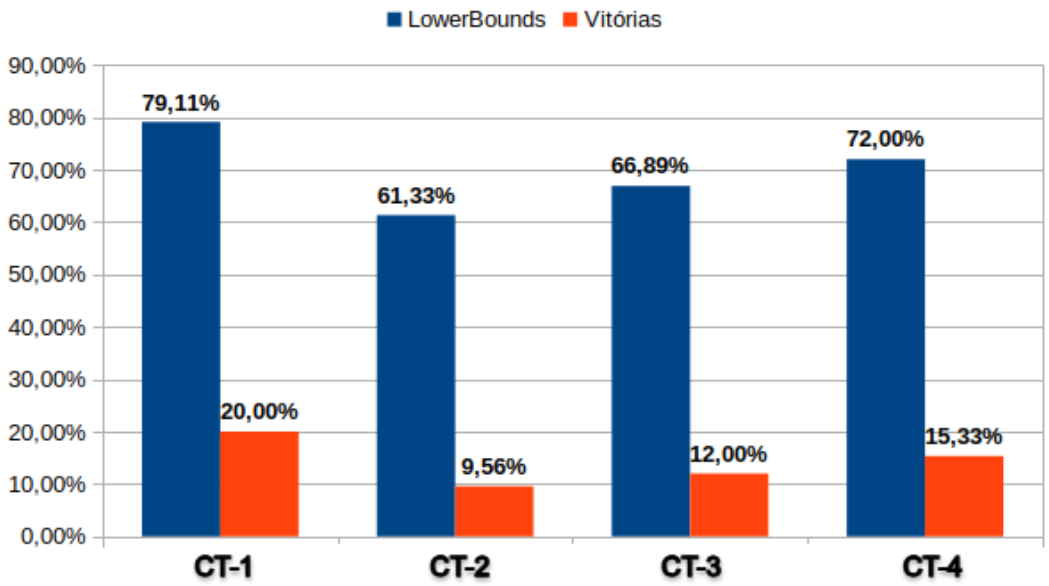


Figura 5.6: *Lower Bounds* e Vitórias do *R-GRASP* - Grupo *Medium*

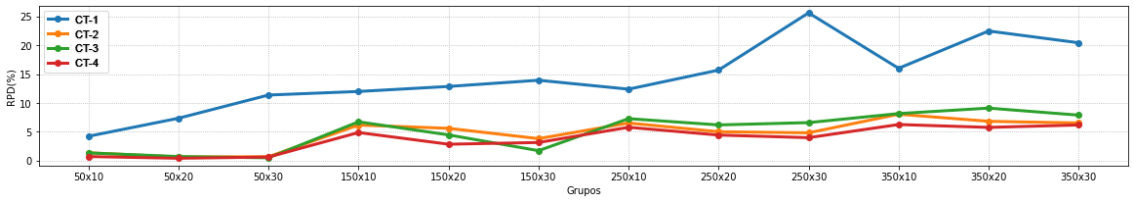


Figura 5.7: Tipos de Construção *R-GRASP* - Grupo *Large*.

Tabela 5.5: Resultados para o conjunto de instâncias *Large*.

$n \times m$		CT-1			CT-2			CT-3			CT-4		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
50	10	2,82%	4,79%	9,34	2,09%	2,89%	5,78	1,90%	2,82%	18,84	1,39%	2,45%	12,16
	20	6,44%	7,94%	15,74	0,56%	1,14%	8,74	0,60%	1,04%	33,52	0,40%	0,73%	27,4
	30	8,37%	9,35%	1,58	**-0,33%**	0,59%	1,42	**-0,14%**	0,43%	5,2	**-0,14%**	0,49%	10
150	10	10,19%	11,77%	194,96	4,33%	5,96%	124,76	5,26%	6,63%	375,16	3,93%	4,67%	238,96
	20	9,55%	12,90%	159,64	5,34%	5,84%	115,14	4,74%	5,04%	255,96	2,62%	3,04%	155,14
	30	13,52%	14,79%	152,48	3,83%	4,39%	79,3	4,42%	1,72%	294,36	2,98%	3,54%	272,62
250	10	10,89%	11,79%	656,56	5,43%	6,19%	893,92	6,30%	7,10%	1405,84	4,52%	5,50%	992,22
	20	14,29%	15,54%	744,76	4,77%	5,33%	803,4	6,16%	6,38%	1134,76	4,31%	4,64%	1093,54
	30	23,58%	26,67%	592,12	4,25%	4,97%	3707,32	5,76%	6,83%	993,44	3,70%	4,42%	646,08
350	10	14,38%	15,41%	1533,94	6,96%	7,86%	1007,74	7,10%	7,85%	2795,72	5,09%	5,90%	1805,26
	20	20,24%	22,12%	1312,32	6,25%	6,92%	825,54	8,25%	9,20%	2041,86	5,04%	5,77%	2324,94
	30	18,37%	21,40%	1213,06	5,01%	5,32%	10226,74	7,07%	7,43%	2456,96	4,76%	5,07%	2097,8
Média		12,72%	14,54%	548,88	4,04%	4,78%	1.483,32	4,79%	5,21%	984,30	3,22%	3,85%	806,34

Tabela 5.6: $p-$Valor para o conjunto de instâncias *Large*.

	CT-4	CT-3	CT-2
CT-1	$5.315e-10$	$5.836e-10$	$1.117e-9$
CT-2	$9.227e-08$	0.001158	
CT-3	$6.323e-09$		

5.4.3 Resultados para as Instâncias do grupo *Large*

Para as instâncias do grupo *Large*, o procedimento de construção CT-4 obteve os melhores valores médios de desvio e o CT-1 o menor tempo médio de execução. As abordagens de construção CT-2, CT-3 e CT-4 se mantiveram próximas nesses resultados, enquanto a abordagem de construção CT-1 obteve o melhor tempo médio, porém, a pior média dos desvios. Esse comportamento dos algoritmos pode ser visualizado no gráfico da Figura 5.7.

Para verificar se há diferença estatística significativa entre os tipos de construção para as instâncias *Large*, o teste *Paired Wilcoxon signed-rank* foi realizado, seguindo a metodologia descrita para o grupo *small*. A Tabela 5.6 mostra os p-valores do teste para cada par de comparação entre os tipos. Todos os pares apresentaram evidências para rejeitar a hipótese nula H_0, ou seja, existe diferença estatisticamente significativa entre todos os tipos com nível de significâncias de 95% ($\alpha = 0,05$).

Observando o *boxplot* da Figura 5.8, os tipos CT-2 e CT-4 competem como os mais promissores. Na comparação das diferenças das medianas dos dois grupos, tem-se a mediana das medianas referente ao tipo CT-2 igual a $0,044$; e, para CT-4, igual

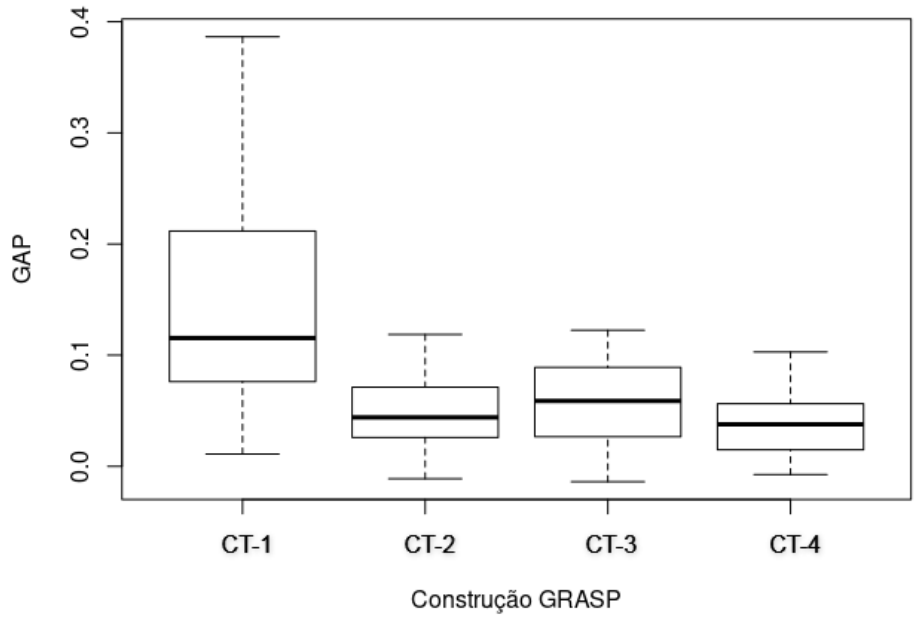


Figura 5.8: Tipos de Construção *R-GRASP* - Grupo *Large*.

a 0,038. A mediana da diferença par-a-par é $dif = 0,007$. Conclui-se, portanto, que, para este grupo de instâncias, o procedimento de construção CT-4 é superior aos demais tipos.

Em relação aos valores médios que coincidem ou superam os valores médios da literatura, observa-se na Figura 5.9 que a construção CT-4 se destacou com 13,33% de médias iguais ou melhores que a literatura, sendo 3,33% superando os melhores resultados conhecido. A construção CT-3 apresentou 5% de médias que superaram os melhores resultados conhecidos e 3,33% que alcançaram os mesmas médias da literatura.

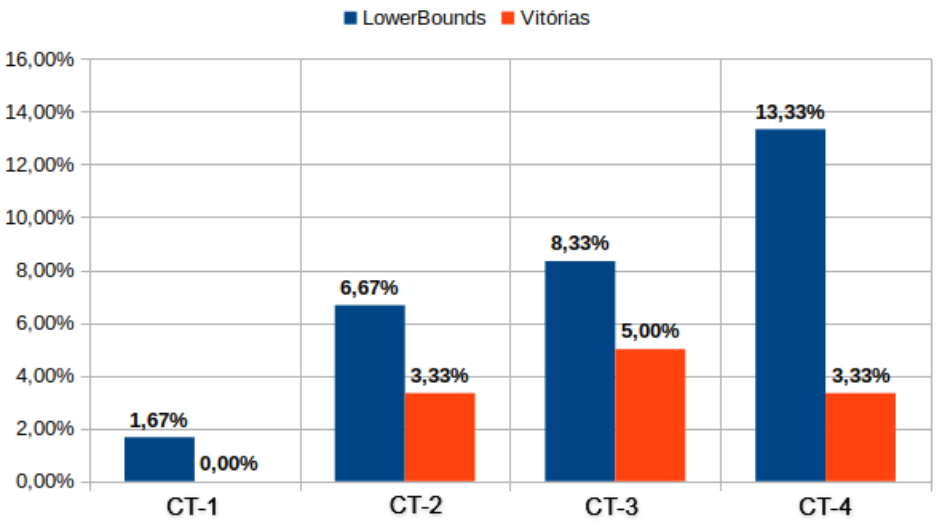


Figura 5.9: *Upperbounds R-GRASP* - Grupo *Large*.

5.5 Comparação: *First Improvement vs. Best Improvement*

A etapa de busca local é responsável por refinar as soluções construídas, ou seja, obter o melhor vizinho de uma vizinhança definida. O método de busca local utilizado foi o algoritmo *Variable Neighborhood Descent* (VND), juntamente com as estratégias *Best Improvement* e *First Improvement*. Essa seção apresenta os resultados obtidos para os grupos de instâncias testados. O método de construção GRASP utilizado neste e nos experimentos seguintes é o denominado CT-2, explicado na Seção 4.2.1.2, que obteve os melhores resultados, como apresentados na Seção 5.4.

5.5.1 Resultados: *R-GRASP* VND grupo *Small*

Para as instâncias do grupo *Small*, os dois métodos de busca local obtiveram resultados animadores, com médias gerais de resultados apresentando melhoria em relação à literatura nos dois procedimentos testados. O método *First Improvement* obteve a melhor média geral e teve o maior tempo médio em relação ao método *Best Improvement*.

Tabela 5.7: Resultados para o conjunto de instâncias *Small*.

$n \times m$		*First Improvement*			*Best Improvement*		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
8	2	**-2,88%**	**-1,31%**	5,10	**-1,08%**	0,14%	3,53
	4	**-0,36%**	0,38%	5,62	0,26%	0,33%	4,35
	6	**-0,58%**	**-0,58%**	2,84	**-0,24%**	**-0,18%**	2,67
12	2	**-2,64%**	**-2,02%**	13,28	**-2,81%**	**-1,64%**	7,70
	4	**-0,23%**	**-0,01%**	13,21	**-0,17%**	0,04%	10,37
	6	**-0,76%**	**-0,66%**	10,91	**-0,83%**	**-0,71%**	8,98
16	2	**-2,28%**	**-1,33%**	28,00	**-2,69%**	**-1,13%**	14,79
	4	**-0,05%**	0,22%	26,44	**-0,21%**	0,21%	18,69
	6	0,20%	0,22%	19,57	0,20%	0,24%	16,82
Média		**-1,06%**	**-0,57%**	13,89	**-0,84%**	**-0,30%**	9,77

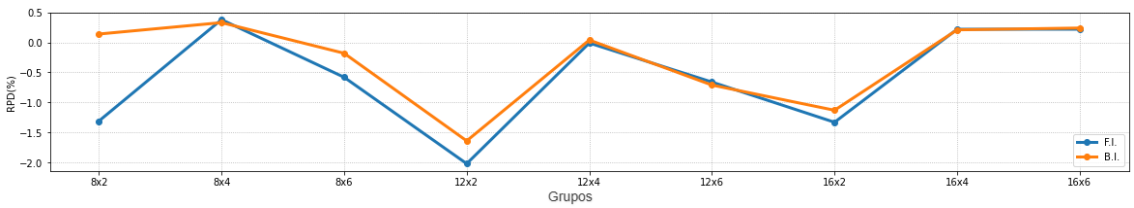


Figura 5.10: VND - Grupo *Small*.

De forma geral, como pode-se observar no gráfico de linhas da Figura 5.10, as médias coincidem, com pequenas diferenças em todos os subgrupos, com exceção

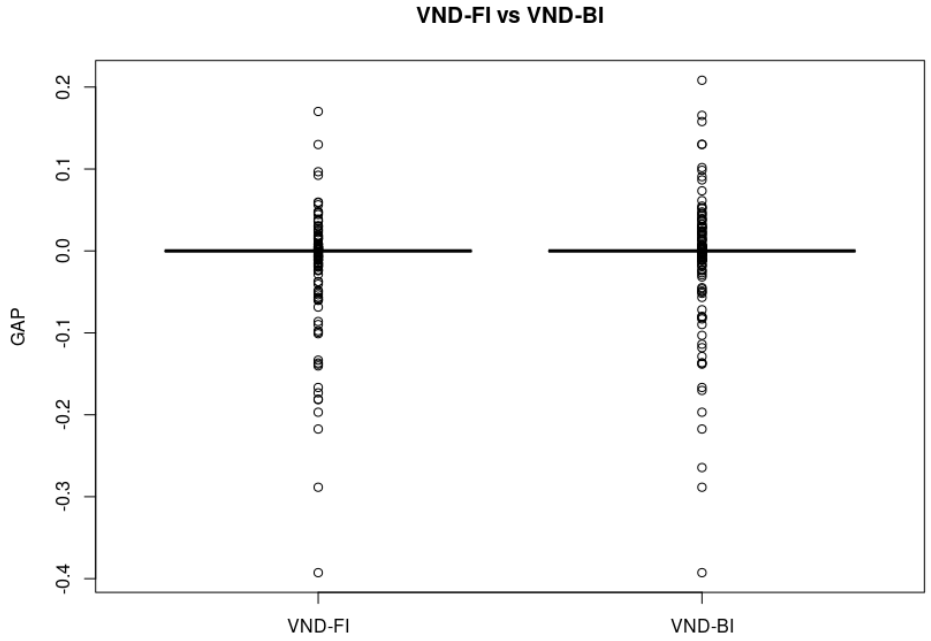


Figura 5.11: VND - Grupo *Small*.

do subgrupo 8×2, em que o método *First Improvement* abre maior diferença em relação aos demais métodos.

O teste pareado de *Wilcoxon* foi conduzido para os resultados do algoritmo VND no grupo *Small*. O resultado obtido foi uma estatística de teste $V = 3817$ e p-valor$= 0,00742$. Portanto, concluí-se que, para o grupo *Small*, há evidências para se rejeitar a hipótese nula $H_0 : m_1 - m_2 = 0$, pois $p = 0,00742 < 0,05$, ou seja, existe diferença significativa entre os métodos *First Improvement* e *Best Improvement*, indicando que o método *First Improvement* apresenta melhores resultados médios quando comparado ao método *Best Improvement*.

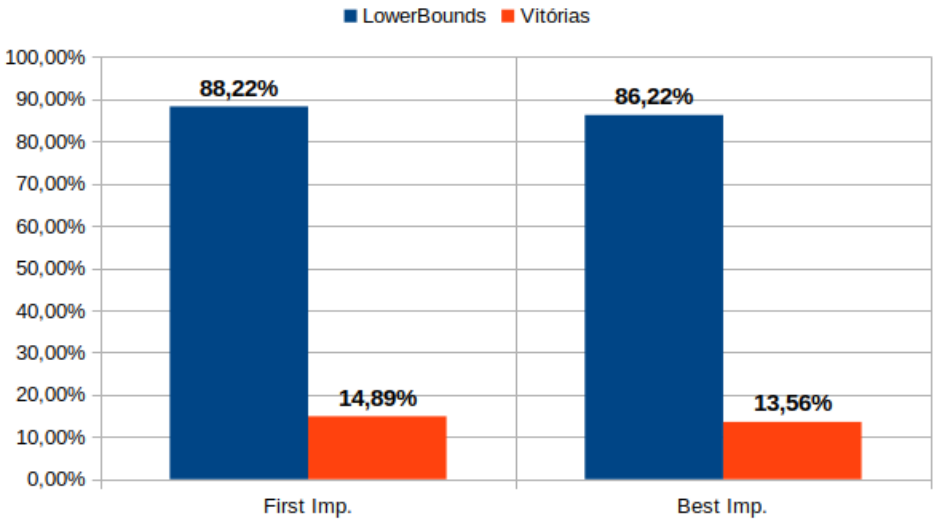


Figura 5.12: *Upperbounds* R-GRASP - Grupo *Small*.

De acordo com o gráfico da Figura 5.12, os dois métodos testados contabilizaram

82, 22% e 86, 22% de soluções que coincidiram ou melhoraram em relação aos valores médios da literatura, com 14, 89% e 13, 56% de soluções que superaram os melhores resultados conhecidos. Em termos quantitativos, o método VND-*First Improvement* esteve ligeiramente à frente do método VND-*Best Improvement*.

5.5.2 Resultados: *R-GRASP* VND grupo *Medium*

Para as instâncias do grupo *Medium*, os dois métodos de busca local mostraram comportamentos semelhantes, com ligeira vantagem qualitativa do método *Best Improvement* sobre o método *First Improvement*. Nesse caso, a diferença do tempo de execução do métodos foi de mais de 10× maior para o método *Best Improvement*. Essas observações são feitas a partir da Tabela 5.8 e do gráfico da Figura 5.13. O método *Best Improvement* obteve melhorias em relação à literatura nas médias de melhores soluções obtidas para todo o subgrupo de 25 tarefas e se manteve superior à literatura no subgrupo 25×4 para a média das médias.

Tabela 5.8: Resultados para o conjunto de instâncias *Medium*.

$n \times m$		*First Improvement*			*Best Improvement*		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
20	2	0,80%	2,26%	6,82	0,02%	1,27%	54,68
	4	0,06%	0,42%	6,63	0,04%	0,27%	52,12
	6	0,12%	0,21%	4,12	0,03%	0,08%	31,56
25	2	0,33%	1,82%	13,86	**-0,17%**	0,87%	108,51
	4	**-0,20%**	0,13%	11,04	**-0,38%**	**-0,18%**	85,81
	6	0,12%	0,33%	9,20	**-0,10%**	0,09%	71,00
30	2	1,25%	2,20%	21,11	0,79%	1,40%	483,36
	4	0,35%	0,57%	14,24	0,10%	0,33%	111,60
	6	0,46%	0,86%	18,47	0,32%	0,54%	145,90
Média		0,37%	0,98%	11,72	0,07%	0,52%	127,17

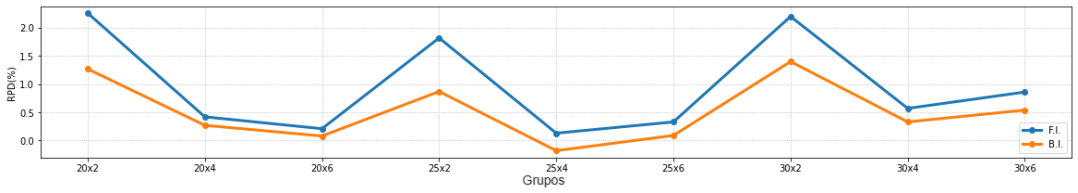


Figura 5.13: VND - Grupo *Medium*.

O teste pareado de *Wilcoxon* foi realizado para os resultados do algoritmo VND no grupo *Medium*. O resultado obtido foi uma estatística de teste $V = 1494$ e p-valor$< 2.2e-16$. Portanto, concluí-se que, para o grupo *Medium*, há evidências para rejeitar a hipótese nula H_0, pois $p < 2,2e-16 < 0,05$, ou seja, existe diferença significativa entre os métodos *First Improvement* e *Best Improvement*, indicando que o

método *Best Improvement* apresenta melhores resultados médios quando comparado ao método *First Improvement*.

Apesar de o teste de *Wilcoxon* pareado indicar diferença significativa entre os conjuntos, a comparação entre as diferenças das medianas dos dois grupos foi inconclusiva, com a mediana das medianas do VND-FI e do VND-BI tendo valor 0. A mediana da diferença par-a-par é $dif = 0,0$.

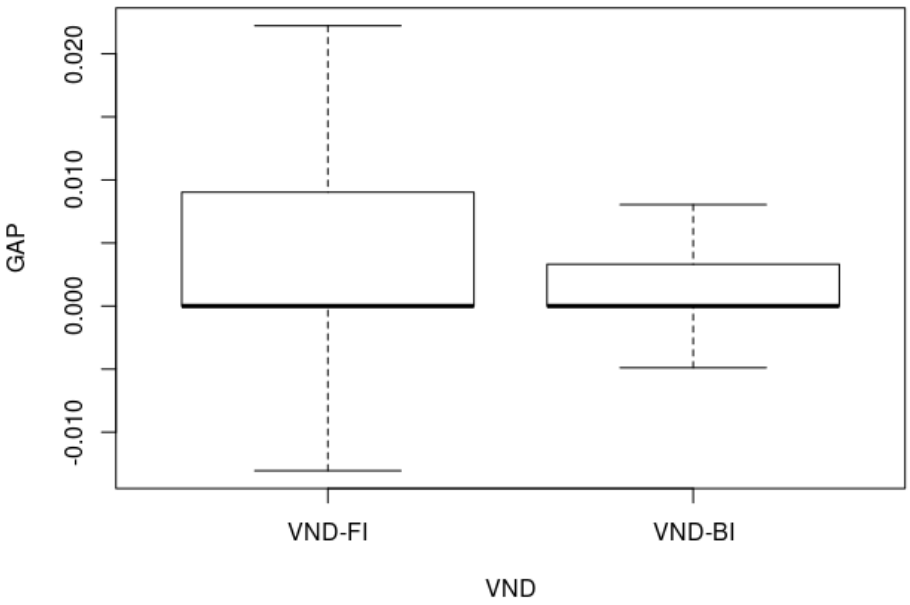


Figura 5.14: VND - Grupo *Medium*.

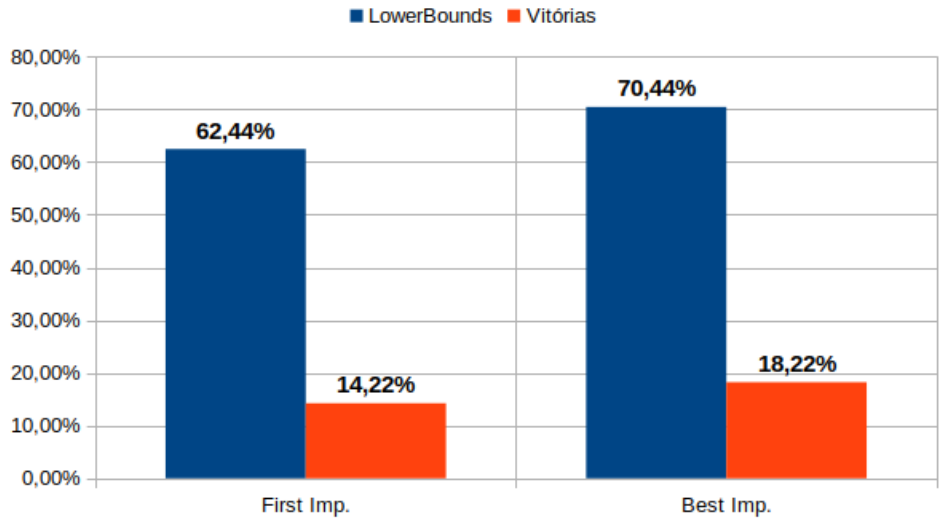


Figura 5.15: *Lower Bounds* R-GRASP - Grupo *Medium*.

Observando o gráfico da Figura 5.15, para instâncias do grupo *Medium*, os dois métodos de busca local obtiveram bons resultados quanto a soluções que alcançaram limites da literatura ou superaram esses limites. A maior marca qualitativa foi

do método *Best Improvement*, com 70, 44% de soluções que alcançaram ou superaram a literatura, sendo 18, 22% que superaram os limites da literatura. O método *First Improvement* alcançou o mesmo quesito qualitativo, com 62, 44% das soluções, sendo 14, 22% apenas vitórias. O método *Best Improvement* obteve melhoras mais expressivas em relação ao método *First Improvement* para este conjunto de instâncias.

5.5.3 Resultados: *R-GRASP* VND grupo *Large*

Para as instâncias do grupo *Large*, os dois métodos de busca local mostraram comportamentos semelhantes, como pode ser observado no gráfico de linhas da Figura 5.16, porém com superioridade do método *First Improvement* em todos os subgrupos de instâncias. Essa superioridade se torna mais expressiva a partir do subgrupo 150×10. O método *First Improvement* para esse grupo de instâncias obteve médias superiores e tempo de execução $2\times$ menor que o método *Best Improvement* observado na Tabela 5.9. Houve superação de limites da literatura nos subgrupos 50×20 e 50×30.

Tabela 5.9: Resultados para o conjunto de instâncias *Large*.

$n \times m$		*First Improvement*			*Best Improvement*		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
50	10	1,02%	1,80%	9,70	1,11%	1,91%	3,70
	20	**-0,54%**	0,06%	7,87	**-0,09%**	0,48%	3,02
	30	**-0,10%**	0,41%	4,96	0,08%	0,81%	1,17
150	10	2,94%	4,01%	112,65	3,52%	4,55%	95,22
	20	2,78%	3,62%	101,31	3,59%	4,58%	74,98
	30	1,79%	2,57%	79,96	3,49%	4,18%	50,07
250	10	3,84%	4,48%	639,33	4,38%	5,41%	625,47
	20	3,45%	4,07%	437,79	4,77%	5,51%	548,77
	30	3,16%	3,70%	340,08	3,98%	4,61%	2.948,41
350	10	4,04%	4,72%	1.394,16	4,88%	5,76%	894,16
	20	3,65%	4,39%	1.275,86	5,26%	5,91%	862,29
	30	3,88%	4,60%	1.524,99	5,21%	5,65%	8.123,92
Média		2,49%	3,20%	494,06	3,35%	4,11%	1.185,93

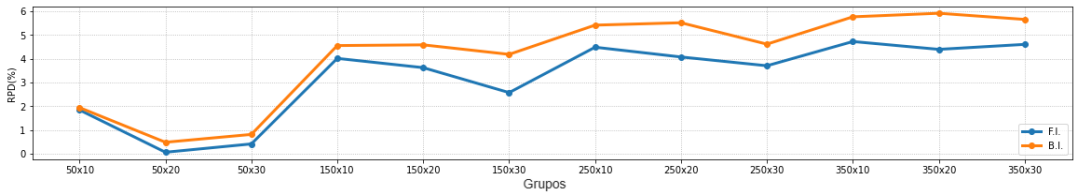


Figura 5.16: VND - Grupo *Large*.

O teste pareado de *Wilcoxon* foi realizado para os resultados do algoritmo VND no grupo *Large*. O resultado obtido foi uma estatística de teste $V = 144098$ e p-valor$< 2, 2e - 16$. Portanto, concluí-se que, para o grupo *Large*, há evidências para

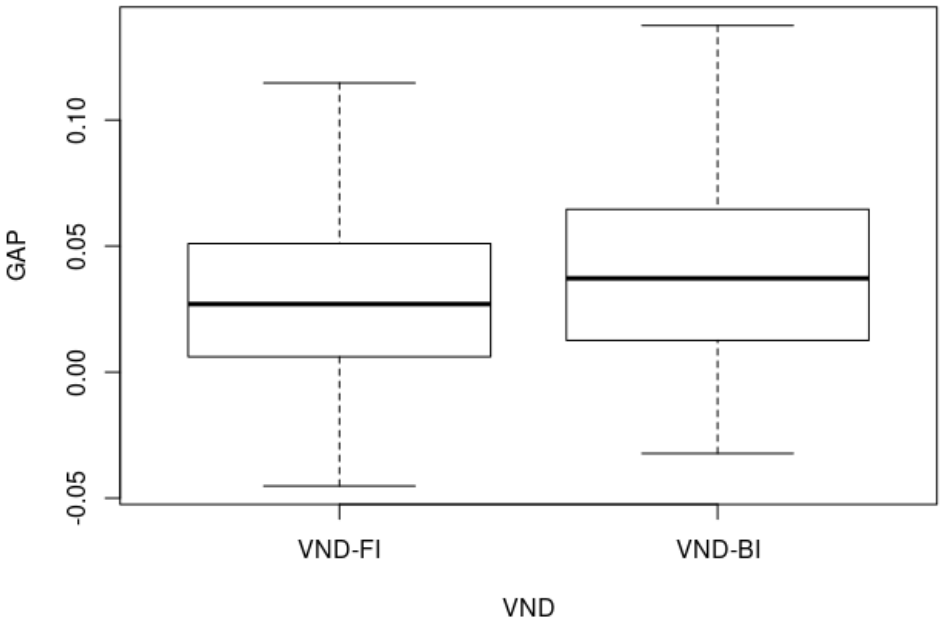


Figura 5.17: VND - Grupo *Large*

rejeitar a hipótese nula H_0, pois $p < 2,2e-16 < 0,05$, ou seja, existe diferença significativa entre os métodos *First Improvement* e *Best Improvement*, indicando que o método *First Improvement* apresenta melhores resultados médios quando comparado ao método *Best Improvement*.

Na comparação das diferenças das medianas dos dois grupos, tem-se a mediana das medianas referente ao VND-FI igual a $0,027$ e para o VND-BI igual a $0,037$. A mediana da diferença par-a-par é $dif = -0,008$. Concluí-se, portanto, que o método *First Improvement* para este grupo de instâncias foi superior ao *Best Improvement*.

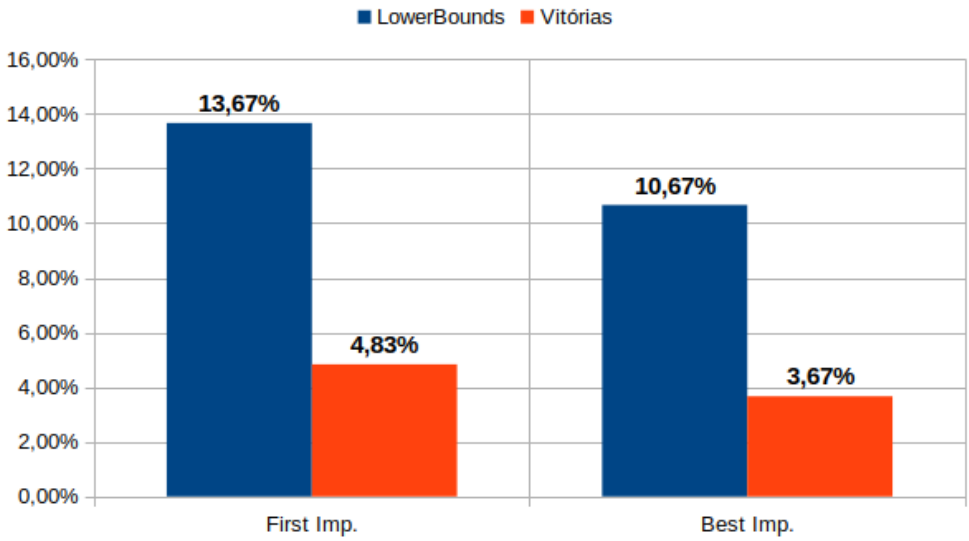


Figura 5.18: *Lower Bounds* R-GRASP - Grupo *Large*

De acordo com o gráfico da Figura 5.18, para instâncias do grupo *Large*, a maior marca qualitativa foi do método *First Improvement*, com 13,67% de soluções que alcançaram ou superaram a literatura, sendo 4,83% que superou os limites da literatura. O método *Best Improvement* alcançou o mesmo nível de qualidade, com 10,67% das soluções iguais ou melhores que a literatura, sendo 3,67% apenas vitórias. O método *First Improvement* foi considerado melhor em relação ao método *Best Improvement* para este conjunto de instâncias.

5.6 Resultados: Conjuntos *Alpha*

Essa seção apresenta os resultados dos testes realizados para três diferentes conjuntos do parâmetro α, conforme mostrados na Tabela 4.1 da Seção 4.2.2. Os conjuntos se expandem de regiões mais gulosas próximas de 0 para regiões mais aleatórias através desses três níveis, como foi explicado na Seção 4.2.2. O método de construção utilizado pelo *R-GRASP* foi o CT-2 para todos os conjuntos α e grupos de instâncias.

5.6.1 Resultados: Conjuntos Alfa grupo *Small*

Para as instâncias do grupo *Small*, os dois conjuntos $C2$ e $C3$ mostraram comportamentos semelhantes, como pode ser observado no gráfico de linhas da Figura 5.19, porém, com superioridade do conjunto $C3$ em todos os subgrupos de instâncias, superando os limites da literatura em parte dos subgrupos. O tempo de execução do algoritmo *R-GRASP* para os três conjuntos não teve diferença relevante, conforme mostra a Tabela 5.10.

Tabela 5.10: Resultados para o conjunto de instâncias *Small*.

$n \times m$		C1			C2			C3		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
8	2	5,27%	5,70%	3,67	**-1,00%**	1,19%	3,62	**-1,08%**	0,14%	3,53
	4	2,57%	2,57%	4,14	1,38%	1,49%	4,44	0,26%	0,33%	4,35
	6	2,23%	2,23%	2,69	0,22%	0,22%	2,73	**-0,24%**	**-0,18%**	2,67
12	2	4,04%	4,26%	7,34	**-1,68%**	**-0,67%**	7,52	**-2,81%**	**-1,64%**	7,70
	4	1,74%	1,76%	10,40	0,47%	0,60%	10,89	**-0,17%**	0,04%	10,37
	6	0,41%	0,46%	9,08	**-0,60%**	**-0,43%**	9,16	**-0,83%**	**-0,71%**	8,98
16	2	2,62%	3,91%	13,91	**-0,68%**	0,53%	14,77	**-2,69%**	**-1,13%**	14,79
	4	0,58%	0,81%	18,01	0,28%	0,52%	18,49	**-0,21%**	0,21%	18,69
	6	0,36%	0,42%	17,65	0,20%	0,24%	17,11	0,20%	0,24%	16,82
Média		2,20%	2,46%	9,65	**-0,16%**	0,41%	9,86	**-0,84%**	**-0,30%**	9,77

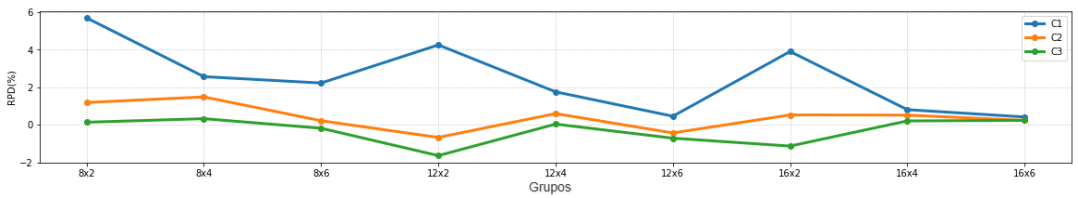


Figura 5.19: Conjunto Alfa - Grupo *Small*.

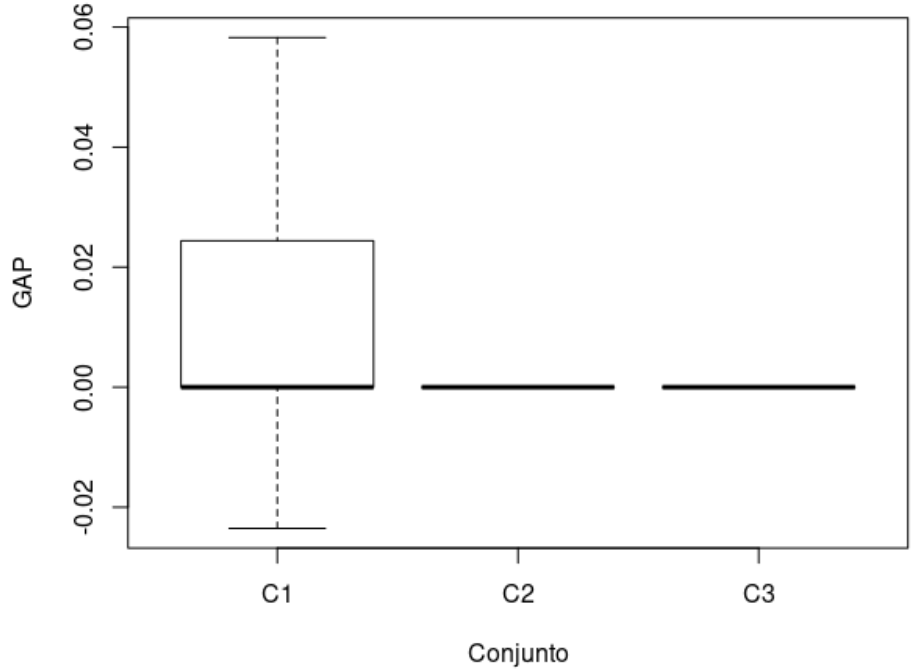


Figura 5.20: Conjunto Alfa - Grupo *Small*.

O teste pareado de *Wilcoxon* foi realizado para os resultados dos três conjuntos para o parâmetro α no grupo *Small*. O resultado obtido para todas as combinações entre os conjuntos foram p-valores$< 0,05$. Portanto, concluí-se que, para o grupo *Small*, há evidências para rejeitar a hipótese nula H_0, indicando que o conjunto $C3$ apresenta os menores valores médios.

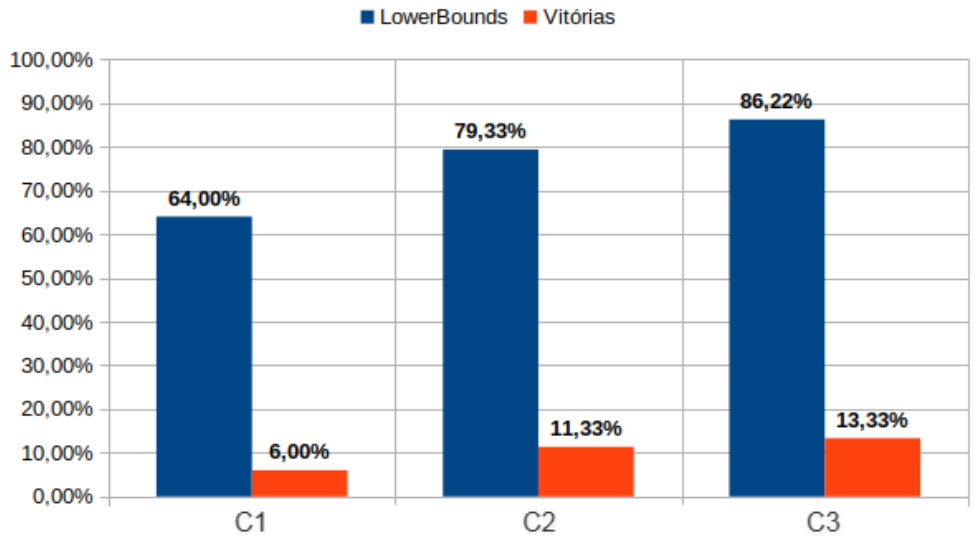


Figura 5.21: *Lower Bounds R-GRASP* - Grupo *Small*.

De acordo com o gráfico da Figura 5.21, para instâncias do grupo *Small*, a maior marca qualitativa foi obtida pelo conjunto $C3$ com $86,22\%$ de soluções que

alcançaram ou superaram a literatura, sendo 13,33% que superou os limites da literatura. O conjunto $C3$, conjunto mais amplo, foi considerado melhor em relação aos demais conjuntos para este grupo de instâncias.

5.6.2 Resultados: Conjuntos Alfa grupo *Medium*

Para as instâncias do grupo *Medium* o conjunto $C3$ se mostrou consideravelmente superior aos demais conjuntos, de acordo com o o gráfico de linhas da Figura 5.22 em todos os subgrupos de instâncias, porém, com tempo de execução do algoritmo consideravelmente superior que a execução do *R-GRASP* para os conjuntos $C1$ e $C2$, como observado na Tabela 5.11.

Tabela 5.11: Resultados para o conjunto de instâncias *Medium*.

$n \times m$		C1			C2			C3		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
20	2	5,58%	6,91%	0,59	3,44%	4,93%	0,74	0,02%	1,27%	54,68
	4	0,96%	1,45%	0,91	0,59%	1,23%	0,99	0,04%	0,27%	52,12
	6	0,37%	0,65%	0,76	0,20%	0,47%	0,78	0,03%	0,08%	31,56
25	2	3,53%	6,02%	1,16	3,73%	4,95%	1,39	**-0,17%**	0,87%	108,51
	4	0,65%	1,12%	1,52	0,43%	0,89%	1,88	**-0,38%**	**-0,18%**	85,81
	6	0,46%	0,88%	1,35	0,41%	0,74%	1,79	**-0,10%**	0,09%	71,00
30	2	4,53%	5,88%	1,81	3,00%	4,72%	2,13	0,79%	1,40%	483,36
	4	0,99%	1,43%	1,81	0,91%	1,28%	2,28	0,10%	0,33%	111,60
	6	1,31%	1,74%	3,00	1,10%	1,63%	3,74	0,32%	0,54%	145,90
Média		2,04%	2,90%	1,43	1,53%	2,32%	1,75	0,07%	0,52%	127,17

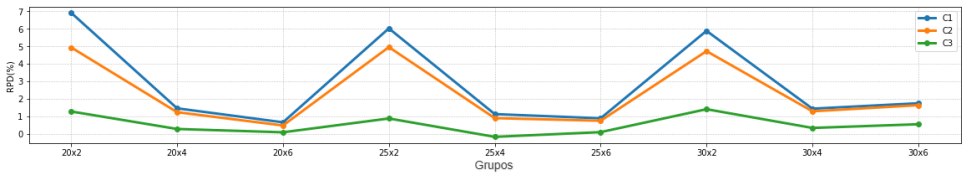


Figura 5.22: Conjunto Alfa - Grupo *Medium*.

No grupo *Medium*, o teste pareado de *Wilcoxon* foi realizado para os resultados dos três conjuntos para o parâmetro α. O resultado do teste obtido para todas as combinações entre os conjuntos foram p-valores $< 0,05$. Portanto, concluí-se que, para o grupo *Medium*, há evidências para rejeitar a hipótese nula H_0, indicando que o conjunto $C3$ apresenta os menores valores médios.

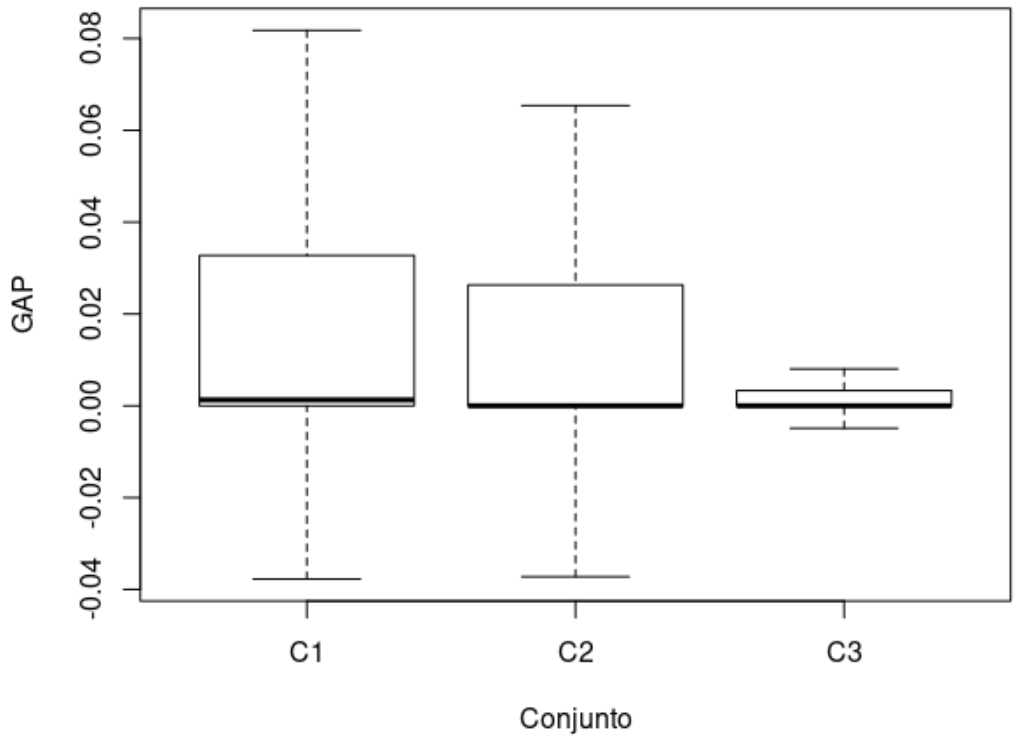


Figura 5.23: Conjunto Alfa - Grupo *Medium*.

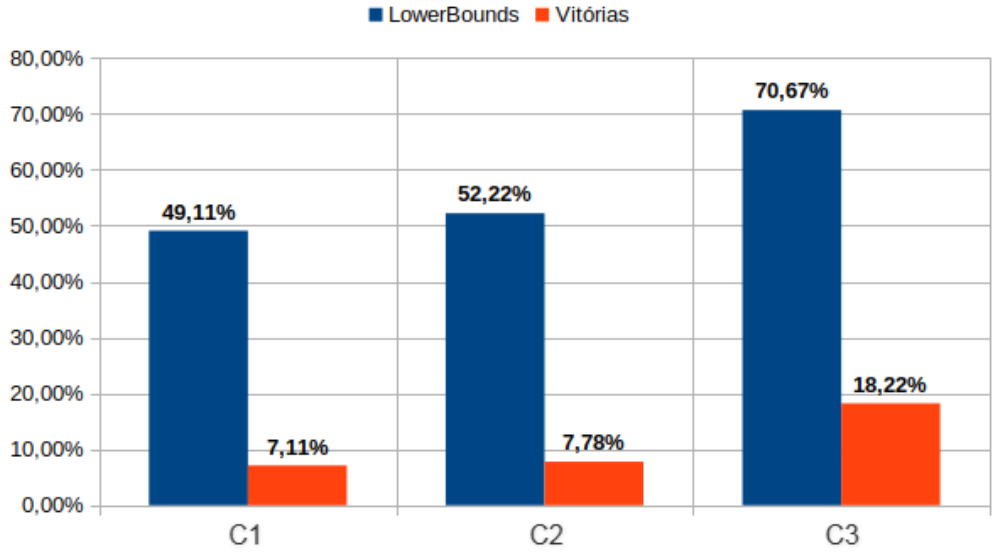


Figura 5.24: *Lower Bounds R-GRASP* - Grupo *Medium*.

De acordo com o gráfico da Figura 5.24, para instâncias do grupo *Medium*, a maior marca qualitativa foi o conjunto $C3$ com $70,67\%$ de soluções que alcançaram ou superaram a literatura, sendo $18,22\%$ que superaram os limites da literatura. O conjunto $C3$, conjunto mais amplo, foi considerado melhor em relação aos demais conjuntos para este grupo de instâncias.

5.6.3 Resultados: Conjuntos Alfa grupo *Large*

Para as instâncias do grupo *Large*, os conjuntos se comportaram de forma semelhante, com médias ligeiramente superiores para o conjunto $C1$ e aproximadamente $1/3$ do tempo de execução referente ao conjunto com maior tempo de execução, conforme mostra a Tabela 5.12. Os três grupos se aproximaram dos limites da literatura no subgrupo 50×, de acordo com a Figura 5.25.

Tabela 5.12: Resultados para o conjunto de instâncias *Large*.

$n \times m$		C1			C2			C3		
		Best	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
50	10	2,47%	3,00%	5,88	1,82%	2,82%	7,66	2,09%	2,89%	5,78
	20	0,24%	0,75%	15,38	0,40%	0,85%	18,74	0,56%	1,14%	8,74
	30	**-0,34%**	0,42%	1,72	**-0,33%**	0,71%	1,74	**-0,33%**	0,59%	1,42
150	10	4,40%	4,98%	130,2	4,16%	5,41%	160,92	4,33%	5,96%	124,76
	20	3,09%	3,71%	86,36	3,37%	4,04%	85,14	5,34%	5,84%	115,14
	30	3,39%	3,99%	144,8	3,57%	4,29%	134,78	3,83%	4,39%	79,3
250	10	4,83%	5,47%	700,72	5,01%	5,64%	937,16	5,43%	6,19%	893,92
	20	4,40%	4,82%	601,68	4,87%	5,13%	634,62	4,77%	5,33%	803,4
	30	4,89%	5,75%	444,74	5,11%	5,88%	387,6	4,25%	4,97%	3707,32
350	10	6,60%	7,35%	1220,76	6,91%	7,48%	1218,34	6,96%	7,86%	1007,74
	20	5,46%	6,19%	1041,3	5,70%	6,26%	1484,96	6,25%	6,92%	825,54
	30	4,32%	4,78%	1807,14	4,61%	5,20%	947,02	5,01%	5,32%	10226,74
Média		3,65%	4,27%	516,72	3,77%	4,48%	501,56	4,04%	4,78%	1.483,32

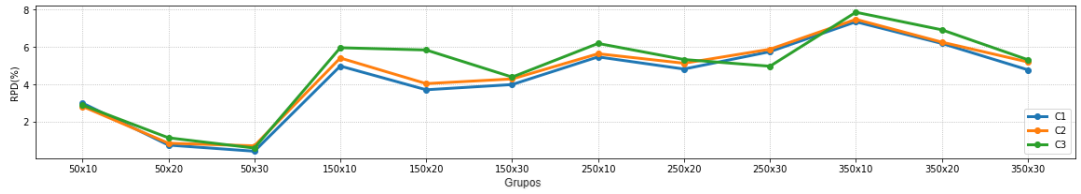


Figura 5.25: Conjunto Alfa - Grupo *Large*.

Para a análise estatística dos resultados obtidos para as instâncias *Large*, foi conduzido um teste de normalidade *Shapiro-Wilk normality test*. Os p-valores da Tabela 5.13 para todos os tipos testados apontaram a falta de evidências para rejeitar a hipótese nula H_0, ou seja, os dados são provenientes de uma distribuição normal com significância de 95%.

Tabela 5.13: Teste de Normalidade - *Large*.

	C1	C2	C3
p-valor	0.04913	0.1784	0.4703

Para verificar se há a existência de diferença significativa entre os dados, foi conduzido uma análise de variância (ANOVA). Como observado na Tabela 5.14 o p-valor$= 0,683 > 0.05$ indica que não há diferença significante entre os conjuntos α utilizados no *R-GRASP* para os conjuntos de amostras testados com grau de significância de 95%.

Tabela 5.14: Anova - *Large*.

	Df	*Sum Sq*	*Mean Sq*	*F value*	*Pr(>F)*
Grupo	2	0.00081	0.0004065	0.382	0.683
Residuals	177	0.18822	0.0010634		

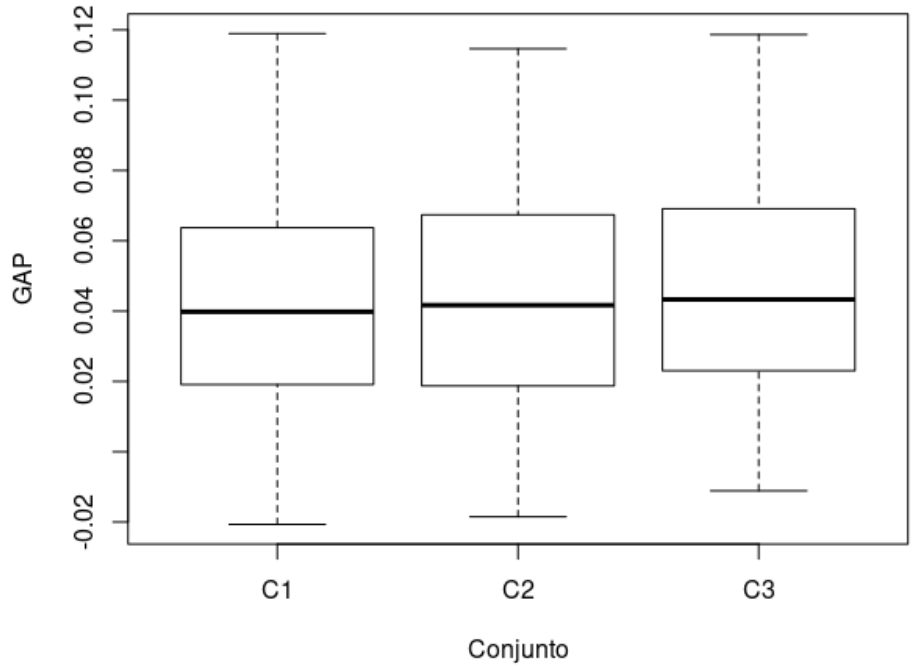


Figura 5.26: Conjunto Alfa - Grupo *Large*.

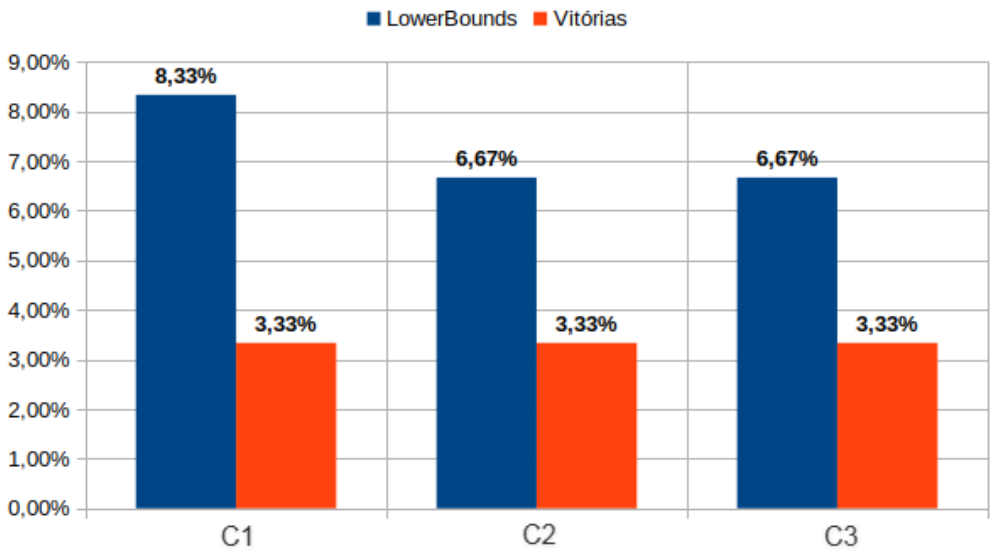


Figura 5.27: *Lower Bounds R-GRASP* - Grupo *Large*.

De acordo com o gráfico da Figura 5.27, para instâncias do grupo *Large*, a maior

marca qualitativa foi o conjunto $C1$, com $8,33\%$ de soluções que alcançaram ou superaram a literatura, sendo $3,33\%$ de soluções que superaram os limites da literatura. O conjunto $C1$, conjunto de menor amplitude, foi considerado melhor em relação aos demais conjuntos para este grupo de instâncias pela vantagem de tempo de execução dado que os resultados são estatisticamente iguais.

5.7 Resultados: Algoritmo *R-GRASP vs.* Algoritmos da Literatura

Esta seção apresenta uma comparação entre o método proposto nesta dissertação com resultados de Fanjul-Peyro et al. (2017) (fonte dos melhores resultados dentre todos os métodos apresentados) e Vallada et al. (2019), que propôs quatro diferentes heurísticas.

5.7.1 Resultados: *R-GRASP vs.* Literatura - grupo *Small*

Para o grupo de instâncias *Small*, o algoritmo *R-GRASP* teve seus valores médios superiores a todos os procedimentos comparados, superando os limites inferiores de Fanjul-Peyro et al. (2017). O tempo médio foi de $13,89$ segundos, com média de $1258,67$ segundos do segundo melhor procedimento nomeado na Tabela 5.15 como *FANJUL*.

Tabela 5.15: Resultados para o conjunto de instâncias *Small*.

$n \times m$		FANJUL		EIG		ESS		M5		SS		R-GRASP		
		Avg	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
8	2	0,00%	548,58	0,00%	0,28	0,09%	6,80	0,24%	0,01	0,17%	0,01	**-2,88%**	**-1,31%**	5,10
	4	0,00%	143,58	0,58%	6,00	0,65%	6,23	1,17%	0,01	1,17%	0,04	**-0,36%**	0,38%	5,62
	6	0,00%	176,22	0,04%	5,64	0,67%	5,57	0,96%	0,01	0,96%	0,02	**-0,58%**	**-0,58%**	2,84
12	2	0,00%	2062,89	0,35%	6,19	0,36%	6,03	0,67%	0,02	0,61%	0,04	**-2,64%**	**-2,02%**	13,28
	4	0,00%	1106,54	1,52%	5,24	1,63%	5,27	2,16%	0,02	2,03%	0,09	**-0,23%**	**-0,01%**	13,21
	6	0,00%	971,74	0,57%	5,37	0,96%	5,36	1,18%	0,03	1,08%	0,10	**-0,76%**	**-0,66%**	10,91
16	2	0,09%	2556,26	0,12%	5,96	0,24%	6,01	0,59%	0,03	0,27%	0,10	**-2,28%**	**-1,33%**	28,00
	4	0,33%	2077,02	1,23%	4,68	1,49%	4,93	1,69%	0,04	1,47%	0,20	**-0,05%**	0,22%	26,44
	6	0,01%	1685,21	0,78%	4,82	1,56%	5,03	1,61%	0,05	1,60%	0,24	0,20%	0,22%	19,57
Média		0,05%	1258,67	0,58%	4,91	0,85%	5,69	1,14%	0,02	1,04%	0,09	**-1,06%**	**-0,57%**	13,89

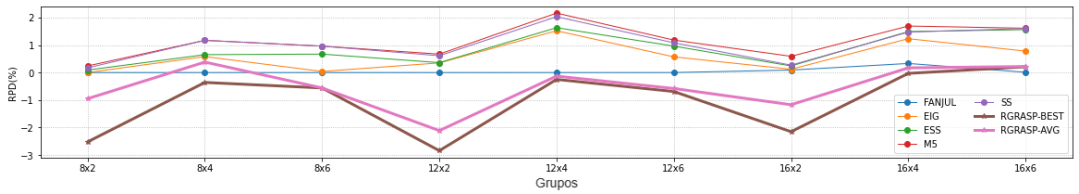


Figura 5.28: Literatura vs. *R-GRASP* - Grupo *Small*.

Para o conjunto de instâncias *Small*, o teste de normalidade *Shapiro-Wilk* apresentou todas as amostras com p-valor$< 2,2e-16$. Logo, há evidências para rejeitar a hipótese nula H_0 (os dados seguem a distribuição normal) e considerar que não há normalidade dos dados nas amostras. O teste de *Wilcoxon* pareado foi conduzido, comparando as medianas do algoritmo *R-GRASP* com todos os demais algoritmos.

Tabela 5.16: Teste de *Wilcoxon* pareado - conjunto de instâncias *Small*.

	FANJUL	EIG	ESS	M5	SS
R-GRASP	$0,001615$	$6,489e-15$	$< 2,2e-016$	$< 2,2e-16$	$7,58e-14$

Em todos as comparações mostradas na Tabela 5.7, o p-valor apontou evidências para rejeitar a hipótese nula H_0. Logo, existe diferença estatística entre os resultados do algoritmo *R-GRASP* com todos os demais algoritmos comparados.

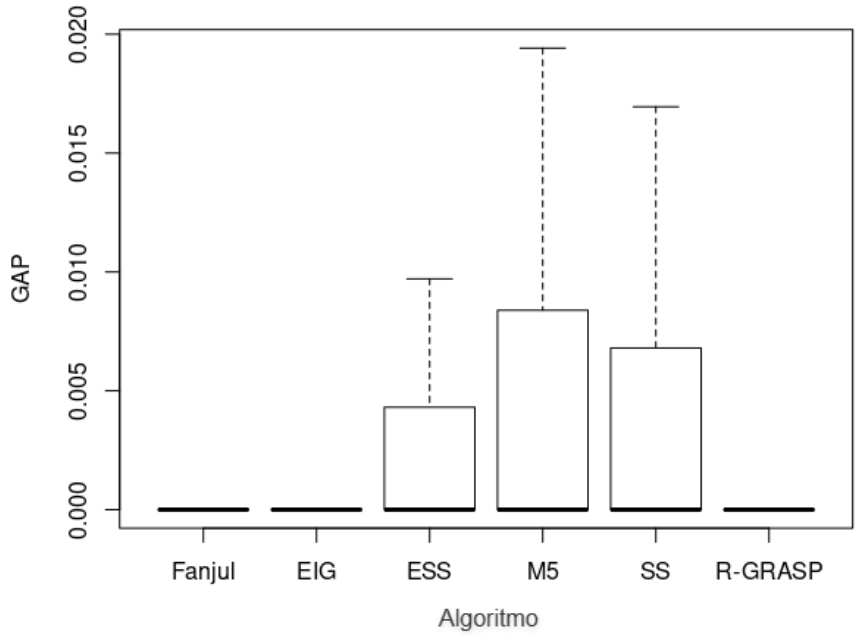


Figura 5.29: Literatura vs. *R-GRASP* - Grupo *Small*.

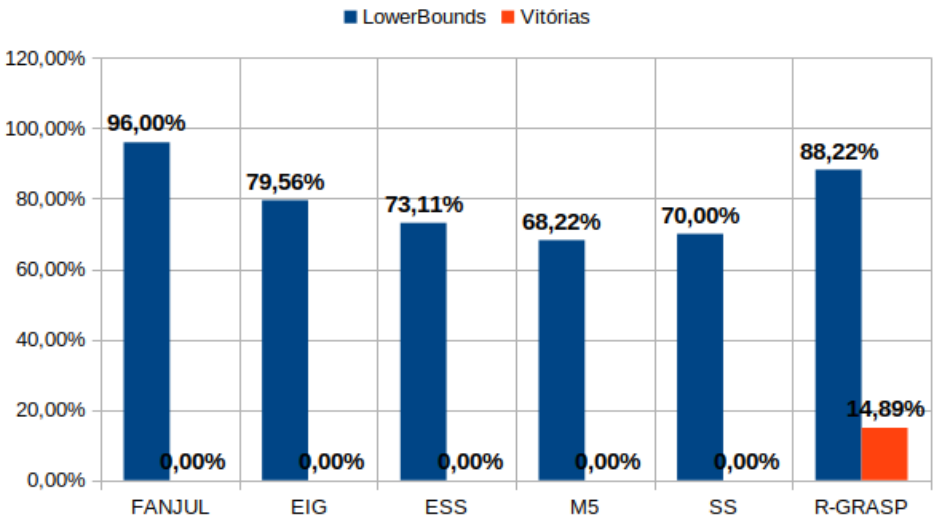


Figura 5.30: *Lower Bounds R-GRASP* - Grupo *Small*.

De acordo com o gráfico da Figura 5.30, para instâncias do grupo *Small*, o algoritmo *R-GRASP* é o que mais se aproxima dos melhores resultados de *FANJUL* com $88,22\%$ das soluções alcançando ou superando os limite da literatura, com $14,89\%$ de vitórias. O método proposto é o único que alcança melhoras em relação a *FANJUL*.

5.7.2 Resultados: *R-GRASP vs.* Literatura - grupo *Medium*

Para o grupo de instâncias *Medium*, de acordo com a Tabela 5.17, o algoritmo *R-GRASP* teve seus valores médios superiores a todos os procedimentos comparados, exceto com relação à média geral do algoritmo EIG. Os limites inferiores foram superados no subgrupo $25\times$ na média de melhores soluções encontradas. O tempo médio foi de $127,17$ segundos, abaixo apenas da média de tempo dos algoritmos de *FANJUL*.

Tabela 5.17: Resultados para o conjunto de instâncias *Medium*.

$n \times m$		FANJUL		EIG		ESS		M5		SS		R-GRASP		
		Avg	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
20	2	0,59%	2882,51	0,27%	17,97	0,69%	18,20	1,09%	0,05	0,78%	0,20	0,02%	1,27%	54,68
	4	1,00%	2462,80	0,54%	14,04	0,88%	14,27	0,99%	0,06	0,88%	0,37	0,04%	0,27%	52,12
	6	0,32%	2185,20	0,50%	12,67	0,67%	12,55	1,02%	0,07	0,68%	0,39	0,03%	0,08%	31,56
25	2	2,05%	2989,28	0,17%	17,89	0,49%	18,11	0,60%	0,08	0,50%	0,34	**-0,17%**	0,87%	108,51
	4	1,14%	2692,08	0,47%	13,48	0,71%	13,65	0,86%	0,10	0,71%	0,53	**-0,38%**	**-0,18%**	85,81
	6	1,43%	2644,40	0,45%	12,94	0,71%	12,89	1,07%	0,12	0,72%	0,59	**-0,10%**	0,09%	71,00
30	2	2,53%	2971,84	0,03%	17,89	0,55%	18,14	0,83%	0,12	0,55%	0,53	0,79%	1,40%	483,36
	4	0,82%	2717,49	0,17%	13,10	0,38%	13,43	0,65%	0,12	0,40%	0,68	0,10%	0,33%	111,60
	6	2,57%	2716,08	0,25%	12,40	0,52%	12,86	0,65%	0,18	0,52%	0,85	0,32%	0,54%	145,90
Média		1,38%	2695,74	0,32%	14,71	0,62%	14,90	0,86%	0,10	0,64%	0,50	0,07%	0,52%	127,17

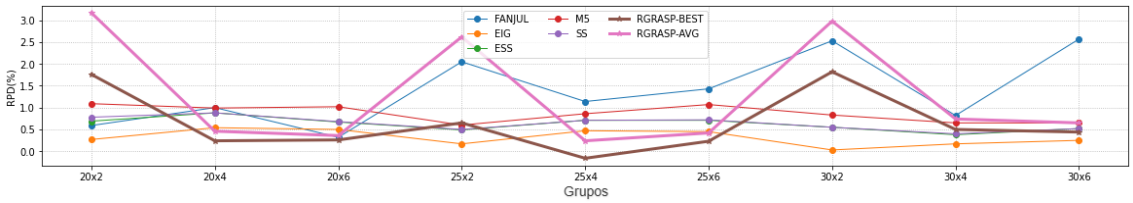


Figura 5.31: Literatura vs. *R-GRASP* - Grupo *Medium*.

Para o conjunto de instâncias *Medium*, o teste de normalidade *Shapiro-Wilk* apresentou todas as amostras com p-valor$< 2,2e-16$. Portanto, há evidências para rejeitarmos a hipótese nula H_0 (os dados seguem a distribuição normal) e, assim, considerar que não há normalidade dos dados nas amostras. O teste de *Wilcoxon* pareado foi conduzido, comparando as medianas do algoritmo *R-GRASP* com todos os demais algoritmos. Nas comparações mostradas na Tabela 5.18, os p-valores $p < \alpha = 0,05$ apontou evidências para rejeitar a hipótese nula H_0 e aceitar a hipótese alternativa H_1 do teste para o algoritmo *R-GRASP* em relação aos métodos de *FANJUL* e os algoritmos *M5* e *SS*, considerando $m_1 - m_2 \neq 0$. Os p-valores do teste para as comparações do algoritmo *R-GRASP* com os algoritmos *EIG* e *ESS* foram $p > \alpha = 0,05$, fornecendo evidências para a hipótese nula H_0, em que $m_1 - m2 = 0$. Conclui-se que não existe diferença estatística entre os resultados do algoritmo *R-GRASP* para os métodos *EIG* e *ESS*.

Tabela 5.18: Teste de *Wilcoxon* pareado - conjunto de instâncias *Medium*.

	FANJUL	EIG	ESS	M5	SS
R-GRASP	$4,102e-05$	$0,7471$	$0,09686$	$0,002057$	$0,009209$

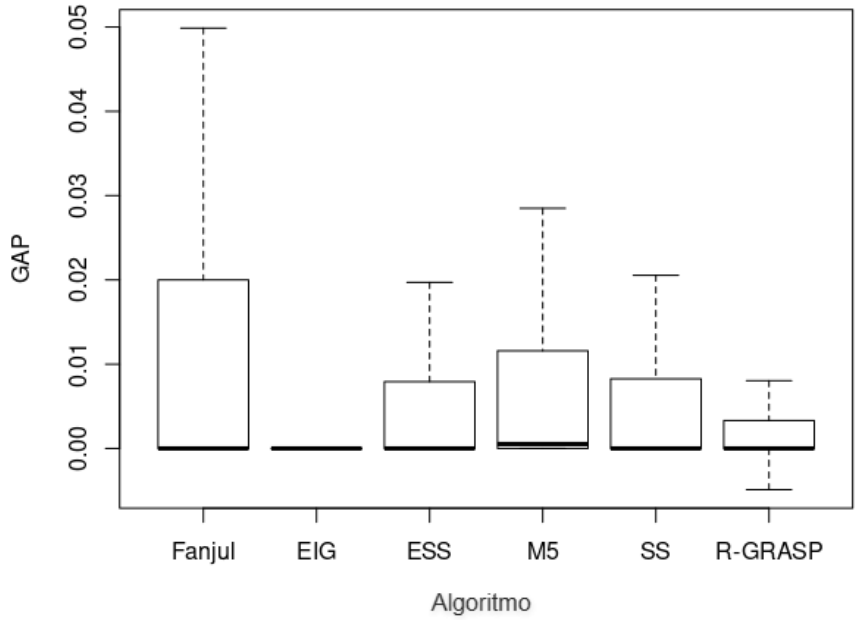


Figura 5.32: Literatura vs. *R-GRASP* - Grupo *Medium*.

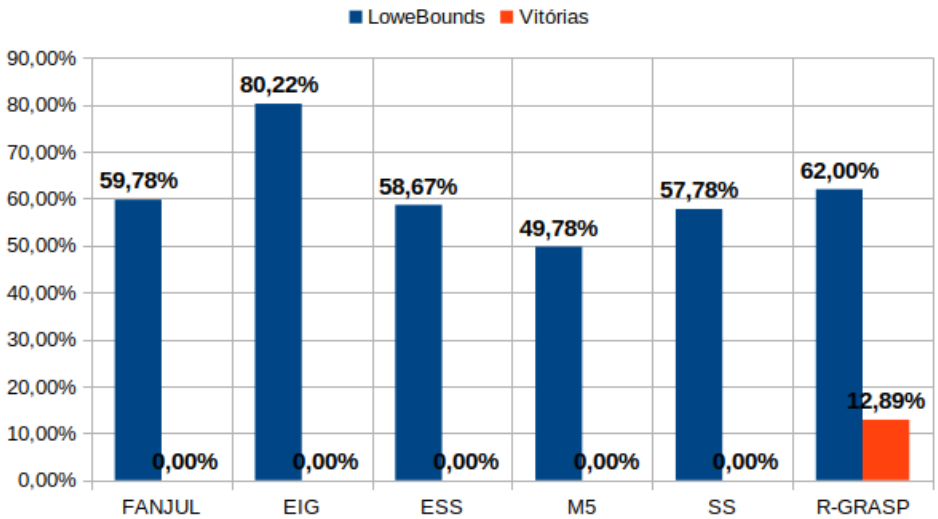


Figura 5.33: *Lower Bounds R-GRASP* - Grupo *Medium*.

De acordo com o gráfico da Figura 5.33, para instâncias do grupo *Medium*, o algoritmo *R-GRASP* é o que mais se aproxima dos melhores resultados obtidos pelo algoritmo *EIG*, atingindo 62% das soluções que alcançam ou superam os limites

da literatura. Com 12,89% de vitórias, o método proposto é o único que alcança melhoras em relação ao algoritmo *EIG*.

5.7.3 Resultados: *R-GRASP vs.* Literatura - grupo *Large*

Para o grupo de instâncias *Large*, de acordo com a Tabela 5.19, o algoritmo *R-GRASP* teve seus valores médios inferiores a todos os procedimentos comparados, exceto com relação ao subgrupo 50× na média de melhores soluções encontradas em que ocorreram superação de limites da literatura. O tempo médio foi o maior de todos os demais algoritmos, com média de 494,06 segundos.

Tabela 5.19: Resultados para o conjunto de instâncias *Large*.

$n \times m$		EIG		ESS		M5		SS		*R-GRASP*		
		Avg	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Avg*	$t(s)$	*Best*	*Avg*	$t(s)$
50	10	0,07%	12,98	0,58%	14,17	0,99%	0,64	0,57%	2,72	1,05%	1,84%	9,70
	20	0,02%	14,14	0,84%	14,38	1,09%	1,06	0,85%	3,67	**-0,54%**	0,06%	7,87
	30	0,00%	14,46	1,03%	14,17	1,44%	1,34	1,18%	3,69	**-0,10%**	0,41%	4,96
150	10	0,04%	32,05	0,30%	36,34	0,58%	8,11	0,30%	25,92	2,94%	4,01%	112,65
	20	0,03%	41,57	0,40%	44,61	1,05%	12,24	0,43%	33,96	2,78%	3,62%	101,31
	30	0,11%	49,40	0,62%	49,72	0,92%	15,97	0,62%	38,57	1,79%	2,57%	79,96
250	10	0,02%	96,63	0,23%	102,97	0,54%	34,59	0,23%	92,71	3,84%	4,48%	639,33
	20	0,08%	105,72	0,29%	112,70	0,55%	38,83	0,28%	102,15	3,45%	4,07%	437,79
	30	0,00%	142,99	0,51%	122,12	0,71%	49,45	0,51%	111,15	3,16%	3,70%	340,08
350	10	0,02%	241,57	0,17%	241,94	0,40%	94,16	0,18%	231,39	4,04%	4,72%	1.394,16
	20	0,00%	222,20	165,50%	242,71	165,68%	90,58	0,18%	231,94	3,65%	4,39%	1.275,86
	30	0,01%	239,29	0,47%	243,59	0,59%	96,72	0,47%	230,73	3,88%	4,60%	1.524,99
Média		0,03%	101,08	14,24%	103,28	14,55%	36,97	0,48%	92,38	2,49%	3,20%	494,06

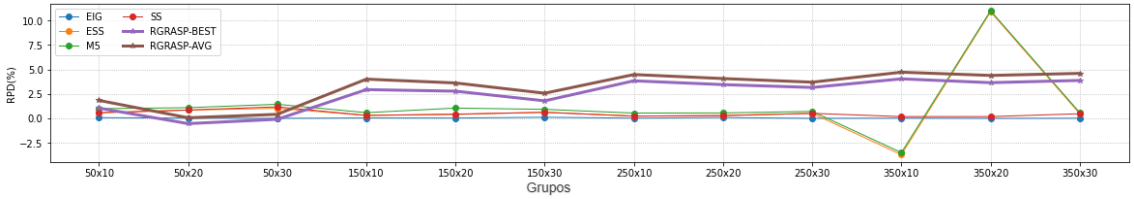


Figura 5.34: Literatura vs. *R-GRASP* - Grupo *Large*

Para o conjunto de instâncias *Large*, o teste de normalidade *Shapiro-Wilk* apresentou todas as amostras com p-valor $< 2,2e-16$. Logo há evidências para rejeitarmos a hipótese nula H_0 (os dados seguem a distribuição normal). O teste de *Wilcoxon* pareado foi conduzido, comparando as medianas do algoritmo *R-GRASP* com todos os demais algoritmos. Nas comparações mostradas na Tabela 5.20, os p-valores $p < \alpha = 0,05$ apontaram evidências para rejeitar a hipótese nula H_0 e aceitar a hipótese alternativa H_1 do teste para o algoritmo *R-GRASP* em relação a todos os métodos comparados, considerando $m_1 - m_2 \neq 0$. Conclui-se que existe diferença estatística entre os resultados do algoritmo *R-GRASP* com todos os métodos comparados para esse grupo de instâncias.

Tabela 5.20: Teste de *Wilcoxon* pareado - conjunto de instâncias *Large*.

	EIG	ESS	M5	SS
R-GRASP	$< 2,2e-16$	$< 2,2e-16$	$< 2,2e-16$	$< 2,2e-16$

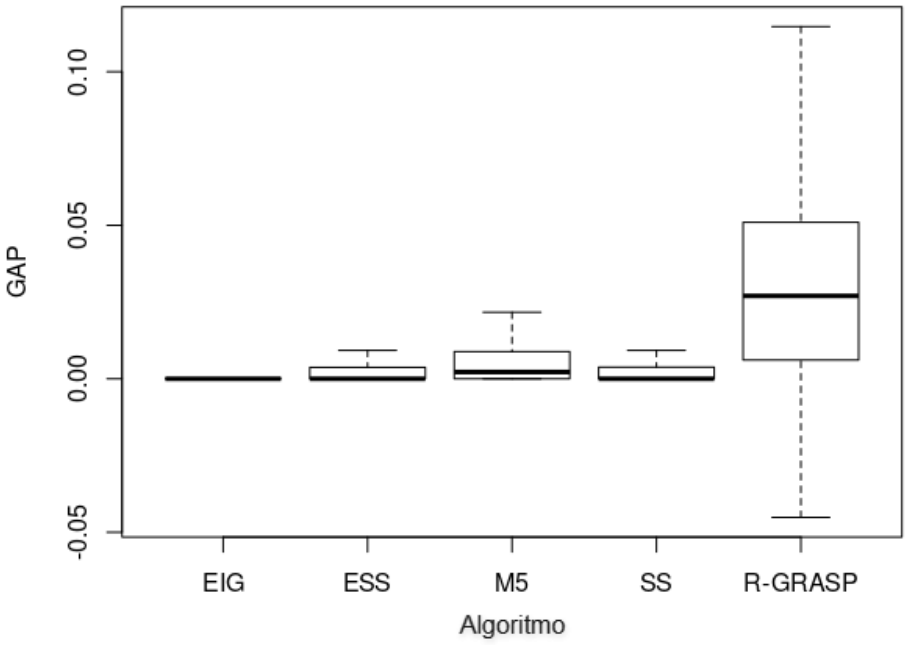


Figura 5.35: Literatura vs. *R-GRASP* - Grupo *Large*.

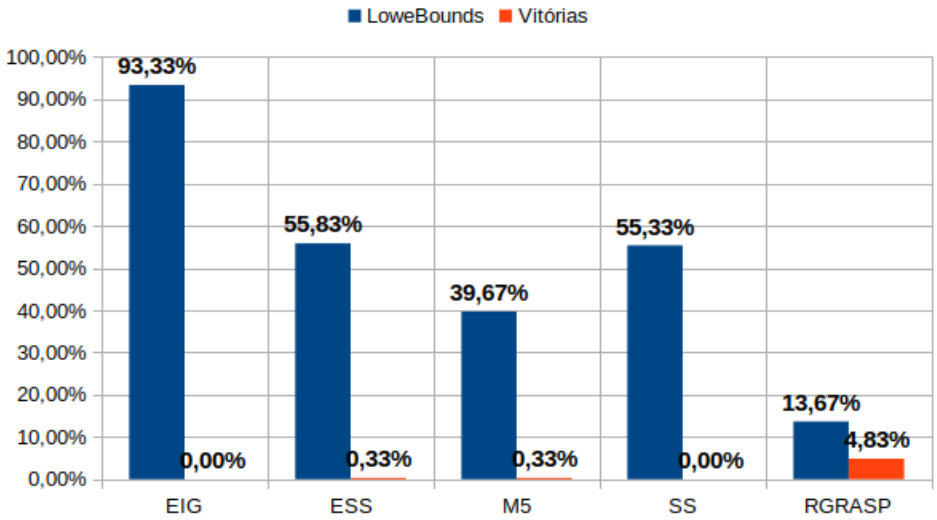


Figura 5.36: *Lower Bounds R-GRASP* - Grupo *Large*.

De acordo com o gráfico da Figura 5.36, para instâncias do grupo *Large*, o algoritmo *R-GRASP* obteve apenas $13,67\%$ de soluções que alcançaram ou superaram

os limite da literatura. Com $4,83\%$ de vitórias, o método proposto é o único que alcança melhoras em relação a literatura.

5.8 Resumo

Os experimentos indicam que o algoritmo *R-GRASP* apresenta desempenho satisfatório em comparação com algoritmos da literatura em boa parte dos testes realizados, alcançando resultados superiores aos conhecidos em boa parte das instâncias e com custo computacional viável.

Os quatro tipos de construção GRASP sugeridos cobriram possibilidades para o critério de construção. Os tipos puderam ser testados para os diferentes grupos de instâncias. Para o grupo *Small*, todos os tipos tiveram bom desempenho, alcançando valores médios superiores à literatura em todos os quatro tipos de construção. Nas instâncias de médio porte *Medium*, as diferenças entre os tipos não foram significativas, embora o tipo CT-1 tenha alcançados 20% de valores médios superiores à literatura e com o menor tempo de execução ($1,93$ segundos em média). Os tipos de construção tiveram menor expressividade para o grupo de instâncias de grande porte (*Large*). O CT-4 obteve as melhores médias e com baixo tempo computacional em comparação com os demais tipos.

O método de busca local foi implementado com *First Improvement* e *Best Improvement* nos quais resultados inesperados foram obtidos nessa etapa de testes. Para o grupo de instâncias *Small*, o método *First Improvement* superou o método *Best Improvement*, considerando os valores médios obtidos. Para as instâncias de médio porte (*Medium*), o método *Best Improvement* exigiu maior tempo computacional e apresentou melhores resultados como esperado considerando os valores médios. Para as instâncias de grande porte (*Large*), o método *First Improvement* exigiu o menor tempo como esperado, mas também alcançou os melhores valores médios.

Foram testados três dimensões de conjuntos para o parâmetro α usado pelo algoritmo *R-GRASP*. O objetivo foi avaliar se grupos mais restritos poderiam ter mais vantagem em relação a grupos menos restritos. Como esperado, o conjunto $C3$, mais amplo, apresentou os melhores valores médios, com exceção do grupo de grande porte (*Large*), em que o conjunto $C1$ foi mais expressivo e produziu melhores resultados, quando analisados os valores médios. Estatisticamente, não houve diferença significante, porém a diferença de tempo foi expressiva colocando o conjunto $C1$ em vantagem em relação aos demais conjuntos para esse caso.

Capítulo 6

Considerações Finais

Neste trabalho foi desenvolvido um algoritmo GRASP Reativo para resolução do problema de minimização do *makespan* em máquinas paralelas não relacionadas com recurso adicional. Um método de reparo guloso também foi desenvolvido, para atender à restrição de recurso do problema.

Para os testes dos métodos implementados, um conjunto de instâncias, proposto por Fanjul-Peyro et al. (2017), foi utilizado. Este mesmo conjunto de instâncias é utilizado em Villa et al. (2018) e Vallada et al. (2019). O conjunto de instâncias citado é constituído por três grupos de instâncias, denominados pequeno, médio e grande porte, nos quais o método proposto apresentou melhoras nos grupos de pequeno e médio porte.

Para a composição do algoritmo GRASP Reativo, foram propostos alguns operadores, como a construção da solução inicial, em que quatro tipos foram propostos e testados. O parâmetro α é calibrando reativamente, considerando o histórico de melhorias recentes. Um operador de busca local VND é proposto para refinar as soluções construídas e conduzi-las para regiões de ótimos locais. As melhores soluções encontradas passam pelo procedimento de reparo. O método de reparo é guloso e faz realocações pontuais nas regiões de excesso de recurso priorizando reparar as regiões sem piorar o *makespan* sempre que possível.

A etapa de busca local utilizada foi testada com as estratégias *First Improvement* e *Best Improvement*. Os métodos apresentaram comportamentos distintos entre os grupos de instâncias, o que pode ser atribuído ao fato de que o método de reparo de soluções é aplicado internamente ao procedimento de busca local, já que as soluções são refinadas sem considerar a restrição de recurso e só então passam pelo método de reparo antes de serem retornadas.

Outro teste realizado foi propor diferentes grupos de possíveis valores para o parâmetro α. Três grupos com amplitudes diferentes foram considerados, sendo que, para os testes com instâncias de pequeno e médio porte, o grupo mais amplo $C3$ apresentou as melhores médias. Em contrapartida, para o grupo de instâncias de grande porte, o grupo de menor amplitude $C1$ apresentou as melhores médias e menor custo computacional.

Embora não tenha havido testes específicos para avaliar o método de reparo proposto neste trabalho, foi possível validar a sua aplicação por meios dos resultados finais. O objetivo de desvincular o problema de otimização da restrição para resolver cada um de forma distinta e recompor as partes resolvidas sem prejuízo foi alcançado

e se mostrou com potencial para exploração em trabalhos futuros.

Este trabalho teve um enfoque simples sobre um problema específico de otimização com o objetivo de estudar sua resolução com técnicas clássicas conhecidas e de fácil implementação. Todos os testes computacionais foram realizados em *hardware* doméstico e com recursos limitados. Ainda assim, as comparações apontaram cenários onde uma boa taxa de melhorias comparadas com a literatura conhecida foi alcançada.

6.1 Trabalhos Futuros

A seguir são apresentadas propostas para trabalhos futuros:

- Adaptar o algoritmo de reparo guloso para outros problemas de sequenciamento que envolvem a restrição de recurso;
- Utilizar em outros problemas de otimização métodos gulosos para atendimento de restrições, uma vez que atender uma restrição é diferente e mais simples que minimizar uma determinada função de avaliação;
- Avaliar o método de reparo de recurso medindo o nível de piora do *makespan* partindo de uma determinada solução que foi refinada sem considerar a restrição de recurso;
- Avaliar um procedimento aleatório de Busca local *Random Variable Neighborhood Descent – RVND*;
- Implementar a hibridização do GRASP Reativo com a heurística de religamento de caminhos *Path Relinking* como método de intensificação. Esta estratégia têm se mostrado promissora em conjunto com o GRASP padrão levando a melhorias significativas no desempenho e qualidade de soluções.
- Direcionar o problema à uma perspectiva bi-objetiva onde *makespan* e número máximo de recursos são minimizados simultaneamente;
- Novos operadores de realocação no método de reparo. Com isso, a ampliação da capacidade de realocação de tarefas do método aumenta a possibilidade da correção de regiões sem piora do *makespan*/*Completion Time*;
- Implementar o GRASP Paralelo, dividindo-se k iterações independentes entre p processadores. Outro enfoque consiste em definir um valor de função objetivo τ a cada processador e executar o algoritmo até que o primeiro processador encontre uma solução pelo menos tão boa quanto τ. Abordagens como esta podem ser úteis em execução de instâncias de grande porte.

Referências Bibliográficas

Alvarez-Valdés, Ramón; Parreño, Francisco e Tamarit, José Manuel. (2008). Reactive GRASP for the strip-packing problem. *Computers & Operations Research*, v. 35, n. 4, p. 1065–1083.

Arbaoui, Taha e Yalaoui, Farouk. (2018). Solving the unrelated parallel machine scheduling problem with additional resources using constraint programming. *Asian Conference on Intelligent Information and Database Systems*, p. 716–725. Springer, (2018).

Binato, S; Hery, WJ; Loewenstern, DM e Resende, MAURICIO GC. (2002). A GRASP for job shop scheduling. *Essays and surveys in metaheuristics*, p. 59–79. Springer.

Binato, Silvio e Oliveira, Gerson Couto. (2002). A reactive GRASP for transmission network expansion planning. *Essays and surveys in metaheuristics*, p. 81–100. Springer.

Bitar, Abdoul; Dauzère-Pérès, Stéphane; Yugma, Claude e Roussel, Renaud. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, v. 19, n. 4, p. 367–376.

Błażewicz, J; Barcelo, J; Kubiak, W e Röck, H. (1986). Scheduling tasks on two processors with deadlines and additional resources. *European journal of operational research*, v. 26, n. 3, p. 364–370.

Błazewicz, J; Kubiak, Wieslaw e Martello, Silvano. (1993). Algorithms for minimizing maximum lateness with unit length tasks and resource constraints. *Discrete applied mathematics*, v. 42, n. 2-3, p. 123–138.

Błażewicz, J; Kubiak, Wieslaw; Röck, Hans e Szwarcfiter, J. (1987). Minimizing mean flow-time with parallel processors and resource constraints. *Acta informatica*, v. 24, n. 5, p. 513–524.

Błażewicz, Jacek. (1979). Deadline scheduling of tasks with ready times and resource constraints. *Information Processing Letters*, v. 8, n. 2, p. 60–63.

Blazewicz, Jacek. (1981). Solving the resource constrained deadline scheduling problem via reduction to the network flow problem. *European Journal of Operational Research*, v. 6, n. 1, p. 75–79.

Błażewicz, Jacek; Ecker, Klaus H; Pesch, Erwin; Schmidt, Günter e Weglarz, Jan. (2007). *Handbook on scheduling: from theory to applications*. Springer Science & Business Media.

Blazewicz, Jacek; Lenstra, Jan Karel e Kan, AHG Rinnooy. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, v. 5, n. 1, p. 11–24.

Boudia, Mourad; Louly, Mohamed Aly Ould e Prins, Christian. (2007). A reactive grasp and path relinking for a combined production–distribution problem. *Computers & Operations Research*, v. 34, n. 11, p. 3402–3419.

Brucker, Peter e Krämer, Andreas. (1996). Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, v. 90, n. 2, p. 214–226.

Cakici, Eray e Mason, SJ. (2007). Parallel machine scheduling subject to auxiliary resource constraints. *Production Planning and Control*, v. 18, n. 3, p. 217–225.

Daniels, Richard L; Hoopes, Barbara J e Mazzola, Joseph B. (1996). Scheduling parallel manufacturing cells with resource flexibility. *Management science*, v. 42, n. 9, p. 1260–1276.

Daniels, Richard L; Hoopes, Barbara J e Mazzola, Joseph B. (1997). An analysis of heuristics for the parallel-machine flexible-resource scheduling problem. *Annals of Operations Research*, v. 70, p. 439–472.

Daniels, Richard L; Hua, Stella Y e Webster, Scott. (1999). Heuristics for parallel-machine flexible-resource scheduling problems with unspecified job assignment. *Computers & operations research*, v. 26, n. 2, p. 143–155.

Davis, Edward W e Heidorn, George E. (1971). An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, v. 17, n. 12, p. B–803.

Delmaire, Hugues; Díaz, Juan A; Fernández, Elena e Ortega, Maruja. (1999). Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *INFOR: Information Systems and Operational Research*, v. 37, n. 3, p. 194–225.

Diana, Rodney Oliveira Marinho; deSouza, Sérgio Ricardo e Wanner, Elizabeth Fialho. (2021). A robust multi-response vns-ainet approach for solving scheduling problems under unrelated parallel machines environments. *Expert Systems with Applications*, v. 182, p. 115140.

Dorigo, Marco. *Optimization, learning and natural algorithms*. Phd thesis, Politecnico di Milano, Milão, Itália, (1992).

Dorigo, Marco; Maniezzo, Vittorio e Colorni, Alberto. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 26, n. 1, p. 29–41.

Eberhart, Russell e Kennedy, James. (1995). Particle swarm optimization. *Proceedings of the IEEE international conference on neural networks*, volume 4, p. 1942–1948. Citeseer, (1995).

Edis, Emrah B; Araz, Ceyhun e Ozkarahan, Irem. (2008). Lagrangian-based solution approaches for a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, p. 337–346. Springer, (2008).

Edis, Emrah B e Oguz, Ceyda. (2011). Parallel machine scheduling with additional resources: a lagrangian-based constraint programming approach. *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, p. 92–98. Springer, (2011).

Edis, Emrah B e Oguz, Ceyda. (2012)a. Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, v. 63, n. 2, p. 433–447.

Edis, Emrah B e Oguz, Ceyda. (2012)b. Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, v. 63, n. 2, p. 433–447.

Edis, Emrah B; Oguz, Ceyda e Ozkarahan, Irem. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, v. 230, n. 3, p. 449–463.

Fanjul-Peyro, Luis. (2020). Models and an exact method for the unrelated parallel machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, v. 5, p. 100022. doi: https://doi.org/10.1016/j.eswax.2020.100022.

Fanjul-Peyro, Luis; Perea, Federico e Ruiz, Rubén. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, v. 260, n. 2, p. 482–493.

Fanjul-Peyro, Luis e Ruiz, Rubén. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, v. 207, n. 1, p. 55–69.

Feo, Thomas A e Resende, Mauricio GC. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, v. 6, n. 2, p. 109–133.

Fleszar, Krzysztof e Hindi, Khalil S. (2018). Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, v. 271, n. 3, p. 839–848.

Garey, Michael R e Johnson, David S. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, v. 4, n. 4, p. 397–411.

Garey, MR e Grehem, RL. (1973). Bounds on scheduling with limited resources. *ACM SIGOPS Operating Systems Review*, v. 7, n. 4, p. 104–111.

Gaspar-Cunha, António; Takahashi, Ricardo e Antunes, Carlos Henggeler. (2012). *Manual de computação evolutiva e metaheurística*. Imprensa da Universidade de Coimbra/Coimbra University Press.

Ghirardi, Marco e Potts, Chris N. (2005). Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, v. 165, n. 2, p. 457–467.

Glover, Fred. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, v. 13, n. 5, p. 533–549.

Gomes, Fernando C; Pardalos, Panos; Oliveira, Carlos S e Resende, Mauricio GC. (2001). Reactive GRASP with path relinking for channel assignment in mobile phone networks. *Proceedings of the 5th international workshop on discrete algorithms and methods for mobile computing and communications*, p. 60–67, (2001).

Graham, Ronald Lewis; Lawler, Eugene Leighton; Lenstra, Jan Karel e Kan, AHG Rinnooy. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, volume 5, p. 287–326. Elsevier.

Hansen, P; Mladenovic, N e Gerad, LCD. (1999). Variable neighborhood search: Methods and recent applications. *Proceedings of MIC*, volume 99, p. 275–280, (1999).

Holland, John H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Kellerer, Hans e Strusevich, Vitaly A. (2008). Scheduling parallel dedicated machines with the speeding-up resource. *Naval Research Logistics (NRL)*, v. 55, n. 5, p. 377–389.

Kirkpatrick, Scott; Gelatt Jr, C Daniel e Vecchi, Mario P. (1983). Optimization by simulated annealing. *science*, v. 220, n. 4598, p. 671–680.

Kravchenko, Svetlana A e Werner, Frank. (2011). Parallel machine problems with equal processing times: a survey. *Journal of Scheduling*, v. 14, n. 5, p. 435–444.

Lourenço, Helena R; Martin, Olivier C e Stützle, Thomas. (2003). Iterated local search. *Handbook of metaheuristics*, p. 320–353. Springer.

Martello, Silvano; Soumis, François e Toth, Paolo. (1997). Exact and approximation algorithms for makespan minimization on unrelated parallel machines. *Discrete applied mathematics*, v. 75, n. 2, p. 169–188.

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

Mokotoff, Ethel e Jimeno, JL. (2002). Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. *Annals of Operations Research*, v. 117, n. 1, p. 133–150.

Moscato, Pablo e others,. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, v. 826, n. 1989, p. 37.

Pinedo, Michael e Hadavi, Khosrow. (1992). Scheduling: theory, algorithms and systems development. *Operations Research Proceedings 1991*, p. 35–42. Springer.

Pinheiro, Júlio CSN; Arroyo, José Elias C e Fialho, Lucas Batista. (2020). Scheduling unrelated parallel machines with family setups and resource constraints to minimize total tardiness. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, p. 1409–1417, (2020).

Prais, M e Ribeiro, Celso C. (1999). Parameter variation in grasp implementations. *Extended abstracts of the third metaheuristics international conference*, p. 375–380, (1999).

Prais, Marcelo e Ribeiro, Celso C. (2000). Reactive grasp: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing*, v. 12, n. 3, p. 164–176.

Ribeiro, Celso C. (1996). Metaheuristics and applications. *Advanced School on Artificial Intelligence, Estoril, Portugal*, v. .

Ruiz, Rubén e Andrés-Romano, Carlos. (2011). Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, v. 57, n. 5-8, p. 777–794.

Ruiz, Ruben; Şerifoğlu, Funda Sivrikaya e Urlings, Thijs. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, v. 35, n. 4, p. 1151–1175.

Ruiz-Torres, Alex J; López, Francisco J e Ho, Johnny C. (2007). Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. *European Journal of Operational Research*, v. 179, n. 2, p. 302–315.

Scaparra, Maria P e Church, Richard L. (2005). A grasp and path relinking heuristic for rural road network development. *Journal of Heuristics*, v. 11, n. 1, p. 89–108.

Słowiński, Roman. (1980). Two approaches to problems of resource allocation among project activities – a comparative study. *Journal of the Operational Research Society*, v. 31, n. 8, p. 711–723.

Slowinski, Roman. (1981). Lórdonnancement des tâches préemptives sur les processeurs indépendants en présence de ressources supplémentaires. *Cybernetica*, v. 24, n. 1, p. 11–24.

Vallada, Eva e Ruiz, Rubén. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 211, n. 3, p. 612–622.

Vallada, Eva; Villa, Fulgencia e Fanjul-Peyro, Luis. (2019). Enriched metaheuristics for the resource constrained unrelated parallel machine scheduling problem. *Computers & Operations Research*, v. 111, p. 415–424.

Villa, Fulgencia; Vallada, Eva e Fanjul-Peyro, Luis. (2018). Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource. *Expert Systems with Applications*, v. 93, p. 28–38.

Weglarz, Jan; Blazewicz, Jacek; Cellary, Wojciech e Slowinski, Roman. (1977). Algorithm 520: an automatic revised simplex method for constrained resource network scheduling [h]. *ACM Transactions on Mathematical Software (TOMS)*, v. 3, n. 3, p. 295–300.

Wilcoxon, Frank. (1947). Probability tables for individual comparisons by ranking methods. *Biometrics*, v. 3, n. 3, p. 119–122.

Yepes-Borrero, Juan C; Villa, Fulgencia; Perea, Federico e Caballero-Villalobos, Juan Pablo. (2020). GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, v. 141, p. 112959.

Zheng, Xiao-long e Wang, Ling. (2016). A two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints. *Expert Systems with Applications*, v. 65, p. 28–39.