UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO E SISTEMAS

EWERTON VICTOR PAREDES DA PENHA TEIXEIRA

UMA HEURÍSTICA HÍBRIDA PARA O PROBLEMA DE FLOW SHOP COM BLOQUEIO E MINIMIZAÇÃO DO MAKESPAN

JOÃO PESSOA
2023

EWERTON VICTOR PAREDES DA PENHA TEIXEIRA

UMA HEURÍSTICA HÍBRIDA PARA O PROBLEMA DE FLOW SHOP COM BLOQUEIO E MINIMIZAÇÃO DO MAKESPAN

Orientador: Prof. Dr. Anand Subramanian

Co-orientador: Prof. Dr. Hugo Harry F. R. Kramer

JOÃO PESSOA
2023

Catalogação na publicação
Seção de Catalogação e Classificação

T266h Teixeira, Ewerton Victor Paredes da Penha.
Uma heurística híbrida para o problema de flow shop com bloqueio e minimização do makespan / Ewerton Victor Paredes da Penha Teixeira. - João Pessoa, 2023.
86 f. : il.

Orientação: Anand Subramanian.
Coorientação: Hugo Harry Frederico Ribeiro Kramer.
Dissertação (Mestrado) - UFPB/CT.

1. Engenharia elétrica - Maquinaria. 2. Sequenciamento da produção. 3. Meta-heurística. 4. Flow shop com bloqueio. I. Subramanian, Anand. II. Kramer, Hugo Harry Frederico Ribeiro. III. Título.

UFPB/BC CDU 621(043)

Elaborado por GRACILENE BARBOSA FIGUEIREDO - CRB-15/794

ATA DE DEFESA DE DISSERTAÇÃO

Ata da 21ª Sessão Pública de Defesa de Dissertação do(a) Mestrando(a) **EWERTON VICTOR PAREDES DA PENHA TEIXEIRA**, realizada como requisito para obtenção do grau de Mestre em Engenharia de Produção e Sistemas, na área de concentração "Sistemas de Produção" e linha de pesquisa "Gestão Integrada de Sistemas de Produção".

No dia 31 de mês de agosto de 2023, às 13 horas, na Sala Virtual criada através da plataforma Google Meet reuniu-se a Banca Examinadora aprovada em processo administrativo pelo Colegiado do Programa de Pós-Graduação em Engenharia de Produção e Sistemas, composta pelos seguintes membros, a saber, o(a) Prof.(ª). Dr.(ª) Anand Subramanian (Orientador(a) – PPGEPS/UFPB), Prof. Dr. Hugo Harry Frederico Ribeiro Kramer (Coorientador(a) – PPGEPS/UFPB), Prof.(ª) Dr.(ª) Luciano Carlos Azevedo da Costa (Examinador(a) Interno (a) – UFPB) e Prof.(ª) Dr.(ª) Pedro Augusto Munari Júnior (Examinador(a) Externo(a) – PPGEP/UFSCar) com a finalidade de julgar a defesa de dissertação do(a) aluno(a) **Ewerton Victor Paredes da Penha Teixeira**, intitulada **"UMA HEURÍSTICA HÍBRIDA PARA O PROBLEMA DE *FLOW SHOP* COM BLOQUEIO E MINIMIZAÇÃO DO *MAKESPAN*"**, como requisito para a obtenção do grau de Mestre em Engenharia de Produção e Sistemas. Além dos examinadores e do mestrando, compareceram como convidados Rafael Praxedes, Camila Eduardo, Renata Mendes, Victor Ferreira, Raquel Patricio, Felipe Lemos, Denise Teixeira, Vinicius Freitas, Eduardo Queiroga, Iderval Neto, Lara Pontes, Valdir Neto, Gabrielle Melo, Lucas Guedes, Beatriz Melchiades, Teobaldo Junior, João Marcos Pereira Silva, Rafael Sobral de Morais, Marcelo Cavalcante, Anderson Coutinho, Igor Malheiros, Petrus Neves e Felipe Silva. Iniciada a sessão, o(a) Prof.(ª) Dr.(ª) Anand Subramanian, na qualidade de presidente da Banca Examinadora, apresentou a finalidade da reunião e os procedimentos regulamentares. Em seguida, passou a palavra ao(à) mestrando(a) para que, nos limites do prazo regimental, realizasse sua apresentação oral. Concluída a exposição, mediante solicitação do(a) senhor(a) presidente, os membros da banca examinadora

teceram os devidos comentários e realizaram a arguição do expositor. Na sequência, a banca reuniu-se em caráter secreto a fim de julgar a defesa e a dissertação, decidindo atribuir-lhe o conceito **APROVADO**, concedendo, assim, o grau de mestre a(à) **Ewerton Victor Paredes da Penha Teixeira**. Nada mais havendo a tratar, o(a) senhor(a) presidente agradeceu a todos pela presença e encerrou a sessão, sugerindo por recomendação dos membros da banca que o prazo máximo para providenciar as correções e entregar os volumes de versão final da dissertação na Secretaria do Programa é de 60 dias a partir desta data. Para constar, eu, Vinícius Pinagé Alves de Lima, secretário do PPGEPS, lavrei a presente ata que assino juntamente com o candidato, o presidente e os demais membros da Banca Examinadora.

João Pessoa/PB, 31 de agosto de 2023.

Documento assinado digitalmente
gov.br
ANAND SUBRAMANIAN
Data: 31/08/2023 22:12:22-0300
Verifique em https://validar.iti.gov.br

Prof.(ª) Dr.(ª) Anand Subramanian
Orientador(a) – PPGEPS/UFPB

Documento assinado digitalmente
gov.br
HUGO HARRY FREDERICO RIBEIRO KRAMER
Data: 01/09/2023 08:53:52-0300
Verifique em https://validar.iti.gov.br

Prof.(ª) Dr.(ª) Hugo Harry Frederico Ribeiro Kramer
Coorientador(a) – PPGEPS/UFPB

Documento assinado digitalmente
gov.br
LUCIANO CARLOS AZEVEDO DA COSTA
Data: 31/08/2023 23:30:10-0300
Verifique em https://validar.iti.gov.br

Prof.(ª) Dr.(ª) Luciano Carlos Azevedo da Costa
Examinador(a) Interno(a) – PPGEPS/UFPB

Documento assinado digitalmente
gov.br
PEDRO AUGUSTO MUNARI JUNIOR
Data: 31/08/2023 22:28:10-0300
Verifique em https://validar.iti.gov.br

Prof.(ª) Dr.(ª) Pedro Augusto Munari Júnior
Examinador(a) Externo(a) – PPGEP/UFSCar

Documento assinado digitalmente
gov.br
VINICIUS PINAGE ALVES DE LIMA
Data: 05/09/2023 12:50:28-0300
Verifique em https://validar.iti.gov.br

Vinícius Pinagé Alves de Lima
Secretário(a) do PPGEPS/UFPB

Documento assinado digitalmente
gov.br
EWERTON VICTOR PAREDES DA PENHA TEIXEIR
Data: 31/08/2023 22:39:34-0300
Verifique em https://validar.iti.gov.br

Ewerton Victor Paredes da Penha Teixeira
Mestrando(a)

Agradecimentos

A minha familia, em especial aos meus pais, Marcos e Denise, por sempre me apoiarem em minhas decisões e me amarem de maneira incondicional.

Ao meu orientador, Anand Subramanian, por todos os conselhos, ensinamentos e parceria desde a minha graduação.

Ao meu co-orientador, Hugo Kramer, pelo suporte que me dá desde a graduação.

Aos membros da banca examinadora, Luciano Costa e Pedro Munari, pela contribuição dada à versão final deste trabalho.

Aos membros do LOG UFPB, por terem feito parte da minha rotina durante os mais de dois anos de mestrado.

A todos os meus amigos, estejam eles próximos ou distantes, por todos os momentos vividos desde a minha infância que me ajudaram a ser quem eu sou hoje.

À CAPES, pela bolsa concedida.

E um agradecimento especial aos meus amigos Alex, Cynthia, Jean, Dan, Daniel, Grazi, Hyou, Key, Markus, Yako e Yeriah. Apesar da distância, completamos quase 15 anos de amizade e formação do grupo.

Resumo

A maximização da eficiência operacional tem se tornado uma prioridade para as empresas modernas, impulsionando a necessidade de otimizar o fluxo de produção, que desempenha um papel fundamental na capacidade de uma organização atender às demandas do mercado de forma ágil e rentável. Nesse sentido, aperfeiçoar o processo de sequenciamento de tarefas é essencial para aumentar a produtividade e reduzir desperdícios em um mercado altamente competitivo. Dentro deste escopo, este trabalho aborda o problema de *flow shop* com bloqueio para minimização do *makespan*. Nesse problema, n tarefas precisam ser escalonadas em um ambiente de m máquinas ordenadas em série, em que todas as tarefas devem seguir a mesma ordem de processamento. Além disso, diferentemente do problema de *flow shop* permutacional, o estoque intermediário entre estas máquinas é considerado nulo. Assim, uma máquina só poderá liberar uma tarefa para a máquina subsquente caso esta não esteja ocupada. É proposta uma heurística populacional híbrida que combina um operador *ruin-and-recreate* com uma busca local baseada em *Variable Neighborhood Descent*, que faz uso de métodos de aceleração da busca. Nesta aceleração, um método da literatura para a vizinhança *insertion* foi adaptado para vizinhanças do tipo *block insertion* e *swap*. Ainda, limitantes inferiores para a vizinhança *swap* também são propostos e possuem como objetivo evitar a avaliação de movimentos que não levariam a melhora da solução corrente. Por fim, um critério de desempate e um mecanismo de controle de diversidade da população são empregados para evitar que o método fique preso em ótimos locais. Extensos experimentos computacionais foram realizados em 150 instâncias de *benchmark*, englobando a calibragem de parâmetros, avaliação do critério de desempate, desempenho das vizinhanças empregadas e demais componentes do método, como o operador *ruin-and-recreate* e a diversidade da população. Em suma, o método proposto foi capaz de obter soluções competitivas, sendo 94,67% delas melhores ou iguais as encontradas na literatura.

Palavras-chave: Sequenciamento da produção; flow shop com bloqueio; meta-heurística; busca local.

Abstract

Maximizing operational efficiency has become a priority for modern companies, driving the need to optimize the production flow, which plays a fundamental role in an organization's ability to meet market demands in an agile and profitable manner. In this regard, enhancing the job sequencing process is essential to increase productivity and reduce waste in a highly competitive market. Within this scope, this work addresses the blocking flow shop scheduling problem with makespan minimization. In this problem, n jobs must be scheduled in an environment of m machines ordered in series, in which all jobs must follow the same processing order. Furthermore, unlike the permutational flow shop problem, the intermediate buffers between these machines are considered zero. Thus, a machine can only release a job to the subsequent machine if this one is not occupied. A hybrid population heuristic that combines a ruin-and-recreate operator with a local search based on Variable Neighborhood Descent which makes use of acceleration methods is proposed to solve the problem. In this sense, a literature method for the insertion neighborhood was adapted for the block insertion and swap neighborhoods. Furthermore, lower bounds for the swap neighborhoods are also proposed with the purpose of avoiding the evaluation of moves that would not lead to an improvement of the current solution. Finally, a tie-breaking criterion and a population diversity control mechanism are employed to prevent the method from getting stuck in local optima. Extensive computational experiments were carried out on 150 benchmark instances, encompassing parameter calibration, evaluation of the tie-breaking criterion, performance of the neighborhood used, and other components of the method, such as the ruin-and-recreate operator and the diversity of the population. In short, the proposed method was able to obtain competitive solutions, with 94.67% being the best or equal to those found in the literature.

Keywords: Production scheduling; blocking flow shop; metaheuristic; local search.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Definição do Tema

Os problemas de *scheduling* estão associados com processos de tomada de decisão presentes em muitos sistemas de manufatura, lidando principalmente com a alocação de recursos em um determinado período de tempo visando otimizar diferentes objetivos (PINEDO, 2018). O sequenciamento da produção, isto é, a ordem com que os produtos são produzidos em um ambiente produtivo, constitui um dos problemas de *scheduling*.

Nesse sentido, as perspectivas inseridas dentro do contexto de sequenciamento de produção são vastas. Entre elas, um ambiente bastante comum encontrado na literatura é o *flow shop*. Nele, existem m máquinas em série e cada tarefa deve ser processada em cada uma destas máquinas. Todas as tarefas a serem processadas devem ter a mesma sequência de processamento, isto é, iniciar na máquina 1 e finalizar na máquina m passando exatamente pela mesma ordem de máquinas. Usualmente, após terminar seu processamento em uma das máquinas, a tarefa segue para uma fila onde aguardará para ser processada pela próxima máquina.

Observa-se que o problema de *flow shop* considera que o estoque intermediário entre as máquinas é infinito, situação que não é comum ocorrer na prática, em que estes estoques são limitados. Esta limitação implica em um fenômeno conhecido como bloqueio. Isto é, caso uma tarefa termine seu processamento em uma máquina i e o estoque entre as máquinas i e $i + 1$ estiver com sua capacidade máxima alcançada, ocorre um bloqueio na máquina i e a próxima tarefa da sequência não poderá ser processada. O objetivo do problema pode variar, sendo a minimização do *makespan*, tempo total de atraso e tempo total de ciclo, objetivos comumente encontrados na literatura (SHAO; PI; SHAO, 2018b, 2017; ABADI, 2007).

O problema de *flow shop* com bloqueio, do inglês *blocking flow shop scheduling problem* (BFSP), com minimização do *makespan* vem sendo amplamente estudado desde o trabalho de Levner (1969), dada sua grande importância prática. Neste problema, o estoque intermediário entre as máquinas é considerado nulo. Entre os métodos empregados para sua resolução, encontram-se os métodos exatos e os métodos heurísticos, sendo o primeiro utilizado para obtenção de soluções ótimas e o segundo utilizado para obtenção de soluções o mais próximo da ótima possível, ou seja, soluções sub-ótimas, que são obtidas em um tempo computacional aceitável tanto para instâncias de pequeno porte, quanto para instâncias de grande porte. Ainda, é importante pontuar que abordagens baseadas em meta-heurísticas são capazes de obterem soluções ótimas para o problema. Entretanto, a comprovação da otimalidade não pode ser comprovada por estas abordagens.

Dentro do exposto, o presente trabalho visa propor um método heurístico para o BFSP com minimização do *makespan*, utilizando instâncias presentes na literatura e comparando os resultados obtidos pelo método proposto com os encontrados na literatura.

1.2 Justificativa

Em um mercado cada vez mais competitivo, ferramentas que auxiliam na tomada de decisão de forma acertiva se fazem cada vez mais necessárias nas empresas. Nesse sentido, *scheduling*, como um processo de tomada de decisão, desempenha um papel importante na maioria dos sistemas de manufatura e produção, bem como na maioria dos ambientes de processamento de informações, sendo também importante em empresas de transportes e outros tipos de serviço (PINEDO, 2018).

Dados os objetivos de cada empresa, um sequenciamento de tarefas adequado pode não ser óbvio, assim como os impactos causados por um mal sequenciamento. No mundo atual, que caminha para uma maior modernização e chegada da Indústria 4.0, o escalonamento da produção faz-se fundamental nas empresas no sentido de se obter vantagem competitiva, agregação de valor e satisfação dos clientes. Para tal, faz-se necessária a utilização de ferramentas modernas para realizar o gerenciamento de recursos, alocando-os onde e quando são necessários de forma a obter melhores indicadores de desempenho.

Nesse cenário, o problema de escalonamento de tarefas em um ambiente de *flow shop* com bloqueio com minimização do *makespan* aparece como um problema prático e que, portanto, merece atenção para garantir com que os recursos em tais ambientes sejam gerenciados da maneira mais adequada. De acordo com Hall e Sriskandarajah (1996), uma

das causas de bloqueio nestes ambientes é a eventual impossibilidade de utilização de estoque intermediário entre as máquinas. Miyata e Nagano (2019) apontam que outra causa para ocorrência de bloqueio reside na própria tecnologia de produção, como temperatura e características dos materiais, que requerem que a tarefa finalizada em uma máquina i permaneça nesta máquina até que a máquina $i + 1$ possa recebê-la de modo a evitar deterioração na tarefa, retrabalho e custos adicionais.

Em termos práticos, o *flow shop* com bloqueio pode ser observado na indústria química, onde produtos químicos parcialmente processados às vezes devem ser mantidos em máquinas devido à falta de armazenamento intermediário (LIU; KOZAN, 2009). Chen et al. (2014) estudaram uma fábrica de eletrônicos que monta e testa placas de circuito impresso. Após a montagem, um lote de placas é submetido a uma sequência de testes em duas câmaras de triagem: a primeira câmara é usada para submeter o lote ao teste de vibração e a segunda câmara é usada para testá-lo a temperaturas extremas. Os requisitos tecnológicos não permitem que um lote espere entre estas duas câmaras. Como resultado, um lote deve ser bloqueado dentro da primeira câmara se a segunda estiver ocupada. Outros sistemas de produção também podem ser modelados como sistemas de *flow shop* com bloqueio, como indústrias farmacêuticas (HALL; SRISKANDARAJAH, 1996), linhas de produção *just-in-time* (PRASAD; RAJENDRAN; CHETTY, 2006) e células robóticas (RIBAS; COMPANYS; TORT-MARTORELL, 2015).

Considerando que o problema de sequenciamento de tarefas em um ambiente de *flow shop* com bloqueio com minimização do makespan é comprovadamente NP-Difícil (RÖCK, 1984; MARTINEZ et al., 2006; FERNANDEZ-VIAGAS; LEISTEN; FRAMINAN, 2016), não é possível garantir que a melhor solução seja encontrada em tempo polinominal. Assim, faz-se necessário a aplicação de métodos heurísticos para resolução do problema, dada a grande aplicabilidade prática demonstrada. Apesar de algumas tentativas de resolver o problema de forma exata (REDDI; RAMAMOORTHY, 1972; SUHAMI; MAH, 1981; RONCONI, 2005; COMPANYS; MATEO, 2007; PITTY; KARIMI, 2008; TOUMI et al., 2017; BAUTISTA et al., 2012), o melhor método, o *improved bounded dynamic programming algorithm* de Ozolins (2019), só é capaz de resolver instâncias com até 20 tarefas de forma ótima. Portanto, a grande maioria dos procedimentos de solução para o BFSP são baseados em abordagens meta-heurísticas, que parecem ser mais adequadas para tratar instâncias de tamanho realista. Assim, o método desenvolvido no presente trabalho baseia-se neste tipo de abordagem.

1.3 Objetivos

1.3.1 Objetivo Geral

Propor um algoritmo heurístico para o problema de sequenciamento de tarefas em um ambiente de *flow shop* com bloqueio e minimização do *makespan*.

1.3.2 Objetivos Específicos

- Realizar uma revisão da literatura acerca dos métodos heurísticos e exatos para resolução do problema.
- Implementar método(s) construtivo(s) para geração de soluções iniciais para o problema.
- Implementar operadores de busca local para melhoria das soluções iniciais obtidas.
- Implementar operadores de perturbação e/ou destruição e reconstrução de soluções previamente exploradas pela busca local.
- Propor e implementar uma abordagem baseada em um algoritmo populacional utilizando os métodos previamente implementados.
- Resolver o problema de sequenciamento de tarefas em um ambiente de *flow shop* com bloqueio e minimização do *makespan*.
- Utilizar instâncias já conhecidas na literatura para testar o método proposto.
- Comparar os resultados obtidos com os resultados encontrados na literatura.

1.4 Organização do Trabalho

O trabalho encontra-se organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica e a revisão da literatura. O Capítulo 3 descreve o algoritmo proposto para a resolução do problema. Os experimentos computacionais são mostrados no Capítulo 4 e as conclusões obtidas são evidenciadas no Capítulo 5.

Capítulo 2

Fundamentação Teórica

2.1 Notações em scheduling

Uma grande variedade de problemas de *scheduling* pode ser encontrada na literatura. Graham et al. (1979) propuseram uma nomenclatura de três campos $\alpha|\beta|\gamma$ para representar tais problemas. O campo α descreve o ambiente de máquinas e geralmente contém apenas uma entrada. O campo β fornece detalhes de características de processamento e restrições e pode conter nenhuma entrada, uma única entrada ou várias entradas. O campo γ descreve o objetivo e geralmente contém uma única entrada. O presente trabalho utiliza-se desta notação e, portanto, é necessário apresentar as possíveis descrições que um problema de *scheduling* pode ter em cada um dos três campos. O objetivo não é esgotar todas as possibilidades, mas sim apresentar as notações que serão utilizadas ao decorrer do texto. Para uma descrição mais detalhada, recomenda-se a leitura do capítulo 2 de Pinedo (2018).

Os possíveis ambientes de máquina especificados no campo α quando o ambiente de *flow shop* é estudado são os seguintes:

Flow shop — F_m: Existem m máquinas em série. Cada tarefa tem que ser processada em cada uma das m máquinas. Todas as tarefas devem seguir a mesma sequência, ou seja, elas devem ser processadas primeiro na máquina 1, depois na máquina 2 e assim por diante. Após a conclusão em uma máquina, uma tarefa entra na fila para a próxima máquina. Normalmente, assume-se que todas as filas operam sob a disciplina *First In, First Out* (FIFO), ou seja, uma tarefa não pode "passar" por outra enquanto espera em uma fila. Se a disciplina FIFO estiver em vigor, o *flow shop* é referido como um *flow shop* permutacional e o campo β inclui a entrada *prmu*.

Flexible flow shop — FF_c: Um *flexible flow shop* é uma generalização do *flow shop* e ambientes de máquinas paralelas. No lugar de m máquinas em série, há c estágios em série em que cada estágio possui um número de máquinas idênticas em paralelo. Cada tarefa deve ser processada primeiro no estágio 1, depois no estágio 2 e assim por diante. Em cada estágio, a tarefa k requer processamento em apenas uma máquina e qualquer máquina pode fazer este processamento. As filas entre os vários estágios podem ou não operar de acordo com a disciplina *First Come, First Served* (FCFS).

No que diz respeito ao campo α, outros ambientes de máquina podem ser encontrados na literatura, como máquina única (1), máquinas paralelas idênticas (P_m), máquinas paralelas com diferentes velocidades (Q_m), *job shop* (J_m), *flexible job shop* (FJ_c), *open shop* (O_m), entre outros. Já com relação às restrições e características de processamento especificadas no campo β, pode-se ter várias entradas. As entradas mais comuns no escopo do presente trabalho são as seguintes:

Tempos de *setup* dependentes da sequência — st_{sd} : O st_{sd} representa o tempo de *setup* dependente da sequência que ocorre entre o processamento de tarefas; st_{0k} denota o tempo de *setup* para uma tarefa k se ela for a primeira na sequência e st_{j0} o tempo de *setup* após a tarefa j se ela for a última na sequência. Se st_{sd} não aparecer no campo β, todos os tempos de *setup* são considerados 0 ou independentes da sequência, caso em que são simplesmente incluídos nos tempos de processamento.

Permutação — *prmu*: Uma restrição que pode aparecer no ambiente de *flow shop* é que as filas de cada máquina operam de acordo com a política *First In, First Out* (FIFO). Isso implica que a ordem (ou permutação) em que as tarefas passam pela primeira máquina é mantida em todo o sistema.

Bloqueio — *block*: O bloqueio é um fenômeno que pode ocorrer em ambientes de *flow shop*. Se um *flow shop* tiver um estoque intermediário limitado entre duas máquinas sucessivas, pode acontecer que, quando o estoque estiver cheio, a máquina anterior não tenha permissão para liberar uma tarefa concluída. O bloqueio implica que esta tarefa deve permanecer na máquina, impedindo (ou seja, bloqueando) que a máquina trabalhe na próxima tarefa. Note que o bloqueio implica em um *flow shop* permutacional. Ainda, a notação *block* é empregada quando o estoque entre as máquinas é considerado nulo. No caso em que este limite não é nulo, a notação geralmente empregada para indicar que há um limite no estoque entre duas máquinas i e $i+1$ é $b_{i,i+1}$.

No-wait — *nwt*: O requisito de *no-wait* é outro fenômeno que pode ocorrer em ambientes de *flow shop*. As tarefas não podem esperar entre duas máquinas sucessivas. Isso significa

que o horário de início de uma tarefa na primeira máquina deve ser atrasado para garantir que ela possa passar pelo *flow shop* sem ter que esperar por nenhuma máquina.

Outras restrições também podem aparecer, a depender do problema a ser estudado. Por fim, tem-se o campo γ que indica o objetivo do problema. Exemplos que podem aparecer neste campo são:

Makespan — C_{max}: O makespan é equivalente ao tempo de conclusão da última tarefa a deixar o sistema.

Maximum Lateness — L_{max}: Mede a pior violação das datas de entrega das tarefas.

Total weighted completion time — $\sum_j w_j C_j$: A soma dos tempos de conclusão ponderados das tarefas fornece uma indicação dos custos totais de manutenção ou estoque incorridos pela programação. A soma dos tempos de conclusão é frequentemente referida na literatura como tempo de fluxo. O tempo de conclusão total ponderado é então referido como o tempo de fluxo ponderado.

Total weighted tardiness — $\sum_j w_j T_j$: Esta também é uma função de custo mais geral do que o tempo total ponderado de conclusão, que está relacionado com o atraso na conclusão das tarefas.

Logo, por meio das definições dessas notações, tem-se que o problema estudado no presente trabalho pode ser definido como $F_m|block|C_{max}$, uma vez que o ambiente de estudo é o *flow shop* (F_m), a restrição presente é o bloqueio ($block$) e o objetivo é a minimização do *makespan* (C_{max}).

2.2 Conceitos Fundamentais

2.2.1 Cálculo do makespan

Conforme explicitado anteriormente, o objetivo problema é realizar a minimização do *makespan* em um ambiente de *flow shop com bloqueio*. Para isso, deve-se encontrar uma permutação π de n tarefas a serem processadas em um conjunto de m máquinas em série, com capacidades dos estoques intermediários iguais a zero, de forma que o *makespan* seja o menor possível.

Então, seja D_{i,π_k} o tempo em que a k-ésima tarefa da permutação π realmente deixa a máquina i. Note que $D_{i,\pi_k} \geq C_{i,\pi_k}$, em que C_{i,π_k} denota o tempo de conclusão do processamento da tarefa π_k na máquina i. O caso em que $D_{i,\pi_k} = C_{i,\pi_k}$ ocorre quando não

há um bloqueio entre as máquinas i e $i+1$. O *makespan* é dado por $C_{max} = D_{m,\pi_n}$. Sendo assim, Pinedo (2018) mostra que o *makespan* de uma permutação π pode ser calculado conforme segue:

$$D_{i,\pi_1} = \sum_{l=1}^{i} p_{l,\pi_1} \qquad i = 1, \ldots, m \tag{2.1}$$

$$D_{i,\pi_k} = max(D_{i-1,\pi_k} + p_{i,\pi_k}, D_{i+1,\pi_{k-1}}) \qquad k = 2, \ldots, n; i = 1, \ldots, m-1 \tag{2.2}$$

$$D_{m,\pi_k} = D_{m-1,\pi_k} + p_{m,\pi_k} \qquad k = 2, \ldots, n; \tag{2.3}$$

Note que p_{i,π_k} denota o tempo de processamento da k-ésima tarefa da permutação na máquina i. A Equação (2.1) calcula os tempos de saída da primeira tarefa da permutação π de cada uma das máquinas, em que a tarefa não encontra nenhum bloqueio. Já a Equação (2.2) calcula os tempos de saída das demais tarefas de cada uma das máquinas, com exceção da máquina m. Neste caso, verifica-se se a tarefa poderá sair da máquina i ao terminar seu processamento ou se deverá esperar a tarefa $k-1$ acabar de ser processada na máquina $i+1$. Por fim, os tempos de saída das tarefas da máquina m são dados pela Equação (2.3). Observe que a complexidade computacional para o cálculo do *makespan* em um *flow shop* com bloqueio com m máquinas, dada uma permutação de n tarefas, é de $O(nm)$.

2.2.2 Representação em grafo

Uma solução π para o $F_m|block|C_{max}$ pode ser representado por um grafo (GRABOWSKI; PEMPERA, 2007; PINEDO, 2018). Tal grafo pode ser definido como sendo $G(\pi) = (S, F^+ \cup F^0 \cup F^-)$, que é constituído por um conjunto de vértices $S = n \times m$, onde n é o número de tarefas e m é o número de máquinas. Além disso, o conjunto de arcos é dado por $F^+ \cup F^0 \cup F^-$, onde:

$$F^+ = \bigcup_{i=2}^{m} \bigcup_{k=1}^{n} \{((i-1,k),(i,k))\} \tag{2.4}$$

$$F^0 = \bigcup_{i=1}^{m} \bigcup_{k=2}^{n} \{((i,k-1),(i,k))\} \tag{2.5}$$

$$F^- = \bigcup_{i=2}^{m} \bigcup_{k=2}^{n} \{((i,k-1),(i-1,k))\}. \tag{2.6}$$

As Equações (2.4) referem-se ao caso em que o tempo de saída de uma tarefa π_k de M_i é igual ao seu tempo de saída da máquina anterior mais o seu tempo de processamento em M_i. Por outro lado, as Equações (2.5) indicam o caso em que o tempo de saída de uma tarefa π_k de M_i é igual ao tempo de saída de π_{k-1} de M_i mais o seu tempo de processamento em M_i. Por fim, as Equações (2.6) representam o bloqueio, isto é, quando $D_{i,\pi_k} = D_{i+1,\pi_{k-1}}$.

Cada nó (i, k) possui um peso associado que é igual a p_{i,π_k} que representa o tempo de processamento da tarefa π_k na máquina i. Cada arco $\{((i, k-1), (i-1, k))\} \in F^-$ garante o bloqueio da tarefa π_k na máquina i se a máquina $i+1$ não estiver liberada. Estes arcos têm pesos iguais a $-p_{i-1,\pi_k}$, enquanto os arcos pertencentes a F^+ e F^0 têm pesos iguais a zero. O *makespan* de uma permutação π é dado pelo caminho crítico entre os nós $(1, 1)$ e (n, m) do grafo $G(\pi)$. Uma representação de um grafo para uma permutação com 5 tarefas a serem processadas em 5 máquinas pode ser visualizado na Figura 2.1.

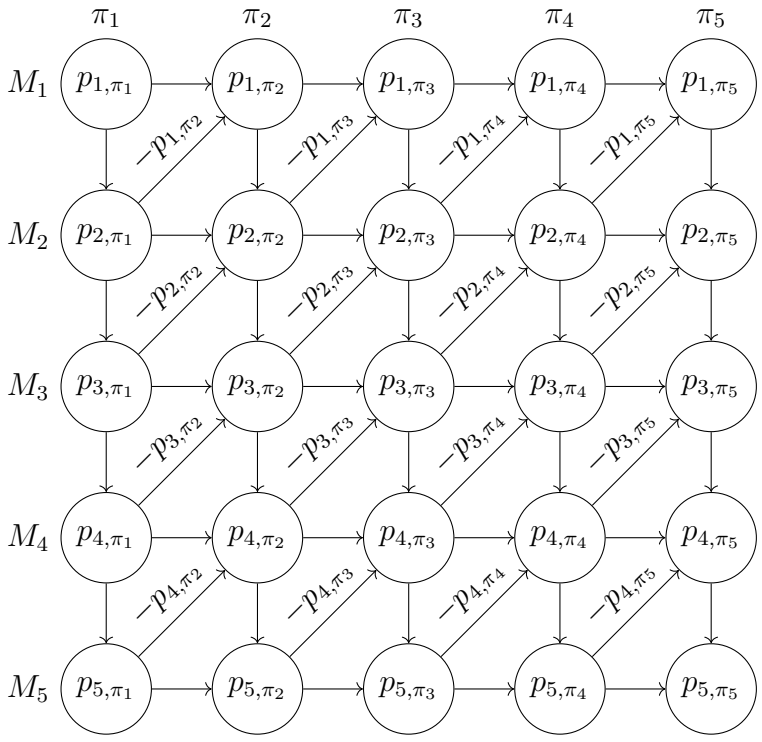

Figura 2.1: Representação em grafo do $F_m|block|C_{max}$ para 5 tarefas e 5 máquinas

Considere uma instância com 5 tarefas e 4 máquinas, em que os tempos de processamentos são descritos na Tabela 2.1. Ainda, seja $\pi = (\pi_1, \pi_2, \pi_3, \pi_4, \pi_5) = (J_4, J_1, J_5, J_3, J_2)$ uma permutação para o problema. A matriz D abaixo mostra todos os tempos de saída das tarefas de cada uma das máquinas, após a aplicação das Equações (2.1)–(2.3) a π. O makepan é igual a $C_{max} = D_{m,\pi_n} = 38$.

Tabela 2.1: Tempo de processamento para uma instância com 5 tarefas e 4 máquinas

	M_1	M_2	M_3	M_4
J_1	5	4	4	3
J_2	5	4	4	6
J_3	3	2	3	3
J_4	6	4	4	2
J_5	3	4	1	5

$$D = \begin{bmatrix} 6 & 11 & 15 & 19 & 24 \\ 10 & 15 & 19 & 22 & 28 \\ 14 & 19 & 22 & 27 & 32 \\ 16 & 22 & 27 & 30 & 38 \end{bmatrix}$$

Conforme explicado, uma outra maneira de cálculo do *makespan* é por meio da obtenção do caminho crítico associado ao grafo que representa π. Assim, a Figura 2.2 ilustra o grafo associado a permutação. O caminho crítico é obtido por meio do método utilizado por Ding et al. (2016):

1. Seja $(i, k) = (m, n)$ o último nó do grafo.
2. Selecione o nó (i, k) imediatamente anterior de acordo com as seguintes equações:

$$(i,k) = \begin{cases} (i, k-1), & \text{se } D_{\pi_k,i} = D_{i,\pi_{k-1}} + p_{i,\pi_k} \\ (i-1, k), & \text{se } D_{i,\pi_k} = D_{i-1,\pi_k} + p_{i,\pi_k} \\ (i+1, k-1), & \text{se } D_{i,\pi_k} = D_{i+1,\pi_{k-1}} \end{cases} \tag{2.7}$$

3. Se $(i, k) = (1, 1)$, pare. Caso contrário, volte ao passo 2.

A Figura 2.3 mostra o caminho crítico em $G(\pi)$ após aplicar os passos acima usando os tempos de saída armazenados em D. A soma dos pesos dos nós e arcos do caminho corresponde a 38, que é igual ao makepan obtido após a aplicação das Equações (2.1)–(2.3) a π. Note que o peso acumulado até chegar em um determinado do caminho crítico indica o tempo de saída da tarefa da máquina correspondente.

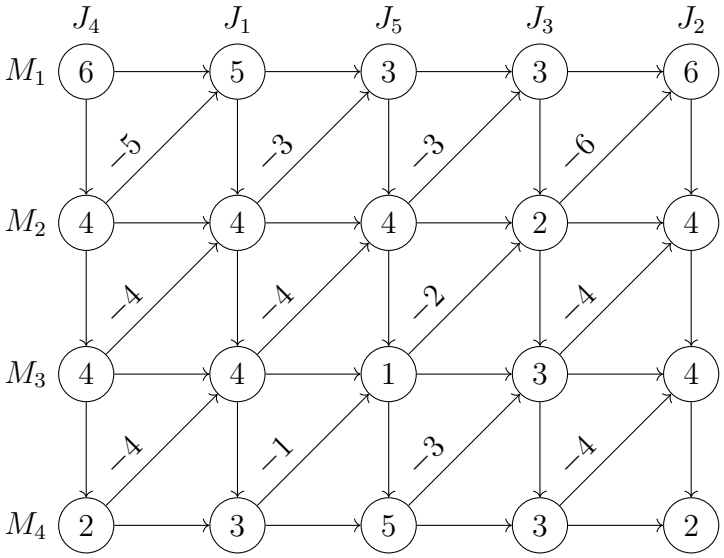

Figura 2.2: Representação em grafo de π

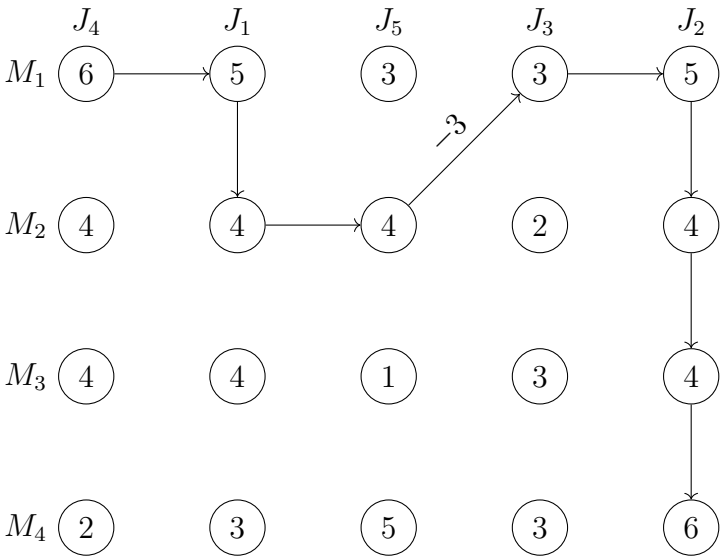

Figura 2.3: Critical path for π

2.2.3 Reversibilidade

De acordo com a propriedade de reversibilidade, caso uma permutação π^r possua as tarefas ordenadas em ordem reversa a π, o *makespan* das duas permutações é igual, caso π^r seja processada na ordem inversa de máquinas. Assim, a reversibilidade do $F_m|block|C_{max}$ considera dois ambientes de *flow shop* com bloqueio com n tarefas e m máquinas. Seja $p^{(1)}_{i,\pi_k}$ e $p^{(2)}_{i,\pi_k}$ os tempos de processamento da tarefa π_k na máquina i nos dois ambientes. Ao se assumir que $p^{(1)}_{i,\pi_k} = p^{(2)}_{i,\pi_{m+1-k}}$, tem-se que a primeira máquina no primeiro ambiente é igual a última máquina no segundo ambiente, a segunda máquina no primeiro ambiente é igual a penúltima máquina no segundo ambiente, e assim por diante. Esta relação fornece o seguinte resultado: o *makespan* da permutação $\pi = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$ no primeiro ambiente é igual ao de uma permutação π^r, que possui as tarefas ordenadas em uma ordem reversa a π, em um ambiente em que as m máquinas também estão em ordem invertida. Ou seja, o *makespan* de uma permutação π não muda se as tarefas atravessarem o *flow shop* na direção reversa e na ordem inversa de processamento.

A fim de exemplificar a propriedade de reversibilidade, um exemplo numérico é apresentado a seguir. Considere a instância para o $F_m|block|C_{max}$ apresentado na Tabela 2.2, que mostra os tempos de processamento de 4 tarefas em cada uma das 4 máquinas do ambiente de *flow shop*. Dada a permutação de tarefas arbitrária $\pi = (\pi_1, \pi_2, \pi_3, \pi_4) = (J_3, J_4, J_1, J_2)$, calcula-se os tempos de saída das tarefas de cada uma das máquinas utilizando as Equações (2.1)-(2.3) e obtem-se um *makespan* de 29. Esta solução está representada por meio de um gráfico de Gantt na Figura 2.4. Note que a solução mostra os tempos de conclusão das tarefas em cada uma das máquinas, e não necessariamente os tempos de saída.

Tabela 2.2: Instância para o $F_m|block|C_{max}$ com 4 tarefas e 4 máquinas

Máquinas / Tarefas	M_1	M_2	M_3	M_4
J_1	2	4	3	1
J_2	3	1	3	4
J_3	8	5	1	2
J_4	2	2	3	3

Aplicando a propriedade da reversibilidade, nota-se que o *makespan* será mantido, ou seja, caso a permutação reversa $\pi^r = (\pi^r_1, \pi^r_2, \pi^r_3, \pi^r_4) = (\pi_4, \pi_3, \pi_2, \pi_1) = (J_2, J_1, J_4, J_3)$ seja processada em uma sequência de máquinas também reversa, o *makespan* ainda será igual a 29. A representação da solução pode ser observada por meio de um gráfico de Gantt mostrado na Figura 2.5. No decorrer do trabalho, os tempos de saída referentes a π^r são

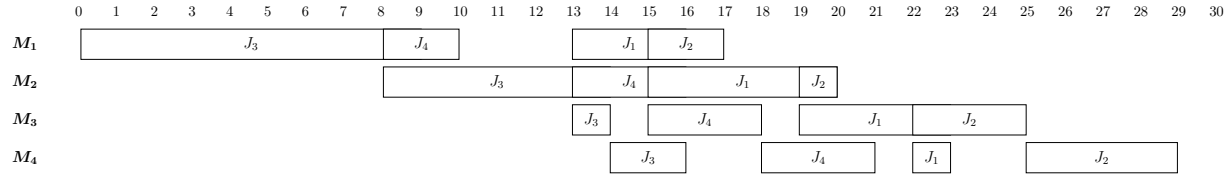

Figura 2.4: Representação direta da solução π

denotados por R_{i,π_k^r}, para toda tarefa k e máquina i.

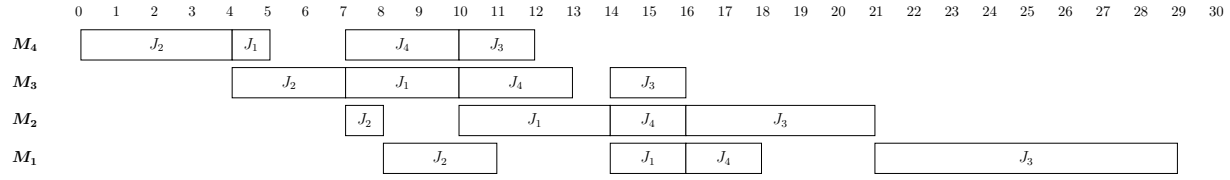

Figura 2.5: Representação reversa da solução π

2.2.4 Propriedades de bloqueio

Baseados na representação em grafo e dos tipos de arcos envolvidos no caminho crítico descritos na Seção 2.2.2, Grabowski e Pempera (2007) propuseram propriedades para o $F_m|block|C_{max}$. Estas propriedades guiam na execução de movimentos em uma permutação π de tarefas para obtenção de uma nova permutação π' de tal modo que $C_{max}(\pi') < C_{max}(\pi)$. Para elucidar tais propriedades, faz-se necessário definir as possíveis subsequências que podem compor o caminho crítico no grafo dado por uma permutação π de tarefas. A Figura 2.6 ilustra tais subsequências em um caminho definido em um grafo para o $F_m|block|C_{max}$.

Definição 1 — *Normal block*: Uma subsequência em π que é composta por nós conectados pelos arcos em F^0 é denominada *normal block* de tarefas em π e é denotada por $NB_l, l \in GH$, onde $GH = \{1, 2, \ldots, l_{NB}\}$ é o conjunto dos indíces de *normal blocks*.

Definição 2 — *Anti-block*: Uma subsequência em π que é composta por nós conectados pelos arcos em F^- é denominada *anti-block* de tarefas em π e é denotada por $AB_l, l \in ST$, onde $ST = \{1, 2, \ldots, l_{AB}\}$ é o conjunto dos indíces de *anti-blocks*.

Definição 3 — *Internal block*: A subsequência excluindo a primeira e a última tarefas de $NB_l(AB_l)$ é denominada *internal block* de $NB_l(AB_l)$ e é denotada por $NB_l^*(AB_l^*)$. Note que podem existir *normal blocks* e *anti-blocks* sem *internal blocks*.

Definição 4 — *T-sequence*: Os nós que são conectados por arcos de F^+ formam uma subsequência denominada *T-sequence*.

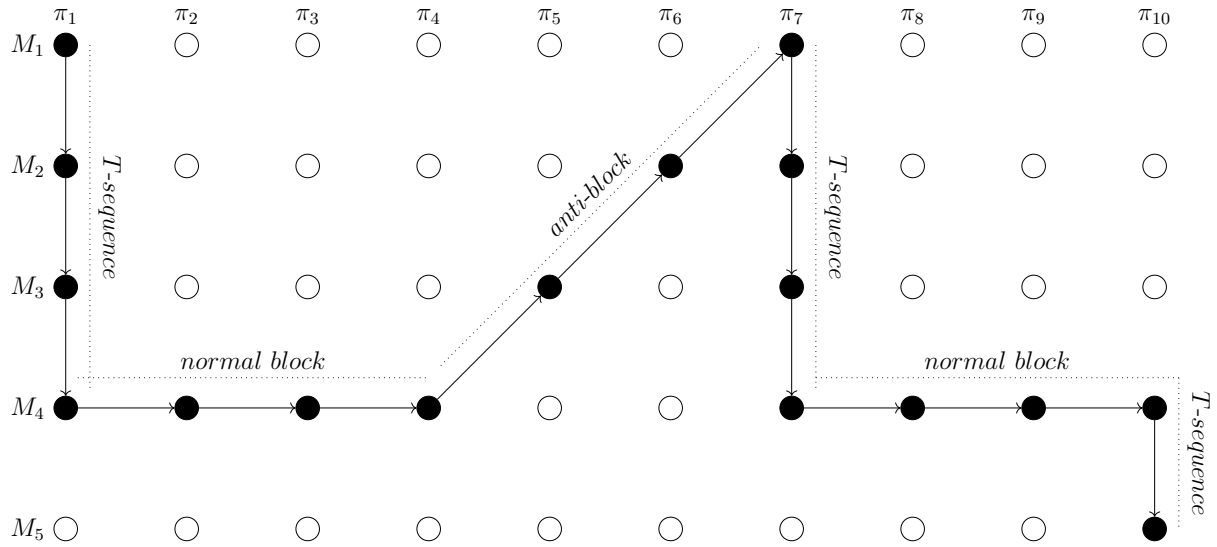

Figura 2.6: Exemplificação dos tipos de subsequência em um grafo. Os nós destacados representam um caminho crítico para uma sequência aleatória.

A partir da definição dos tipos de subsequência de uma permutação, Grabowski e Pempera (2000) propuseram as seguintes propriedades:

Propriedade 1: Para qualquer *normal block* NB_l em π e uma permutação $\pi^{'}$ obtida de π por uma troca de tarefas do *internal block* NB_l^*, tem-se que $C_{max}(\pi^{'}) \geq C_{max}(\pi)$.

Propriedade 2: Para qualquer *anti-block* AB_l em π e uma permutação $\pi^{'}$ obtida de π por uma troca de tarefas do *internal block* AB_l^*, tem-se que $C_{max}(\pi^{'}) \geq C_{max}(\pi)$.

A partir destas propriedades, Grabowski e Pempera (2007) propuseram o Teorema 1 definido abaixo. Note que o teorema fornece condições suficientes para obtenção de uma permutação $\pi^{'}$ a partir de π de tal modo que $C_{max}(\pi^{'}) \leq C_{max}(\pi)$.

Teorema 1: Seja π uma permutação com *normal block* NB_l e *anti-block* AB_l. Se uma permutação π' foi obtida a partir de π por meio de uma troca de posição de tarefas de tal forma que $C_{max}(\pi') \leq C_{max}(\pi)$, então em π' uma das das condições abaixo deve ser atendida:

(i) pelo menos uma tarefa $k \in NB_l$ precede a primeira tarefa de NB_l para algum $l \in GH$
(ii) pelo menos uma tarefa $k \in NB_l$ sucede a última tarefa de NB_l para algum $l \in GH$
(iii) pelo menos uma tarefa $k \in AB_l$ precede a primeira tarefa de AB_l para algum $l \in ST$
(iv) pelo menos uma tarefa $k \in NB_l$ sucede a última tarefa de AB_l para algum $l \in ST$

A Figura 2.7 destaca as tarefas em que a aplicação de movimentos de inserção ou troca entre elas não acarretará na obtenção de uma permutação $\pi^{'}$ com *makespan* menor que ao da solução corrente. Em suma, movimentos de busca local não devem ser feitos

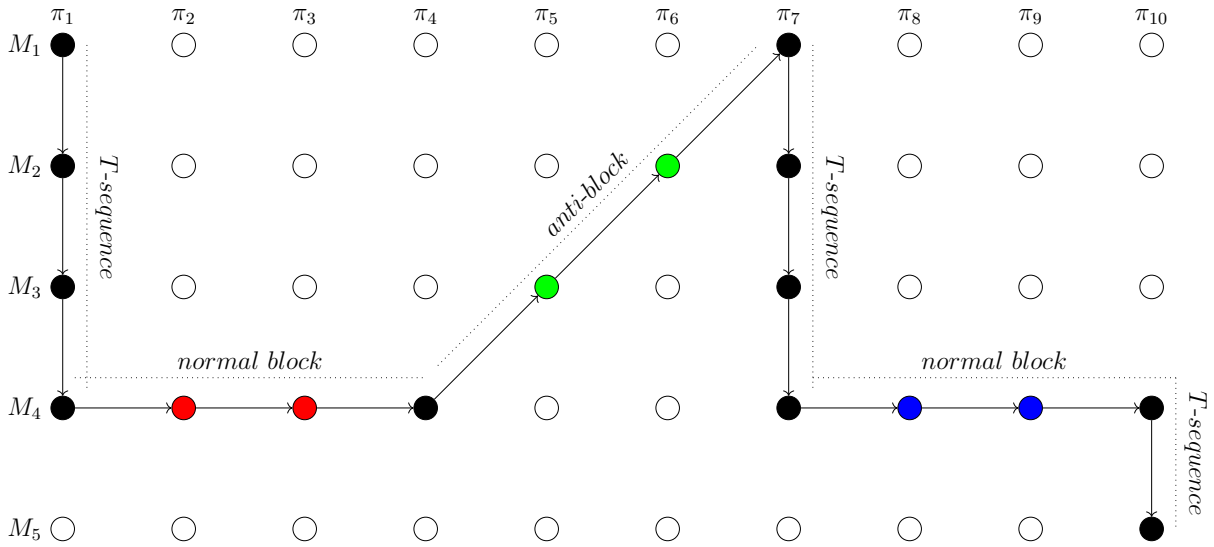

Figura 2.7: Exemplo do Teorema 1. A troca ou a inserção entre as tarefas coloridas de π não produzirá uma permutação π' em que $C_{max}(\pi') < C_{max}(\pi)$.

em tarefas que fazem parte do interior de um *normal block* ou *anti-block*.

Apesar do Teorema 1 fornecer condições suficientes para se obter uma sequência π' a partir de π com um *makespan* melhor, experimentos computacionais mostram que o tamanho dos *normal blocks* e *anti-blocks* é pequeno. A partir disso, Ding et al. (2016) propuseram propriedades para aumentar a eficiência de algoritmos para resolver o $F_m|block|C_{max}$. De forma a compreender as propriedades propostas, a Definição 5 faz-se necessária, em que s_l e e_l indicam a máquina em que uma subsequência inicia e termina, respectivamente. A Figura 2.8 exemplifica a Definição 5.

Definição 5 — ***Joined block***: Dado um bloco AB_l ou NB_l, ele é chamado de *joined block* se ele inicia na mesma máquina que o bloco anterior terminou, ou seja, $s_l = e_{l-1}$.

Dada a definição acima, o conjunto de posições $\{\pi(u_{l-1}), \pi(u_{l-1}+1), \ldots, \pi(u_l)\}$ para um bloco B_l é dado por:

$$U_l = \begin{cases} \{u_{l-1}, u_{l-1}+1, \ldots, u_l\}, se\ B_l\ for\ um\ joined\ block \\ \{u_{l-1}+1, \ldots, u_l\}, caso\ contrario \end{cases} \tag{2.8}$$

Dada uma tarefa $\pi(a)$, os autores definem um conjunto de movimentos para a esquerda como $L_a(x,y) = \{(a,b)|x \leq b \leq y, y < a\}$ e um conjunto de movimentos para a direita como $R_a(x,y) = \{(a,b)|x \leq b \leq y, a < x\}$. Definir esses movimentos auxilia na formulação de operações de deslocamento de uma tarefa $\pi(a)$ entre diferentes blocos. Por fim, para construir as propriedades, a seguinte definição é proposta:

Fonte: Adaptado de Ding et al. (2016)

Figura 2.8: Exemplificação de um *joined block*

Definição 6: Seja π qualquer permutação para o $F_m|block|C_{max}$. Seja π_v uma permutação obtida por um movimento $v = (a, b)$ de $L_a(u_{q-1} + 1, u_q), a \in U_k, q < k$ ou $R_a(u_{q-1}, u_q - 1), a \in U_k, q > k$. Defina:

$$\pi_0 = \begin{cases} \pi(u_q - 1), se\ b\ \neq u_q,\ q\ <\ k,\ ou\ b\ \neq\ u_q - 1,\ q\ >\ k, \\ \pi(a), caso\ contrario \end{cases} \tag{2.9}$$

A partir das definições apresentadas, Ding et al. (2016) propuseram três propriedades para o cálculo de um limite inferior de uma sequência π_v obtida a partir de π por um movimento de deslocamento de uma tarefa $\pi(a)$. Tais propriedades são definidas a seguir.

Teorema 2: Seja π qualquer permutação para o $F_m|block|C_{max}$. Seja π_v uma permutação obtida por um movimento $v = (a, b)$ de $L_a(u_{q-1} + 1, u_q), q < k$ ou $R_a(u_{q-1}, u_q - 1), q > k, a \in U_k, k \in GH$. Então, $C_{max}(\pi_v) \geq C_{max}(\pi) + \delta_{kq}(\pi)$, onde:

$$\delta_{kq}(\pi) = \begin{cases} p_{s_q,\pi(a)} - p_{s_k,\pi(a)},\ q \in GH, \\ p_{e_q,\pi_0} - p_{s_k,\pi(a)},\ q \in ST \end{cases} \tag{2.10}$$

Teorema 3: Seja π qualquer permutação para o $F_m|block|C_{max}$. Seja π_v uma permutação obtida por um movimento $v = (a, b)$ de $L_a(u_{q-1} + 1, u_q), q < k$ ou $R_a(u_{q-1}, u_q - 1), q > k, a \in U_k, k \in ST$. Então, $C_{max}(\pi_v) \geq C_{max}(\pi) + \delta_{kq}(\pi)$, onde:

$$\delta_{kq}(\pi) = \begin{cases} p_{s_q,\pi(a)} - p_{s_k,\pi(u_{k-1}-1)},\ q \in GH,\ q \neq k-1, ou\ q = k-1, b \neq u_{q-1}, \\ p_{e_q,\pi_0} - p_{s_k,\pi(u_{k-1}-1)},\ q \in ST \end{cases} \tag{2.11}$$

Teorema 4: Seja π qualquer permutação para o $F_m|block|C_{max}$. Seja π_v uma permutação obtida por um movimento $v = (a, b)$ de $L_a(u_{q-1} + 1, u_q), q < a$ ou $R_a(u_{q-1}, u_q - 1), q >$

$a, a \in V$. Então, $C_{max}(\pi_v) \geq C_{max}(\pi) + \delta_{aq}(\pi)$, onde:

$$\delta_{aq}(\pi) = \begin{cases} p_{s_q,\pi(a)} - (\sum_{i=s_a}^{e_a} p_{i,\pi(a)} - \sum_{i=s_a+1}^{e_a} p_{i,\pi(a-1)}), \ q \in GH, \\ p_{e_q,\pi_0} - (\sum_{i=s_a}^{e_a} p_{i,\pi(a)} - \sum_{i=s_a+1}^{e_a} p_{i,\pi(a-1)}), \ q \in ST \end{cases} \tag{2.12}$$

Com o intuito de elucidar os teoremas, um exemplo é apresentado a seguir utilizando o caminho crítico mostrado na Figura 2.3. Considere o caso em que J_1 deve ser inserida entre J_3 e J_2. Observando o caminho crítico, tem-se que J_1 pertence a uma *T-sequence* e as tarefas J_3 e J_2 formam um *normal-block*. Sendo assim, esta situação é apresentada no primeiro caso do Teorema 4. A Figura 2.9 ilustra a obtenção de um caminho, que não é necessariamente crítico, ao simular a inserção. O cálculo de $\delta_{aq}(\pi)$ é feito em seguida.

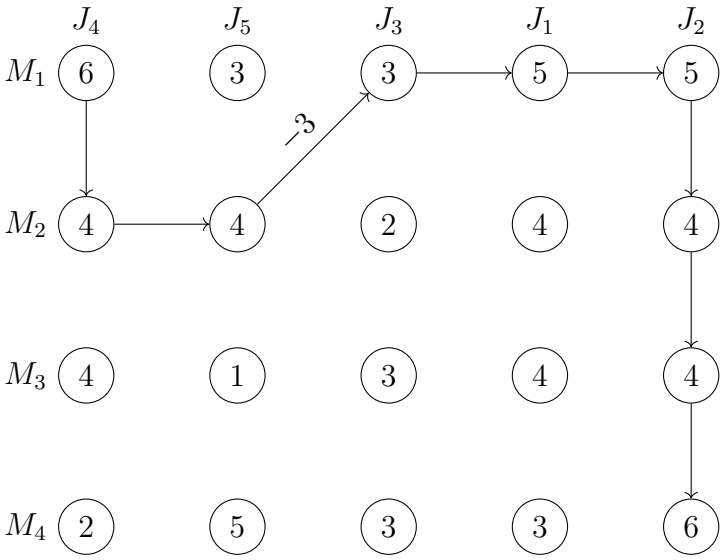

Figura 2.9: Simulação de um movimento de inserção

$$\delta_{aq}(\pi) = p_{s_q,\pi(a)} - (\sum_{i=s_a}^{e_a} p_{i,\pi(a)} - \sum_{i=s_a+1}^{e_a} p_{i,\pi(a-1)}) \tag{2.13}$$

Note que a *T-sequence* formada por J_1 não existe mais neste novo caminho e corresponde a parcela $\sum_{i=s_a}^{e_a} p_{i,\pi(a)}$. Em seu lugar, o nó (M_2, J_4), representado pela parcela $\sum_{i=s_a+1}^{e_a} p_{i,\pi(a-1)}$, passa a formar uma *T-sequence* com (M_1, J_4). Por fim, a parcela $p_{s_q,\pi(a)}$ corresponde ao nó (M_1, J_1) que foi inserido entre os nós (M_1, J_3) e (M_1, J_2). Substituindo os valores de cada parcela da Equação 2.13 pelos pesos dos nós correspondentes, tem-se:

$$\delta_{aq}(\pi) = 5 - ((5 + 4) - (4)) = 0 \tag{2.14}$$

Como $\delta_{aq}(\pi) \geq 0$, o movimento não precisa ser avaliado, uma vez que o peso asso-

ciado ao caminho não irá diminuir. Como o caminho utilizado para o cálculo de $\delta_{aq}(\pi)$ não é necessariamente crítico, este valor fornece um limitante inferior válido para a permutação que seria obtida pela inserção de J_1 entre J_3 e J_2. É importante pontuar que o cálculo de $\delta_{aq}(\pi)$ levou em consideração apenas os nós menciodados porque os demais nós permaneceram inalterados no caminho.

Observe que os Teoremas 2-4 fornecem condições para o cálculo de um limite inferior dado o deslocamento de uma tarefa π_a encontrada em *normal blocks*, *anti-blocks* e *T-sequences*, respectivamente. Além disso, tal limite inferior pode ser calculado em $O(1)$, enquanto que o cálculo do *makespan* de uma sequência π_v é feito em $O(nm)$ utilizando as Equações (2.1)-(2.3). Então, caso o limite inferior de $C_{max}(\pi_v)$ seja maior que $C_{max}(\pi)$, π_v não será uma permutação melhor que π e, portanto, não há necessidade de avaliar o movimento. Tais propriedades auxiliam na busca local ao explorarmos uma vizinhança de inserção, garantindo menor tempo computacional na busca.

2.3 Métodos de Resolução

2.3.1 Heurísticas Construtivas

2.3.1.1 Algoritmo NEH

O algoritmo NEH foi proposto por Nawaz, Jr e Ham (1983) para o $F_m|prmu|C_{max}$. O algoritmo assume que uma tarefa com um maior tempo total de processamento em todas as máquinas deve ter prioridade mais alta do que uma tarefa com menos tempo total de processamento. Uma visão geral do algoritmo é apresentada no Algoritmo 2.1.

Algoritmo 2.1 Algoritmo NEH

1: **procedimento** NEH(n, m)
2: Calcule o tempo de processamento total T_k para $k = \{1, \ldots, n\}$
3: Gere uma sequência $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ em ordem decrescente de T_k
4: $\pi \leftarrow \beta_1$
5: **para** $k \leftarrow 2, \ldots, n$ **faça**
6: Pegue a tarefa β_k de β e teste em todas as k posições possíveis de π
7: Insira β_k em π na posição em que o *makespan* foi menor
8: **retorne** π

As duas tarefas com os tempos de processamento totais mais altos são selecionadas inicialmente. Em seguida, a melhor sequência parcial para essas duas tarefas é encontrada por uma busca, ou seja, considerando as duas permutações parciais possíveis. As posições relativas dessas duas tarefas em relação uma à outra são fixadas nas etapas restantes do

algoritmo.

Após isso, a tarefa com o terceiro maior tempo total de processamento é selecionada e as três sequências parciais são testadas nas quais essa tarefa é colocada no início, meio e fim da sequência parcial encontrada na primeira etapa. Então, a melhor sequência parcial fixará as posições relativas dessas três tarefas para as etapas restantes. O processo é repetido até que todas as tarefas sejam alocadas e uma sequência completa seja encontrada.

Observe que o algoritmo possui complexidade computacional de $O(n^3m)$. Entretanto, Taillard (1990) mostrou que a complexidade da fase de inserção do algoritmo NEH pode ser reduzida de $O(n^2m)$ para $O(nm)$ utilizando a propriedade de reversibilidade do problema de *flow shop* apresentada na Seção 2.2.3. Assim, a complexidade computacional do algoritmo NEH é de $O(n^2m)$.

2.3.1.2 Profile Fitting

A heurística construtiva *Profile Fitting* (PF) foi originalmente proposta por McCormick et al. (1989) para o $F_m|block|CC$, mas pode ser facilmente modificada para o $F_m|block|C_{max}$. A heurística utiliza o perfil formado nas máquinas para determinar a tarefa que é inserida a cada iteração. Este perfil é baseado no tempo total que as máquinas permanecem ociosas ou bloqueadas para cada tarefa candidata na iteração corrente.

A heurística inicia escolhendo uma tarefa para iniciar a sequência de acordo com algum critério pré-estabelecido, como o menor tempo total de processamento, por exemplo. Observe que pela Equação (2.1), a primeira tarefa da sequência não encontra nenhum bloqueio e é processada tranquilamente entre as máquinas do sistema. Este processamento gera um perfil que é determinado pelos tempos de saída da tarefa de cada uma das máquinas e é dado pela Equação (2.1).

Para saber qual tarefa deve ser selecionada em seguida, cada uma das tarefas ainda não inseridas na sequência é testada. Então, para cada uma destas tarefas será computado o tempo total de ociosidade e bloqueio do sistema. Por exemplo, os tempos de saída de uma tarefa candidata π_2 de cada uma das máquinas podem ser calculados aplicando as Equações (2.1)-(2.3) conforme segue:

$$D_{1,\pi_2} = max(D_{1,\pi_1} + p_{i,\pi_2}, D_{2,\pi_1}) \tag{2.15}$$

$$D_{\pi_2,i} = max(D_{i-1,\pi_2} + p_{i,\pi_2}, D_{i+1,\pi_1}) \tag{2.16}$$

$$D_{\pi_2,m} = D_{m-1,\pi_2} + p_{m,\pi_2} \tag{2.17}$$

Observe que o tempo desperdiçado em determinada máquina i, isto é, o tempo em que ela fica ociosa ou bloqueada, para uma determinada tarefa candidata π_2 é dado por $D_{i,\pi_2} - D_{i,\pi_1} - p_{i\pi_2}$. Assim, a soma destes tempos é feita para cada uma das tarefas candidatas e a que obtiver o menor tempo total é selecionada como a segunda tarefa da sequência. Após a segunda tarefa ser selecionada, um novo perfil é gerado para as máquinas e o processo se repete até que todas as tarefas sejam incluídas na sequência. A heurística *Profile Fitting* pode ser observada em detalhes no Algoritmo 2.2. A complexidade computacional do algoritmo é de $O(n^2m)$.

Algoritmo 2.2 *Profile Fitting*

procedimento PF(n, m)
  Calcule o tempo de processamento total T_k para $k = \{1, \ldots, n\}$
  Gere uma sequência $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ em ordem crescente de T_k
  $\pi \leftarrow \beta_1$
  $\beta \leftarrow \beta - \{\beta_1\}$
  para $k \leftarrow 1, \ldots, n-2$ **faça**
    Calcule os tempos de saída D_{i,π_k} para a última tarefa inserida em π
    Para cada tarefa $j \in \beta$, calcule os possíveis tempos de saída $D_{i,\pi_{k+1}}$
    Calcule os tempos ocioso e de bloqueio $\delta_{j,k} = \sum_{i=1}^{m} D_{i,\pi_{k+1}} - D_{i,\pi_k} - p_{i\pi_{k+1}}$
    Selecione a tarefa $j \in \beta$ com o menor $\delta_{j,k}$
    $\pi \leftarrow \pi \cup \{j\}$
    $\beta \leftarrow \beta - \{j\}$
  Insira a última tarefa $j \in \beta$ como a última tarefa da permutação π
  retorne π

A partir da permutação formada pela *Profile Fitting*, Ronconi (2004) aplicou a fase de inserção do algoritmo NEH, originando uma heurística construtiva denominada PFE.

2.3.1.3 MinMax

A heurística construtiva MinMax (MM) foi proposta por Ronconi (2004). Esta heurística é baseada nas propriedades do *makespan* propostas por Ronconi e Armentano (2001), em que os autores propuseram um limite inferior para o tempo de saída de uma tarefa do *flow shop*.

Para construir uma solução viável para o problema, a heurística MM define na primeira e última posição da sequência as tarefas que os tempos de processamento são os mais curtos na primeira e na última máquina, respectivamente. Em seguida, a heurística adiciona a tarefa da próxima posição na sequência, a partir da segunda posição, considerando que o caminho crítico provavelmente será composto pelas somas dos valores máximos entre

os tempos de processamento de máquinas consecutivas e pela soma de m tempos de processamento. A fim de diminuir essas somas, o algoritmo seleciona a tarefa (c) como a tarefa consecutiva àquela já definida $(c-1)$, o que leva ao menor valor da Equação (2.18).

$$\alpha \sum_{i=1}^{m-1} |p_{i,c} - p_{i+1,c-1}| + (1-\alpha) \sum_{i=1}^{m} p_{i,c} \tag{2.18}$$

O pseudocódigo para heurística MM pode ser observado no Algoritmo 2.3.

Algoritmo 2.3 *MinMax*

procedimento MM(n, m, α)
  Seja $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ uma permutação de tarefas que se deseja construir
  $next_{pos} \leftarrow 1$
  $\pi_1 \leftarrow$ Tarefa com o tempo de procesamento mais curto na primeira máquina
  $next_{pos} \leftarrow next_{pos} + 1$
  $\pi_n \leftarrow$ Tarefa com o tempo de procesamento mais curto na última máquina
  enquanto $next_{pos} \neq n$ **faça**
    $\pi_{next_{pos}} \leftarrow$ A tarefa não alocada que leva ao menor valor da expressão 2.18
    $next_{pos} \leftarrow next_{pos} + 1$
  retorne π

Observe que, ao fim do algoritmo, será obtida uma sequência de tarefas π. Se aplicarmos a fase de inserção do algoritmo NEH a essa sequência, obteremos uma nova heurística construtiva denominada MME, também proposta por Ronconi (2004).

2.3.1.4 Heurísticas de Pan e Wang (2012)

Pan e Wang (2012) propuseram um total de oito heurísticas construtivas para o $F_m|block|C_{max}$. Inicialmente, os autores propuseram duas heurísticas chamadas wPF e PW, baseadas na heurística *Profile Fitting* e características do problema. Em seguida, combinando as heurísticas anteriores com a etapa de inserção do algoritmo NEH, os autores propuseram mais três heurísticas construtivas: $PF\text{-}NEH$, $wPF\text{-}NEH$ e $PW\text{-}NEH$. Por fim, os autores combinaram estas heurísticas com um processo de busca local baseada em inserção, dando origem a mais três heurísticas: $PF\text{-}NEH_{LS}$, $wPF\text{-}NEH_{LS}$ e $PW\text{-}NEH_{LS}$. A seguir, as oito heurísticas de Pan e Wang (2012) serão elucidadas com maiores detalhes.

A heurística construtiva wPF baseia-se diretamente na heurística *Profile Fitting*. Entretanto, a PF assume que os tempos ocioso e de bloqueio em cada uma das máquinas possuem o mesmo efeito no *makespan*. Todavia, dadas as Equações (2.1) - (2.3), percebe-se que os tempos ocioso e de bloqueio nas máquinas iniciais do sistema podem levar a um

atraso maior no processamento de tarefas sucessivas quando comparados a estes tempos nas máquinas mais adiante no sistema. Por outro lado, os tempos de ociosidade e bloqueio causados pelas tarefas iniciais de uma sequência tendem a possuir um maior impacto no *makespan* quando comparados às tarefas mais avançadas da permutação. Assim, a heurística *weighted Profile Fitting* (wPF) leva em consideração estas duas características e seleciona a tarefa j para assumir a posição $k+1$ de uma sequência $\pi = (\pi_1, \pi_2, \ldots, \pi_k)$ conforme Equações (2.19) - (2.20). O pseudocódigo para a heurística wPF pode ser visualizado no Algoritmo 2.4. Observa-se que a única diferença com relação ao Algoritmo 2.2 é com relação ao peso utilizado para medir o grau de importância dos tempos de ociosidade e de bloqueio de cada tarefa candidata. Assim como a *Profile Fitting*, a heurística *weighted Profile Fitting* possui complexidade computacional $O(n^2m)$.

$$\delta_{j,k} = \sum_{i=1}^{m} w_i(D_{k+1,i} - D_{k,i} - p_{j,i}) \tag{2.19}$$

$$w_i = \frac{m(n-2)}{i + k(m-i)} \tag{2.20}$$

Algoritmo 2.4 *weighted Profile Fitting*

procedimento WPF(n, m)
  Calcule o tempo de processamento total T_k para $k = \{1, \ldots, n\}$
  Gere uma sequência $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ em ordem crescente de T_k
  $\pi \leftarrow \beta_1$
  $\beta \leftarrow \beta - \{\beta_1\}$
  para $k \leftarrow 1, \ldots, n-2$ **faça**
    Calcule os tempos de saída D_{i,π_k} para a última tarefa inserida em π
    Para cada tarefa $j \in \beta$, calcule os possíveis tempos de saída $D_{i,\pi_{k+1}}$
    Calcule os tempos ocioso e de bloqueio $\delta_{j,k} = \sum_{i=1}^{m} w_i(D_{i,\pi_{k+1}} - D_{i,\pi_k} - p_{i,\pi_{k+1}})$
    Selecione a tarefa $j \in \beta$ com o menor $\delta_{j,k}$
    $\pi \leftarrow \pi \cup \{j\}$
    $\beta \leftarrow \beta - \{j\}$
  Insira a última tarefa $j \in \beta$ como a última tarefa da permutação π
  retorne π

A heurística construtiva PW gera uma sequência inserindo as tarefas considerando uma função de inserção que contém duas partes distintas. A primeira parte da função considera o tempo que as máquinas permanecem ociosas, enquanto a segunda parte representa todas as tarefas ainda não alocadas na sequência (U) como uma tarefa artificial v como se ela fosse ser inserida após a tarefa j que está sendo considerada para inserção. O tempo de processamento da tarefa artificial v é dado pela média dos tempos de processamento das tarefas ainda não inseridas, conforme a Equação (2.21).

$$p_{i,v} = \sum_{q \in U q \neq j} \frac{p_{i,q}}{n-k-1} \qquad i = 1, 2, \ldots, m \tag{2.21}$$

Assumindo que a tarefa $j \in U$ irá assumir a posição $k+1$ da sequência, calcula-se os tempos de saída da tarefa artifical v, que assumiria a posição $k+2$, de cada uma das máquinas. Então, a soma ponderada dos tempos de ociosidade e de bloqueio causados pela tarefa artificial v é dada pela Equação (2.22). Por fim, a função de inserção considerada é dada pela Equação (2.23). O pseudocódigo da heurístia PW pode ser visualizado no Algoritmo 2.5. Note que a complexidade computacional do algoritmo também é $O(n^2m)$.

$$\chi_{j,k} = \sum_{i=1}^{m} w_i (D_{i,\pi_{k+2}} - D_{i,\pi_{k+1}} - p_{i,v}) \tag{2.22}$$

$$f_{j,k} = (n-k-2)\delta_{j,k} + \chi_{j,k} \tag{2.23}$$

Algoritmo 2.5 *PW*

procedimento PW(π, n, m)
  Seja U o conjunto de tarefas ainda não selecionadas
  Selecione $j \in U$ com o menor $f_{j,0}$ como a primeira tarefa da sequência $\pi = \{\pi_1\}$
  $U \leftarrow U - \pi_1$
  para $k \leftarrow 1, \ldots, n-2$ **faça**
    Calcule os tempos de saída D_{i,π_k} para a última tarefa π_k inserida em π
    Para cada tarefa $j \in U$, calcule $p_{i,v}$, $D_{i,\pi_{k+1}}$ para j e $D_{i,\pi_{k+2}}$ para v
    Para cada tarefa $j \in U$, calcule $f_{j,k}$
    Selecione a tarefa com o menor $f_{j,k}$ para a posição $k+1$ de π
    Remova a tarefa inserida de U
  Insira a última tarefa $j \in U$ como a última tarefa da permutação π
  retorne π

Além disso, os autores combinaram o algoritimo NEH com as heurísticas PW, PF, wPF, criando as heurísticas $PW-NEH$, $PF\text{-}NEH$ e $wPF\text{-}NEH$, respectivamente. Tais heurísticas consistem em criar uma permutação de tarefas π utilizando uma das heurísticas construtivas mencionadas e, em seguida, aplicar a etapa de inserção do algoritmo NEH utilizando as últimas λ tarefas de π, isto é, as últimas λ tarefas são retiradas de π e reinseridas nas posições que resultem em um C_{max} menor. Ainda, os autores propuseram as heurísticas $PW\text{-}NEH(x)$, $PF\text{-}NEH(x)$ e $wPF\text{-}NEH(x)$ que constroem x permutações diferentes utilizando o procedimento mencionado e mantém aquela que possui o melhor *makespan*.

Por fim, os autores ainda integram uma *referenced local search* (RLS) conforme o Algoritmo 2.6 nas heurísticas $PW\text{-}NEH(x)$, $PF\text{-}NEH(x)$ e $wPF\text{-}NEH(x)$ criando heu-

rísticas compostas denominadas $PW\text{-}NEH_{LS(x)}$, $PF\text{-}NEH_{LS}(x)$ e $wPF\text{-}NEH_{LS}(x)$. O Algoritmo 2.7 mostra o procedimento da $PF\text{-}NEH_{LS}(x)$. Os procedimentos para as heurísticas $PW\text{-}NEH_{LS}(x)$ e $wPF\text{-}NEH_{LS}(x)$ são similares.

Algoritmo 2.6 *RLS*

procedimento RLS(π, π^{ref})
  enquanto π é melhorada **faça**
    para $k \leftarrow 1, \ldots, n$ **faça**
      $\pi' \leftarrow \pi$
      Encontre a tarefa π_k^{ref} em π' e a remova
      Teste π_k^{ref} em todas as posições possíveis de π'
      Insira π_k^{ref} em π' na posição que resulta no menor C_{max}
      if $C_{max}(\pi') < C_{max}(\pi)$
        $\pi \leftarrow \pi'$
  retorne π

Algoritmo 2.7 $PF\text{-}NEH_{LS}(x)$

procedimento $PF\text{-}NEH_{LS}(x)(n, m)$
  Gere uma sequência $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ de acordo com o tempo total de processamento em ordem não-decrescente
  para $l \leftarrow 1, \ldots, x$ **faça**
    Seja α_l a primeira tarefa para gerar a sequência $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ utilizando a heurística *Profile Fitting*
    $\pi^l \leftarrow (\beta_1, \beta_2, \ldots, \beta_{n-\lambda})$
    para $k \leftarrow n - \lambda + 1, \ldots, n$ **faça**
      Pegue a tarefa β_k de β e teste em todas as posições possíveis de π^l
      Insira β_k na posição que resultar no menor *makespan*
    $\pi^l \leftarrow RLS(\pi^l, \pi^l)$
  retorne $\pi \in \{\pi^1, \pi^2, \ldots, \pi^x\}$ com o menor *makespan*

2.3.2 Meta-heurísticas

O primeiro algoritmo baseado em uma meta-heurística proposto para o $F_m|block|C_{max}$ encontrado na literatura foi desenvolvido por Caraffa et al. (2001). Os autores desenvolveram um algoritmo genético (GA) que gera a população de forma aleatória e usa o operador *roulette wheel parent* na fase de seleção, *crossover* de dois pontos e operadores de mudança de turno nas fases de *crossover* e mutação, respectivamente. Grabowski e Pempera (2007) propuseram algoritmos de busca tabu (TS), sendo um deles implementando múltiplos operadores de busca local (TS + M). Algoritmos de busca tabu iniciam com uma solução inicial e procuram em sua vizinhança uma solução para minimizar a função objetivo (no caso, C_{max}). Então, a melhor solução se torna a solução inicial e o

processo continua até que um critério de parada seja atendido. Durante o processo, uma memória do histórico de pesquisa é usada para evitar ciclos, bem como para guiar a busca por regiões espaciais de solução promissora. Esta lista registra, para um determinado número de iterações, os movimentos realizados, tratando-os como sendo proibidos para movimentos futuros. A busca para quando um determinado número de iterações sem valor de melhoria é alcançado. Tanto o TS quanto o TS + M propostos aplicam as propriedades de bloqueio propostas por Grabowski e Pempera (2000). Experimentos computacionais mostraram que os métodos propostos obtiveram resultados melhores que o GA proposto por Caraffa et al. (2001).

Jarboui et al. (2009) propuseram um *estimation distribution algorithm* (EDA), que é um algoritmo evolutivo baseado em população que explora o espaço de soluções por amostragem de um modelo probabilístico atualizado de acordo com as melhores soluções promissoras. No H-EDA proposto, a população inicial é gerada pelo algoritmo NEH e método aleatório. Como critério de seleção, as soluções são classificadas e selecionadas com base no menor C_{max}. Um modelo de distribuição probabilística é desenvolvido com base na posição das tarefas na sequência. Os piores indivíduos são substituídos pelos novos indivíduos de acordo com seu valor de aptidão. Uma busca local é aplicada aos novos indivíduos, dado um valor de probabilidade de melhoria. Tal algoritmo demonstrou melhores resultados que o TS + M proposto por Grabowski e Pempera (2007).

Companys, Ribas e Mateo (2010) propuseram uma heurística de refinamento, utilizando heurísticas construtivas presentes na literatura para construirem sequências iniciais para o método, que é uma variação de um algoritmo denominado *non-exhaustive descent algorithm* (NEDA), conhecida como *soft simulated annealing* (SSA). Duan et al. (2010) propuseram um algoritmo denominado *hybrid harmony search* (HHS) que mostrou resultados melhores que os algoritmos propostos por Caraffa et al. (2001) e Grabowski e Pempera (2007). Wang et al. (2010) propuseram versões de um algoritmo do tipo *differential evolution* (DE). Como DE é, originalmente, um algoritmo estocástico, os autores o adaptaram para uma nova versão para o $F_m|block|C_{max}$, denominado *discrete differential evolution* (DDE). Ainda, uma segunda versão também foi proposta, denominada *hybrid discrete differential evolution* (HDDE) que incorpora uma fase de busca local após a operação de *crossover* com uma certa probabilidade. Os métodos propostos obtiveram resultados computacionais superiores aos obtidos por Caraffa et al. (2001) e Grabowski e Pempera (2007). Liang et al. (2011) propuseram um algoritmo denominado *dynamic multi-swarm particle swarm optimization* (DMS-PSO), cujos resultados foram melhores que os obtidos por Grabowski e Pempera (2007).

Ribas, Companys e Tort-Martorell (2011) desenvolveram duas versões de um *iterated greedy algorithm* (IG) que utilizam o algoritmo NEH como heurística construtiva para a solução inicial. Uma solução incumbente é então obtida a partir de um operador de destruição e reconstrução que remove uma certa quantidade de tarefas da solução e as reinsere na posição em que o *makespan* obtido é o menor possível. A solução se torna a nova solução corrente de acordo com um critério de aceitação. Uma fase de busca local é aplicada na solução com o intuito de melhorá-la. A diferença entre as duas versões do IG propostas reside nesta fase, pois um dos algoritmos propostos utiliza um método denominado *revolver tool* para decidir a sequência de movimentos que serão executados na busca local, de forma a evitar que o algoritmo explore a vizinhança sempre na mesma ordem. Ambos os algoritmos foram capazes de superar o HDDE proposto por Wang et al. (2010). Tais resultados foram então superados pelas três versões de um algoritmo *harmony search* (HS) propostas por Wang, Pan e Tasgetiren (2011).

Davendra et al. (2012) propuseram um algoritmo do tipo *differential evolution* denominado EDE_C com resultados melhores que os obtidos por Wang et al. (2010). Bao, Zheng e Jiang (2012) utilizaram um algoritmo *improved harmony search* (IHS) que incorpora um algoritmo genético para melhorar a população, assim como uma busca local que faz uso de um operador de inserção. Han et al. (2012) desenvolveram um algoritmo denominado *improved artificial bee colony* (IABC) que obteve melhores resultados que Caraffa et al. (2001), Grabowski e Pempera (2007) e Wang et al. (2010). Wang e Tang (2012) desenvolveram um algoritmo chamado *discrete particle swarm optimization* (DPSO) que obteve resultados melhores que o método proposto por Grabowski e Pempera (2007). Wang e Tang (2012) implementaram um *three phase algorithm* (TPA). Na primeira fase, uma lista inicial é composta por tarefas classificadas de acordo com a ordem não decrescente da soma ponderada entre os tempos médios de processamento e o desvio padrão dos tempos de processamento. Na segunda fase, a etapa de inserção do algoritmo NEH é aplicada à lista inicial gerada na fase anterior. Por fim, na terceira fase, um algoritmo do tipo *simulated annealing* (SA) modificado é usado para melhorar a solução dada pela segunda fase. O TPA foi capaz de superar os resultados das duas versões do IG propostas por Ribas, Companys e Tort-Martorell (2011).

Chowdhury et al. (2013) propuseram um algoritmo genético para resolver o problema do caixeiro viajante e o $F_m|block|C_{max}$. O algoritmo aplica uma mutação forçada às piores sequências da população. Além disso, um mecanismo multinível é usado para a fase de mutação, que é composta por mais de um operador, que são de troca e inversão. Os resultados experimentais mostraram que o método superou os métodos de Grabowski

e Pempera (2007) e Wang et al. (2010). O *discrete self organism migrating algorithm* (DSOMA) de Davendra e Bialic-Davendra (2013) também superou os trabalhos citados. Um DSOMA também foi proposto por Davendra et al. (2013). O algoritmo incorpora um *lozi map* que faz uso de um gerador de números pseudo-aleatórios. Além disso, um operador de busca local 2-opt é aplicado no melhor indivíduo.

Um algoritmo baseado nas características dos sistemas imunológicos, denominado *revised artificial immune system* (RAIS), foi desenvolvido por Lin e Ying (2013). Um método também incorpora característica da meta-heurística *simulated annealing*. Os autores compararam a performance do método proposto com outros algoritmos já conhecidos na literatura. Outro algoritmo bio-inspirado foi proposto por Han, Gong e Sun (2015), chamado *discrete artificial bee colony with differential evolution* (DE-ABC). O algoritmo faz uso de operadores distintos que incoporam características de busca local, com uma vizinhança do tipo *insertion*, além de mutação de soluções e *crossover*. Han et al. (2016) implementaram um método chamado *modified fruit fly optimisation* (MFFO) que também utiliza de operadores distintos para guiar a convergência do algoritmo e realizar busca local a partir de uma vizinhança *insertion*.

Pan et al. (2012) propuseram um *modified memetic algorithm* (MMA). Como método para gerar indivíduos para compor a população inicial, os autores utilizaram uma heurística construtiva PF + NEH. Nas fases de seleção, *crossover* e mutação, o MMA usa a seleção de torneio, técnica de *path relinking* e operadores de deslocamento, respectivamente. Ainda, é utilizado métodos de refinamento para melhoria de soluções. Os resultados experimentais mostraram que o MMA superou os métodos de Wang et al. (2010) e Ribas, Companys e Tort-Martorell (2011). Ribas, Companys e Tort-Martorell (2013) desenvolveram dois algoritmos de *variable neighborhood search* (VNS). Os dois métodos diferem apenas na fase de refinamento da solução. Na etapa de construção, os autores utilizaram as heurística PW/PW2 de Pan e Wang (2012). Como mecanismo de perturbação, a sequência é destruída removendo-se d tarefas e reinserindo-as em posições cujo *makespan* seja o menor possível. No *parallel* VNS (PVNS) proposto, uma das buscas locais é aplicada com base em uma porcentagem de seleção. Já no *series structures* (SVNS), as buscas locais são aplicadas em série. Os resultados experimentais mostraram que ambos os VNS propostos superaram o HDDE de Wang et al. (2010) e os IG de Ribas, Companys e Tort-Martorell (2011).

Ding et al. (2016) implementaram um IG com propriedades de bloqueio, propostas por eles, para eliminar movimentos que não resultarão em melhora da solução corrente. O algoritmo utiliza um algoritmo NEH modificado para construir a solução inicial, assim

como aplica as propriedades propostas. Resultados computacionais mostraram que o IG proposto superou o HDDE de Wang et al. (2010), o IG de Ribas, Companys e Tort-Martorell (2011), TPA de Wang et al. (2012), e o MMA de Pan et al. (2012). Tasgetiren et al. (2015) desenvolveram um algoritmo de busca local populacional com DE denominado DE_PLS. O método utiliza a heurística PF-NEH(x) proposta por Pan e Wang (2012), uma meta-heurística do tipo GRASP e um procedimento aleatório para construir a população inicial. O algoritmo inclui um mecanismo que aplica IG ou *iterated local search* (ILS) a um indivíduo em estudo. O novo indivíduo é submetido a uma busca local que é orientada pela melhor solução encontrada até o momento. Os resultados computacionais mostraram que o DE_PLS proposto gerou melhores resultados do que todos os algoritmos avaliados da literatura com exceção do MMA proposto por Pan et al. (2012).

Zhang et al. (2015) implementaram um *hybrid variable neighborhood search* com *simulated annealing* em que cinco estruturas de vizinhança são consideradas para serem aplicadas: *swap*, *forward insert*, *afterward insert*, *forward swap* e *inverse*. HVNSSA usa um método aleatório para gerar uma solução inicial e integra um SA que faz uso de um procedimento de construção e descontrução. Tasgetiren et al. (2017) desenvolveram diferentes versões de um IG que mostraram resultados superiores aos obtidos por Wang et al. (2010), Ribas, Companys e Tort-Martorell (2011) e Ribas, Companys e Tort-Martorell (2013). Moslehi e Khorasanian (2014) propuseram um algoritmo hibridizado baseado em *variable neighborhood descent* e *simulated annealing*. Os autores utilizaram duas vizinhanças distintas, denominadas *insert* e *edge-insert*. O critério de aceitação foi feito a partir de característica da meta-heurística SA, e um procedimento de perturbação baseado nas vizinhanças citadas e da vizinhança *swap* também foi utilizado.

Shao, Pi e Shao (2018a) desenvolveram um *estimation distribution algorithm with variable neighborhood search* (P-EDA) em que a população inicial é gerada por meio de métodos heurísticos e aleatórios baseados no algoritmo NEH. A fase de seleção usa uma classificação linear modificada para selecionar os indivíduos superiores da população. Um método probabilístico é proposto para guiar o algoritmo a um espaço de solução promissor e um método de *path relinking* é usado para aprimorar a propriedade de convergência do EDA. Para aumentar a capacidade de exploração, uma busca local é aplicada aos indivíduos. Finalmente, um esquema de manutenção da diversidade é adotado para evitar a deterioração da população. Resultados computacionais foram melhores que os de Wang et al. (2010), Ribas, Companys e Tort-Martorell (2011), Wang et al. (2012), Ribas, Companys e Tort-Martorell (2013), Pan et al. (2012) e Ding et al. (2016).

Por fim, Shao et al. (2019) propuseram um algoritmo denominado *discrete invasive*

weed optimization (DIWO) que faz uso de uma heurística eficaz e um método aleatório combinados para gerar uma população inicial com alta qualidade e diversidade. Para manter a capacidade de busca e eficiência, uma dispersão espacial baseada em inserção aleatória é apresentada por meio da distribuição normal. Além disso, uma busca local referenciada baseada em embaralhamento é incorporada para aumentar ainda mais a capacidade de exploração. Uma exclusão competitiva aprimorada é desenvolvida para determinar uma população de descendentes com boa qualidade e diversidade. Os experimentos mostraram que o DIWO produz resultados melhores do que todos os algoritmos comparados por uma margem significativa.

2.3.3 Métodos Exatos

Quando comparados com as meta-heurísticas, a quantidade de métodos exatos propostos para o $F_m|block|C_{max}$ encontrados na literatura é consideravelmente inferior. O resumo destes métodos pode ser encontrado na Tabela 2.3. Conforme mostrado na revisão sistemática de Miyata e Nagano (2019),a concentração de trabalhos que propoem métodos exatos para o $F_m|block|C_{max}$ é mais expressiva antes dos anos 2000. O primeiro destes trabalhos foi o *branch-and-bound* (B&B) proposto por Levner (1969), em que o autor considera os limites utilizados a partir das propriedades de bloqueio.

Conforme mencionado anteriormente, o $F_m|block|C_{max}$ é conhecidamente NP-Difícil. Entretanto, Reddi e Ramamoorthy (1972) mostraram que o $F_2|block|C_{max}$ pode ser reduzido a um caso especial do conhecido problema do caixeiro viajante. Os autores também mostraram que o problema pode ser resolvido em tempo polinomial a partir da aplicação do algoritmo de Gilmore e Gomory (1964).

Suhami e Mah (1981) propuseram um B&B para o problema, desenvolvendo um limite inferior a partir de estimativas da média e da variância do *makespan* de sequências parciais. Os autores mostraram exemplos numéricos, assim como resultados computacionais do B&B proposto. Ronconi (2005) também propôs um B&B para o $F_m|block|C_{max}$, desenvolvendo um limite inferior que considera uma estimativa dos tempos de saída das tarefas das máquinas a partir de um conjunto de tarefas ainda não alocadas. Um algoritmo denominado LOMPEN (*lomnicki pendular algorithm*) foi proposto por Companys e Mateo (2007). Tal algoritmo aplica um B&B para resolver instâncias diretas e inversas para o $F_m|block|C_{max}$ e $F_m|prmu|C_{max}$. Os autores desenvolveram um limite inferior similar ao proposto por Ronconi e Armentano (2001) para o $F_m|block|\sum_i T_i$. Toumi et al. (2017) também propuseram um B&B para o problema e desenvolveram limites infe-

riores baseados no algoritmo de Gilmore e Gomory (1964) e no limite inferior proposto por Ronconi (2005). Assim, os autores utilizaram estes limites limitando o número de nós que usariam cada um deles. Os resultados computacionais mostraram a superioridade do limite inferior baseado no algoritmo de Gilmore e Gomory (1964) em relação aos demais existentes na literatura.

Modelos de programação linear inteira podem ser encontrados no trabalho de Pitty e Karimi (2008). Os autores classificaram modelos matemáticos para o problema de *flow shop* e propuseram modelos para o $F_m|block|C_{max}$ e $F_m|block, st_{sd}|C_{max}$. Além disso, os autores também realizaram uma avaliação computacional dos modelos encontrados na literatura.

Por fim, dois trabalhos da literatura utilizaram *bounded dynammic programming* (BDP) para solucionar o $F_m|block|C_{max}$. O BDP é um método que combina características de programação dinâmica e *branch-and-bound*. Bautista et al. (2012) adaptaram os limites inferiores de Lageweg, Lenstra e Kan (1978) como um limite geral parcial considerando que o $F_m|prmu|C_{max}$ é uma relaxação do $F_m|block|C_{max}$. Ozolins (2019) também utilizou-se de BDP, propondo dois limites inferiores baseados no limite de Bautista et al. (2012) e no algoritmo de Gilmore e Gomory (1964). Os resultados demonstraram que o limite inferior proposto é melhor que o proposto por Ronconi (2005). O método foi capaz de resolver instâncias com até 20 tarefas na otimalidade.

Tabela 2.3: Resumo dos métodos exatos para o $F_m|block|C_{max}$

Referência	**Problema**	**Método**
Levner (1969)	$F_m\|block\|C_{max}$	B&B
Reddi e Ramamoorthy (1972)	$F_2\|block\|C_{max}$	GGA
Suhami e Mah (1981)	$F_m\|block\|C_{max}$	B&B
Ronconi (2005)	$F_m\|block\|C_{max}$	B&B
Companys e Mateo (2007)	$F_m\|block\|C_{max}$ e $F_m\|prmu\|C_{max}$	B&B
Pitty e Karimi (2008)	$F_m\|block\|C_{max}$ e $F_m\|block, st_{sd}\|C_{max}$	MILP
Toumi et al. (2017)	$F_m\|block\|C_{max}$	B&B
Bautista et al. (2012)	$F_m\|block\|C_{max}$	BDP
Ozolins (2019)	$F_m\|block\|C_{max}$	BDP

Fonte: Adaptado de Miyata e Nagano (2019)

Capítulo 3

Algoritmo Proposto

Este capítulo descreve o algoritmo a ser utilizado para a resolução do problema: um algoritmo populacional híbrido que incorpora um operador *ruin-and-recreate* (R&R), bem como um procedimento de busca local baseado em VND. Este último é composto por três tipos de estruturas de vizinhança que empregam métodos de aceleração na busca. Além disso, um critério de desempate e um mecanismo de gerenciamento da diversidade populacional são implementados para evitar que o algoritmo fique preso em ótimos locais.

O Algoritmo 3.1 apresenta o pseudocódigo do método proposto, denominado *hybrid population-based ruin-and-recreate algorithm* (HyPRR; foneticamente, "*hyper*"). O método inicia gerando uma população de tamanho μ que é formada por indivíduos (soluções) representados por permutações de tarefas, em que cada um desses indivíduos é submetido a busca local. Então, iterativamente, um indivíduo é selecionado por torneio binário e submetido a um operador R&R, em que d tarefas são removidas da permutação e reinseridas utilizando o *makespan* como critério de reinserção. Em seguida, o indivíduo gerado é submetido à busca local. A busca é composta por três estruturas de vizinhança distintas que são exploradas de forma determinística, conforme complexidade computacional de cada uma delas.

Tal procedimento é repetido até que a população atinja um tamanho igual a $\mu + \lambda$ indivíduos. Quando isto acontece, o método aplica um procedimento de seleção de sobreviventes para descartar λ indivíduos. Para realizar tal procedimento, é levado em consideração a qualidade da função objetivo dos indivíduos descartados, assim como sua contribuição para a diversidade da população. O algoritmo chega ao fim quando o critério de parada é atingido. Nas seções que seguem, cada etapa do HyPRR é descrita em detalhes.

Algoritmo 3.1 HyPRR

1: **procedimento** HyPRR($\mu, \lambda, d, \mu^{elite}, \mu^{close}$)
2: Gere uma população P de μ indivíduos utilizando a heurística $PF\text{-}NEH(v)$
3: Submeta os μ indivíduos à busca local
4: **enquanto** o critério de parada não é atingido **faça**
5: Selecione um indivíduo π de P utilizando torneio binário
6: $\pi^{'} \leftarrow$ `R&R(`π, d`)`
7: Submeta $\pi^{'}$ à busca local
8: Insira $\pi^{'}$ na população P
9: **se** $|P| = \mu + \lambda$ **então**
10: $P \leftarrow$ `GerenciamentoPopulação(`μ^{elite}, μ^{close}`)`
11: **retorne** a melhor solução da população

3.1 Geração da População Inicial

A fim de gerar a população inicial P com μ indivíduos, foi utilizada uma variação da heurística construtiva $PF\text{-}NEH(v)$ proposta por Pan e Wang (2012). O método original consiste em ordernar as tarefas em ordem não-descrescente de acordo com seus tempos totais de processamento e construir v permutações utilizando a heurística *Profile Fitting* (PF), descrita na Seção 2.3.1.2. Em cada uma dessas v permutações, a tarefa utilizada como tarefa inicial para a PF é aquela com o maior tempo total de processamento que ainda não foi testada. Por fim, um número α de tarefas é removido do final de cada uma das v permutações e depois é reinserido utilizando-se a fase de inserção do Algoritmo NEH, descrito na Seção 2.3.1.1. O método visa tirar proveito das características específicas do problema exploradas pela *Profile Fitting*.

Conforme pode ser observado no Algoritmo 3.1, a população inicial é composta por um total de μ indivíduos. Nesse sentido, é importante que os indivíduos dessa população sejam diferentes entre si. O intuito é fazer com que haja maior diversidade de soluções para que a busca local seja aplicada e evitar que o algoritmo fique preso em ótimos locais. Portanto, durante a construção da população, faz-se $v = 1$ e a tarefa inicial para construção de uma solução é escolhida de forma aleatória. Assim, para a construção de cada um dos μ indivíduos, é escolhida uma tarefa inicial diferente. Dessa maneira, a $PF\text{-}NEH(v)$ construirá soluções distintas entre si. Isto acontece porque a tarefa inicial gerará perfis diferentes de bloqueio e ociosidade, conforme explicado na Seção 2.3.1.2, e, assim, as ordens com que as tarefas serão inseridas nas soluções tendem a ser distintas.

Após a construção de cada solução, as α últimas tarefas são reinseridas em ordem aleatória na fase de inserção, enquanto que Pan e Wang (2012) realizam a reinserção na mesma ordem que as tarefas são removidas da permutação inicial. Conforme adotado por

Pan e Wang (2012), $\alpha = 25$ quando $n \geq 25$ e $\alpha = 20$, caso contrário. O método pode ser visualizado no Algoritmo 3.2.

Algoritmo 3.2 *PF-NEH* Adaptado

```
1: procedimento PF-NEH ADAPTADO(n, m)
2:     Selecione uma tarefa aleatoriamente para ser a tarefa inicial de π
3:     Construa a permutação π utilizando a heurística Profile Fitting
4:     Remova α tarefas no final de π e as insira em uma lista LC
5:     enquanto LC não estiver vazia faça
6:         Selecione uma tarefa β ∈ LC
7:         Reinsira β em π na posição que resultar no menor makespan
8:         Remova β de LC
9:     retorne π
```

3.2 Geração de Novos Indivíduos

Inicialmente, com o intuito de gerar novos indivíduos da população, um indivíduo é selecionado a partir de torneio binário. Nessa seleção, dois indivíduos são pré-selecionados e é mantido aquele que possui o melhor valor da função de aptidão, que é descrita na Seção 3.4. Após esta etapa, este indivíduo é submetido ao operador *ruin-and-recreate* (R&R), similar ao utilizado por Ruiz e Stützle (2007). Em oposição aos movimentos realizados na busca local, essa operação não possui como objetivo principal obter uma solução π' derivada de π de modo que $C_{max}(\pi') < C_{max}(\pi)$. Na realidade, o R&R é responsável por gerar uma nova solução $\pi^{'}$, a partir de π, com boa qualidade para ser submetida a busca local. Além disso, este operador também é responsável por fazer com que o método não fique preso em ótimos locais.

O R&R é mostrado no Algoritmo 3.3. As Linhas 2-3 iniciam o método removendo um total de d tarefas, escolhidas aleatoriamente, de π, formando uma permutação parcial denominada π_D. Em seguida, tais tarefas são inseridas em uma lista para posterior realocação na permutação (Linha 4). Enquanto esta lista (LC) não estiver vazia, isto é, enquanto ainda houverem tarefas a serem reinseridas na permutação π_D, o algoritmo seleciona de forma aleatória uma tarefa $j \in LC$ para reinserir na permutação (Linha 6). Na Linha 7, obtem-se a posição da tarefa j em π_D que resulta no menor *makespan* possível, excluindo a posição atual da tarefa. Esta operação pode ser feita a partir da fase de inserção do Algoritmo NEH. Nas Linhas 8 e 9, a tarefa j é removida de LC e reinserida em π_D. Conforme já mencionado, este processo é repetido até que LC se esvazie. Ao fim, retorna-se uma nova permutação $\pi^{'}$ que será submetida a uma nova busca local.

Algoritmo 3.3 Operador ruin-and-recreate

```
1: procedimento R&R(d)
2:     Selecione, de forma aleatória, d tarefas de π
3:     Remova as d tarefas de π, obtendo-se uma permutação parcial π_D
4:     Insira as d tarefas em uma lista LC
5:     enquanto LC ≠ ∅ faça
6:         Selecione, de forma aleatória, uma tarefa j ∈ LC
7:         Reinsira j em π_D na posição do menor makespan, que não seja a posição atual
8:         LC ← LC − j
9:         π_D ← π_D + j
10:    retorne π'
```

3.3 Busca Local

A fase de busca local utiliza um conjunto de estruturas de vizinhança que são exploradas de forma determinística. Quando uma vizinhança não melhora a solução corrente, o algoritmo prossegue para a próxima vizinhança. Por outro lado, quando uma vizinhança consegue obter uma melhora, o algoritmo retorna para a primeira vizinhança ao finalizar a exploração da vizinhança corrente. O procedimento chega ao fim quando todas as vizinhanças são exploradas e nenhuma melhora é obtida. Um total de 3 estruturas de vizinhança foram utilizadas e são descritas a seguir.

Insertion: Uma tarefa é retirada da sua posição na solução π e reinserida em uma nova posição.

Block Insertion: Um bloco de x tarefas consecutivas é removido de sua posição na solução π e reinserido em uma nova posição. Uma vizinhança distinta é associada a cada tamanho de bloco. Por exemplo, se $2 \leq x \leq 4$, então existem 3 vizinhanças com tamanhos de bloco 2, 3 e 4, respectivamente. Observe que a vizinhança *Insertion* é um caso particular quando $x = 1$.

Swap: Duas tarefas da solução π são trocadas de posição entre si.

A ordem em que as vizinhanças são exploradas é a mesma em que estão descritas. O critério para a adoção dessa ordem é o tempo computacional que se leva para explorar cada uma das vizinhanças. Wang et al. (2010) propuseram um método que reduz a complexidade computacional de explorar a vizinhança *insertion* de $O(n^3m)$ para $O(n^2m)$. Com o intuito de utilizar vizinhanças do tipo *block insertion*, esse método foi adaptado para considerar um bloco de x tarefas a ser movido em uma permutação π, reduzindo a complexidade computacional da estrutura de vizinhança de $O(n^3m)$ para $O(n^2m)$.

Ainda, Tasgetiren et al. (2017) utilizaram um método de aceleração para a vizinhança

do tipo *swap* que é capaz de reduzir o tempo computacional de exploração da vizinhança em até 50%, mas ainda com uma complexidade computacional de $O(n^3m)$. Neste trabalho, um método de aceleração que faz uso da propriedade de reversibilidade é proposto para a vizinhança *swap* juntamente com limites inferiores que auxiliam na aceleração da busca.

Dessa maneira, a ordem de exploração das vizinhanças é justificada a partir da complexidade computacional de cada uma das estruturas adotadas. Assim, o número de chamadas de vizinhanças que possuem um custo computacional maior é diminuído. A Figura 3.1 ilustra os movimentos executados pelas vizinhanças.

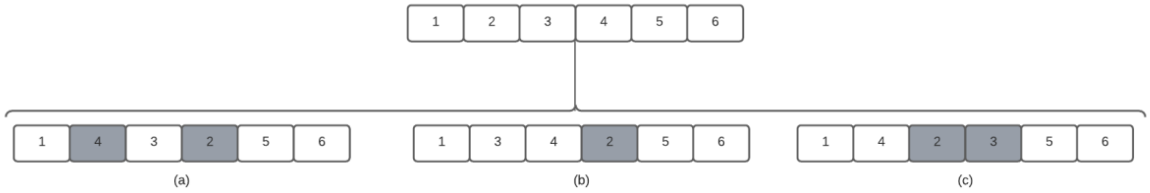

Figura 3.1: Exemplos das vizinhanças: (a) *Swap*, (b) *Insertion*, (c) *Block Insertion*

3.3.1 Critério de Aceitação

A melhora da solução foi adotado como critério de aceitação, ou seja , toda vez que a busca local encontra uma solução melhor, esta solução é aceita. Todavia, foi implementada a política de *first improvement*, isto é, ao encontrar uma solução melhor, aplica-se o movimento, e a busca continua a partir da nova solução obtida.

Além disso, caso a solução avaliada possua o mesmo *makespan* que a solução corrente, aplica-se um critério de desempate a partir de um objetivo secundário. Tal objetivo é dado pelo somatório dos tempos ociosos e de bloqueio em decorrência de cada tarefa π_k. O tempo desperdiçado em uma máquina i com bloqueio e ociosidade devido a uma tarefa π_k é dado por $D_{i,\pi_k} - D_{i,\pi_{k-1}} - p_{i,\pi_k}$. Dessa forma, o tempo total T associado a todas tarefas é dado por:

$$T = \sum_{i=1}^{m} \sum_{k=1}^{n} D_{i,\pi_k} - D_{i,\pi_{k-1}} - p_{i,\pi_k} \tag{3.1}$$

Portanto, em situações em que duas soluções possuem o mesmo *makespan*, é aceita aquela com o menor valor de T. Note que, na Equação (3.1), $D_{i,\pi_0} = 0, 1 \leq i \leq m$.

3.3.2 Métodos de Aceleração

Observa-se que a complexidade computacional para se explorar cada uma das vizinhanças é de $O(n^2)$. Dada a forma de cálculo do *makespan* explicado na Seção 2.2.1, tem-se que a complexidade computacional desse cálculo é de $O(nm)$. Portanto, para realizar a exploração de cada uma das vizinhanças tem-se uma complexidade computacional de $O(n^3m)$. Dessa forma, faz-se necessário procurar formas de acelerar a exploração de cada uma dessas vizinhanças. A seguir, serão mostrados os métodos implementados para acelerar cada uma delas, que são compostas por avaliações eficientes dos movimentos e limitantes inferiores utilizados para evitar as avaliações, quando possível.

3.3.2.1 Insertion

De forma a acelerar a exploração da vizinhança de inserção, Wang et al. (2010) propuseram um método fazendo uso da propriedade de reversibilidade apresentado na Seção 2.2.3. Assim, a complexidade computacional de se explorar esta vizinhança pode ser reduzida de $O(n^3m)$ para $O(n^2m)$. O método proposto consiste nos seguintes passos:

1. Calcule os tempos de saída D_{i,π_k}, para cada tarefa k e máquina i.
2. Calcule os tempos de saída R_{i,π_k^r}, para cada tarefa k e máquina i.
3. Faça $j = 1$.
4. Faça $\bar{\pi} = (\bar{\pi}_1, \bar{\pi}_2, \ldots, \bar{\pi}_{n-1})$ uma permutação parcial após a remoção de π_j. Gere uma permutação $\bar{\pi}^r$ depois de remover π^r_{n-j+1} de π^r.
5. Calcule $D_{i,\bar{\pi}_k}, k = 1, \ldots, n-1, i = 1, \ldots, m$. Note que se $k < j$, $D_{i,\bar{\pi}_k} = D_{\pi_k,i}$.
6. Calcule $R_{i,\bar{\pi}_k^r}$, para $k = 1, \ldots, n-1$ e $i = 1, \ldots, m$. Note que se $k < j$, $R_{i,\bar{\pi}_k^r} = R_{i,\pi_k^r}$.
7. Repita os seguintes passos até todas as $q \in \{j+1, \ldots, n\}$ possíveis posições serem avaliadas:
   (a) Insira a tarefa π_j na posição q e gere uma permutação π'.
   (b) Calcule D_{i,π'_q} para $i = 1, \ldots, m$. Note que se $k < q$, $D_{i,\pi'_k} = D_{i,\bar{\pi}_k}$.
   (c) O *makespan* de π' é dado por:

$$C_{max} = \max_{i=1}^{m}(D_{i,\pi'_q} + R_{i,\bar{\pi}^r_{n-q}}) \tag{3.2}$$

8. Faça $j = j + 1$. Se $j > n$, pare. Caso contrário, volte para o Passo 4.

Observe que há um total de n iterações para os Passos de 5 a 7, que podem ser executados em $O(nm)$. Assim, para explorar toda vizinhança de inserção, a complexidade computacional é de $O(n^2m)$. Além disso, observe que o Passo 7(c) não precisa ser executado completamente. Caso, em uma determinada iteração, o valor supere ou seja igual ao *makespan* corrente, não há a necessidade de avaliar as demais iterações.

A Figura 3.2 ilustra a avaliação de um movimento de inserção. Nela, a tarefa J_5 foi retirada da permutação para ser movida, resultando na permutação parcial $\bar{\pi}$ (Passo 4). Em seguida, os tempos de saída parciais são calculados em (a) e (b). Por fim, é avaliado o movimento em que J_5 é reinserido em $q = 2$. Note que o *makespan* é dado por $C_{max} = max(D_{1,\pi'_2} + R_{2,\bar{\pi}^r_2}, D_{2,\pi'_2} + R_{1,\bar{\pi}^r_2})$.

Neste trabalho, além de aplicar o procedimento de Wang et al. (2010) na vizinhança de inserção, também foram implementadas adaptações deste procedimento para as vizinhanças de *swap* e *block insertion*, também reduzindo o tempo computacional destas estruturas. Por fim, também foram aplicadas as propriedades de bloqueio descritas na Seção 2.2.4.

3.3.2.2 Block Insertion

O método de aceleração proposto por Wang et al. (2010) descrito anteriormente foi elaborado para a vizinhança de inserção para uma única tarefa. Entretanto, durante os experimentos realizados, observou-se que o método pode ser adaptado para um valor x de tarefas, sendo $x > 1$. Dessa forma, o método proposto consiste nos seguintes passos, para duas posições j e q, com $q > j$, para a vizinhança *block insertion*:

1. Calcule os tempos de saída D_{i,π_k}, para cada tarefa k e máquina i.
2. Calcule os tempos de saída R_{i,π^r_k}, para cada tarefa k e máquina i.
3. Faça $j = 1$.
4. Seja $\bar{\pi} = (\bar{\pi}_1, \bar{\pi}_2, \ldots, \bar{\pi}_{n-x})$ uma permutação parcial após a remoção do bloco $b = (\pi_j, \pi_{j+1}, \ldots, \pi_{j+x-1})$. Gere uma permutação $\bar{\pi}^r$ depois de remover b^r de π^r.
5. Calcule $D_{i,\bar{\pi}_k}$, para $k = 1, \ldots, n-x$ e $i = 1, \ldots, m$. Note que se $k < j$, $D_{i,\bar{\pi}_k} = D_{i,\pi_k}$.
6. Calcule $R_{i,\bar{\pi}^r_k}$, para $k = 1, \ldots, n-x$ e $i = 1, \ldots, m$. Note que se $k < j$, $R_{i,\bar{\pi}^r_k} = R_{i,\pi^r_k}$.

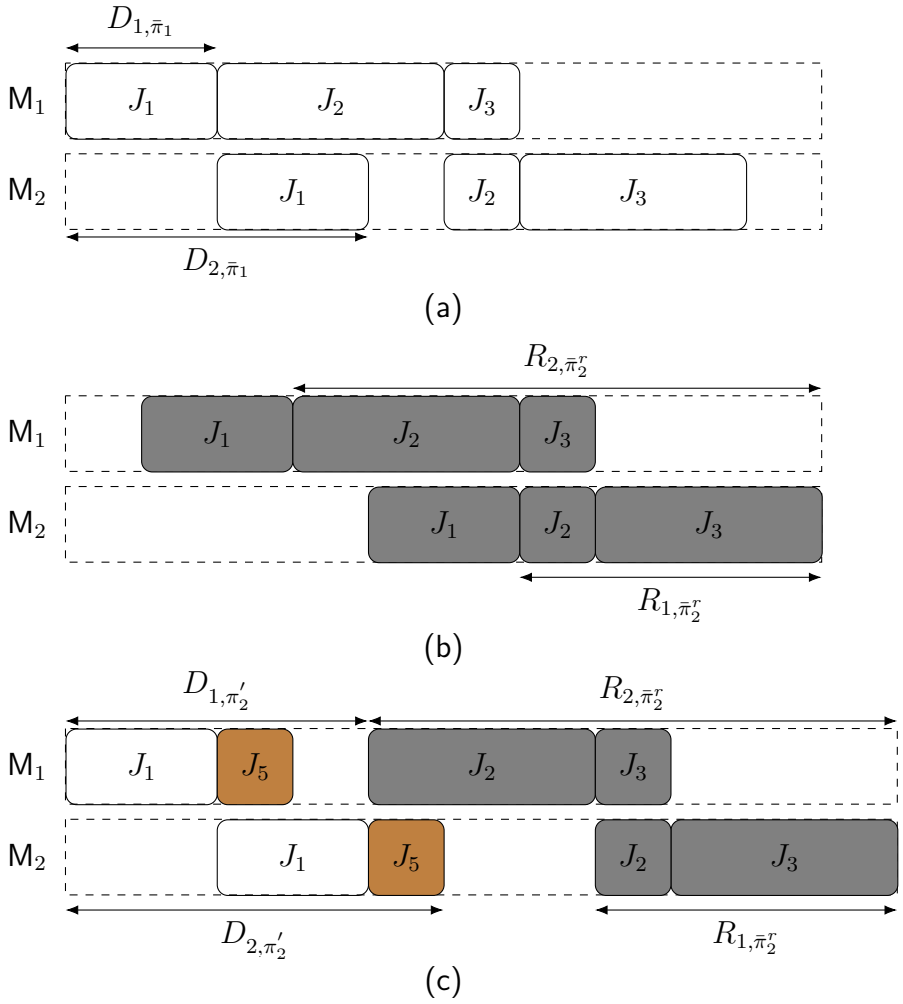

Figura 3.2: Ilustração de uma avaliação de um movimento de inserção com as seguintes etapas: cálculo dos tempo de saída direto (a) e reverso (b) ao retirar a tarefa a ser movida, e reinserção da tarefa movida (c)

7. Repita os seguintes passos até todas as $q \in \{j+x, \ldots, n\}$ possíveis posições serem avaliadas:

   (a) Insira o bloco b na posição q e gere uma permutação π'.

   (b) Calcule $D_{i,\pi_k'}$ para $q \leq k < q+x$, $i = 1, \ldots, m$, que corresponde aos tempos de saíde das tarefas do bloco b. Note que se $k < q$, $D_{i,\pi_k'} = D_{i,\bar{\pi}_k}$.

   (c) O *makespan* de π' é dado por:

$$C_{max} = \max_{i=1}^{m}(D_{i,\pi'_{q+x-1}} + R_{i,\bar{\pi}^r_{n-q-x+1}}) \tag{3.3}$$

8. Faça $j = j+1$. Se $j > n-x+1$, pare. Caso contrário, volte para o passo 4.

A Figura 3.3 ilustra um movimento do tipo *block insertion* ao retirar um bloco $b = \{J_5, J_6\}$ e avaliá-lo ao reinserir na posição 2 da permutação parcial $\bar{\pi}$. Note que os tempos

D_{1,π'_3} e D_{2,π'_3} são calculados a partir de $D_{1,\bar{\pi}_1}$ e, em seguida, o *makespan* é dado por $C_{max} = max(D_{1,\pi'_3} + R_{2,\bar{\pi}^r_2}, D_{2,\pi'_3} + R_{1,\bar{\pi}^r_2})$.

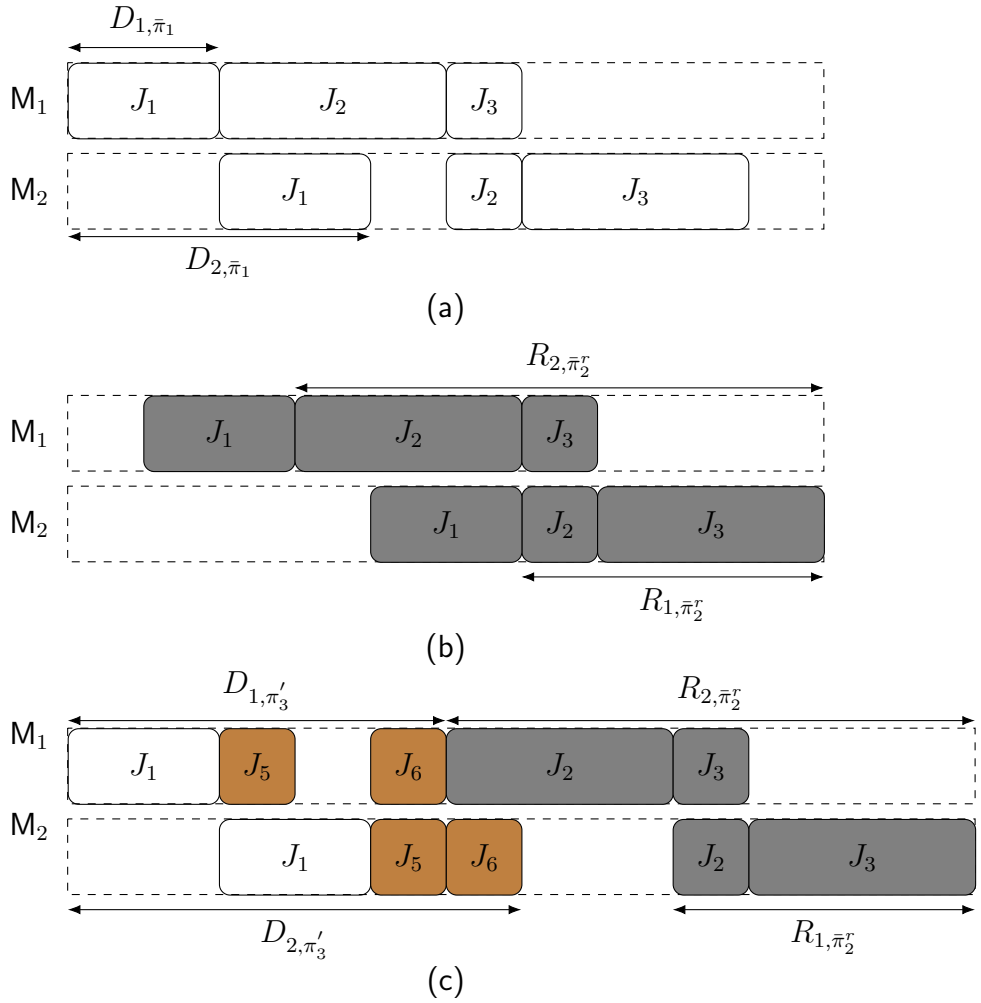

Figura 3.3: Ilustração de uma avaliação de um movimento de inserção com as seguintes etapas: cálculo dos tempo de saída direto (a) e reverso (b) ao retirar o bloco a ser movido, e reinserção do bloco (c)

3.3.2.3 Swap

3.3.2.3.1 Avaliação do movimento

Conforme descrito anteriormente, a vizinhança *swap* consiste em trocar de posições duas tarefas de uma permutação π e avaliar o *makespan* obtido. Nesse sentido, como o cálculo do *makespan* de uma permutação é feito em $O(nm)$ e o tamanho da vizinhança *swap* é $O(n^2)$, a exploração de toda a estrutura é feita em $O(n^3m)$.

Contudo, observe que, ao trocar duas tarefas de posição em uma permutação π, não há a necessidade de se calcular o *makespan* desta nova permutação do início. Dada uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_5, \ldots, \pi_9, \ldots, \pi_n)$, ao trocar as tarefas π_5 e π_9 de posição,

bastaria aplicar as Equações (2.1) - (2.3) a partir de $k = 5$, que é a posição da menor tarefa que está sendo trocada de posição. Sendo assim, o calculo do *makespan* desta nova permutação pode ser feito em $O((n - y + 1)m)$ sendo y igual a menor posição entre as tarefas que estão sendo trocadas. No pior caso, quando a primeira e a última tarefa são trocadas de posição, todos tempos devem ser calculados e a complexidade é de $O(nm)$. Tasgetiren et al. (2017) fez uso desse método juntamente de uma probabilidade de utilização da vizinhança *swap* devido ao seu alto custo computacional.

Apesar do método anterior reduzir o tempo computacional de exploração da vizinhança *swap*, é possível adaptar o método proposto por Wang et al. (2010), originalmente para a vizinhança de inserção, para a vizinhança *swap*. O método consiste nos seguintes passos:

1. Calcule os tempos de saída D_{i,π_k}, para cada tarefa k e máquina i.
2. Calcule os tempos de saída R_{i,π_k^r}, para cada tarefa k e máquina i.
3. Sejam j e l as posições que serão trocadas em uma permutação π gerando $\pi^{'}$, faça:
   (a) Calcule os novos tempos de saída $D_{i,\pi_k^{'}}$, para $j \leq k \leq l$, $i \leq m$.
   (b) O *makespan* de $\pi^{'}$ é dado por:

$$C_{max} = \max_{i=1}^{m}(D_{i,\pi_l^{'}} + R_{i,\pi_{n-l}^r}) \tag{3.4}$$

4. Se todos os possíveis pares $(j, l), l > j$ já tiverem sido avaliados, pare. Caso contrário, prossiga para o próximo par.

O método proposto realiza menos operações do que o descrito anteriormente, uma vez que não há a necessidade de calcular todos os tempos de saída das tarefas das máquinas a partir de j, mas apenas os tempos até a tarefa l. Além disso, note que os itens (a) e (b) do Passo 3 não necessitam ser executados completamente. No primeiro caso, se algum valor $D_{i,\pi_k^{'}}$, para $j \leq k \leq l$, $i \leq m$, for maior ou igual ao *makespan* corrente, o método passa para o próximo par (j, l). Já no segundo caso, o método prosseguirá para o próximo par caso o valor resultante de uma iteração seja maior ou igual ao *makespan* corrente.

A Figura 3.4 ilustra a avaliação de movimento do tipo *swap*. Nela, as tarefas J_1 e J_2 são trocadas de posição. Para avaliar o *makespan* desta troca, basta calcular os novos tempos de saída das tarefas que se encontram entre as tarefas trocadas, incluindo as próprias. Neste caso, como as tarefas envolvidas são as duas primeiras tarefas da permutação, basta

calcular os novos tempos de saída até a posição 2. Em seguida, aplica-se a propriedade de reversibilidade e o *makespan* é dado por $C_{max} = (D_{1,\pi_2'} + R_{2,\pi_2^r}, D_{2,\pi_2'} + R_{1,\pi_2^r})$.

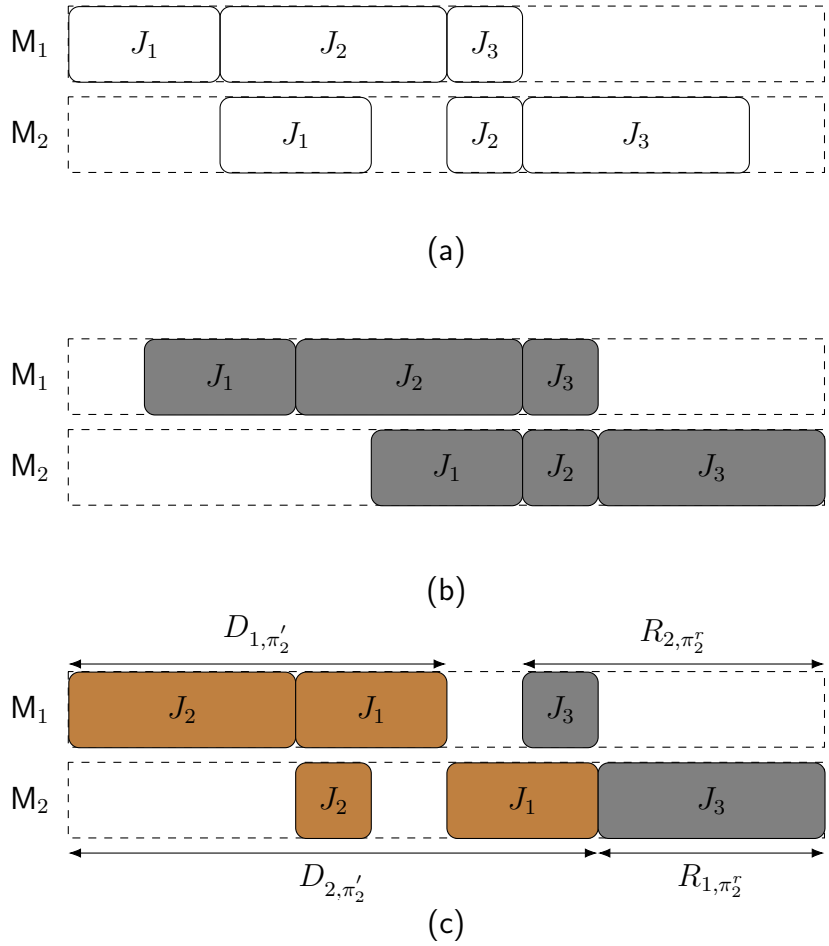

Figura 3.4: Ilustração de uma avaliação de um movimento de troca com as seguintes etapas: cálculo dos tempo de saída direto (a) e reverso (b), e troca de duas tarefas (c)

3.3.2.3.2 Limitantes Inferiores

Além do método de aceleração, estão sendo propostos limitantes inferiores para os valores do *makespan* que podem ser obtidos a partir de um movimento de troca. Tais limitantes são similares aos propostos por Ding et al. (2016) para a vizinhança de inserção. Desta forma, eles se baseiam na representação em grafo (Seção 2.2.2) e nas definições de *normal block* (GH), *anti-block* (ST) e *T-sequence* (T) (Seção 2.2.4) e são utilizados para evitar a avaliação de movimentos que não produziriam uma solução melhor do que a corrente. Diante disso, calcula-se um fator δ_{jl} ao considerar duas tarefas j e l para troca. Dado um caminho crítico no grafo $G(\pi)$, uma variação (δ_{jl}) no comprimento total do caminho é obtida ao trocar os nós associados a j e l. Essa operação resulta em um novo caminho que não é necessariamente crítico e fornece um limite inferior válido no makespan

obtido pela troca. Assim, somente vale a pena avaliar o movimento quando $\delta_{jl} < 0$. Dessa maneira, os limitantes inferiores são definidos a seguir.

Limitante Inferior 1 — GH: Seja π uma permutação para o $F_m|block|C_{max}$. Seja $\pi^{'}$ uma permutação obtida por um movimento de troca entre duas tarefas j e l, $j < l$ em que j pertence a um *normal block* t e l a um tipo de bloco q. Então, $C_{max}(\pi^{'}) \geq C_{max}(\pi)+\delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{s_t,\pi_l} - p_{s_t,\pi_j} + p_{s_q,\pi_j} - p_{s_q,\pi_l}, & q \in GH \\ p_{s_t,\pi_l} - p_{s_t,\pi_j}, & q \in ST \\ p_{s_t,\pi_l} - p_{s_t,\pi_j} + \sum_{i=s_q}^{e_q} p_{i,\pi_j} - p_{i,\pi_l} & q \in T \end{cases} \tag{3.5}$$

Demonstração. A ideia consiste em encontrar um caminho no grafo $G(\pi^{'})$ que não é, necessariamente, o caminho crítico. Dessa forma, o comprimento desse caminho fornece um limite inferior para a permutação $\pi^{'}$. Conforme Ding et al. (2016), o comprimento do caminho crítico associado a π é dado pela Equação (3.6), onde n_i é um nó pertencente ao caminho crítico, V_{GH} é o conjunto de nós que compõem os *normal-blocks* do caminho, V_T o conjunto de nós que compõem as *T-sequences* e p_{n_i} o peso relacionado ao nó n_i. Note que os nós pertencentes a *anti-blocks* (ST) não influenciam no cálculo do comprimento do caminho crítico.

$$C_{max}(\pi) = \sum_{n_i \in V_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i} \tag{3.6}$$

No caso de uma permutação π' obtida a partir da troca de duas tarefas j, que está em um bloco $t \in GH$, e l, que está em um bloco $q \in GH$, o *makespan* associado a $\pi^{'}$ é dado pela Equação 3.7. Perceba que a diferença entre as Equações (3.6) e (3.7) ocorre em relação aos conjuntos de nós referentes aos *normal-blocks*, em que $V^{'}_{GH}$ é derivado de V_{GH} conforme a Equação (3.8).

$$C_{max}(\pi') \geq \sum_{n_i \in V^{'}_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i} \tag{3.7}$$

$$V^{'}_{GH} = V_{GH} + \{(s_t, \pi_l)\} + \{(s_q, \pi_j)\} - \{(s_t, \pi_j)\} - \{(s_q, \pi_l)\} \tag{3.8}$$

Como os nós associados às *T-sequences* não são alterados, o comprimento do caminho

associado a $\pi^{'}$ é dado pelas Equações (3.10) e (3.9). Os demais casos para $t \in GH$ podem ser demonstrados de forma similiar e, portanto, são omitidos.

$$C_{max}(\pi') \geq \underbrace{\sum_{n_i \in V_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i}}_{C_{max}(\pi)} + \{(s_t, \pi_l)\} + \{(s_q, \pi_j)\} - \{(s_t, \pi_j)\} - \{(s_q, \pi_l)\} \quad (3.9)$$

$$C_{max}(\pi') \geq C_{max}(\pi) + \underbrace{p_{s_t,\pi_l} - p_{s_t,\pi_j} + p_{s_q,\pi_j} - p_{s_q,\pi_l}}_{\delta_{jl}(\pi)} \quad (3.10)$$

□

Limitante Inferior 2 — *ST*: Seja π uma permutação para o $F_m|block|C_{max}$. Seja $\pi^{'}$ uma permutação obtida por um movimento de troca entre duas tarefas j e l, $j < l$ em que j pertence a um *anti-block* t e l a um tipo de bloco q. Então, $C_{max}(\pi^{'}) \geq C_{max}(\pi) + \delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{s_q,\pi_j} - p_{s_q,\pi_l}, & q \in GH \\ 0, & q \in ST \\ \sum_{i=s_q}^{e_q} p_{i,\pi_j} - p_{i,\pi_l} & q \in T \end{cases} \quad (3.11)$$

Demonstração. O caso em que ambas as tarefas pertencem a *anti-blocks* é trivial. Como estes nós não influenciam no valor final do comprimento do caminho crítico, conforme a Equação (3.6), $C_{max}(\pi^{'}) \geq C_{max}(\pi)$. Tal resultado implica que a troca de tarefas que pertencentes a *anti-blocks* não resultará em uma melhora do *makespan* corrente. Neste caso, uma demonstração do limitante inferior para o caso em que $t \in ST$ e $q \in GH$ é realizada a seguir.

Da mesma forma em que $t \in GH$ e $q \in GH$, a única alteração a ser feita na Equação (3.6) será no conjunto V_{GH}, uma vez que os nós associados a *anti-blocks* não influenciam no comprimento do caminho crítico. Tal alteração consiste em realizar a retirada do nó $\{(\pi_l, s_q)\}$ e acrescentar o nó $\{(\pi_j, s_q)\}$, resultando na Equação (3.12).

$$V_{GH}^{'} = V_{GH} + \{(s_q, \pi_j)\} - \{(s_q, \pi_l)\} \quad (3.12)$$

A troca realizada é válida, uma vez que desejamos retirar o nó associado a j do *anti-block*

t e colocá-lo no *normal-block* q, fazendo o oposto com o nó associado a tarefa l. Dessa forma, o nó associado a t passa a ter um peso no comprimento do caminho dado por $p_{\pi_j s_q}$. Como os nós associados às *T-sequences* não são alterados, o comprimento associado a $\pi^{'}$ é dado pelas Equações (3.13) e (3.14). O caso em que $q \in T$ pode ser demonstrado de forma similar e, portanto, será omitido.

$$C_{max}(\pi') \geq \underbrace{\sum_{n_i \in V_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i}}_{C_{max}(\pi)} + \{(s_q, \pi_j)\} - \{(s_q, \pi_l)\} \tag{3.13}$$

$$C_{max}(\pi') \geq C_{max}(\pi) + \underbrace{p_{s_q,\pi_j} - p_{s_q,\pi_l}}_{\delta_{jl}(\pi)} \tag{3.14}$$

□

Limitante Inferior 3 — T: Seja π uma permutação para o $F_m|block|C_{max}$. Seja $\pi^{'}$ uma permutação obtida por um movimento de troca entre duas tarefas j e l, $j < l$ em que j pertence a uma $T - sequence$ t e l a um tipo de bloco q. Então, $C_{max}(\pi^{'}) \geq C_{max}(\pi) + \delta_{jl}(\pi)$, onde:

$$\delta_{jl}(\pi) = \begin{cases} p_{s_q,\pi_l} - p_{s_q,\pi_j} + \sum_{i=s_t}^{e_t} p_{i,\pi_l} - p_{i,\pi_j}, & q \in GH \\ \sum_{i=s_t}^{e_t} p_{i,\pi_l} - p_{i,\pi_j}, & q \in ST \\ \sum_{i=s_t}^{e_t} p_{i,\pi_l} - p_{i,\pi_j} + \sum_{i=s_q}^{e_q} p_{i,\pi_j} - p_{i,\pi_l} & q \in T \end{cases} \tag{3.15}$$

Demonstração. Considere o caso mais simples, em que $q \in ST$. Diferente dos casos demonstrados anteriormente, a alteração na Equação (3.6) acontece no conjunto V_T. Neste caso, os nós associados a j deixarão de formar uma $T - sequence$ para fazer parte de um $anti - block$, e o contrário ocorrerá com o nó associado a l. A alteração no conjunto V_T é dado pela Equação (3.16).

$$V_T^{'} = V_T + \sum_{i=s_t}^{e_t} \{(i, \pi_l)\} - \sum_{i=s_t}^{e_t} \{(i, \pi_j)\} \tag{3.16}$$

Por fim, o comprimento do caminho associado a $\pi^{'}$ é dado pelas Equações (3.17) e (3.18).

$$C_{max}(\pi') \geq \underbrace{\sum_{n_i \in V_{GH}} p_{n_i} + \sum_{n_i \in V_T} p_{n_i}}_{C_{max}(\pi)} + \sum_{i=s_t}^{e_t} p_{i,\pi_l} - p_{i,\pi_j} \tag{3.17}$$

$$C_{max}(\pi') \geq C_{max}(\pi) + \underbrace{\sum_{i=s_t}^{e_t} p_{i,\pi_l} - p_{i,\pi_j}}_{\delta_{jl}(\pi)} \tag{3.18}$$

□

Com o intuito de elucidar os limitantes propostos, um exemplo numérico é descrito a seguir. Considere a instância e a permutação $\pi = (\pi_1, \pi_2, \pi_3, \pi_4, \pi_5) = (J_4, J_1, J_5, J_3, J_2)$ apresentados na Seção 2.2.2. A Figura 3.5 ilustra a representação dessa solução em um gráfico de Gantt para o $F_m|block|C_{max}$. Observe que as áreas em branco representam tempos de bloqueio ou ociosidade das máquinas.

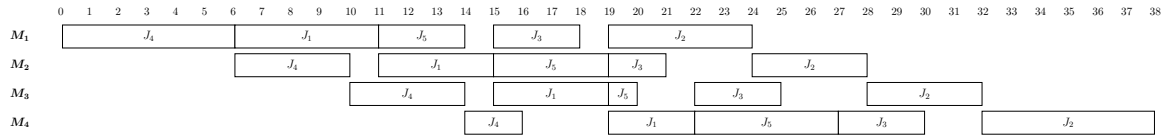

Figura 3.5: Representação da solução π em um gráfico de Gantt

Além da representação pelo gráfico de Gantt, a solução π pode ser representada a partir de um grafo $G(\pi)$, conforme mencionado anteriormente. O caminho crítico associado a esse grafo corresponderá ao *makespan* associado à solução e pode ser visualizado na Seção 2.2.2. A partir deste caminho é possível derivar limitantes inferiores para a vizinhança *swap* realizando a troca dos nós associados às tarefas que estão sendo trocadas. Assim, caso as tarefas J_4 e J_2 fossem trocadas de posição, bastaria trocar os nós envolvidos com as duas tarefas e manter os demais. O exemplo é mostrado na Figura 3.6. Nota-se que o caso em questão consiste em trocar duas tarefas que pertencem a um *normal block* e a uma *T-sequence*. O limite inferior (LI) é dado por $LI = 38 + \delta$, onde δ corresponde a variação no *makespan* que ocorre devido a troca dos nós. Neste caso $\delta = 5 + (6 + 4 + 4 + 2) - 6 - (5 + 4 + 4 + 6) = -4$ e, portanto, $LI = 38 - 4 = 34$. Tal limite é válido, pois o caminho gerado não é necessariamente crítico e, portanto, o *makespan* da solução gerada pela troca dos dois nós é maior ou igual a 34.

Dados os limitantes apresentados, nota-se que um movimento de troca entre duas tarefas j e l só poderá resultar em uma permutação π' com $C_{max}(\pi') < C_{max}(\pi)$ se, e somente se, $\delta_{jl}(\pi) < 0$. Ainda, o cálculo de $\delta_{jl}(\pi)$ pode ser feito em $O(1)$, o que

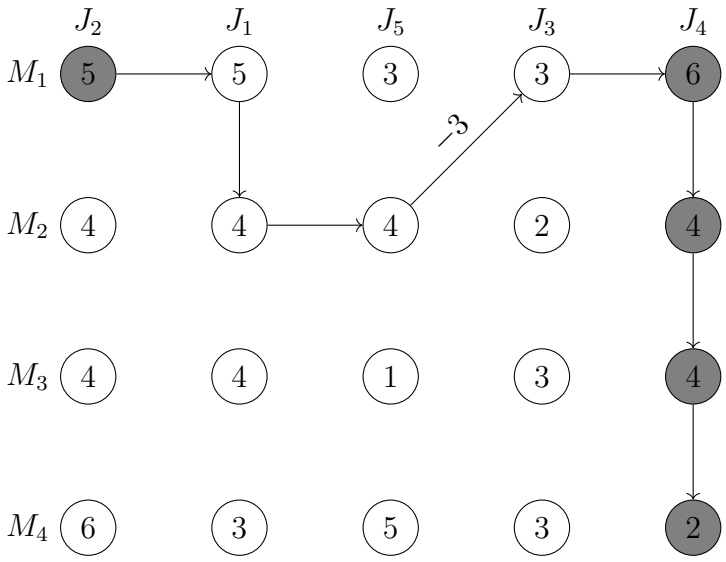

Figura 3.6: Simulação de troca das tarefas J_4 e J_2 de π

proporciona uma redução do tempo computacional ao se explorar a vizinhança *swap* em sua totalidade. Como a utilização de cada um dos limitantes depende exclusivamente do tipo de bloco (GH, ST ou T) em que cada tarefa se encontra, é importante mapeá-los por meio da obtenção do caminho crítico para a permutação, que pode ser feito a partir do método descrito por Ding et al. (2016) apresentado na Seção 2.2.2.

3.4 Gerenciamento de Diversidade

O gerenciamento de diversidade da população incorporado no HyPRR é similar ao empregado nos trabalhos de Vidal et al. (2012) e Vidal et al. (2014), para variantes do problema de roteamento de veículos, e no trabalho de Mecler, Subramanian e Vidal (2021) para o *job sequencing and tool switching problem*.

Tal gerenciamento consiste na utilização de uma função de aptidão que leva em consideração o valor da função objetivo (*makespan*) e a diversidade dos indivíduos presentes na população. Dessa maneira, o cálculo da função de aptidão $f(\pi)$ de cada indivíduo π da população P depende de dois parâmetros: o número μ^{close} de indivíduos mais próximos na população considerada na medida de distância, e o número de indivíduos de elite μ^{elite} que se deseja preservar. Os indivíduos são mantidos ordenados em termos de qualidade (considerando objetivos primários e de desempate) e diversidade.

A fim de se obter a contribuição de diversidade de cada indivíduo, o algoritmo calcula sua distância média para seus indivíduos mais próximos, sendo a distância entre dois indivíduos, que são representados como permutações de tarefas, definida como o número

de arestas diferentes entre eles, isto é, ao comparar duas permutações, a distância entre elas é incrementada em 1 sempre que um elemento possui um vizinho diferente, seja à direta ou à esquerda. Em seguida, a população é ordenada em ordem decrescente de contribuição da diversidade, e em ordem crescente de função objetivo, associando a cada indivíduo um *rank* de diversidade $f_{div}(\pi)$ e um *rank* de qualidade $f_{obj}(\pi)$.

O ranqueamento é feito da seguinte maneira: uma vez que a população é ordenada em ordem decrescente de diversidade, o valor de $f_{div}(\pi)$ é dado pelo quociente entre a posição da permutação na população e o número de indivíduos, por outro lado, uma vez que a população é ordenada em ordem crescente de qualidade, o valor de $f_{obj}(\pi)$ é dado pelo quociente entre a posição da permutação e o número de indivíduos da população. Dessa maneira, a função de aptidão de cada indivíduo pode ser calculada conforme a Equação (3.19).

$$f(\pi) = f_{obj}(\pi) + (1 - \frac{\mu^{elite}}{|P|}) \times f_{div}(\pi) \tag{3.19}$$

Nota-se que pequenos valores de aptidão correspondem a indivíduos promissores, com um pequeno valor da função objetivo e uma grande contribuição para a diversidade da população. Esta medida de aptidão é usada ao selecionar os indivíduos por torneio binário e ao selecionar indivíduos para excluir da população durante as seleções de sobreviventes. No último caso, o algoritmo exclui iterativamente o pior indivíduo (ou seja, com maior aptidão) tendo um clone sempre que existirem soluções duplicadas na população, ou o pior indivíduo caso contrário. Este processo é repetido até que o tamanho desejado da população (μ) seja atingido. Conforme pode ser observado no trabalho de Vidal et al. (2012), esse procedimento de seleção de sobreviventes preserva a diversidade e, ao mesmo tempo, garante que os melhores indivíduos (μ^{elite}) da população permaneçam preservados.

Capítulo 4

Experimentos Computacionais

Os experimentos computacionais foram realizados nas instâncias propostas por Taillard (1993). Tais instâncias foram inicialmente desenvolvidas para o $F_m|prmu|C_{max}$, mas podem ser utilizadas para o $F_m|block|C_{max}$ sem qualquer alteração, uma vez que os problemas possuem como informações o número de tarefas (n), o número de máquinas (m) e o tempo de processamento de cada tarefa k em cada uma das máquinas i ($p_{i,k}$). Além disso, as instâncias possuem três diferentes níveis de máquinas, com $m = \{5, 10, 20\}$, e cinco diferentes níveis de tarefas, com $n = \{20, 50, 100, 200, 500\}$. Em particular, os problemas com 200 tarefas possuem apenas dois níveis de máquina, 10 e 20, enquanto que os problemas com 500 tarefas possuem apenas um nível de máquina, 20. Como cada combinação possível de número de tarefas com número de máquinas possui um total de 10 instâncias distintas, o total de problemas é igual a 120. Por fim, os tempos de processamento foram gerados de forma aleatória por Taillard (1993) a partir de uma distribuição uniforme, com os tempos variando entre 1 e 99.

O algoritmo proposto foi implementado na linguagem de programação C++ e os experimentos foram realizados em um Intel Xeon E5-2650 v4 2.20GHz, 128 GB de memória RAM e sistema operacional Linux Ubuntu 16.04 e o critério de parada (CP) adotado foi o tempo de execução dado por $CP = 100 \times n \times m$ milissegundos, assim como feito por Shao et al. (2019) e Tasgetiren et al. (2017). Experimentos computacionais foram realizados para averiguar a qualidade do método proposto e seu desempenho computacional quando comparado com outros algoritmos da literatura. Em termos de qualidade da solução, foi utilizado o *Relative Percent Deviation* (RPD), conforme aplicado por Shao et al. (2019), Tasgetiren et al. (2017) e outros trabalhos da literatura. O cálculo do RPD é realizado conforme a Equação 4.1, em que C_{HyPRR} é o *makespan* obtido pelo método proposto e C_{min} é a melhor *makespan* reportado por Ribas, Companys e Tort-Martorell (2011).

A escolha pelo trabalho de Ribas, Companys e Tort-Martorell (2011) foi feita porque é uma prática comum na literatura, uma vez que os referidos autores reportam todos os limitantes superiores obtidos, para cada uma das instâncias, pelo método proposto por eles.

$$RPD = \frac{C_{HyPRR} - C_{min}}{C_{min}} \times 100 \tag{4.1}$$

O presente capítulo encontra-se organizado conforme segue. A Seção 4.1 aborda como foi realizada a parametrização do HyPRR. A Seção 4.2 realiza uma análise das vizinhanças utilizadas na busca local, justificando a escolha de cada uma delas. Experimentos computacionais com relação ao critério de desempate adotado são descritos na Seção 4.3. Na Seção 4.4, os resultados alcançados pela utilização dos limitantes inferiores da vizinhança *swap* são descritos. Uma análise com relação ao operador *ruin-and-recreate* e a utilização da população é realizada na Seção 4.5. Por fim, a comparação dos resultados alcançados com os já existentes na literatura são descritos na Seção 4.6.

4.1 Parâmetros do algoritmo

Conforme pode ser observado no Algoritmo 3.1, cinco parâmetros precisam ser calibrados, sendo eles: tamanho da populaçao inicial (μ), o número máximo de novos indivíduos (λ), o número de tarefas que serão removidas e reinseridas na geração de novos indivíduos (d), número de indivíduos elite (μ^{elite}) e o número de indivíduos mais próximos (μ^{close}).

Nesse sentido, optou-se por utilizar valores já conhecidos na literatura que demonstraram ser eficazes nos trabalhos em que foram utilizados. Assim, nos parâmetros referentes ao gerenciamento da população foram utilizados os valores conforme o trabalho de Mecler, Subramanian e Vidal (2021): $\mu = 20$, $\lambda = 40$, $\mu^{elite} = 10$ e $\mu^{close} = 3$. Com relação ao valor de d, Ruiz e Stützle (2007) conduziram experimentos computacionais variando o valor entre 2 e 8 para um *iterated greedy algorithm* desenvolvido para o $F_m|prmu|C_{max}$. Os testes, realizados nas mesmas instâncias utilizadas no presente trabalho, apontaram que o melhor valor para o parâmetro foi de $d = 4$. Assim, optou-se por utilizar este mesmo valor nos experimentos realizados.

4.2 Seleção de vizinhanças

Conforme elucidado na Seção 3.3, as vizinhanças *insertion*, *block insertion* e *swap* foram utilizadas. Entretanto, com o intuito de determinar quantas vizinhanças do tipo *block insertion* seriam empregadas, amostras de três instâncias de cada um dos 12 grupos foram selecionadas aleatoriamente. A escolha de realizar os testes em uma amostra de instâncias é para evitar que a calibração seja tendenciosa. Em seguida, cinco versões distintas do algoritmo foram implementadas: considerando apenas a vizinhança *insertion* ($x = 1$) e acrescentando novas vizinhanças do tipo *block insertion* até $x \leq 5$. Cada versão foi executada 10 vezes e os resultados foram comparados com os limites superiores apresentados no trabalho de Ribas, Companys e Tort-Martorell (2011) por meio do cálculo do RPD médio agrupado para cada grupo de instâncias (n x m). A Tabela 4.1 reporta os resultados obtidos.

Tabela 4.1: RPD médio para as cinco versões diferentes com vizinhanças *block insertion*

Instâncias	$x = 1$	$x \leq 2$	$x \leq 3$	$x \leq 4$	$x \leq 5$
20x5	0,000	0,000	0,000	0,000	0,000
20x10	0,000	0,000	0,000	0,000	0,000
20x20	0,000	0,000	0,000	0,000	0,000
50x5	-0,324	-0,427	-0,483	-0,586	**-0,616**
50x10	-0,199	-0,244	**-0,258**	-0,226	-0,224
50x20	-0,335	-0,318	-0,340	**-0,345**	-0,296
100x5	-1,057	-1,316	-1,491	-1,590	**-1,693**
100x10	**-1,781**	-1,728	-1,717	-1,721	-1,749
100x20	-1,682	-1,673	-1,721	**-1,728**	-1,682
200x10	**-1,520**	-1,405	-1,503	-1,489	-1,442
200x20	-2,209	-2,166	-2,158	**-2,310**	-2,045
500x20	-3,123	-3,018	-3,061	**-3,125**	-2,992
Avg.	-1,019	-1,025	-1,061	**-1,093**	-1,061

A partir dos resultados obtidos, conclui-se que as vizinhanças do tipo *block insertion* auxiliam na obtenção de soluções com alta qualidade. Nesse sentido, destaca-se a versão para $x \leq 4$, em que quatro vizinhanças de inserção são utilizadas na busca local, cujo RPD médio foi o menor para quatro grupos de instâncias distintos, assim como obteve a menor média geral. Além disso, é importante notar o impacto das vizinhanças de bloco nos grupos de instâncias que possuem apenas cinco máquinas. O valor do RPD médio teve um comportamento decresente nestes grupos conforme o valor de x aumentava, indicando a obtenção de soluções de maior qualidade. Como o melhor resultado, em média, foi alcançado para $x \leq 4$, optou-se por adotar este valor no presente trabalho.

Em seguida, a partir da validação das vizinhanças do tipo *block insertion*, uma sexta

versão do algoritmo, considerando a vizinhança do tipo *swap*, foi testada na mesma amostra de instâncias. Nesta versão, a vizinhança *swap* é incorporada à busca local e executada após a execução da vizinhança *block insertion* para $x = 4$, caso não seja encontrada uma melhora na solução corrente nesta vizinhança. Esta escolha é justificada a partir da análise de complexidade das vizinhanças realizada na Seção 3.3.

A Tabela 4.2 reporta o RPD médio obtido para cada um dos grupos de instâncias e se encontra organizada da seguinte forma: a primeira coluna indica o grupo de instância, a segunda coluna mostra o RPD médio obtido ao manter-se as vizinhanças considerando até $x = 4$ e a terceira coluna indica o RPD médio obtido pela versão em que a vizinhança *swap* é acrescentada.

Tabela 4.2: RPD médio obtido acrescentando a vizinhança *swap*

Instâncias	$x \leq 4$	$x \leq 4$+ Swap
20x5	0	0
20x10	0	0
20x20	0	0
50x5	-0,586	**-0,710**
50x10	-0,226	**-0,377**
50x20	**-0,345**	-0,336
100x5	-1,59	**-1,947**
100x10	-1,721	**-2,057**
100x20	-1,728	**-1,855**
200x10	-1,489	**-1,968**
200x20	**-2,31**	-2,236
500x20	**-3,125**	-2,839

Os resultados indicam que o *swap* possui um impacto de relevância na qualidade das soluções obtidas para todos os grupos, com exceção para o grupo de 500 tarefas e 20 máquinas. Devido ao tempo de execução limitado, a complexidade da vizinhança *swap* ($O(n^3m)$) torna o algoritmo mais lento e, dessa forma, não foi possível obter soluções com a mesma qualidade quando comparado com a versão sem o *swap*. Note que os grupos de 50 e 200 tarefas com 20 máquinas também possuiram um RPD médio maior com relação ao valor obtido pela versão que considera $x = 4$. Contudo, tal variação se mostrou mínima, menor que 0,1%. Desta forma, optou-se por adotar a versão com a vizinhança do tipo *swap* para todos os grupos de instância, com exceção para o que contém 500 tarefas e 20 máquinas.

4.3 Critério de desempate

A partir desta seção, todo o conjunto de instâncias é considerado e o algoritmo é executado 10 vezes para cada uma delas para a realização dos testes. Inicialmente, averiguou-se se o objetivo secundário utilizado para realizar o desempate convergeria de forma decrescente conforme a população evoluísse para soluções com maior qualidade em relação ao *makespan*. Para isso, foi computada a média do objetivo secundário T em três instantes distintos do tempo de execução, sendo eles o instante em que a população inicial é construída ($t = 0$), o instante em que o algoritmo finaliza sua execução ($t = CP$) e o instante correspondente a metade do tempo total de execução ($t = CP/2$). Os resultados são descritos na Tabela 4.3, que está organizada da seguinte forma: para cada instante de tempo t são reportados o tamanho da população e a média do objetivo secundário naquele instante, para cada um dos 12 grupos de instâncias.

Os resultados mostram que o T médio é maior no instante em que a população inicial é construída em relação aos demais instantes para todos os casos. Além disso, a média no instante final é menor do que no instante intermediário para todos os grupos de instâncias. Assim, o valor do objetivo secundário tende a diminuir à medida que a população evolui para soluções de alta qualidade em termos de makespan, levando em consideração que o método também considera a diversidade das soluções ao inseri-las na população, e não apenas sua qualidade.

Tabela 4.3: Média do objetivo secundário na população em três instantes distintos

Instances	$t = 0$		$t = CP/2$		$t = CP$	
	$\|P\|$	Avg.	$\|P\|$	Avg.	$\|P\|$	Avg.
20x5	20	2075,35	40,43	1619,99	39,90	1605,77
20x10	20	6885,68	37,33	5526,37	39,36	5520,85
20x20	20	22400,37	39,16	18542,55	38,71	18476,86
50x5	20	3596,26	40,54	2661,47	40,79	2614,23
50x10	20	11708,48	41,77	9340,85	40,90	9254,57
50x20	20	34456,19	39,20	28522,53	41,22	28477,19
100x5	20	5625,12	38,69	3988,05	39,14	3866,48
100x10	20	19562,76	39,71	15191,82	40,80	14916,01
100x20	20	56703,72	38,15	46464,46	40,30	45981,98
200x10	20	33990,25	39,57	27488,38	41,99	26769,55
200x20	20	98055,49	41,15	83179,45	39,35	81364,76
500x20	20	215019,10	40,77	172878,74	39,03	170857,79

O comportamento decrescente do objetivo secundário em relação ao tempo de execução trouxe a motivação de executar duas versões do HyPRR: com e sem desempate. A tabela 4.4 informa os valores médios de RPD para cada grupo de instâncias, exceto

Tabela 4.4: RPD médio obtido pelo algoritmo com e sem o critério de desempate

Instância	Com desempate	Sem desempate
50x5	**-0,662**	-0,572
50x10	**-0,620**	-0,458
50x20	**-0,409**	-0,305
100x5	**-1,987**	-1,777
100x10	**-1,941**	-1,671
100x20	**-1,687**	-1,339
200x10	**-1,984**	-1,648
200x20	**-2,342**	-1,899
500x20	**-3,098**	-2,915

para aquelas contendo 20 tarefas, pois ambas as versões obtiveram os mesmos resultados em todas as execuções. Os experimentos mostram que a versão que utiliza o critério de desempate sistematicamente obteve melhor desempenho.

Testes adicionais foram conduzidos para determinar se havia uma diferença estatisticamente significativa entre os resultados obtidos pelas duas versões para cada grupo de instância. Ao contrário dos experimentos relatados na Seção 4.2, os teste estatísticos foram realizados porque os experimentos com relação ao critério de desempate foram realizados em todo o conjunto de instâncias.

Dessa forma, as diferenças entre os valores de *makespan* obtidos pelas versões foram calculadas e agrupadas por tamanho de instância e o teste de normalidade de Anderson-Darling foi realizado em cada um desses grupos. A Tabela 4.5 mostra os p-valores para cada grupo de instância. Para os casos em que não foi encontrado desvio significativo da normalidade, foi aplicado o Teste-t Pareado. Por outro lado, quando os valores não seguiram uma distribuição normal (50x5), foi aplicado o teste não-paramétrico de Wilcoxon. Esse teste é utilizado para substituir o Teste-t Pareado em situações em que as amostras não seguem a distribuição normal. Como todos os p-valores para ambos os testes ficaram abaixo de $0,01$ para um nível de significância de 5%, pode-se afirmar que há uma diferença estatística significativa ao comparar os resultados obtidos pelas duas versões, justificando assim a adoção do critério de desempate no HyPRR.

Tabela 4.5: P-valores para o teste de normalidade do critério de desempate

Instâncias	50x5	50x10	50x20	100x5	100x10	100x20	200x10	200x20	500x20
p-valor	0,019	0,203	0,099	0,893	0,507	0,974	0,282	0,628	0,241

4.4 Limitantes Inferiores

Os limitantes inferiores propostos (LB) para a vizinhança *swap* foram submetidos a diversos testes de forma a comprovar sua efetividade. Inicialmente, os limites foram comparados com propriedades clássicas (CL) encontradas na literatura no trabalho de Grabowski e Pempera (2007) que foram explicadas na Seção 2.2.4. Apesar de tais propriedades serem propostas para a vizinhança do tipo *insertion*, elas podem ser facilmente aplicadas na vizinhança *swap*. Dessa forma, tais propriedades foram incorporadas no método proposto.

A Figura 4.1 mostra o percentual de avaliações de movimentos evitados por cada método. Note que os limitantes inferiores propostos conseguem evitar uma quantidade de avaliações bastante superior quando comparados com as propriedades propostas por Grabowski e Pempera (2007) por uma margem considerável. Além disso, quando os limitantes inferiores são aplicados, a quantidade de avaliações evitadas se mantém estável conforme o tamanho da instância aumenta, o que não acontece com as propriedades clássicas que conseguem evitar apenas 0,19% dos movimentos nas instâncias de 500 tarefas, o pior caso, e 4,07% nas instâncias com 20 tarefas e 5 máquinas, o melhor caso. Por outro lado, os limitantes inferiores possuem como pior caso 30,23% de avaliações evitadas para o grupo de 20 tarefas e 20 máquinas, e 46,29% no melhor caso, quando a instância é de 100 tarefas e 5 máquinas.

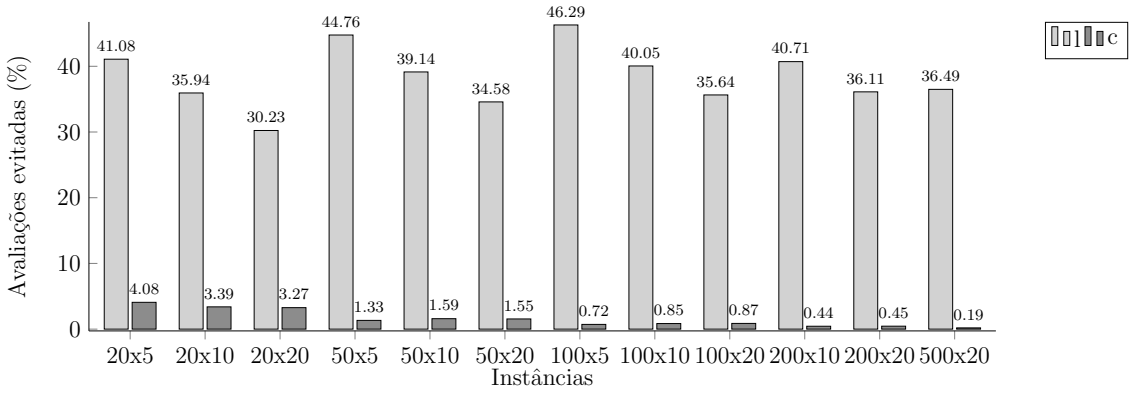

Figura 4.1: Porcentagem de avaliações de movimentos evitadas pelos limitantes inferiores propostos (l) e as propriedades clássicas (c)

A discrepância em questão de efetividade entre as propriedades clássicas e os limitantes propostos pode ser explicada a partir da quantidade e comprimento de blocos presentes em cada grupo de instâncias. Conforme Ding et al. (2016), o aumento no número de tarefas leva a um aumento no número de blocos, enquanto um aumento no número de máquinas não afeta significativamente este número. Além disso, o comprimento destes

Tabela 4.6: Impacto dos limitantes inferiores no número de iterações

Instâncias	**Com limitante**	**Sem limitante**	**Variação**
20x5	29.347,18	26.449,62	9,87%
20x10	38.331,35	35.616,42	7,05%
20x20	52.711,5	47.954,55	9,02%
50x5	10.128,13	9.675,29	4,47%
50x10	15.593,26	13.872,33	11,04%
50x20	18.816,44	16.162,22	14,11%
100x5	4.587,92	3.883,09	15,36%
100x10	6.428,46	5.409,37	15,85%
100x20	6.919,91	5.443,78	21,33%
200x10	2.207,85	1.717,99	22,19%
200x20	2.017,63	1.357,71	32,71%
500x20	226,66	166,48	26,55%

blocos não varia de forma considerável conforme aumenta o número de tarefas. Assim, como as propriedades clássicas são aplicadas apenas entre tarefas de um mesmo bloco, sua efetividade está diretamente ligada ao tamanho deles. Como este aumento não acontece conforme o número de tarefas aumenta, explica-se a baixa efetividade destas propriedades. Por outro lado, os limitantes propostos são capazes de atuar em tarefas que pertencem a um mesmo bloco e a blocos distintos e, portanto, a efetividade é mantida conforme o número de tarefas aumenta.

Dada a efetividade apresentada no que tange a avaliação de movimentos, é importante mensurar o impacto tanto na quantidade de iterações que o algoritmo é capaz de fazer, dado que o tempo de execução foi adotado como critério de parada, quanto na proporção de tempo gasto entre as vizinhanças presentes na busca local. Para isso, o algoritmo foi executado com e sem os limitantes inferiores para a vizinhança *swap*. A Tabela 4.6 mostra o impacto dos limitantes na quantidade de iterações executadas pelo algoritmo e encontra-se organizada da seguinte forma: a primeira coluna indica o grupo de instâncias, as próximas duas colunas indicam o número de iterações com e sem os limitantes inferiores e, por fim, a última coluna mostra a variação obtida. Note que houve aumento no número de iterações para todos os grupos de instâncias, cujo ganho pode chegar em até 32,71%. Além disso, é importante notar que para o grupo com 500 tarefas o número de iterações se torna bastante baixo quando comparado com os outros grupos, corroborando com os resultados apresentados na seção anterior e justificando uma vez mais a escolha de não executar o método proposto com a vizinhança *swap* para este grupo.

Por fim, foi avaliado a proporção do tempo de execução de cada uma das vizinhanças. Os resultados são mostrados nas Figuras 4.2 e 4.3. Observa-se que a utilização dos

limitantes possui um impacto direto no tempo de execução das demais vizinhanças. Os resultados agregados mostram uma diminuição de cerca de 7,25% no tempo de execução da vizinhança *swap*, fornecendo maior tempo para que as demais vizinhanças atuem. Além disso, note que a vizinhança *insertion* é a segunda com maior tempo de execução, apesar de ser a que executa mais rapidamente. Isto ocorre devido a natureza da busca, já que toda vez que cada uma das outras vizinhanças encontram uma melhora a busca retorna para *insertion*.

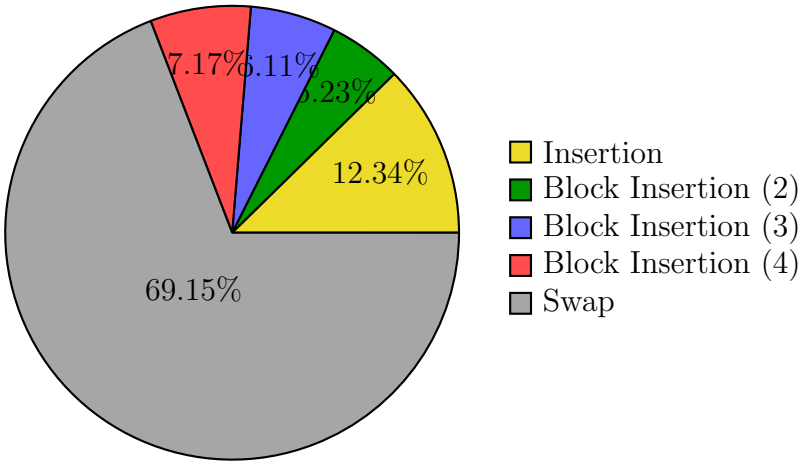

Figura 4.2: Proporção de tempo de execução de cada vizinhança sem utilização dos limitantes

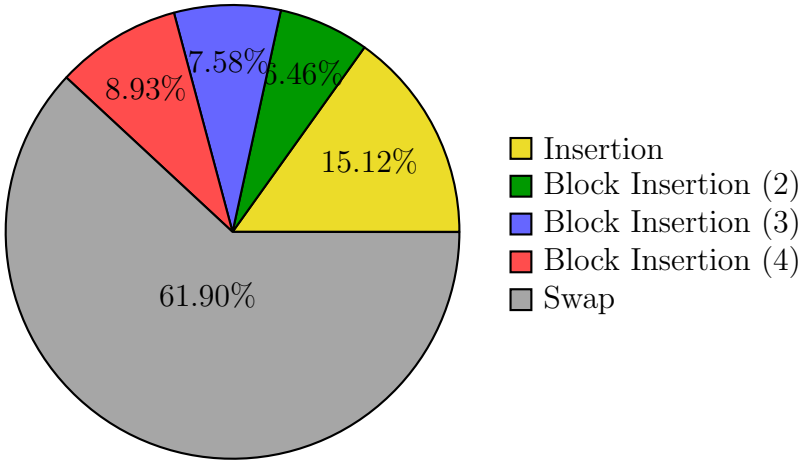

Figura 4.3: Proporção de tempo de execução de cada vizinhança com utilização dos limitantes

4.5 Operador ruin-and-recreate e população

O operador *ruin-and-recreate* e a diversidade da população também foram submetidos a testes computacionais. O intuito destes testes é validar cada componente incorporado no

algoritmo e demonstrar que a utilização deles possui uma eficácia maior quando comparados com outros métodos comumente utilizados na literatura. Assim, foram implementadas um total de 7 versões distintas do método proposto.

Entre essas versões, o operador *ruin-and-recreate* (RR) foi substituido por dois operadores de *crossover* diferentes, sendo eles o *crossover* OX, o mesmo utilizado por Mecler, Subramanian e Vidal (2021) para o *job sequencing and tool switching problem*, e o *crossover* SJOX, proposto por Ruiz, Maroto e Alcaraz (2006) para o $F_m|prmu|C_{max}$. Além disso, esses operadores também foram utilizados em conjunto com o R&R, isto é, depois de realizado o *crossover* entre duas soluções distintas, o operador *ruin-and-recreate* é aplicado na solução resultante. Em todas essas versões, a busca local é aplicada na solução resultante, da mesma forma que é feita no método proposto.

Com relação à população, duas novas versões foram testadas: uma com solução única e outra sem o gerenciamento de diversidade. Na primeira, a população é formada por uma única solução que é submetida ao operador *ruin-and-recreate* e, em seguida, é submetida a busca local. Na segunda versão, a função de aptidão descrita na Equação 3.19 é modificada para considerar apenas a qualidade das soluções, desconsiderando a diversidade da população.

A Tabela 4.7 sumariza os resultados obtidos. Todas as versões do algoritmo conseguiram obter a melhor solução encontrada na literatura para os grupos de instância com 20 tarefas, com exceção da que não utiliza população, mas sim uma única solução. Nela, os grupos 20x10 e 20x20 apresentam um RPD médio de $0,002$ e $0,004$, respectivamente. Apesar disso, quando comparamos a média considerando todos os grupos de instância, os piores resultados são aqueles obtidos pelas versões que utilizam o *crossover* OX. A diferença de qualidade das soluções obtidas pelas versões que utilizam os dois tipos de *crossover* distintos pode ser justificada pelo fato de que o *crossover* SJOX foi proposto para um problema de *flow shop*, aproveitando-se de características do problema para realizar a operação. Ainda, observa-se que a utilização do operador *ruin-and-recreate* de forma conjunta com os operadores de *crossover* não apresentou uma melhora significativa, havendo, inclusive, piorado a média no caso do *crossover* SJOX. Por fim, a versão que não considera a diversidade da população foi a que mais se aproximou dos resultados obtidos pelo HyPRR, com um RPD médio inferior a -1.

Após realizar a comparação do RPD entre as diferentes versões do algoritmo, foram realizados testes para averiguar se há uma diferença estatística significativa entre os resultados obtidos por cada uma das versões, da mesma forma que foi feito na Seção 4.3. Os

Tabela 4.7: RPD médio obtido por cada versão do algoritmo

Instância	OX	OX+R&R	SJOX	SJOX+R&R	Unica	SD	HyPRR
20x5	0,000	0,000	0,000	0,000	0,000	0,000	0,000
20x10	0,000	0,000	0,000	0,000	0,002	0,000	0,000
20x20	0,000	0,000	0,000	0,000	0,004	0,000	0,000
50x5	-0,586	-0,561	-0,622	-0,608	-0,352	-0,551	**-0,662**
50x10	-0,448	-0,452	-0,491	-0,548	-0,103	-0,404	**-0,620**
50x20	-0,230	-0,281	-0,313	-0,364	0,102	-0,204	**-0,409**
100x5	-1,325	-1,180	-1,792	-1,614	-1,555	-1,783	**-1,987**
100x10	-1,057	-1,050	-1,663	-1,582	-1,201	-1,598	**-1,941**
100x20	-0,537	-0,600	-1,338	-1,309	-0,797	-1,241	**-1,687**
200x10	-0,692	-0,691	-1,071	-0,975	-1,348	-1,657	**-1,984**
200x20	-0,738	-0,840	-1,301	-1,266	-1,591	-1,845	**-2,342**
500x20	-2,416	-2,462	-2,662	-2,574	-2,667	-2,909	**-3,098**
Avg	-0,669	-0,676	-0,938	-0,903	-0,792	-1,016	**-1,228**

p-valores do teste de normalidade de Anderson-Darling podem ser observados na Tabela 4.8. Para os casos em que não foi encontrado desvio significativo da normalidade, foi aplicado o Teste-t Pareado. Nos casos em que os valores não seguem a distribuição normal, o teste de Wilcoxon foi aplicado. Após a aplicação dos testes referidos, observou-se que todos os p-valores ficaram abaixo de 0,05 para um nível de significância de 5%. Mais precisamente, todos os valores obtidos foram menores ou iguais a 0,002. Tal resultado indica que há uma diferença significativa entre os resultados obtidos por cada uma das versões modificadas quando comparados com os resultados obtidos pelo HyPRR. Assim, conclui-se que a presença de cada um dos elementos incorporados no algoritmo original possui impacto significativo nos resultados obtidos.

Tabela 4.8: P-valores do teste de normalidade Anderson-Darling para cada versão

Instâncias	OX	OX+RR	SJOX	SJOX+RR	Única	SD
50x5	0,141	0,028	0,026	0,005	0,084	0,023
50x10	0,07	0,316	0,184	0,333	0,536	0,497
50x20	0,22	0,068	0,24	0,005	0,006	0,101
100x5	0,118	0,022	0,747	0,934	0,606	0,554
100x10	0,646	0,691	0,944	0,941	0,965	0,025
100x20	0,142	0,224	0,118	0,005	0,801	0,929
200x10	0,544	0,648	0,8	0,106	0,964	0,025
200x20	0,923	0,545	0,149	0,716	0,086	0,586
500x20	0,119	0,026	0,496	0,404	0,755	0,024

Com relação ao tempo de execução de cada etapa do método proposto, a Tabela 4.9 mostra os resultados obtidos. É importante salientar que a construção da população inicial é finalizada após a inclusão de cada indivíduo. Como cada um deles deve passar por uma busca local antes da inclusão, a proporção de tempo de execução desta etapa

tende a crescer conforme aumenta-se o número de tarefas. Todavia, percebe-se que se o número de tarefas é mantido e o número de máquinas é aumentado, a construção da população inicial tende a ser feita de forma mais rápida. Já com relação ao gerenciamento da população, que é feito de acordo com a função de aptidão de cada indivíduo, percebe-se que a proporção de tempo de execução tende a diminuir conforme o tamanho da instância aumenta, uma vez que esta etapa do algoritmo não depende do tamanho dos dados de entrada. Em seguida, conforme esperado, a etapa de busca local é a etapa com maior demanda de tempo computacional, aumentando conforme o tamanho das instâncias também aumenta. Apesar de haver um decréscimo nas instâncias de 500 tarefas, o comportamento é justificado por haver um aumento maior no tempo necessário para se construir a população inicial que, por sua vez, também utiliza uma fase de busca local conforme explicado previamente. Por fim, observa-se que o operador *ruin-and-recreate* tende a levar um menor tempo de execução, proporcionalmente, conforme o número de tarefas aumenta. Dentro de grupos de instâncias que possuem um mesmo número de tarefas, o operador utiliza um maior tempo de execução naqueles em que o número de máquinas é maior. Tal comportamento é esperado, uma vez que a fase de reinserção de uma tarefa em uma permutação é feita em $O(m)$.

Tabela 4.9: Proporção de tempo de execução de cada etapa do algoritmo

Instâncias	População Inicial	Gerenciamento da População	Busca Local	*Ruin-and-Recreate*
20x5	0,12%	19,94%	77,39%	2,55%
20x10	0,09%	14,67%	82,39%	2,85%
20x20	0,07%	10,07%	86,76%	3,09%
50x5	0,51%	4,18%	94,54%	0,78%
50x10	0,28%	3,22%	95,53%	0,97%
50x20	0,23%	2,13%	96,60%	1,04%
100x5	1,20%	1,29%	97,20%	0,30%
100x10	0,72%	0,96%	97,94%	0,37%
100x20	0,61%	0,58%	98,45%	0,37%
200x10	2,04%	0,26%	97,58%	0,12%
200x20	1,86%	0,14%	97,90%	0,10%
500x20	13,79%	0,15%	85,93%	0,14%

4.6 Comparação com a literatura

A comparação dos resultados obtidos com os encontrados na literatura acerca do $F_m|block|C_{max}$ é apresentada a seguir. Nela, foram escolhidos os trabalhos de Shao et al. (2019) e Tasgetiren et al. (2017) para a realização da análise dos resultados obtidos pelo HyPRR quando confrontados com resultados obtidos por métodos estado-da-arte da literatura. Essa escolha é justificada por acreditar-se que esses trabalhos reportam

as melhores soluções conhecidas (BKS) para o problema em estudo, de acordo com a revisão bibliográfica desenvolvida. Testes estatísticos não foram realizados porque não houve acesso aos resultados detalhados dos métodos.

Tasgetiren et al. (2017) propuseram dois algoritmos baseados na meta-heurística *iterated greedy algorithm*, nomeadas IG_RIS e IG_IJ. Ambos foram implementados em Visual C++13 e os experimentos computacionais foram realizados em um Intel(R) Core(TM) i7-2600 3.40GHZ com 8 GB de memória RAM. Shao et al. (2019) propuseram um algoritmo denomidado *discrete invasive weed optimization* (DIWO) que foi implementado em Java e os experimentos computacionais realizados em um Intel Core (TM) 3.20 GHz, com 4 GB de memória RAM e em um sistema operacional Windows 7. Apesar do modelo exato do processador utilizado não ser reportado, foi encontrado um trabalho acerca do $F_m|block, st_{sd}|C_{max}$ (SHAO; PI; SHAO, 2018c) realizado pelos mesmos autores em que o ambiente computacional utilizado para a realização dos experimentos é similar ao empregado por Shao et al. (2019): Intel Core™ i5-3470 3.20GHZ, com 4 GB de memória RAM e sistema operacional Windows 7. Dessa forma, acredita-se que o modelo do processador utilizado nos experimentos de Shao et al. (2019) é o mesmo reportado por Shao, Pi e Shao (2018c).

Assim, de acordo com o site CPU *Benchmark* (`cpubenchmark.net/compare/2797vs1vs822/Intel-Xeon-E5-2650-v4-vs-Intel-i7-2600-vs-Intel-i5-3470`), os processadores utilizados pelos referidos autores são mais rápidos do que o utilizado no presente trabalho, sendo o utilizado por Tasgetiren et al. (2017), 1,012 vezes mais rápido e o utilizado por Shao et al. (2019), 1,116 vezes. Portanto, as análises realizadas a seguir podem ser consideradas justas, uma vez que não são esperados ganhos na qualidade das soluções obtidas por realização de experimentos utilizando-se um processador mais rápido do que os utilizados pelos estudos que servem de objetos de comparação.

Inicialmente, os resultados obtidos pelo HyPRR são comparados com os reportados por Shao et al. (2019). Para isso, cada uma das instâncias foi executada 10 vezes para cada um dos critérios de parada adotados pelos autores, sendo eles $CP = p \times n \times m$ milissegundos, em que $p = \{30, 60, 90\}$. Ainda, os autores reimplementaram os seguintes algoritmos da literatura para comparação com o DIWO: HDDE (WANG et al., 2010), hmgHS (WANG; PAN; TASGETIREN, 2011), IG (RIBAS; COMPANYS; TORT-MARTORELL, 2011), TPA (WANG et al., 2012), RAIS (LIN; YING, 2013), MA (PAN et al., 2012), SVNS_D (RIBAS; COMPANYS; TORT-MARTORELL, 2013), HVNS (MOSLEHI; KHORASANIAN, 2014), DE-ABC (HAN; GONG; SUN, 2015), DE_PLS (TASGETIREN et al., 2015), MFFO (HAN et al., 2016), SaDIWO (SHAO; PI; SHAO, 2017) e P-EDA (SHAO; PI; SHAO, 2018a). Os métodos

são comparados por meio do RPD médio e agrupados por tamanho das instâncias. Assim, Shao et al. (2019) reportam os resultados para cada um dos valores p. As Tabelas 4.10 - 4.12 reportam esses resultados, acrescentando os RPD médios obtidos pelo HyPRR.

Tabela 4.10: Comparação de RPD médio para $p = 30$ conforme Shao et al. (2019)

Instâncias	HDDE	hmgHS	IG	TPA	RAIS	MA	SVSNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P-EDA	DIWO	HyPRR
20x5	0,013	0,121	0,065	0,008	0,230	0,005	0,069	0,025	0,014	0,007	0,005	0,017	0,060	0,000	0,000
20x10	0,012	0,057	0,107	0,014	0,079	0,007	0,034	0,004	0,016	0,006	0,007	0,008	0,009	0,000	0,000
20x20	0,001	0,031	0,104	0,000	0,017	0,001	0,003	0,000	0,009	0,001	0,003	0,010	0,000	0,000	0,000
50x5	0,414	0,574	1,220	0,346	0,634	0,058	0,196	0,298	1,842	0,229	0,705	0,021	0,155	0,128	**-0,488**
50x10	0,191	0,398	1,374	-0,045	0,336	-0,039	0,296	0,086	1,463	-0,040	0,395	0,105	0,040	-0,073	**-0,380**
50x20	0,223	0,360	1,429	-0,063	0,978	0,009	0,561	0,037	1,210	-0,016	0,342	0,312	0,104	0,086	**-0,225**
100x5	1,354	0,719	0,929	-0,284	0,062	-0,405	-0,609	-0,351	2,428	0,111	0,916	-0,746	-0,647	-0,775	**-1,564**
100x10	0,596	0,127	1,041	-0,830	0,095	-0,811	-0,465	-0,696	1,568	-0,357	0,135	-0,657	-0,800	-0,972	**-1,392**
100x20	0,403	0,090	1,149	-0,951	1,765	-0,780	0,120	-0,734	1,211	-0,387	0,016	-0,139	-0,616	-0,741	**-1,102**
200x10	2,059	1,727	1,935	-0,789	1,722	-0,737	-0,742	-0,530	2,286	-0,224	1,180	-0,763	-0,939	-1,200	**-1,248**
200x20	0,909	0,614	1,282	-1,480	2,802	-1,343	-0,857	-1,191	1,045	-0,606	-0,028	-0,889	-0,281	-1,591	**-1,598**
500x20	1,171	1,078	1,376	-1,933	4,083	-2,540	-2,425	-0,148	0,768	-2,139	0,038	-2,463	-2,583	-2,835	**-2,855**
Avg.	0,612	0,491	1,001	-0,501	1,067	-0,548	-0,318	-0,267	1,155	-0,285	0,310	-0,432	-0,542	-0,664	**-0,904**

Tabela 4.11: Comparação de RPD médio para $p = 60$ conforme Shao et al. (2019)

Instâncias	HDDE	hmgHS	IG	TPA	RAIS	MA	SVSNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P-EDA	DIWO	HyPRR
20x5	0,004	0,094	0,031	0,012	0,227	0,000	0,050	0,017	0,007	0,000	0,002	0,012	0,000	0,000	0,000
20x10	0,008	0,039	0,068	0,028	0,080	0,005	0,007	0,002	0,012	0,006	0,003	0,003	0,000	0,000	0,000
20x20	0,002	0,014	0,058	0,000	0,017	0,000	0,008	0,000	0,004	0,000	0,000	0,004	0,000	0,000	0,000
50x5	0,237	0,438	1,085	0,317	0,575	-0,074	0,147	0,202	1,695	0,062	0,563	-0,066	0,050	0,016	**-0,546**
50x10	0,069	0,347	1,200	-0,028	0,283	-0,081	0,120	-0,036	1,376	-0,113	0,284	-0,004	-0,037	-0,115	**-0,448**
50x20	0,078	0,312	1,293	-0,018	0,363	-0,051	0,497	-0,053	1,115	-0,130	0,232	0,212	0,136	0,000	**-0,303**
100x5	0,985	0,401	0,699	-0,324	0,048	-0,577	-0,733	-0,554	2,277	-0,210	0,682	-0,926	-0,813	-0,809	**-1,736**
100x10	0,270	-0,086	0,814	-0,831	-0,178	-1,002	-0,638	-0,834	1,495	-0,618	-0,025	-0,868	-0,951	-1,060	**-1,565**
100x20	0,127	-0,161	0,924	-0,966	0,036	-0,848	-0,134	-0,911	1,103	-0,636	-0,161	-0,396	-0,695	-0,848	**-1,245**
200x10	1,816	1,446	1,542	-0,786	0,587	-0,891	-0,850	-0,732	2,190	-0,347	0,810	-0,919	-1,070	-1,285	**-1,500**
200x20	0,645	0,437	0,916	-1,555	0,419	-1,482	-0,986	-1,419	0,961	-0,833	-0,262	-1,118	-1,442	-1,691	**-1,732**
500x20	0,957	0,933	1,270	-2,030	1,454	-2,623	-2,483	-1,883	0,595	-2,230	-0,243	-2,464	-2,669	-2,925	**-2,930**
Avg	0,433	0,351	0,825	-0,515	0,326	-0,635	-0,416	-0,517	1,069	-0,421	0,157	-0,544	-0,624	-0,726	**-1,000**

Tabela 4.12: Comparação de RPD médio para $p = 90$ conforme Shao et al. (2019)

Instâncias	HDDE	hmgHS	IG	TPA	RAIS	MA	SVSNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P-EDA	DIWO	HyPRR
20x5	0,001	0,077	0,010	0,013	0,246	0,000	0,036	0,013	0,001	0,003	0,000	0,000	0,000	0,000	0,000
20x10	0,008	0,033	0,050	0,027	0,064	0,003	0,003	0,000	0,008	0,002	0,002	0,000	0,000	0,000	0,000
20x20	0,000	0,010	0,033	0,000	0,020	0,000	0,001	0,000	0,000	0,000	0,001	0,000	0,000	0,000	0,000
50x5	0,148	0,365	0,998	0,292	0,556	-0,111	0,097	0,149	1,644	-0,062	0,447	-0,168	0,054	-0,080	**-0,588**
50x10	-0,049	0,265	1,083	-0,036	0,291	-0,141	0,126	-0,067	1,277	-0,255	0,218	-0,057	-0,067	-0,173	**-0,482**
50x20	-0,004	0,275	1,209	-0,032	0,394	-0,034	0,441	-0,073	1,078	-0,210	0,222	0,191	0,092	-0,094	**-0,305**
100x5	0,737	0,211	0,591	-0,340	0,019	-0,836	-0,817	-0,553	2,188	-0,281	0,393	-1,053	-0,858	-0,948	**-1,814**
100x10	0,019	-0,191	0,616	-0,903	-0,243	-1,051	-0,758	-0,879	1,436	-0,823	-0,173	-0,938	-0,107	-1,068	**-1,653**
100x20	-0,024	-0,222	0,783	-0,963	-0,109	-0,958	-0,247	-0,965	1,064	-0,819	-0,230	-0,555	-0,767	-0,971	**-1,341**
200x10	1,670	1,324	1,298	-0,810	0,473	-0,988	-0,904	-0,743	2,121	-0,414	0,630	-1,038	-1,169	-1,359	**-1,635**
200x20	0,532	0,266	0,721	-1,579	0,123	-1,580	-1,110	-1,468	0,934	-0,944	-0,407	-1,262	-1,543	-1,782	**-1,785**
500x20	0,874	0,863	1,041	-2,053	0,936	-2,676	-2,529	-1,976	0,480	-2,245	-0,414	-2,547	-2,710	-2,970	**-2,976**
Avg	0,326	0,273	0,703	-0,532	0,231	-0,698	-0,472	-0,547	1,019	-0,504	0,057	-0,617	-0,633	-0,787	**-1,048**

Desconsiderando os casos em que $n = 20$, os valores de RPD médio obtidos pelo HyPRR são menores aos obtidos por todos os algoritmos, em todos os grupos de instância. Para todos os valores de p considerados, o HyPRR supera os demais métodos por uma margem significativa, principalmente quando são considerados os grupos de instância com $n \leq 100$. Além disso, há um destaque para os grupos 50x5 e 100x5, em que a discrepância tende a ser maior devido a eficácia das vizinhanças do tipo *block insertion* nestes grupos conforme explicado na Seção 4.2. Além disso, o HyPRR, juntamente com o DIWO, foram

os únicos métodos a conseguirem obter as melhores soluções para as instâncias de 20 máquinas em todas as execuções.

Além das comparações anteriores, Shao et al. (2019) comparam os resultados obtidos para $p = 100$ com os seguintes métodos: HDDE (WANG et al., 2010), hmgHS (WANG; PAN; TASGETIREN, 2011), IG (RIBAS; COMPANYS; TORT-MARTORELL, 2011), TPA (WANG et al., 2012), MA (PAN et al., 2012), SVNS_S e SVNS_D (RIBAS; COMPANYS; TORT-MARTORELL, 2013), HVNS (MOSLEHI; KHORASANIAN, 2014), DE_PLS (TASGETIREN et al., 2015) e P-EDA (SHAO; PI; SHAO, 2018a). A Tabela 4.13 mostra os resultados em termos de RPD médio. Novamente, o HyPRR foi capaz de obter o melhor RPD médio em todos os casos. A média geral foi de -1,228, enquanto que o segundo melhor algoritmo, DIWO, obteve uma média de -0,844.

Tabela 4.13: Comparação de RPD médio para $p = 100$ conforme Shao et al. (2019)

Instâncias	HDDE	hmgHS	IG	TPA	RAIS	SVNS_S	SVNS_D	HVNS	DE_PLS	MA	P-EDA	DIWO	HyPRR
20x5	0.000	0.090	0.390	0.400	0.000	0.120	0.160	0.010	0.000	0.000	0.000	0.000	0.000
20x10	0.000	0.030	0.480	0.210	0.000	0.150	0.130	0.010	0.000	0.010	0.000	0.000	0.000
20x20	0.000	0.030	0.310	0.030	0.000	0.080	0.070	0.000	0.000	0.000	0.000	0.000	0.000
50x5	0.780	0.370	2.710	1.560	-0.190	0.690	0.770	0.170	-0.280	-0.170	-0.110	-0.100	**-0.662**
50x10	0.530	0.500	3.240	1.070	-0.200	0.880	0.750	-0.060	-0.380	-0.180	-0.240	-0.180	**-0.620**
50x20	0.350	0.290	2.880	0.760	0.000	0.850	0.700	-0.060	-0.340	-0.200	-0.120	-0.120	**-0.409**
100x5	1.780	0.800	3.820	1.710	-0.820	-0.190	-0.140	-0.600	-0.810	-0.810	-0.960	-0.980	**-1.987**
100x10	1.010	0.460	3.340	0.590	-0.880	0.110	0.140	-0.900	-1.250	-1.160	-1.210	-1.190	**-1.941**
100x20	0.860	0.310	0.030	0.070	-0.710	0.370	0.440	-1.030	-1.140	-1.090	-1.020	-1.170	**-1.687**
200x10	2.490	1.270	3.850	1.220	-0.330	0.040	0.040	-0.800	-0.930	-0.890	-1.410	-1.440	**-1.984**
200x20	1.240	0.370	2.310	0.030	-0.630	-0.280	-0.270	-1.500	-0.950	-1.060	-1.820	-1.870	**-2.342**
500x20	1.540	0.170	1.320	-0.050	-0.020	-1.750	-1.740	-2.000	-2.620	-2.650	-2.940	-3.080	**-3.098**
Avg.	0.882	0.391	2.057	0.633	-0.315	0.089	0.088	-0.563	-0.725	-0.683	-0.819	-0.844	**-1.228**

Com relação ao trabalho de Tasgetiren et al. (2017), os autores compararam o IG_RIS e IG_IJ, para $p = 100$, com os seguintes métodos: HDDE (WANG et al., 2010), SVNS_S e SVNS_D (RIBAS; COMPANYS; TORT-MARTORELL, 2013), SVNS* (RIBAS; COMPANYS; TORT-MARTORELL, 2013; TASGETIREN et al., 2017), VND1 e VND2 (TASGETIREN et al., 2017) e MA (PAN et al., 2012). Cada instância foi executada 10 vezes por cada um dos métodos. A Tabela 4.14 mostra os resultados obtidos. Nela, percebe-se que o HyPRR foi capaz de obter melhores resultados que todos os outros métodos em todos os grupos com $n \geq 50$, com exceção do 100x20, em que obteve um resultado inferior ao IG_IJ por uma diferença de 0,022 no RPD médio. Com relação à média geral, o HyPRR obteve um valor igual a -1,228 enquanto o IG_IJ e o IG_RIS obtiveram -1,100. Assim, conclui-se que o HyPRR obteve resultados melhores com relação aos obtidos pelos demais métodos considerados por Tasgetiren et al. (2017).

A Tabela 4.15 reporta os resultados detalhados para cada uma das 120 instâncias que foram executadas 10 vezes pelo HyPRR para $CP = 100 \times n \times m$ milissegundos. Para cada instância, são reportados o melhor limitante superior obtido Shao et al. (2019)

Tabela 4.14: Comparação de RPD médio para $p = 100$ conforme Tasgetiren et al. (2017)

Instâncias	HDDE	SVNS_S	SVNS_D	SVNS*	VND2	VND1	MA	IG_RIS	IG_IJ	HyPRR
20x5	0,000	0,021	0,028	0,000	0,000	0,000	0,000	0,000	0,000	0,000
20x10	0,000	0,010	0,024	0,006	0,000	0,000	0,000	0,000	0,000	0,000
20x20	0,000	0,004	0,000	0,001	0,000	0,000	0,000	0,000	0,000	0,000
50x5	0,110	-0,042	-0,003	-0,051	-0,374	-0,457	-0,400	-0,529	-0,519	**-0,662**
50x10	0,210	0,091	0,109	-0,019	-0,392	-0,434	-0,420	-0,567	-0,556	**-0,620**
50x20	0,160	0,344	0,320	0,095	-0,217	-0,296	-0,320	-0,396	-0,397	**-0,409**
100x5	0,190	-0,985	-1,020	-1,048	-1,304	-1,376	-1,120	-1,507	-1,520	**-1,987**
100x10	0,650	-0,879	-0,866	-0,978	-1,374	-1,418	-1,610	-1,819	-1,805	**-1,941**
100x20	0,560	-0,498	-0,509	-0,747	-1,047	-1,240	-1,540	-1,677	**-1,709**	-1,687
200x10	-1,100	-1,085	-1,083	-1,235	-1,317	-1,386	-1,350	-1,798	-1,788	**-1,984**
200x20	-0,630	-0,866	-0,879	-1,097	-1,095	-1,260	-1,600	-1,826	-1,816	**-2,342**
500x20	-0,530	-2,562	-2,564	-2,743	-2,650	-2,735	-2,800	-3,077	-3,086	**-3,098**
Avg.	-0,032	-0,537	-0,537	-0,651	-0,814	-0,884	-0,930	-1,100	-1,100	**-1,228**

(DIWO) e Tasgetiren et al. (2017) (IG), a melhor solução conhecida (BKS), isto é, o menor valor entre os dois mencionados anteriormente, a melhor e a média das 10 execuções do HyPRR (*Best*, Avg.) e os RPD mínimo e médio (RPD_{Best}, $\text{RPD}_{\text{Avg.}}$) quando compara-se os resultados obtidos pelo algoritmo proposto com a BKS.

Com base nos resultados apresentados, o HyPRR foi capaz de obter melhores soluções do que as encontradas na literatura em um total de 54 instâncias (45%) e empatou em 58 outras (48,33%), perdendo em apenas 8 instâncias (6,67%). Além disso, quando comparado individualmente com os limites superiores reportados por Shao et al. (2019), o método proposto foi capaz de melhorar as soluções de 74 instâncias (61,67%), empatar em 43 outras (35,83%) e perder em apenas 3 (2,5%). Por outro lado, quando comparado com os limites reportados por Tasgetiren et al. (2017), o HyPRR melhorou os resultados em 55 instâncias (45,83%), empatou em 58 (48,33%) e perdeu em apenas 7 (5,83%). Assim, percebe-se que o HyPRR foi capaz de obter soluções competitivas quando comparadas com a literatura, alcançando soluções melhores ou iguais em 93,33% dos casos.

Além das instâncias propostas por Taillard (1993), Tasgetiren et al. (2017) também realizaram experimentos computacionais em grupos de instâncias de 200x5, 500x5 e 500x10. De acordo com os autores, a literatura acerca do $F_m|block|C_{max}$ não trata estes grupos. Entretanto, Pan e Ruiz (2012) propuseram estas instâncias para o $F_m|prmu|\sum_j C_j$. Assim, o grupo 200x5 é gerado a partir das instâncias 200x10, em que são considerados apenas os tempos de processamento das primeiras 5 máquinas. Da mesma forma, os grupos 500x5 e 500x10 são derivados das instâncias 500x20, em que são considerados os tempos de processamentos das primeiras 5 e 10 máquinas, respectivamente. Assim, este trabalho trata as instâncias propostas por Pan e Ruiz (2012) como instâncias do $F_m|block|C_{max}$, da mesma forma que Tasgetiren et al. (2017).

A Tabela 4.16 mostra os resultados obtidos nesses grupos de instâncias e encontra-se organizada da seguinte forma: a primeira coluna indica o grupo de instâncias considerado,

Tabela 4.15: Resultados obtidos nas instâncias introduzidas por Taillard (1993)

Instância	DIWO	IG	BKS	*Best*	Avg.	RPD_{Best}	$RPD_{Avg.}$	Instance	DIWO	IG	BKS	*Best*	Avg.	RPD_{Best}	$RPD_{Avg.}$
				20x5								100x5			
Ta001	1374	1374	1374	1374	1374	0,00	0,00	Ta061	6065	6038	6038	**6004**	6027,9	-0,57	-0,17
Ta002	1408	1408	1408	1408	1408	0,00	0,00	Ta062	5934	5925	5925	**5888**	5908,9	-0,63	-0,27
Ta003	1280	1280	1280	1280	1280	0,00	0,00	Ta063	5851	5831	5831	**5805**	5819,7	-0,45	-0,19
Ta004	1448	1448	1448	1448	1448	0,00	0,00	Ta064	5656	5659	5656	**5623**	5645	-0,59	-0,19
Ta005	1341	1341	1341	1341	1341	0,00	0,00	Ta065	5896	5873	5873	**5845**	5859,8	-0,48	-0,23
Ta006	1363	1363	1363	1363	1363	0,00	0,00	Ta066	5755	5732	5732	**5711**	5728,3	-0,37	-0,06
Ta007	1381	1381	1381	1381	1381	0,00	0,00	Ta067	5915	5886	5886	**5866**	5878	-0,34	-0,14
Ta008	1379	1379	1379	1379	1379	0,00	0,00	Ta068	5809	5778	5778	**5753**	5769,8	-0,43	-0,14
Ta009	1373	1373	1373	1373	1373	0,00	0,00	Ta069	6027	6019	6019	**6004**	6017,6	-0,25	-0,02
Ta010	1283	1283	1283	1283	1283	0,00	0,00	Ta070	6059	6049	6049	**6025**	6041,3	-0,40	-0,13
				20x10								100x10			
Ta011	1698	1698	1698	1698	1698	0,00	0,00	Ta071	6906	6896	6896	**6891**	6911,3	-0,07	0,22
Ta012	1833	1833	1833	1833	1833	0,00	0,00	Ta072	6656	6622	6622	**6618**	6641,1	-0,06	0,29
Ta013	1659	1659	1659	1659	1659	0,00	0,00	Ta073	6797	6766	6766	**6751**	6782,8	-0,22	0,25
Ta014	1535	1535	1535	1535	1535	0,00	0,00	Ta074	7035	7028	7028	**7019**	7037,4	-0,13	0,13
Ta015	1617	1617	1617	1617	1617	0,00	0,00	Ta075	6728	6690	6690	**6689**	6712,2	-0,01	0,33
Ta016	1590	1590	1590	1590	1590	0,00	0,00	Ta076	6537	6517	6517	**6496**	6534,7	-0,32	0,27
Ta017	1622	1622	1622	1622	1622	0,00	0,00	Ta077	6689	6679	6679	**6653**	6686,7	-0,39	0,12
Ta018	1731	1731	1731	1731	1731	0,00	0,00	Ta078	6746	6707	6707	**6705**	6733,6	-0,03	0,40
Ta019	1747	1747	1747	1747	1747	0,00	0,00	Ta079	6928	6900	6900	**6894**	6923,1	-0,09	0,33
Ta020	1782	1782	1782	1782	1782	0,00	0,00	Ta080	6855	6824	6824	**6803**	6821,4	-0,31	-0,04
				20x20								100x20			
Ta021	2436	2436	2436	2436	2436	0,00	0,00	Ta081	7709	7664	7664	**7663**	7698	-0,01	0,44
Ta022	2234	2234	2234	2234	2234	0,00	0,00	Ta082	7744	7725	7725	**7724**	7743	-0,01	0,23
Ta023	2479	2479	2479	2479	2479	0,00	0,00	Ta083	7723	7694	7694	7694	7724	0,00	0,39
Ta024	2348	2348	2348	2348	2348	0,00	0,00	Ta084	7743	7722	7722	**7706**	7731,6	-0,21	0,12
Ta025	2435	2435	2435	2435	2435	0,00	0,00	Ta085	7730	7694	7694	7703	7725,2	0,12	0,40
Ta026	2383	2383	2383	2383	2383	0,00	0,00	Ta086	7779	7745	7745	7752	7769,8	0,09	0,32
Ta027	2390	2390	2390	2390	2390	0,00	0,00	Ta087	7857	7840	7840	**7839**	7869,6	-0,01	0,38
Ta028	2328	2328	2328	2328	2328	0,00	0,00	Ta088	7898	7866	7866	7875	7894,4	0,11	0,36
Ta029	2363	2363	2363	2363	2363	0,00	0,00	Ta089	7818	7771	7771	**7770**	7806,7	-0,01	0,46
Ta030	2323	2323	2323	2323	2323	0,00	0,00	Ta090	7842	7816	7816	7816	7836,1	0,00	0,26
				50x5								200x10			
Ta031	2980	2974	2974	2974	2975,6	0,00	0,05	Ta091	13149	13100	13100	**13098**	13155,9	-0,02	0,42
Ta032	3180	3171	3171	3171	3172,3	0,00	0,04	Ta092	13085	13003	13003	**13002**	13042,1	-0,01	0,30
Ta033	2995	2988	2988	2988	2993,6	0,00	0,19	Ta093	13183	13135	13135	**13094**	13175,2	-0,31	0,31
Ta034	3115	3111	3111	**3109**	3112,5	-0,06	0,05	Ta094	13097	13098	13097	**13044**	13106,4	-0,41	0,07
Ta035	3139	3138	3138	3138	3140,2	0,00	0,07	Ta095	13100	13097	13097	**13055**	13092,3	-0,32	-0,04
Ta036	3158	3158	3158	3158	3158,8	0,00	0,03	Ta096	12883	12855	12855	**12828**	12871,6	-0,21	0,13
Ta037	3005	3004	3004	**3002**	3005,4	-0,07	0,05	Ta097	13408	13342	13342	**13318**	13344,2	-0,18	0,02
Ta038	3042	3039	3039	**3036**	3040,3	-0,10	0,04	Ta098	13266	13225	13225	**13180**	13235,4	-0,34	0,08
Ta039	2889	2889	2889	2889	2889,8	0,00	0,03	Ta099	13080	13022	13022	**13012**	13055,4	-0,08	0,26
Ta040	3097	3094	3094	3094	3098,3	0,00	0,14	Ta100	13193	13119	13119	**13112**	13147,9	-0,05	0,22
				50x10								200x20			
Ta041	3611	3605	3605	3605	3609,9	0,00	0,14	Ta101	14192	14484	14192	14494	14533,2	2,08	2,35
Ta042	3470	3470	3470	3470	3474	0,00	0,12	Ta102	14714	14682	14682	**14675**	14689,6	-0,05	0,05
Ta043	3465	3465	3465	3465	3466,1	0,00	0,03	Ta103	14831	14776	14776	**14771**	14818,8	-0,03	0,29
Ta044	3650	3643	3643	3643	3648,8	0,00	0,16	Ta104	14801	14694	14694	14734	14764,6	0,27	0,48
Ta045	3582	3582	3582	3614	3619,4	0,89	1,03	Ta105	14609	14547	14547	14571	14600,9	0,16	0,37
Ta046	3571	3571	3571	**3570**	3574,3	-0,03	0,09	Ta106	14737	14718	14718	**14713**	14750,7	-0,03	0,22
Ta047	3667	3667	3667	3667	3673,2	0,00	0,17	Ta107	14770	14708	14708	14708	14762	0,00	0,37
Ta048	3549	3546	3546	3546	3551,5	0,00	0,15	Ta108	14823	14747	14747	**14745**	14795,8	-0,01	0,33
Ta049	3508	3508	3508	3508	3513,3	0,00	0,15	Ta109	14691	14643	14643	14643	14691	0,00	0,33
Ta050	3608	3603	3603	3603	3610	0,00	0,19	Ta110	14711	14683	14683	**14664**	14714,2	-0,13	0,21
				50x20								500x20			
Ta051	4479	4479	4479	4479	4485,8	0,00	0,15	Ta111	35380	35372	35372	**35369**	35449,8	-0,01	0,22
Ta052	4262	4262	4262	4262	4263,4	0,00	0,03	Ta112	35736	35743	35736	**35716**	35813,9	-0,06	0,22
Ta053	4261	4261	4261	**4259**	4260,8	-0,05	0,00	Ta113	35406	35445	35406	**35402**	35525	-0,01	0,33
Ta054	4339	4338	4338	4338	4340,6	0,00	0,06	Ta114	35030	35672	35030	35628	35730,7	1,68	1,96
Ta055	4249	4249	4249	4249	4254,2	0,00	0,12	Ta115	35417	35417	35417	**35379**	35486,5	-0,11	0,20
Ta056	4271	4271	4271	4271	4278,2	0,00	0,17	Ta116	35740	35739	35739	**35686**	35809,8	-0,15	0,20
Ta057	4291	4289	4289	4289	4297,5	0,00	0,20	Ta117	35299	35395	35299	**35289**	35379,5	-0,03	0,23
Ta058	4298	4298	4298	4298	4301,3	0,00	0,08	Ta118	35515	35525	35515	**35505**	35571	-0,03	0,16
Ta059	4304	4304	4304	4304	4305,5	0,00	0,03	Ta119	35268	35278	35268	**35248**	35362,3	-0,06	0,27
Ta060	4398	4398	4398	4398	4402,8	0,00	0,11	Ta120	35609	35623	35609	**35585**	35679,3	-0,07	0,20

a segunda coluna mostra o valor a melhor solução conhecida, as duas próximas colunas indicam a melhor e a média de 10 execuções do HyPRR para a instância no grupo, e as duas últimas colunas indicam o RPD mínimo e médio, respectivamente, quando os resultados são comparados com o apresentado na primeira coluna.

Tabela 4.16: Resultados obtidos nas instâncias utilizadas por Pan e Ruiz (2012)

Instância	BKS	$\text{HyPRR}_{\text{Best}}$	$\text{HyPRR}_{\text{Avg.}}$	RPD_{Best}	$\text{RPD}_{\text{Avg.}}$
200x5	11746	**11640**	11677,8	-0.91	-0.58
	11330	**11253**	11285,5	-0.68	-0.39
	11636	**11555**	11577,2	-0.70	-0.51
	11659	**11587**	11614	-0.62	-0.39
	11621	**11584**	11604,1	-0.32	-0.15
	11314	**11245**	11271,5	-0.61	-0.38
	11711	**11643**	11667,2	-0.58	-0.38
	11519	**11467**	11493	-0.45	-0.23
	11520	**11454**	11478,7	-0.58	-0.36
	11437	**11387**	11404,1	-0.44	-0.29
500x5	28441	**28418**	28456,1	-0.08	0.05
	28835	**28736**	28785,6	-0.34	-0.17
	28760	**28639**	28692	-0.42	-0.24
	28442	**28416**	28433,1	-0.09	-0.03
	28493	**28435**	28470,3	-0.20	-0.08
	28398	**28386**	28415,3	-0.04	0.06
	28239	**28152**	28204,7	-0.31	-0.12
	28462	**28454**	28494	-0.03	0.11
	28383	**28377**	28416	-0.02	0.12
	28341	**28261**	28305,7	-0.28	-0.12
500x10	31923	31923	32016	0.00	0.29
	32212	**32211**	32254,5	0.00	0.13
	32079	**32005**	32032,6	-0.23	-0.14
	32206	**32205**	32261,8	0.00	0.17
	31758	**31733**	31779,3	-0.08	0.07
	32123	**32122**	32258,3	0.00	0.42
	31856	**31816**	31873,3	-0.13	0.05
	32103	**32102**	32196,1	0.00	0.29
	31938	**31937**	31958,7	0.00	0.06
	32225	**32224**	32315,4	0.00	0.28

A partir do exposto, conclui-se que os resultados encontrados pelo HyPRR superam aqueles reportados por Tasgetiren et al. (2017), uma vez que foram encontrados 29 melhores limites superiores para as instâncias consideradas e 1 único empate. Além disso, o HyPRR foi capaz de obter uma média inferior ao melhor valor encontrado na literatura em 17 dos 30 casos considerados. Esse resultado pode ser justificado pela eficácia das vizinhanças do tipo *block insertion*, que possuem um grande impacto na qualidade das soluções de instâncias que consideram apenas 5 máquinas, conforme foi evidenciado na Seção 4.2.

Em suma, quando todas as 150 instâncias são consideradas, o método proposto foi capaz de encontrar melhores soluções do que as reportadas na literatura em 83 delas (55,33%), empatou em 59 outras (39,33%) e perdeu em apenas 8 (5,33%). No geral, conclui-se que o HyPRR é competitivo em todos os conjuntos de instâncias, encontrando soluções melhores ou iguais em 94,67% dos casos testados.

Capítulo 5

Considerações Finais

Neste trabalho, uma abordagem heurística foi proposta para o problema de *flow shop* com bloqueio e minimização do *makespan*, ou $F_m|block|C_{max}$. Neste problema, um conjunto de n tarefas deve ser processado em um conjunto de m máquinas organizadas em série. Não há estoque intermediário entre as máquinas. Assim, caso uma tarefa termine de ser processada na máquina i, só poderá sair dela caso a máquina $i+1$ não esteja processando nenhuma tarefa. O objetivo consiste em encontrar uma permutação π de tarefas, que indica a ordem em que cada uma delas será processada, de forma que o tempo total de processamento seja mínimo.

Dentro do exposto, as principais contribuições do trabalho foram: (i) um algoritmo populacional híbrido que combina um operador *ruin-and-recreate* com uma busca local baseada em *variable neighborhood descent* e a incorporação de um mecanismo de gerenciamento de diversidade empregado para evitar que o método fique preso em ótimos locais, além de melhorar sua convergência; (ii) aplicação de aceleração da busca na vizinhança *block insertion*, que é capaz de reduzir a complexidade computacional desse tipo de vizinhança de $O(n^3m)$ para $O(n^2m)$; (iii) introdução de diferentes limitantes inferiores para a vizinhança *swap* que podem ser calculados em O(1) e evitar a avaliação do movimento, que é feita em $O(nm)$; e (iv) utilização de um critério de desempate durante a fase de busca local, que é usado para evitar ótimos locais e melhorar a convergência do algoritmo para soluções de maior qualidade.

Experimentos computacionais foram realizados em 150 instâncias de *benchmark* e os testes conduzidos demonstraram a eficácia dos componentes utilizados no algoritmo. O limitante inferior para a vizinhança *swap* foi comparado com propriedades clássicas da literatura e foi observado uma grande discrepância com relação ao número de movimentos evitados entre os métodos comparados. Além disso, os limitantes utilizados foram capazes

de aumentar o número de iterações do algoritmo em até 32,71%, a depender do tamanho da instância considerada.

Com relação à vizinhança *block insertion*, os experimentos indicaram que a qualidade da solução tende a aumentar conforme o número de vizinhanças deste tipo aumenta até um certo limite, ou seja, conforme são considerados novos tamanhos de bloco de tarefas. Especialmente, esse comportamento foi mais evidenciado para os grupos de instâncias com 5 máquinas. Além disso, o critério de desempate também foi analisado, sendo observado que o método consegue obter, em média, soluções de maior qualidade quando ele é considerado.

Ainda, o operador *ruin-and-recreate* e a população também foram submetidos a testes computacionais. Diferentes versões do método proposto foram implementadas, sendo considerados operadores de *crossover*, solução única e/ou desconsideração da diversidade da população. Os testes indicaram que o método proposto é melhor do que todas as outras versões consideradas.

Por fim, foi realizada uma comparação com diferentes métodos considerados estado-da-arte para o problema estudado. Em todos os casos analisados, o método proposto no presente trabalho conseguiu melhores resultados em todos os grupos de instância, com exceção do 100x20 quando comparado ao trabalho de Tasgetiren et al. (2017). Apesar disso, a diferença observada pode ser considerada mínima.

De forma geral, o método foi capaz de encontrar melhores soluções do que as reportadas na literatura em 83 instâncias (55,33%), empatou em 59 outras (39,33%) e perdeu em apenas 8 (5,33%), quando todos os 150 casos são considerados. Assim, conclui-se que o HyPRR é competitivo em todos os conjuntos de instâncias, encontrando soluções melhores ou iguais em 94,67% dos casos testados.

No contexto da engenharia de produção, o problema possui diversas aplicações, conforme evidenciado anteriormente. Nesse sentido, o método proposto surge como possibilidade de resolução de problemas do tipo *blocking flow shop* em diversos setores, como linhas de produção *just-in-time*, indústrias química e farmacêutica, dentre outros. Assim, o algoritmo pode ser empregado para obtenção de soluções de alta qualidade em um curto período de tempo, auxiliando no processo de tomada de decisão e, por consequência, atendendo possíveis demandas de mercado em tempo hábil.

Como sugestão de trabalhos futuros, a aplicação do HyPRR para outras variantes do problema pode ser considerada. A incorporação de outras restrições além do bloqueio, como *due dates* e tempos de *setup*, tornam o problema mais desafiador, mas mais próximo

de casos práticos. Nesse sentido, a adaptação dos métodos de aceleração e limitantes inferiores utilizados neste trabalho para outras variantes surge como possibilidade de estudo. Além disso, visto que a busca local é a grande responsável pelo aumento do tempo de execução do método, a elaboração de métodos que sejam capazes de identificar movimentos promissores pode ser fundamental para que haja uma convergência mais rápida para soluções de alta qualidade, principalmente em instâncias de grande porte.

Referências

ABADI, I. K. A new algorithm for minimizing makespan, c max, in blocking flow-shop problem through slowing down the operations. *Journal of the Operational Research Society*, Taylor & Francis, v. 58, n. 1, p. 134–140, 2007.

BAO, Y.; ZHENG, L.; JIANG, H. An improved hs algorithms for the blocking flow shop scheduling problems. In: IEEE. *2012 International Conference on Computer Science and Information Processing (CSIP)*. [S.l.], 2012. p. 1289–1291.

BAUTISTA, J. et al. Solving the $f_m|block|c_{max}$ problem using bounded dynamic programming. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 25, n. 6, p. 1235–1245, 2012.

CARAFFA, V. et al. Minimizing makespan in a blocking flowshop using genetic algorithms. *International Journal of Production Economics*, Elsevier, v. 70, n. 2, p. 101–115, 2001.

CHEN, H. et al. A hybrid differential evolution algorithm for a two-stage flow shop on batch processing machines with arbitrary release times and blocking. *International Journal of Production Research*, Taylor & Francis, v. 52, n. 19, p. 5714–5734, 2014.

CHOWDHURY, A. et al. A novel genetic algorithm to solve travelling salesman problem and blocking flow shop scheduling problem. *International Journal of Bio-Inspired Computation*, Inderscience Publishers Ltd, v. 5, n. 5, p. 303–314, 2013.

COMPANYS, R.; MATEO, M. Different behaviour of a double branch-and-bound algorithm on fm| prmu| cmax and fm| block| cmax problems. *Computers & Operations Research*, Elsevier, v. 34, n. 4, p. 938–953, 2007.

COMPANYS, R.; RIBAS, I.; MATEO, M. Note on the behaviour of an improvement heuristic on permutation and blocking flow-shop scheduling. *International Journal of Manufacturing Technology and Management*, Inderscience Publishers, v. 20, n. 1-4, p. 331–357, 2010.

DAVENDRA, D.; BIALIC-DAVENDRA, M. Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. *International Journal of Production Research*, Taylor & Francis, v. 51, n. 8, p. 2200–2218, 2013.

DAVENDRA, D. et al. Scheduling the flow shop with blocking problem with the chaos-induced discrete self organising migrating algorithm. In: *ECMS*. [S.l.: s.n.], 2013. p. 386–392.

DAVENDRA, D. et al. Clustered enhanced differential evolution for the blocking flow shop scheduling problem. *Central European Journal of Operations Research*, Springer, v. 20, n. 4, p. 679–717, 2012.

DING, J.-Y. et al. New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. *International Journal of Production Research*, Taylor & Francis, v. 54, n. 16, p. 4759–4772, 2016.

DUAN, J.-H. et al. Scheduling the lot-streaming flow shop problem using a shuffled frog-leaping algorithm. In: IEEE. *2010 Sixth International Conference on Natural Computation*. [S.l.], 2010. v. 8, p. 4263–4266.

FERNANDEZ-VIAGAS, V.; LEISTEN, R.; FRAMINAN, J. M. A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. *Expert Systems with Applications*, Elsevier, v. 61, p. 290–301, 2016.

GILMORE, P. C.; GOMORY, R. E. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations research*, INFORMS, v. 12, n. 5, p. 655–679, 1964.

GRABOWSKI, J.; PEMPERA, J. Sequencing of jobs in some production system. *European journal of operational research*, Elsevier, v. 125, n. 3, p. 535–550, 2000.

GRABOWSKI, J.; PEMPERA, J. The permutation flow shop problem with blocking. a tabu search approach. *Omega*, Elsevier, v. 35, n. 3, p. 302–311, 2007.

GRAHAM, R. L. et al. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In: *Discrete Optimization II*. [S.l.]: Elsevier, 1979, (Annals of Discrete Mathematics, v. 5). p. 287–326.

HALL, N. G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, INFORMS, v. 44, n. 3, p. 510–525, 1996.

HAN, Y. et al. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *International Journal of Production Research*, Taylor & Francis, v. 54, n. 22, p. 6782–6797, 2016.

HAN, Y.-Y.; GONG, D.; SUN, X. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. *Engineering Optimization*, Taylor & Francis, v. 47, n. 7, p. 927–946, 2015.

HAN, Y.-Y. et al. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 60, n. 9-12, p. 1149–1159, 2012.

JARBOUI, B. et al. An estimation of distribution algorithm for minimizing the makespan in blocking flowshop scheduling problems. In: *Computational Intelligence in Flow Shop and Job Shop Scheduling*. [S.l.]: Springer, 2009. p. 151–167.

LAGEWEG, B.; LENSTRA, J. K.; KAN, A. R. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, INFORMS, v. 26, n. 1, p. 53–67, 1978.

LEVNER, E. Optimal planning of parts' machining on a number of machines. *Automatin and Remote Control*, v. 12, n. 12, p. 1972–1978, 1969.

LIANG, J. et al. Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 55, n. 5-8, p. 755–762, 2011.

LIN, S.-W.; YING, K.-C. Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. *Omega*, Elsevier, v. 41, n. 2, p. 383–389, 2013.

LIU, S. Q.; KOZAN, E. Scheduling a flow shop with combined buffer conditions. *International Journal of Production Economics*, Elsevier, v. 117, n. 2, p. 371–380, 2009.

MARTINEZ, S. et al. Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, Elsevier, v. 169, n. 3, p. 855–864, 2006.

MCCORMICK, S. T. et al. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, INFORMS, v. 37, n. 6, p. 925–935, 1989.

MECLER, J.; SUBRAMANIAN, A.; VIDAL, T. A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, Elsevier, v. 127, p. 105153, 2021.

MIYATA, H. H.; NAGANO, M. S. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, Elsevier, v. 137, p. 130–156, 2019.

MOSLEHI, G.; KHORASANIAN, D. A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion. *Computers & Operations Research*, Elsevier, v. 52, p. 260–268, 2014.

NAWAZ, M.; JR, E. E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, Elsevier, v. 11, n. 1, p. 91–95, 1983.

OZOLINS, A. Improved bounded dynamic programming algorithm for solving the blocking flow shop problem. *Central European Journal of Operations Research*, Springer, v. 27, n. 1, p. 15–38, 2019.

PAN, Q.-K.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, Elsevier, v. 222, n. 1, p. 31–43, 2012.

PAN, Q.-K.; WANG, L. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, Elsevier, v. 40, n. 2, p. 218–229, 2012.

PAN, Q.-k. et al. A high performing memetic algorithm for the flowshop scheduling problem with blocking. *IEEE Transactions on Automation Science and Engineering*, IEEE, v. 10, n. 3, p. 741–756, 2012.

PINEDO, M. L. *Scheduling: theory, algorithms, and systems*. [S.l.]: Springer, 2018.

PITTY, S. S.; KARIMI, I. A. Novel milp models for scheduling permutation flowshops. *Chemical Product and Process Modeling*, De Gruyter, v. 3, n. 1, 2008.

PRASAD, S. D.; RAJENDRAN, C.; CHETTY, O. K. A genetic algorithmic approach to multi-objective scheduling in a kanban-controlled flowshop with intermediate buffer and transport constraints. *The International Journal of Advanced Manufacturing Technology*, Springer, v. 29, n. 5, p. 564–576, 2006.

REDDI, S.; RAMAMOORTHY, C. On the flow-shop sequencing problem with no wait in process. *Journal of the Operational Research Society*, Springer, v. 23, n. 3, p. 323–331, 1972.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, Elsevier, v. 39, n. 3, p. 293–301, 2011.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. A competitive variable neighbourhood search algorithm for the blocking flow shop problem. *European Journal of Industrial Engineering*, Inderscience Publishers Ltd, v. 7, n. 6, p. 729–754, 2013.

RIBAS, I.; COMPANYS, R.; TORT-MARTORELL, X. An efficient discrete artificial bee colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, Elsevier, v. 42, n. 15-16, p. 6155–6167, 2015.

RÖCK, H. Some new results in flow shop scheduling. *Zeitschrift für Operations Research*, Springer, v. 28, n. 1, p. 1–16, 1984.

RONCONI, D. P. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, Elsevier, v. 87, n. 1, p. 39–48, 2004.

RONCONI, D. P. A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. *Annals of Operations Research*, Springer, v. 138, n. 1, p. 53–65, 2005.

RONCONI, D. P.; ARMENTANO, V. A. Lower bounding schemes for flowshops with blocking in-process. *Journal of the Operational Research Society*, Taylor & Francis, v. 52, n. 11, p. 1289–1297, 2001.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, Elsevier, v. 34, n. 5, p. 461–476, 2006.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research*, Elsevier, v. 177, n. 3, p. 2033–2049, 2007.

SHAO, Z.; PI, D.; SHAO, W. Self-adaptive discrete invasive weed optimization for the blocking flow-shop scheduling problem to minimize total tardiness. *Computers & Industrial Engineering*, Elsevier, v. 111, p. 331–351, 2017.

SHAO, Z.; PI, D.; SHAO, W. Estimation of distribution algorithm with path relinking for the blocking flow-shop scheduling problem. *Engineering Optimization*, Taylor & Francis, v. 50, n. 5, p. 894–916, 2018.

SHAO, Z.; PI, D.; SHAO, W. A multi-objective discrete invasive weed optimization for multi-objective blocking flow-shop scheduling problem. *Expert Systems with Applications*, Elsevier, v. 113, p. 77–99, 2018.

SHAO, Z.; PI, D.; SHAO, W. A novel discrete water wave optimization algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, Elsevier, v. 40, p. 53–75, 2018.

SHAO, Z. et al. An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem. *Engineering Applications of Artificial Intelligence*, Elsevier, v. 78, p. 124–141, 2019.

SUHAMI, I.; MAH, R. S. An implicit enumeration scheme for the flowshop problem with no intermediate storage. *Computers & Chemical Engineering*, Elsevier, v. 5, n. 2, p. 83–91, 1981.

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, Elsevier, v. 47, n. 1, p. 65–74, 1990.

TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, Elsevier, v. 64, n. 2, p. 278–285, 1993.

TASGETIREN, M. F. et al. Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Computers & Operations Research*, Elsevier, v. 77, p. 111–126, 2017.

TASGETIREN, M. F. et al. A populated local search with differential evolution for blocking flowshop scheduling problem. In: IEEE. *2015 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2015. p. 2789–2796.

TOUMI, S. et al. Branch-and-bound algorithm for solving blocking flowshop scheduling problems with makespan criterion. *International Journal of Mathematics in Operational Research*, Inderscience Publishers (IEL), v. 10, n. 1, p. 34–48, 2017.

VIDAL, T. et al. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, INFORMS, v. 60, n. 3, p. 611–624, 2012.

VIDAL, T. et al. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, Elsevier, v. 234, n. 3, p. 658–673, 2014.

WANG, C. et al. A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. *Computers & operations research*, Elsevier, v. 39, n. 11, p. 2880–2887, 2012.

WANG, L. et al. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research*, Elsevier, v. 37, n. 3, p. 509–520, 2010.

WANG, L.; PAN, Q.-K.; TASGETIREN, M. F. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering*, Elsevier, v. 61, n. 1, p. 76–83, 2011.

WANG, X.; TANG, L. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Applied Soft Computing*, Elsevier, v. 12, n. 2, p. 652–662, 2012.

ZHANG, C. et al. An effective vnssa algorithm for the blocking flowshop scheduling problem with makespan minimization. In: IEEE. *2015 International Conference on Advanced Mechatronic Systems (ICAMechS)*. [S.l.], 2015. p. 86–89.