Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação
Programa de Pós-Graduação em Ciência da Computação

Algoritmos Multiobjetivos para o Problema de Sequenciamento de Tarefas em Uma Máquina com Tempo de Preparação Dependente da Sequência e da Família

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto, como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Aluno: Marcelo Ferreira Rego

Orientador: Marcone Jamilson Freitas Souza

Co-orientador: José Elias Claudio Arroyo

Ouro Preto – MG
2013

Algoritmos Multiobjetivos para o Problema de Sequenciamento de Tarefas em Uma Máquina com Tempo de Preparação Dependente da Sequência e da Família

Marcelo Ferreira Rego

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto, como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Aprovada por:

Marcone Jamilson Freitas Souza, D.Sc. / DECOM-UFOP
(Orientador)

José Elias Claudio Arroyo, D.Sc. / DPI-UFV
(Co-orientador)

André Gustavo dos Santos, D.Sc. / DPI-UFV

Haroldo Gambini Santos, D.Sc. / DECOM-UFOP

Ouro Preto, 14 de março de 2013.

Dedico este trabalho ao meu pai (in memoriam) e à minha mãe

Agradecimentos

Primeiramente, agradeço à Deus pela vida e por tudo de bom que tem me proporcionado.

À minha mãe, meu irmão, minha irmã, minha avó, e à toda a minha família que é um recanto de amor, pelo apoio e o incentivo em toda a minha vida.

À minha namorada Caroline, pessoa que amo e que me acompanhou por todo esse percurso, sempre disposta a me ajudar.

Ao professor Marcone pela excelente orientação para o desenvolvimento deste trabalho, pela amizade e incentivo, e principalmente pelos ensinamentos de vida.

Ao professor José Elias Claudio Arroyo pela sua valiosa orientação que enriqueceu muito o trabalho.

Ao Ricardo Brasil, professor Valter Carvalho e os colegas da Divisão de Tecnologia da Informação da UFVJM que me ajudaram a conseguir o afastamento do meu trabalho.

À UFVJM que me concedeu um período de afastamento para que eu pudesse concretizar mais esse objetivo em minha vida.

À UFOP e ao programa de pós-graduação em Ciência da Computação pela oportunidade de me qualificar.

Aos ótimos professores do DECOM que contribuíram muito para a minha formação e foram a base para o desenvolvimento deste trabalho.

Aos colegas do mestrado pelo companheirismo.

À dona Hercília que me proporcionou um lar em Ouro Preto.

Ao meu amigo Rafael por sempre me incentivar.

Resumo

Este trabalho trata do Problema de Sequenciamento de Tarefas em Uma Máquina com Tempo de Preparação Dependente da Sequência e da Família. Nesse problema, um conjunto de tarefas devem ser processadas por uma máquina, sendo que antes da execução de cada tarefa é necessário um tempo para preparar a máquina, o qual é definido de acordo com a sequência e a família da tarefa. Desta forma, o tempo de preparação da máquina é requerido, apenas, para executar duas tarefas consecutivas que pertencem a famílias diferentes. Consideram-se os objetivos de minimizar o *makespan* e o atraso total ponderado. Para resolvê-lo, foram analisados sete algoritmos de otimização multiobjetivo. O primeiro é o *Multi-objective Variable Neighborhood Search* (MOVNS), que é um método de otimização multiobjetivo baseado na metaheurística *Variable Neighborhood Search* (VNS). O segundo e o terceiro são duas variantes do MOVNS encontradas na literatura, denominadas MOVNS_Ottoni e MOVNS_Arroyo, que consistem em adicionar um procedimento de intensificação no MOVNS. O quarto é o *Pareto Iterated Local Search* (PILS), que é um algoritmo multiobjetivo de busca local com características semelhantes à metaheurística *Iterated Local Search* (ILS). O quinto é uma variante do PILS proposta neste trabalho, denominada PILS1, em que um novo procedimento de perturbação é desenvolvido. O sexto e o sétimo são o *Nondominated Sorting Genetic Algorithm* II (NSGA-II) e o *Strength Pareto Evolutionary Algorithm* 2 (SPEA2), os quais são métodos de otimização baseados no processo de evolução natural para problemas multiobjetivos. Entre os sete algoritmos, cinco são de busca local: MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS e PILS1, e outros dois são de busca populacional: NSGA-II e SPEA2. Esses algoritmos foram comparados em relação às métricas de cardinalidade, distância média, distância máxima, diferença de hipervolume e *epsilon*. Os resultados computacionais realizados em instâncias-teste geradas aleatoriamente mostraram que o algoritmo PILS1 é estatisticamente superior a todos os outros algoritmos em relação às métricas cardinalidade, distância média, diferença de hipervolume e métrica *epsilon*, em termos de resultados médios. O PILS1 conseguiu também o melhor resultado médio para a métrica distância máxima; entretanto, a partir da análise estatística não foi possível afirmar que a diferença observada entre ele o NSGA-II era significativa.

PALAVRAS-CHAVE: Problema de sequenciamento em única máquina, Otimização multiobjetivo, Pareto Iterated Local Search, Multi-objective Variable Neighborhood Search, Nondominated Sorting Genetic Algorithm, Strength Pareto Evolutionary Algorithm

Abstract

This work addresses the Single Machine Scheduling Problem with Sequence Dependent Family Setup Times. In this problem, there is a set of jobs to be processed by a single machine, and before the execution of each job a time to prepare the machine is required, which is defined according to the sequence and the family of the job. Thus, the setup time is required only to perform two consecutive jobs belonging to different families. Two objectives were considered: the minimization of the makespan and the total weighted tardiness. In order to solve this problem, seven multi-objective algorithms were analyzed. The first one is the Multi-objective Variable Neighborhood Search (MOVNS), which is a multi-objective optimization method based on Variable Neighborhood Search (VNS). The second and third ones are two variants of MOVNS found in the literature, here named MOVNS_Ottoni and MOVNS_Arroyo, which consists in adding an intensification procedure in the MOVNS. The fourth one is the Pareto Iterated Local Search (PILS), which is a multi-objective local search algorithm with similar features to the metaheuristics Iterated Local Search (ILS). The fifth one is a variant of PILS proposed in this work, called PILS1, in which a new perturbation procedure is developed. The sixth and seventh ones are the Nondominated Sorting Genetic Algorithm II (NSGA-II) and the Strength Pareto Evolutionary Algorithm 2 (SPEA2), which are optimization methods based on the process of natural evolution for multi-objective problems. Among the seven algorithms, five are based on local search: MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS and PILS1, and two other are population-based: NSGA-II and SPEA2. These algorithms were compared according to cardinality, average distance, maximum distance, hypervolume difference and epsilon metrics. The Computational results realized over test instances randomly generated show that PILS1 is statistically better than all other algorithms in relation to the cardinality, average distance, difference of hypervolume and epsilon metrics, in terms of average results. The PILS1 produced also the best average result for the metric maximum distance; however, from the statistical analysis is not possible to affirm that the difference between it and the NSGA-II is significant.

Keywords: Single machine scheduling, Multi-objective optimization, Pareto Iterated Local Search, Multi-objective Variable Neighborhood Search, Nondominated Sorting Genetic Algorithm, Strength Pareto Evolutionary Algorithm

Sumário

Lista de Figuras **ix**

Lista de Algoritmos **x**

Lista de Tabelas **xi**

Lista de Abreviaturas e Siglas **xii**

1 Introdução **1**
1.1 Objetivos . . . 3
1.1.1 Objetivo Geral . . . 3
1.1.2 Objetivos Específicos . . . 3
1.2 Motivação . . . 4
1.3 Estrutura do Trabalho . . . 4

2 Caracterização do Problema **5**

3 Revisão Bibliográfica **7**
3.1 Otimização Multiobjetivo . . . 7
3.1.1 Definição . . . 7
3.1.2 Métodos Clássicos . . . 9
3.2 Métricas de Avaliação de Desempenho . . . 11
3.2.1 Métrica de Cardinalidade . . . 11
3.2.2 Métricas de Distância . . . 12
3.2.3 Métrica hipervolume . . . 12
3.2.4 Métrica *epsilon* . . . 13
3.3 Algoritmos Multiobjetivos . . . 14
3.3.1 MOVNS . . . 15
3.3.2 MOVNS_Ottoni . . . 16
3.3.3 MOVNS_Arroyo . . . 16
3.3.4 PILS . . . 18
3.3.5 NSGA-II . . . 20
3.3.6 SPEA2 . . . 22
3.4 Trabalhos Relacionados . . . 26
3.4.1 *Makespan* . . . 26
3.4.2 Atraso Total . . . 26
3.4.3 Famílias de Tarefas . . . 27
3.4.4 Problemas de Otimização Multiobjetivo . . . 28

4 Metodologia **29**
4.1 Representação da Solução . . . 29
4.2 Geração da Solução Inicial . . . 29
4.3 Estruturas de Vizinhança . . . 30
4.3.1 Vizinhança de Inserção . . . 30
4.3.2 Vizinhança de Troca . . . 31
4.4 Perturbação . . . 31
4.5 Seleção . . . 33
4.5.1 Seleção para Reprodução . . . 33
4.5.2 Seleção Populacional . . . 33
4.6 Reprodução . . . 33
4.6.1 Cruzamento . . . 33
4.6.2 Mutação . . . 39
4.7 Algoritmos Implementados . . . 40
4.7.1 MOVNS . . . 40
4.7.2 MOVNS_Ottoni . . . 41
4.7.3 MOVNS_Arroyo . . . 41
4.7.4 PILS . . . 41
4.7.5 PILS1 . . . 41
4.7.6 NSGA-II . . . 42
4.7.7 SPEA2 . . . 43
4.8 Normalização . . . 43
4.9 Parâmetros . . . 44

5 Resultados Computacionais **45**
5.1 Instâncias . . . 45
5.2 Experimentos Computacionais . . . 46
5.3 Análise Estatística . . . 51

6 Conclusões e Trabalhos Futuros **56**

A Publicações **58**

B Resultados Completos **59**

Referências **60**

Lista de Figuras

1.1 Produtos 2

2.1 Sequência relativa ao exemplo 6

3.1 Mapeamento das Variáveis de Decisão 8
3.2 Função Ponderada 10
3.3 Hipervolume 13
3.4 Diferença do Hipervolume 13
3.5 Epsilon 14
3.6 Perturbação Original 20

4.1 Representação da Solução 29
4.2 Movimento de inserção para $x < y$ 30
4.3 Movimento de inserção para $x > y$ 31
4.4 Movimento de troca 31
4.5 Nível 1 da Perturbação Proposta 32
4.6 Nível 2 da Perturbação Proposta 32
4.7 Nível p da Perturbação Proposta 32
4.8 Operador OX 35
4.9 Operador LOX 37
4.10 Mapeamento do primeiro elemento 38
4.11 Mapeamento do segundo elemento 39

Lista de Algoritmos

1 MOVNS 15
2 Intensificação de Ottoni et al. (2011) 17
3 Intensificação de Arroyo et al. (2011) 18
4 PILS 19
5 Fast Non-Dominated Sorting 21
6 *crowding distance* 22
7 NSGA-II 23
8 SPEA2 24

9 PILS1 42

Lista de Tabelas

2.1 Instância do problema 6
2.2 Tempo de Preparação entre as Famílias 6

3.1 Relações de Dominância 9

4.1 Valores dos melhores parâmetros 44

5.1 Número de soluções não-dominadas 46
5.2 Comparação dos algoritmos com relação à métrica de Cardinalidade ($C1_R$) 47
5.3 Comparação dos algoritmos com relação à métrica Distância Média ($D1_R$) 48
5.4 Comparação dos algoritmos com relação à métrica Distância Máxima ($D_{\max}$) 48
5.5 Comparação dos algoritmos com relação à métrica Diferença de Hipervolume (H^-) 49
5.6 Comparação dos algoritmos com relação à métrica *Epsilon*($I^1_\epsilon(.)$) . . . 50
5.7 Resumo da comparação dos algoritmos por métrica em relação aos resultados médios 50
5.8 Resumo da comparação dos algoritmos por métrica em relação aos melhores resultados encontrados 51
5.9 Teste Kruskal-Wallis: cardinalidade ($C1_R$) 52
5.10 Teste Kruskal-Wallis: distância média ($D1_R$) 52
5.11 Teste Kruskal-Wallis: distância máxima ($D_{\max}$) 53
5.12 Teste Kruskal-Wallis: diferença de hipervolume (H^-) 53
5.13 Teste Kruskal-Wallis: métrica *epsilon* ($I^1_\epsilon(.)$) 54
5.14 Teste Kruskal-Wallis de comparação múltipla 55

Lista de Abreviaturas e Siglas

ILS *Iterated Local Search*

LOX *Linear Order Crossover*

MOVNS *Multi-objective Variable Neighborhood Search*

NSGA-II *Nondominated Sorting Genetic Algorithm* II

OX *Order Crossover*

PMX *Partial Mapped Crossover*

PILS *Pareto Iterated Local Search*

SPEA2 *Strength Pareto Evolutionary Algorithm* 2

VNS *Variable Neighborhood Search*

Capítulo 1

Introdução

No atual ambiente competitivo industrial, em que nota-se um aumento da concorrência entre as empresas, coloca-se o sistema de planejamento, programação e controle da produção como uma área de decisão prioritária para as organizações (Pedroso e Corrêa, 1996). Neste contexto, realizar o planejamento do sequenciamento de tarefas implica em melhorar o controle do fluxo da cadeia produtiva. Esse planejamento de tarefas deve ser feito de modo a permitir que a empresa consiga cumprir os prazos de entrega que foram acordados, sob o risco de perder clientes em caso de descumprimento. Além disso, deseja-se programar as atividades de tal maneira a utilizar os recursos disponíveis de uma forma mais eficiente possível. Os benefícios de um bom planejamento são o aumento da produtividade organizacional, redução dos custos de produção, agilidade no processo de tomada de decisão e aumento da eficiência organizacional.

O problema de realizar o sequenciamento de tarefas está presente em diversas áreas, como na indústria, na manufatura, no gerenciamento de processos de um computador, dentre outras.

Os primeiros estudos, de maior relevância, feitos acerca desses problemas são da década de 50 e, desde então, as abordagens se aproximaram cada vez mais de aplicações industriais, aumentando assim a complexidade dos problemas. Nessa época, os estudos eram feitos com problemas relativamente simples, o que possibilitou o desenvolvimento de algoritmos capazes de fornecer uma solução ótima. Com o passar do tempo as abordagens foram se tornando mais sofisticadas e na década de 70 muitos problemas foram provados ser NP-difíceis. Após os anos 80, os estudos tomaram duas direções, de um lado abordagens utilizando algoritmos aproximados e do outro, estudos de métodos heurísticos (Leung, 2004).

No processo de planejamento e operação industrial, é usual que mais de uma função objetivo seja desejável e essencial para uma análise mais precisa do problema. No entanto, por uma questão de simplicidade a maioria das abordagens modelam o problema com um único objetivo.

Observa-se que em problemas de sequenciamento, existem muitos objetivos diferentes e conflitantes, dos quais, segundo Pinedo (2008), pode-se destacar:

- *Makespan*, que consiste em minimizar o maior tempo de término de um conjunto de tarefas. A minimização do *makespan* implica em uma boa utilização da máquina;

- Soma ponderada dos tempos de término de cada tarefa ou fluxo total ponderado, que mede o estoque em processamento;
- Atraso total ponderado, que consiste em determinar um sequenciamento cuja soma ponderada dos atrasos das tarefas seja mínima;
- Adiantamento total ponderado, que consiste em determinar um sequenciamento com o menor valor para adiantamento total ponderado;
- Soma ponderada de atrasos e adiantamentos, que consiste em determinar um sequenciamento com um valor mínimo para o adiantamento e atraso das tarefas. Esse objetivo está relacionado à filosofia *Just-In-Time* de produção, na qual uma tarefa deve ser finalizada na sua data de entrega.
- *Lateness* máximo, que consiste em minimizar a maior diferença entre o tempo de término e a data de entrega das tarefas. A diferença entre o *Lateness* e o atraso é que aquele pode assumir valores negativos enquanto que esse não.

Neste trabalho discute-se o problema de sequenciamento em uma máquina em que o tempo de preparação da máquina dependente da sequência e da família das tarefas. O agrupamento de tarefas em famílias ocorre, por exemplo, na siderurgia. Em Bustamante (2007), o autor mostra um processo de fabricação de produtos siderúrgicos (cantoneira, vergalhão, barra chata etc.) no setor de laminação, em que as tarefas são agrupadas em famílias de acordo com as semelhanças dos produtos. No caso, os produtos da mesma família são aqueles que diferenciam entre si apenas pela espessura. Em tais circunstâncias, o tempo de preparação é tão pequeno e insignificante em relação ao tempo de processamento das tarefas, que é comum considerá-lo igual a zero. A vantagem, portanto, de se fazer esse agrupamento, é que as tarefas pertencentes a uma mesma família, quando processadas em sequência, não requerem tempo de preparação.

Na Figura 1.1 apresenta-se uma aplicação prática do problema de sequenciamento com famílias de tarefas. Neste caso, existem diversos produtos, cada um com tamanho, espessura e seção diferentes. Produtos que pertencem a uma mesma família possuem a mesma seção e o mesmo tamanho, ou seja, eles se diferenciam apenas pela espessura.

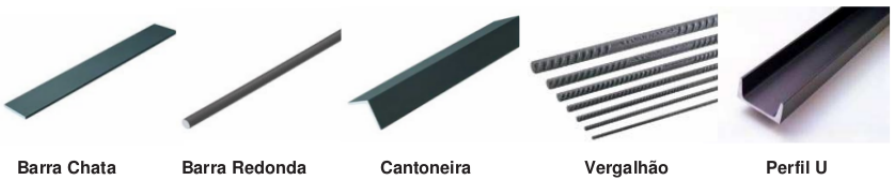

Figura 1.1: Produtos

No problema tratado são considerados dois critérios de otimização a serem atendidos, a saber: a minimização do *makespan* e a minimização do atraso total ponderado. Isto é, ao invés de se procurar uma solução que satisfaça a um ou outro

objetivo isoladamente, a meta, neste caso, é obter um conjunto de soluções não-dominadas, de modo que cada solução pertencente a esse conjunto não seja pior que nenhuma outra nos dois objetivos simultaneamente.

Em termos de algoritmos para solução de problemas de otimização multiobjetivo, foi apenas em meados dos anos 80 que uma quantidade considerável de pesquisas neste campo começaram a ser feitas. A partir daí, surgiram os primeiros algoritmos multiobjetivos, a maioria deles baseados em algoritmos evolutivos. Atualmente, existem diversos algoritmos multiobjetivos propostos na literatura (Coello e Lamont, 2004).

Tendo em vista a complexidade computacional dos problemas de sequenciamento, os métodos mais utilizados para resolvê-los são os metaheurísticos. Revisões da literatura têm mostrado que métodos inspirados em procedimentos evolutivos, tais como os algoritmos *Nondominated Sorting Genetic Algorithm II* - NSGA-II (Deb et al., 2002) e *Strength Pareto Approach* – SPEA2 (Zitzler e Thiele, 1999), estão entre os mais utilizados em otimização multiobjetivo. Entretanto, recentemente surgiram na literatura relatos de aplicações bem sucedidas de métodos multiobjetivos baseados em busca local, tais como *Multi-objective Variable Neighborhood Search* – MOVNS (Geiger, 2004) e *Pareto Iterated Local Search* – PILS (Geiger, 2009), conforme mostram os trabalhos de Arroyo et al. (2011), Ottoni et al. (2011) e Minella et al. (2010). Neste último, por exemplo, mostra-se a superioridade do PILS sobre o SPEA2.

Em vista desse bom desempenho de algoritmos baseados em busca local na solução de problemas similares ao abordado, no presente trabalho são implementadas adaptações dos algoritmos multiobjetivos MOVNS e PILS na solução do problema de sequenciamento em estudo. Os algoritmos MOVNS de Arroyo et al. (2011) e Ottoni et al. (2011) foram adaptados para resolver o problema, dando origem às variantes MOVNS_Ottoni e MOVNS_Arroyo; enquanto que um nova variante, denominada PLS1, foi desenvolvida a partir da proposição de um novo procedimento de perturbação para o PILS. Para verificar a eficiência desses algoritmos de busca local, também foram adaptados ao problema os algoritmos evolutivos NSGA-II e SPEA2.

1.1 Objetivos

Nesta seção são apresentados os principais objetivos deste trabalho.

1.1.1 Objetivo Geral

O objetivo do presente trabalho é desenvolver um algoritmo multiobjetivo eficiente para resolver o Problema de Sequenciamento de Tarefas em uma Máquina com Tempo de Preparação Dependente da Sequência e da Família, tendo como objetivos a minimização do *makespan* e do atraso total ponderado.

1.1.2 Objetivos Específicos

São objetivos específicos:

- Investigar os principais trabalhos da literatura que tratam de algoritmos multiobjetivos, assim como aqueles que tratam problemas correlatos ao abordado.
- Implementar algoritmos multiobjetivos baseados em busca local e em busca populacional para resolver o problema tratado.
- Criar um conjunto de instâncias para o problema abordado.
- Realizar experimentos com os algoritmos implementados.
- Comparar os resultados obtidos por cada algoritmo, segundo as métricas de cardinalidade, distância média, distância máxima, hipervolume e *epsilon*.
- Verificar a diferença entre os resultados dos algoritmos implementados a partir do uso de testes estatísticos.
- Publicar os resultados da pesquisa em eventos e periódicos.

1.2 Motivação

O interesse no estudo de problemas de sequenciamento de tarefas ocorre devido a pelo menos dois aspectos. O primeiro é o interesse prático, uma vez que há diversas aplicações dessa classe de problemas na indústria, como, por exemplo, nas áreas têxtil (Gendreau et al., 2001), eletrônica (Kim et al., 2002) e siderúrgica (Tang e Wang, 2009). Outro aspecto que releva o estudo desse tipo de problema, é o interesse teórico, visto que a maioria dos problemas de sequenciamento pertencem à classe NP-difícil (Brucker, 2007).

De nosso conhecimento, não foram encontradas referências na literatura sobre o estudo do problema abordado. Por outro lado, problemas de sequenciamento de tarefas podem envolver diversos objetivos, no entanto, grande parte das pesquisas nesse campo consideram apenas um único objetivo, ou quando mais de um é considerado, geralmente, define-se um único objetivo representado pela combinação linear dos objetivos envolvidos, ou seja, o problema é tratado normalmente em uma abordagem mono-objetivo. Ao contrário desse procedimento comum, neste trabalho o problema é tratado com algoritmos de otimização multiobjetivo. Observou-se que geralmente os problemas multiobjetivos são tratados por meio de algoritmos baseados em busca populacional; entretanto, há relatos na literatura de aplicações bem sucedidas de algoritmos baseados em busca local, por isso, optou-se, neste trabalho, por desenvolver e comparar o desempenho dessas duas classes de algoritmos.

1.3 Estrutura do Trabalho

O restante deste trabalho está organizado como segue. No capítulo 2 são descritas as características do problema abordado e no capítulo 3 apresenta-se a revisão bibliográfica, enquanto que no 4 está a metodologia utilizada para o desenvolvimento do trabalho. No capítulo 5 são apresentados e analisados os resultados dos testes realizados. O capítulo 6 conclui o trabalho e aponta sugestões para trabalhos futuros.

Capítulo 2

Caracterização do Problema

O problema abordado pode ser definido da seguinte forma: existe um conjunto $J = \{1, 2, 3, \ldots, n\}$ com n tarefas que estão disponíveis para serem sequenciadas em uma única máquina a partir do instante zero. Cada tarefa $j \in J$ possui um tempo de processamento p_j, uma data de entrega d_j e uma penalidade por atraso β_j. As tarefas são agrupadas em f famílias de acordo com suas características, sendo que cada família i possui n_i tarefas. Um tempo de preparação s_{ik} é requerido entre a execução de duas tarefas consecutivas de diferentes famílias i e k e caso elas sejam da mesma família nenhum tempo de preparação é necessário. A primeira tarefa a ser executada não requer tempo de preparação. Além disso, não é permitida preempção, ou seja, o processamento de uma tarefa não pode ser interrompido. Dada uma sequência π, a cada tarefa j está associado um atraso T_j. Sendo C_j o instante de conclusão (*completion time*) da tarefa j, seu atraso é calculado pela equação (2.1):

$$T_j = \max\{C_j - d_j, 0\} \tag{2.1}$$

Os objetivos do problema abordado são minimizar, simultaneamente, o *makespan* $f_1(\pi)$ e o atraso total ponderado $f_2(\pi)$. Os valores de $f_1(\pi)$ e $f_2(\pi)$ são calculados por meio das equações (2.2) e (2.3):

$$f_1(\pi) = \max_{1 \leqslant j \leqslant n} \{C_j\} \tag{2.2}$$

$$f_2(\pi) = \sum_{1 \leqslant j \leqslant n} \beta_j T_j \tag{2.3}$$

Para ilustrar, considere a instância dada na Tabela 2.1, que contém 5 tarefas (nomeadas de 1 a 5) e duas famílias (1 e 2). Nessa tabela são apresentados, para cada tarefa j, a família i à qual pertence, a data de entrega d_j, o tempo de processamento p_j e a penalidade por atraso β_j, respectivamente:

A Tabela 2.2 mostra o tempo de preparação entre as tarefas de uma família i e outra tarefa da família k, neste caso tem-se duas famílias 1 e 2.

Neste exemplo, o tempo de preparação entre uma tarefa da família 2 e outra tarefa da família 1, que serão processadas em sequência, é igual a 1, conforme a Tabela 2.2. É importante observar que o tempo de preparação depende da sequência,

Tabela 2.1: Instância do problema

Tarefa j	Família i	d_j	p_j	β_j
1	2	22	2	15
2	1	16	4	37
3	2	4	6	12
4	2	8	1	21
5	1	13	3	18

Tabela 2.2: Tempo de Preparação entre as Famílias

s_{ik}	1	2
1	0	3
2	1	0

ou seja, o tempo de preparação de i para k não necessariamente é igual ao tempo de preparação de k para i.

Considere uma sequência de exemplo $\{3, 4, 5, 2, 1\}$, nesse caso, verifica-se que a tarefa 3 é a primeira a ser executada e a tarefa 1 é a última. De acordo com as Equações (2.2) e (2.3) do problema, os valores das funções objetivos são: $f_1 = 20$ e $f_2 = 24$.

Na Figura 2.1 apresenta-se o gráfico de Gantt correspondente a essa solução.

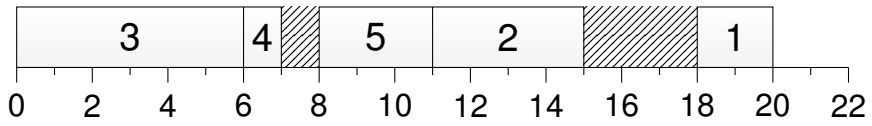

Figura 2.1: Sequência relativa ao exemplo

Conforme a Figura 2.1, nenhum tempo de preparação é necessário entre as tarefas 3 e 4, uma vez que elas são da mesma família. Entretanto, entre as tarefas 4 e 5 existe um tempo de preparação que ocorre entre os instantes 7 e 8. Esse tempo de preparação é necessário porque a tarefa 4 pertence à família 2 e a tarefa 5 pertence à família 1. Na Figura 2.1 o tempo de preparação é representado pela área hachurada.

Capítulo 3

Revisão Bibliográfica

3.1 Otimização Multiobjetivo

Um problema de otimização multiobjetivo envolve dois ou mais objetivos, normalmente conflitantes, que devem ser otimizados. A principal diferença, em relação à otimização mono-objetivo, é que em problemas de otimização multiobjetivo não existe uma única solução, mas em vez disso, um conjunto de soluções ótimas, e cada uma representa um compromisso entre os objetivos do problema.

Uma opção para reduzir a complexidade de um problema de otimização multiobjetivo é convertê-lo para um problema mono-objetivo atribuindo pesos para os diferentes objetivos. Dessa forma, uma única função de avaliação é calculada. A principal dificuldade dessa abordagem é que pesos são atribuídos aos objetivos, de forma subjetiva, de acordo com importância atribuída pelo tomador de decisão (*decision maker* – DM). Essa abordagem assume que o DM tem, a *priori*, conhecimento da importância dos diferentes objetivos, o que é muito difícil de conseguir. Por outro lado, em uma abordagem multiobjetivo, as soluções são avaliadas e classificadas conforme o conceito de dominância.

3.1.1 Definição

Geralmente, em problemas de otimização multiobjetivo, tem-se um conjunto de m variáveis de decisão e um conjunto de n funções objetivos. Um vetor de decisão $x \in X$ do problema especifica um valor para cada umas das m variáveis de decisão. X é denotado como o espaço de decisão. Os n objetivos, normalmente, são conflitantes. O resultado para as n funções objetivos define o espaço de objetivos Y de dimensão n, também chamado de espaço de soluções. Assim, a n-ésima função objetivo $f_n(x)$ é descrita por $f_n(x) : \mathbb{R}^m \to \mathbb{R}$ e o vetor de funções $f(x)$ definida por $f(x) : \mathbb{R}^m \to \mathbb{R}^n$. A região factível do problema X_f é um subconjunto de X definido pelas restrições do problema. Assim, o problema de otimização multiobjetivo pode ser formulado da seguinte maneira:

$$\begin{aligned} \text{Minimizar} \quad & f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) & & \\ \text{sujeito a} \quad & g_j(x) \geq 0 & & j = 1, 2, \ldots, J \\ & h_k(x) = 0 & & k = 1, 2, \ldots, K \end{aligned} \tag{3.1}$$

A região factível com o espaço de objetivos Y_f pode ser descrita por $Y_f = f(X_f)$.

A função de avaliação multiobjetivo $f : X \rightarrow Y$ mapeia as variáveis de decisão $x = \{x_1, x_2, \cdots, x_m\}$ para o vetor de objetivos $y = \{y_1, y_2, \cdots, y_n\}$. Essa situação é representada na Figura 3.1, que considera $m = 2$ e $n = 3$.

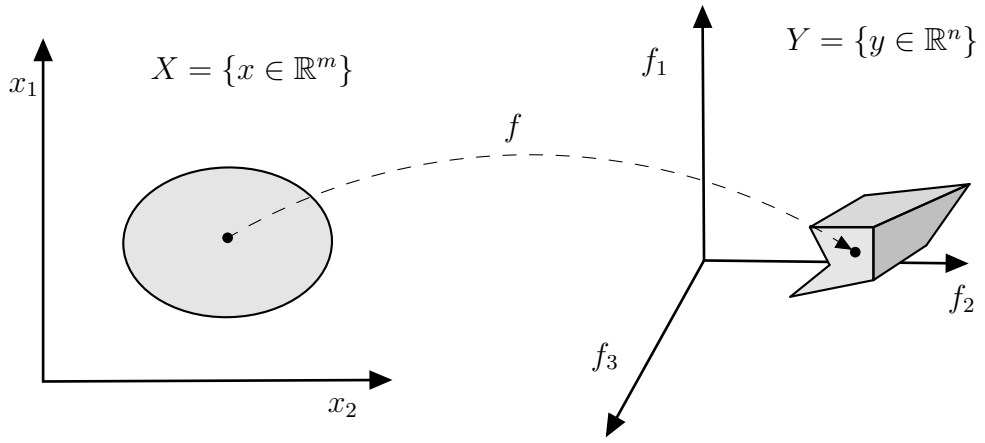

Figura 3.1: Mapeamento das Variáveis de Decisão

Alguns conceitos de otimização multiobjetivo (Arroyo, 2002; Coello et al., 2007; Takahashi, 2004) são necessários para o melhor entendimento dos algoritmos propostos:

Definição 1 (Pareto Dominância) Para quaisquer duas soluções $x_1, x_2 \in X_f$, sendo X_f o conjunto de soluções factíveis, e para uma função vetorial $f(x) = (f_1(x), f_2(x), ..., f_n(x)) : X \rightarrow Z$, tem-se que, x_1 domina x_2 se a imagem de x_1 domina a imagem de x_2, isto é, $f(x_1) \leqslant f(x_2)$ e $f(x_1) \neq f(x_2)$. O conceito de Pareto dominância estabelece a relação $\prec$ entre dois vetores de decisão. Assim, $x_1 \prec x_2$ denota que o vetor de decisão x_1 domina o vetor de decisão x_2.

Definição 2 (Soluções Não-Comparáveis) Para quaisquer duas soluções $x_1, x_2 \in X_f$, sendo X_f o conjunto de soluções factíveis, e para uma função vetorial $f(x) = (f_1(x), f_2(x), ..., f_n) : X \rightarrow Z$, diz-se que x_1 e x_2 são não comparáveis se nem x_1 domina x_2 e nem x_2 domina x_1.

Definição 3 (Solução Pareto-ótima) Uma solução $x^* \in X_f$ é uma solução Pareto-ótima se não existe qualquer outra solução $x \in X_f$, tal que x domine x^*; $z^* = f(x^*)$ é chamado de ponto eficiente ou ponto Pareto-ótimo.

Definição 4 (Conjunto Pareto-ótimo) O conjunto Pareto-ótimo $\mathcal{P}$, ou conjunto Pareto-ótimo global para um problema de otimização multiobjetivo é definido como:

$$\mathcal{P} = \{x^* \in X_f \mid \nexists x \in X_f : f(x) \prec f(x^*)\} \tag{3.2}$$

Definição 5 (Fronteira Pareto-ótima) A fronteira Pareto-ótima $\mathcal{F}$, ou fronteira Pareto-ótima global para um problema de otimização multiobjetivo corresponde à imagem do conjunto Pareto-ótimo, sendo definida como:

$$\mathcal{F} = \{y = f(x) : x \in \mathcal{P}\} \tag{3.3}$$

Existem diversas relações de ordem entre duas soluções $x_1, x_2 \in X_f$. Uma lista das mais comuns é apresenta na Tabela 3.1

Tabela 3.1: Relações de Dominância

relação	**notação**	**interpretação**
domina estritamente	$x_1 \prec\prec x_2$	$f_n(x_1) < f_n(x_2)\ \forall n$
domina	$x_1 \prec x_2$	$f_n(x_1) \leq f_n(x_2)\ \forall n \wedge \exists i : f_i(x_1) < f_i(x_2)$
domina fracamente	$x_1 \preceq x_2$	$f_n(x_1) \leq f_n(x_2)\ \forall n$
não-comparáveis	$x_1 \parallel x_2$	$\neg(x_1 \preceq x_2) \wedge \neg(x_2 \preceq x_1)$
indiferente	$x_1 \sim x_2$	$f_n(x_1) = f_n(x_2)\ \forall n$

3.1.2 Métodos Clássicos

Nesta seção são apresentados os métodos tradicionais para resolver problemas de otimização multiobjetivo.

3.1.2.1 Método da Soma Ponderada

Este método consiste em criar um modelo mono-objetivo a partir da ponderação das n funções objetivos através da atribuição de pesos para cada função. Dessa forma, o método de soma ponderada para um problema de otimização multiobjetivo pode ser definido da seguinte forma:

$$f(x) = \sum_{i=1}^{n} w_i f_i(x) \tag{3.4}$$

ou seja, cada função objetivo f_i é multiplicada por um peso w_i, com $w_i \geq 0$. Além disso, geralmente a soma dos pesos é normalizada:

$$\sum_{i=i}^{n} w_i = 1 \tag{3.5}$$

Analisando a função $f(x)$ obtida pela combinação linear das funções f_i, pode-se afirmar que a solução fornecida por esse método será um ponto da frente Pareto-ótima. A definição de qual é o ponto da fronteira Pareto-ótima depende da atribuição dos pesos para cada função objetivo.

A Figura 3.2 representa a interpretação do problema de soma ponderada no espaço de objetivos: O processo inicia a partir de um hiperplano, cuja inclinação é definida pelo vetor w. Então o algoritmo de otimização determina outros hiperplanos

paralelos ao inicial buscando a minimização da distância entre o hiperplano e a origem do espaço de objetivos Y, com restrição de que o hiperplano contenha algum ponto de Y_f. O algoritmo converge para o hiperplano h^*, que possui um único ponto de interseção com o conjunto Pareto Y^*, que é o ponto y^* (Takahashi, 2004).

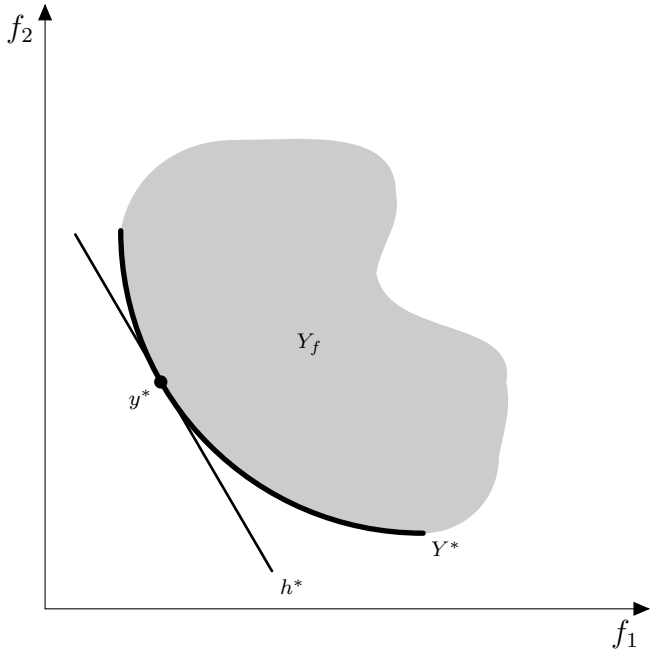

Figura 3.2: Função Ponderada

3.1.2.2 Método ϵ-Restrito

O método ϵ-restrito, introduzido por Haimes et al. (1971), é um das técnicas de otimização multiobjetivo mais conhecidas. Nele apenas um dos objetivos do problema é otimizado, de modo que os outros objetivos são transformados em restrições. Utilizando esse método, um problema de otimização multiobjetivo pode ser modelado da seguinte maneira:

$$\begin{aligned} \text{Minimizar} \quad & f_\mu(x), & \\ \text{sujeito a} \quad & f_m(x) \geq \epsilon_m & m = 1, 2, \ldots, M \text{ e } m \neq \mu \\ & g_j(x) \geq 0 & j = 1, 2, \ldots, J \\ & h_k(x) = 0 & k = 1, 2, \ldots, K \end{aligned}$$

O método ϵ-restrito tem algumas vantagens sobre o método de soma ponderada (Mavrotas, 2009):

- Para problemas lineares, o método de soma ponderada é aplicado para a região factível original e o resultado é uma solução localizada na extremidade do conjunto. Por outro lado, o método ϵ altera a região factível original e

assim consegue produzir soluções eficientes que não estão localizadas nas extremidades do conjunto. Dessa forma, o método ϵ-restrito pode obter uma representação mais rica do conjunto eficiente.

- No método de soma ponderada a escala das funções objetivos tem forte influência na obtenção dos resultados. Por isso, é necessário definir uma escala comum para as funções objetivos antes de formar a soma ponderada. No método ϵ-restrito isso não é preciso.

Uma desvantagem do método ϵ-restrito é a dificuldade em escolher o melhor valor mais adequado para ϵ_m. Pode-se destacar os seguintes problemas causados pela configuração indevida desse parâmetro (Takahashi, 2004):

1. O ponto obtido pode não ser eficiente;
2. Os valores de ϵ_m podem tornar o problema infactível.

3.2 Métricas de Avaliação de Desempenho

A comparação de dois conjuntos de pontos não-dominados, A e B, obtidos respectivamente por dois algoritmos de otimização multiobjetivo não é uma tarefa trivial. Diversas métricas de avaliação de desempenho de algoritmos multiobjetivo têm sido propostas na literatura (Hansen e Jaszkiewicz, 1998; Zitzler et al., 2000; Deb e Jain, 2002; Fonseca et al., 2005). Elas devem ser escolhidas de forma adequada para realizar uma comparação justa dos algoritmos.

Neste trabalho são utilizadas cinco métricas de avaliação de desempenho, a saber: cardinalidade (Hansen e Jaszkiewicz, 1998), distância média (Czyzżak e Jaszkiewicz, 1998), distância máxima (Knowles e Corne, 2002), hipervolume (Zitzler e Thiele, 1998) e *epsilon* (Fonseca et al., 2005). Em Fonseca et al. (2005) é mostrado que as métricas hipervolume e *epsilon* fornecem medidas confiáveis principalmente quando dois algoritmos possuem desempenhos semelhantes.

A qualidade do conjunto de pontos não-dominados obtidos por um algoritmo, para uma determinada instância, é avaliada com relação ao conjunto constituído por todos os pontos não-dominados encontrados em todos os experimentos realizados. Este conjunto é denominado conjunto de pontos de referência R.

A seguir apresentam-se as definições das métricas utilizadas:

3.2.1 Métrica de Cardinalidade

A métrica de cardinalidade $C1_R$ mensura a quantidade de soluções não-dominadas (geradas por um algoritmo A) presentes no conjunto de referência, sendo pela Equação (3.6):

$$C1_R(A) = \frac{|A \cap R|}{|R|} \times 100 \tag{3.6}$$

em que $C1_R(A)$ é a cardinalidade do conjunto A de soluções não-dominadas em relação ao conjunto de referência R. O valor de $C1_R(A)$ está no intervalo $[0, 100]$,

de modo que $C1_R(A) = 100$ significa que todas as soluções do conjunto R estão em A e $C1_R(A) = 0$ significa que nenhuma solução de R está em A.

Uma desvantagem da métrica de cardinalidade é que ela não consegue avaliar a distribuição das soluções do conjunto em relação ao conjunto de referência, tampouco a proximidade de um conjunto de soluções em relação ao conjunto de referência.

3.2.2 Métricas de Distância

As métricas de distância avaliam a proximidade entre o conjunto de referência R e um conjunto de soluções não-dominadas A. Segundo Czyżżak e Jaszkiewicz (1998), a métrica $D1_R$ informa sobre a distância média de $s \in R$ para a solução mais próxima em A, enquanto que a métrica $D_{\max}$ informa sobre o pior caso. Os valores de $D1_R$ e $D_{\max}$ são obtidos pela aplicação das equações (3.7) e (3.8), respectivamente:

$$D1_R = \frac{1}{|R|} \sum_{s \in R} \left\{ \min_{s' \in A} \{d(s, s')\} \right\} \tag{3.7}$$

$$D_{\max} = \max_{s \in R} \left\{ \min_{s' \in A} \{d(s, s')\} \right\} \tag{3.8}$$

sendo $d(s, s')$ calculado pela expressão (3.9):

$$d(s, s') = \sqrt{(f_1^*(s) - f_1^*(s'))^2 + (f_2^*(s) - f_2^*(s'))^2} \tag{3.9}$$

3.2.3 Métrica hipervolume

O hipervolume ou *S metric* é uma métrica frequentemente utilizada para comparar resultados de algoritmos multiobjetivos. Ela avalia a distribuição do conjunto de soluções em relação ao espaço de busca e foi proposta por Zitzler e Thiele (1998). O hipervolume de um conjunto de soluções A mede a área coberta ou dominada pelo conjunto A de pontos obtido por um algoritmo. Para calcular o hipervolume de um conjunto de soluções é necessário definir um ponto de referência. Em problemas de maximização é comum utilizar o ponto $(0, 0)$, enquanto que em problemas de minimização um ponto de referência (x, y) é usado para limitar essa área. A área sombreada da Figura 3.3 define o hipervolume de um conjunto de soluções A para um problema com duas funções objetivo f_1 e f_2, em que o ponto (x, y) foi definido como limite superior. Denotamos por $H(A)$ o hipervolume de um conjunto de soluções A em relação a um ponto de referência.

A métrica diferença do hipervolume $H^-(A)$ fornece a área do espaço de objetivos de um conjunto de soluções A que é fracamente dominada pela área do espaço de objetivos do conjunto R. O valor de $H(A)$ é calculado pela Equação (3.10).

$$H^-(A) = H(R) - H(A) \tag{3.10}$$

Na Figura 3.4 a área hachurada representa a diferença de hipervolume $H^-(A)$ para um conjunto de soluções A em relação a um conjunto de referência R:

Como $H(R) \geq H(A)$, quanto menor o valor de $H^-(A)$, melhor será a qualidade do conjunto A, sendo que o valor mínimo para essa métrica é zero.

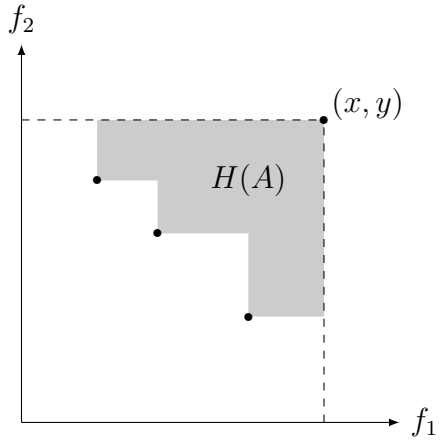

Figura 3.3: Hipervolume

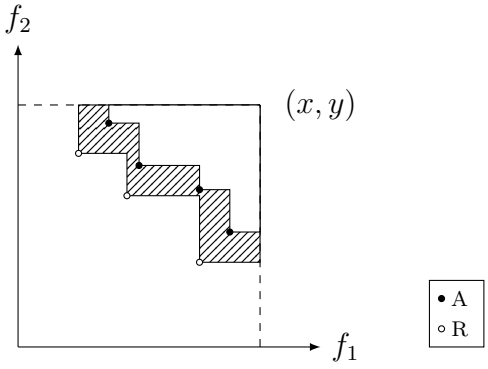

Figura 3.4: Diferença do Hipervolume

3.2.4 Métrica *epsilon*

A métrica *epsilon* multiplicativo foi proposta por Zitzler et al. (2003). Essa métrica é baseada no conceito de ϵ-dominância. Dizemos que um vetor de objetivos $z^a = (z_1^a, z_2^a, \ldots, z_n^a) \in Z$ ϵ-domina um outro vetor de objetivos $z^b = (z_1^b, z_2^b, \ldots, z_n^b) \in Z$, em um problema de minimização, se e somente se:

$$\forall i \in \{1, \ldots, n\} : z_i^a \leqslant \epsilon \cdot z_i^b \tag{3.11}$$

A Figura 3.5 ilustra o funcionamento da ϵ-dominância para uma solução z^a em relação a três diferentes valores de ϵ. A Figura 3.5(a) considera um valor de ϵ menor que 1; nesta situação é possível observar que a solução z^a ϵ-domina as soluções z^b, z^d e z^e. A Figura 3.5(b) considera um valor de ϵ é igual a 1; neste caso, a solução z^a ϵ-domina as soluções z^d e z^e. Por fim, na Figura 3.5(c), em que o valor de ϵ é maior que 1, a solução z^a ϵ-domina apenas a solução z^d.

O valor da métrica *epsilon* multiplicativo (I_ϵ) para dois conjuntos aproximados A e B representa o valor mínimo pelo qual cada ponto pertencente a B deve ser multiplicado de forma que o resultado seja fracamente dominado por A. O valor de

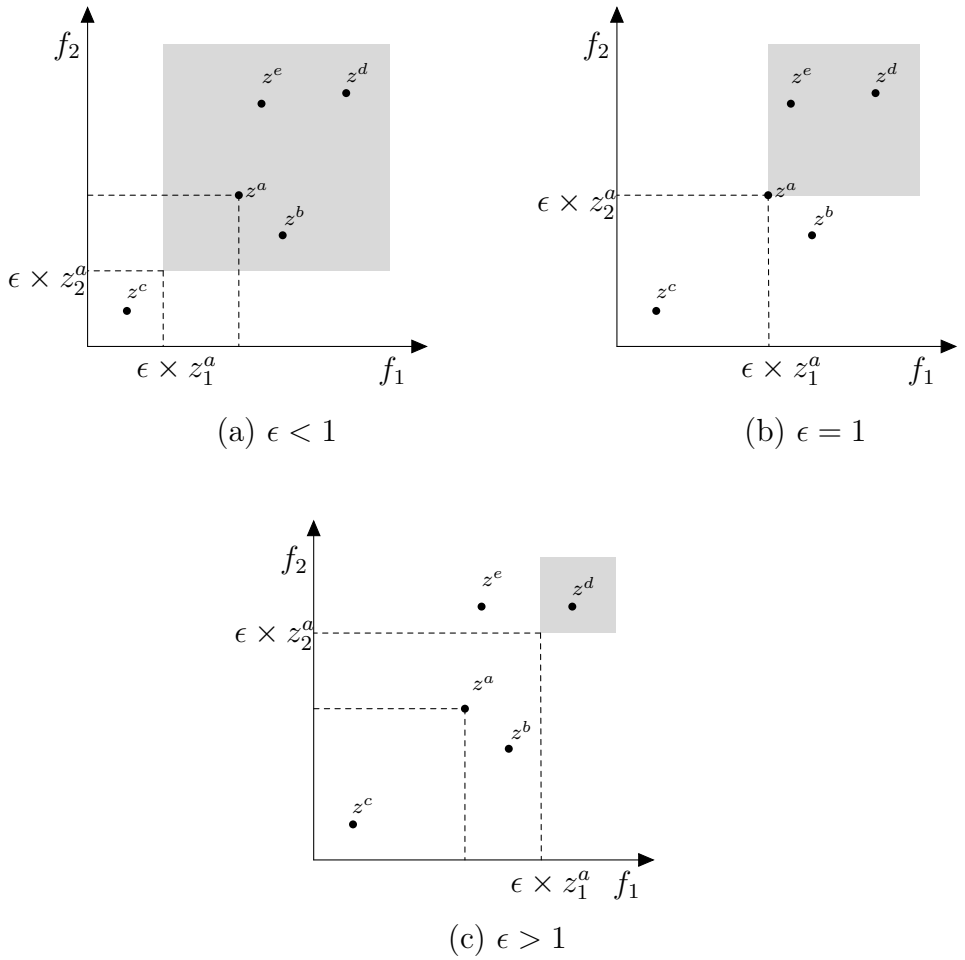

Figura 3.5: Epsilon

ϵ pode ser calculado conforme a Equação (3.12):

$$I_\epsilon(A, B) = \max_{z^b \in B} \left\{ \min_{z^a \in A} \left\{ \max_{1 \leqslant i \leqslant r} \frac{z_i^a}{z_i^b} \right\} \right\} \tag{3.12}$$

Observa-se que o valor de ϵ precisaria ser calculado para todos os pares de resultados com a finalidade de comparar os algoritmos. Todavia, Fonseca et al. (2005) propuseram uma versão unária em que A é o conjunto de soluções não-dominadas fornecida por cada algoritmo e o valor de ϵ é calculado apenas em relação ao conjunto das melhores soluções conhecidas R. Desta forma, temos que $I_\epsilon^1(A) = I_\epsilon(A, R)$.

Como $I_\epsilon(A, R)$ mede a distância máxima do conjunto A com relação a R, então um valor próximo a 1 da medida $I_\epsilon^1(A)$ indica uma boa qualidade do conjunto A.

3.3 Algoritmos Multiobjetivos

Nesta seção são apresentados os algoritmos multiobjetivos estudados.

3.3.1 MOVNS

Geiger (2004) trata o problema de *flow shop* em que existe um conjunto de tarefas e cada uma deve ser processada em série por um conjunto de máquinas, sendo que a sequência de processamento deve ser a mesma para todas as tarefas. Sete objetivos foram considerados e combinados para a realização dos experimentos. Os objetivos são: minimização do *makespan*; minimização do tempo de término total; minimização do atraso máximo; minimização do atraso total; minimização do número de tarefas atrasadas; minimização do tempo máximo ocioso e minimização do tempo total ocioso. Os objetivos foram combinados em funções com 2 a 6 objetivos. O autor propôs o *Multi-objective Variable Neighborhood Search* (MOVNS), que é um algoritmo multiobjetivo baseado na metaheurística *Variable Neighborhood Search* (VNS) de Mladenovic e Hansen (1997). O algoritmo foi executado para resolver 100 instâncias testes com 10 tarefas e 10 máquinas. As métricas utilizadas foram a distância média e a distância máxima. Os resultados do algoritmos foram comparadas com o conjunto Pareto-ótimo obtido a partir de enumeração completa.

O Algoritmo 1 apresenta a versão básica do MOVNS.

Algoritmo 1: MOVNS

Entrada: critério de parada
Saída: ND

```
1  ND ← Solução Inicial();
2  enquanto critério de parada não satisfeito faça
3  |  Selecione uma solução s ∈ ND | s ainda não foi visitada;
4  |  Marque s como visitada ;
5  |  Selecione uma estrutura de vizinhança N^i ∈ N^1, ..., N^k ;
6  |  Selecione aleatoriamente uma solução vizinha s' ∈ N^i(s) ;
7  |  para cada s'' ∈ N^i(s') faça
8  |  |  ND ← Conjunto de soluções não-dominadas de ND ∪ {s''} ;
9  |  fim
10 |  se ∄s ∈ ND | s não foi visitada então
11 |  |  Marque todas as soluções de ND como não visitadas ;
12 |  fim
13 fim
14 retorna ND ;
```

O Algoritmo 1 inicia, na linha 1, com um conjunto de soluções não-dominadas (ND). A cada iteração seleciona-se aleatoriamente uma solução ainda não visitada do conjunto ND (linha 3), marcando-a como visitada (linha 4). Em seguida, seleciona-se aleatoriamente uma estrutura de vizinhança N^i (linha 5) com a finalidade de gerar uma solução vizinha s' (linha 6). Na próxima etapa, o conjunto ND é atualizado com todas as soluções s'' pertencentes a $N^i(s')$ de acordo com o conceito de Pareto dominância (linha 8). Por fim, é verificado se todas as soluções do conjunto ND estão marcadas como visitadas; caso afirmativo, marca-se todas as soluções de ND como não visitadas. O laço é repetido enquanto o critério de parada não estiver satisfeito.

3.3.2 MOVNS Ottoni et al. (2011)

Em Ottoni et al. (2011) os autores examinam o problema de programação de tarefas *job scheduling* em que n tarefas devem ser processadas em uma única máquina, que pode processar uma tarefa de cada vez. As tarefas estão disponíveis para serem processadas a partir do instante zero. Para processar duas tarefas consecutivas um tempo de preparação é necessário. Podem ocorrer tempos de ociosidade, ou seja, a máquina pode ficar parada em alguns instantes. O autor propôs a incorporação de um novo método de intensificação ao algoritmo MOVNS. Os experimentos foram realizados em um conjunto de 72 instâncias com número de tarefas entre 20 e 100. O resultado foi comparado com a versão original do MOVNS considerando as métricas de cardinalidade, distância média, distância máxima e hipervolume. O algoritmo proposto foi superior ao MOVNS original em relação à todas as métricas avaliadas.

A variante do MOVNS proposta por Ottoni et al. (2011), nomeada neste trabalho de MOVNS_Ottoni, consiste em adicionar um procedimento de intensificação entre as linhas 12 e 13 do Algoritmo 1. O pseudocódigo desse procedimento é apresentado no Algoritmo 2.

A intensificação é iniciada com a etapa de destruição (linhas 4 – 8), na qual n_r tarefas são removidas da solução s. Esta estratégia resulta na formação de uma solução parcial s_p e de uma sequência de tarefas removidas s_r, sendo esta última composta pelas tarefas que foram retiradas de s enquanto a primeira é composta pelas $(n - n_r)$ tarefas restantes. Em seguida, a solução é reconstruída inserindo-se $(n_r - 1)$ tarefas em s_p (linhas 9 – 19). Para tanto, uma tarefa de s_r é inserida em todas as possíveis posições de s_p, sendo escolhida a posição que fornece o melhor sequenciamento parcial avaliado segundo uma função de avaliação f dada pela ponderação dos objetivos f_1 e f_2, com pesos w_1 e w_2, respectivamente. O procedimento anterior é repetido até que as $(n_r - 1)$ tarefas de s_r sejam inseridas em s_p. Em seguida, (linhas 20 – 23), a solução parcial s_p sofre a inserção da última tarefa de s_r em todas as possíveis posições de s_p. Todas as soluções geradas por essa inserção (linha 21) são adicionadas no conjunto *ND*, mas somente as soluções não-dominadas são mantidas (linha 22).

3.3.3 MOVNS Arroyo et al. (2011)

O problema de sequenciamento em uma máquina com janelas de tempo é abordado por Arroyo et al. (2011). Neste problema existe um conjunto de n tarefas que devem ser processadas por uma única máquina. A máquina pode processar apenas uma tarefa de cada vez. Cada tarefa está disponível para ser processada a partir do instante zero. Um tempo de preparação da máquina é necessário entre a execução de duas tarefas. Os autores consideraram dois objetivos, o primeiro é minimizar a soma do atraso e adiantamento ponderado, e o segundo é minimizar o fluxo total. Para resolver o problema foi proposto um novo algoritmo MOVNS combinado com um processo de intensificação, sendo este, diferente do proposto em Ottoni et al. (2011).

A outra variante do MOVNS analisada neste trabalho foi proposta por Arroyo et al. (2011), sendo nomeada MOVNS_Arroyo. Ela consiste em adicionar uma intensificação diferente entre as linhas 12 e 13 do Algoritmo 1. O pseudocódigo

Algoritmo 2: Intensificação de Ottoni et al. (2011)

Entrada: s, n_r, ND
Saída: ND

```
1  s_r ← ∅ ;
2  s_p ← s ;
3  Selecione aleatoriamente pesos w_1, w_2 ∈ [0,1] tal que w_1 + w_2 = 1;
4  para i ← 1 até n_r faça
5  |  Seja s_p(j) a j-ésima tarefa de s_p escolhida aleatoriamente;
6  |  Remova s_p(j) de s_p;
7  |  Adicione s_p(j) à s_r;
8  fim
9  para i ← 1 até (n_r − 1) faça
10 |  f_p^* ← ∞ ;
11 |  para j ← 1 até (n − n_r + i) faça
12 |  |  s' ← resultado da inserção da i-ésima tarefa de s_r na j-ésima posição
   |  |  de s_p ;
13 |  |  se f(s') < f_p^* então
14 |  |  |  s_p^* ← s' ;
15 |  |  |  f_p^* ← f(s') ;
16 |  |  fim
17 |  fim
18 |  s_p ← s_p^* ;
19 fim
20 para j ← 1 até n faça
21 |  s' ← inserção da última tarefa de s_r na j-ésima posição de s_p ;
22 |  ND ← soluções não-dominadas obtidas de ND ∪ {s'} ;
23 fim
24 retorna ND ;
```

desse procedimento é apresentado no Algoritmo 3.

A intensificação é iniciada com a etapa de destruição (linhas 3 – 7), na qual n_r tarefas são removidas da solução s. Esta estratégia resulta na formação de uma solução parcial s_p e de uma sequência de tarefas removidas s_r, sendo esta última composta pelas tarefas que foram retiradas de s enquanto a primeira é composta pelas $(n - n_r)$ tarefas restantes. O conjunto de soluções parciais ND_a é inicializado com a solução parcial s_p (linha 8).

Em seguida, nas linhas 9 – 18, a solução é reconstruída inserindo-se n_r tarefas em s_p. Para tanto, um outro conjunto ND_b de soluções parciais é inicializado como um conjunto vazio. Em seguida, para cada solução s_p presente no conjunto ND_a uma tarefa i de s_r é inserida em todas as possíveis posições de s_p. As soluções geradas são adicionadas ao conjunto de soluções parciais ND_b, sendo que apenas as soluções não-dominadas são mantidas em ND_b. Na sequência (linha 17), as soluções do conjunto ND_b são salvas em ND_a. O procedimento entre as linhas 10 e 17 é repetido para as n_r tarefas de s_r. Ao final da etapa de reconstrução, um conjunto de soluções não-dominadas completas será encontrado no conjunto ND_a. Por fim,

Algoritmo 3: Intensificação de Arroyo et al. (2011)

Entrada: s, n_r, ND
Saída: ND

```
1  s_r ← ∅ ;
2  s_p ← s ;
3  para i ← 1 até n_r faça
4  |  Seja s_p(j) a j-ésima tarefa de s_p escolhida aleatoriamente;
5  |  Remova s_p(j) de s_p ;
6  |  Adicione s_p(j) à s_r ;
7  fim
8  ND_a ← s_p ;
9  para i ← 1 até n_r faça
10 |  ND_b ← ∅ ;
11 |  para cada s_p ∈ ND_a faça
12 |  |  para j ← 1 até (n − n_r + i) faça
13 |  |  |  s' ← resultado da inserção da i-ésima tarefa de s_r na j-ésima
   |  |  |  posição de s_p ;
14 |  |  |  ND_b ← soluções não-dominadas obtidas de ND_b ∪ {s'} ;
15 |  |  fim
16 |  fim
17 |  ND_a ← ND_b ;
18 fim
19 ND ← soluções não-dominadas obtidas de (ND ∪ ND_a);
20 retorna ND ;
```

na linha 19, o conjunto ND receberá o conjunto não-dominado da união entre ND e ND_a.

3.3.4 PILS

O algoritmo *Pareto Iterated Local Search* PILS foi proposto em Geiger (2006). Posteriormente, outros trabalhos propuseram novos aperfeiçoamentos para este método.

Em Geiger (2009) apresenta-se uma versão do algoritmo PILS para resolver o problema de *flow shop* multiobjetivo em que um conjunto de J tarefas que precisam ser processadas em um conjunto de M máquinas. Neste último trabalho, foram feitas duas diferentes abordagens: na primeira considerou-se dois objetivos para serem minimizados: o *makespan* e o atraso total ponderado; enquanto que na segunda modelagem examinou-se três objetivos a serem minimizados o *makespan*, o tempo médio de término e o tempo médio de atraso. Geiger (2009) realizou experimentos em dois conjuntos de instâncias, utilizando as métricas de distância média e distância máxima. A comparação foi feita em relação às soluções ótimas para as instâncias pequenas, enquanto que para as instâncias maiores a análise foi feita baseada nos melhores resultados encontrados na literatura. Os experimentos foram realizados em instâncias de 20 a 100 tarefas e 5 a 20 máquinas.

O PILS é um algoritmo de otimização multiobjetivo cuja estrutura se baseia na

metaheurística *Iterated Local Search* – ILS (Lourenço et al., 2003). Apresenta-se no Algoritmo 4 o pseudocódigo básico do PILS.

Algoritmo 4: PILS

```
   Entrada: critério de parada
 1 ND ← Solução Inicial();
 2 Selecione uma solução s ∈ ND ;
 3 enquanto critério de parada não satisfeito faça
 4 |   i ← 1 ;
 5 |   enquanto i < k ∧ critério de parada não satisfeito faça
 6 |   |   para cada s' ∈ N^i(s) faça
 7 |   |   |   ND ← soluções não-dominadas obtidas de ND ∪ {s'} ;
 8 |   |   fim
 9 |   |   se ∃s' ∈ N^i(s) | s' domina s então
10 |   |   |   s ← s' ;
11 |   |   |   Reordene as vizinhanças N^1, ..., N^k, em uma ordem aleatória ;
12 |   |   |   i ← 1 ;
13 |   |   fim
14 |   |   senão
15 |   |   |   i++;
16 |   |   fim
17 |   fim
18 |   Marque s como visitada ;
19 |   se ∃s' ∈ ND | s'ainda não foi visitada então
20 |   |   s ← s';
21 |   fim
22 |   senão
23 |   |   Selecione uma solução s' ∈ ND ;
24 |   |   s'' ← Perturbação(s') ;
25 |   |   s ← s'' ;
26 |   fim
27 fim
28 retorna ND ;
```

No Algoritmo 4, obtém-se, inicialmente, um conjunto de soluções não-dominadas (*ND*) (linha 1). Em seguida, seleciona-se aleatoriamente uma das soluções do conjunto *ND* (linha 2) para ser a solução corrente s. A cada iteração do laço mais externo (linhas 3 – 27), explora-se toda a vizinhança de s (linhas 5 – 17). Nesse laço, caso uma solução vizinha domine s (linha 9), então essa solução vizinha passa a ser a nova solução corrente. Além disso, as vizinhanças são reordenadas aleatoriamente e o procedimento retorna à primeira vizinhança na nova ordem gerada. Após visitar toda a vizinhança da solução corrente, marca-se s como visitada (linha 18). Em seguida, caso exista solução não-visitada no conjunto *ND* (linha 19), uma solução não-visitada é selecionada aleatoriamente como a solução corrente. Por outro lado, se todas as soluções ja tiverem sido visitas, seleciona-se aleatoriamente uma solução do conjunto *ND* (linha 23) e aplica-se uma perturbação (linha 24), que gera

uma solução s''; em seguida essa solução passa a ser a nova solução corrente. O laço mais externo (linhas 3 – 27) é repetido enquanto o critério de parada não for satisfeito.

Uma solução é perturbada com o objetivo de explorar outras regiões do espaço de busca. A estratégia de perturbação original do PILS, de Geiger (2009), funciona da seguinte forma: inicialmente uma solução π do conjunto *ND* é selecionada aleatoriamente. A seguir, escolhe-se aleatoriamente uma posição $j \leq n-4$ e quatro tarefas consecutivas de π nas posições $j, j+1, j+2, j+3$. Uma solução vizinha s'' é, então, gerada aplicando-se o movimento de troca das tarefas das posições j e $j+3$, assim como das tarefas $j+1$ e $j+2$. Dessa forma, as tarefas anteriores à tarefa j e as tarefas seguintes às tarefas $j+3$ continuam em suas respectivas posições após a aplicação da perturbação.

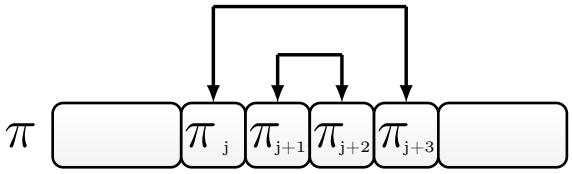

Figura 3.6: Perturbação Original

A Figura 3.6 ilustra o funcionamento da perturbação realizada pelo método PILS proposto por Geiger (2009). Para realizar esse procedimento, são selecionadas quatro tarefas consecutivas de uma solução nas posições $j, j+1, j+2, j+3$, e em seguida são realizadas duas trocas, a primeira entre as tarefas j e $j+3$ e a segunda entre as tarefas $j+1$ e $j+2$.

3.3.5 NSGA-II

O *Nondominated Sorting Genetic Algorithm* II (NSGA-II) é um algoritmo de otimização multiobjetivo proposto por Deb e Jain (2002). Ele é resultado de uma melhoria feita no algoritmo NSGA, proposto por Srinivas e Deb (1994). Basicamente, foram feitas três principais mudanças: aperfeiçoamento do processo de ordenação das soluções não-dominadas, adição de elitismo e a eliminação da necessidade do parâmetro σ_{share} para incrementar a variedade da população.

3.3.5.1 Fast Non-Dominated Sorting

Para ordenar uma população, o NSGA-II utiliza o método *Fast Non-Dominated Sorting*, que tem complexidade $O(MN^2)$, sendo M o número de objetivos e N o tamanho da população. Este método recebe como parâmetro de entrada uma população P e retorna um conjunto de frentes não-dominadas $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k)$.

O Algoritmo 5 apresenta o pseudocódigo do *Fast Non-Dominated Sorting*, que compara todas as possíveis soluções p e q presentes na população P, sendo p diferente de q. Caso p domine q, então a solução q é inserida no conjunto S_p, caso q domine p, então n_p é incrementado em uma unidade. Na linha 12, verifica-se o valor de n_p, caso ele seja igual a zero, então a solução p é inserida na primeira fronteira $\mathcal{F}_1$ (linha 13),

Algoritmo 5: Fast Non-Dominated Sorting

```
   Entrada: P
1  para cada p ∈ P faça
2  |  S_p ← ∅ ;
3  |  n_p = 0 ;
4  |  para cada q ∈ P faça
5  |  |  se (p ≺ q) então
6  |  |  |  S_p ← S_p ∪ {q} ;
7  |  |  fim
8  |  |  senão se (q ≺ p) então
9  |  |  |  n_p ← n_p + 1 ;
10 |  |  fim
11 |  fim
12 |  se (n_p = 0) então
13 |  |  F ← F ∪ {p} ;
14 |  fim
15 fim
16 i ← 1 ;
17 enquanto F ≠ ∅ faça
18 |  Q ← ∅ ;
19 |  para cada p ∈ F_i faça
20 |  |  para cada q ∈ S_p faça
21 |  |  |  n_q ← n_q − 1 ;
22 |  |  |  se (n_q = 0) então
23 |  |  |  |  Q ← Q ∪ q ;
24 |  |  |  fim
25 |  |  fim
26 |  fim
27 |  i ← i + 1 ;
28 |  F_i ← Q ;
29 fim
30 retorna F ;
```

pois ela não é dominada por nenhuma outra solução. Em seguida, cria-se um novo laço (linha 17 – 29) para gerar as demais fronteiras. Nesse laço, percorre-se cada solução p da fronteira $\mathcal{F}_i$, e todas as q soluções de S_p realizando um decremento em n_q. Caso o valor de n_q alcance o valor zero, então a solução q é incluída em Q. Após percorrer todas as soluções de $\mathcal{F}_i$, o valor de i é incrementado e atribui-se Q para a fronteira $\mathcal{F}_i$. Esse laço é repetido enquanto for possível gerar uma nova fronteira para as soluções de P.

3.3.5.2 *Crowding Distance*

Para estimar a densidade de soluções em volta de um certo ponto da população, é calculada a distância média entre os dois pontos adjacentes deste, para cada um dos objetivos. A medida $i_{distance}$ serve como uma estimativa do tamanho do maior cubóide que envolve o ponto i, sem incluir nenhum outro ponto da população. Soluções localizadas em regiões com menor número de pontos recebem um valor maior do que soluções localizadas em regiões com maior número de pontos em relação ao espaço de objetivos.

O algoritmo que calcula o valor para a *crowding distance* é apresentado pelo

Algoritmo 6:

Algoritmo 6: *crowding distance*

Entrada: $\mathcal{I}$
1 $l \leftarrow |\mathcal{I}|$;
2 **para cada** $i \in \mathcal{I}$ **faça**
3 | $\mathcal{I}[i] \leftarrow 0$;
4 **fim**
5 **para cada** objetivo m **faça**
6 | $\mathcal{I} \leftarrow ordenar(\mathcal{I}, m)$;
7 | $\mathcal{I}[1] \leftarrow \infty$;
8 | $\mathcal{I}[l] \leftarrow \infty$;
9 | **para** $i = 2$ to $(l-1)$ **faça**
10 | | $\mathcal{I}[i] \leftarrow \mathcal{I}[i] + (\mathcal{I}[i+1].m - \mathcal{I}[i-1].m)/(f_m^{\max} - f_m^{\min})$;
11 | **fim**
12 **fim**
13 **retorna** $\mathcal{I}$;

O Algoritmo 6 recebe como entrada um conjunto de soluções $\mathcal{I}$. Este conjunto tem tamanho l e, inicialmente, é atribuído o valor zero para a *crowding distance* de todos os indivíduos de $\mathcal{I}$. A cada iteração do laço mais externo (linha 5 – 12): 1) ordenam-se as soluções do conjunto $\mathcal{I}$ (linha 6), considerando cada objetivo m; 2) atribui-se infinito para o valor da *crowding distance* da primeira (linha 7) e da última solução de $\mathcal{I}$ (linha 8). Em seguida, o algoritmo entra no laço mais interno (linha 9 – 11), no qual é feito o cálculo da *crowding distance* para cada solução i (linha 10), com i variando de 2 a $l-1$. O laço mais externo é repetido para os m objetivos. O método retorna o valor da *crowding distance* para todas as soluções do conjunto $\mathcal{I}$.

No Algoritmo 7 apresenta-se o pseudocódigo do algoritmo NSGA-II.

O algoritmo NSGA-II parte de um população inicial P_0 (linha 2) de tamanho *pop*, em que *pop* é um parâmetro de entrada. A cada iteração do laço principal (linha 5 – 25), as populações de pais P_t e de filhos Q_t se unem formando a população R_t (linha 6). O método *fast non-dominated sorting* é aplicado em R_t (linha 7), para dividir a população em conjuntos não-dominados, denominados fronteiras, $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k$, onde, $\mathcal{F}_i$ domina $\mathcal{F}_j$, se e somente se, $i < j$ e $R_t = \mathcal{F}_1 \cup \mathcal{F}_2 \ldots \mathcal{F}_k$. Em seguida, selecionam-se as i melhores fronteiras de $\mathcal{F}$, com o objetivo de formar a população P_{t+1}. Caso as fronteiras selecionadas não formem um conjunto de tamanho N, ordenam-se os indivíduos da fronteira $i+1$ de acordo com *crowding distance* e escolhem-se os $N - |P_{t+1}|$ primeiros para a população P_{t+1}. Na linha 23 uma população filha Q_t é criada a partir da aplicação de operadores de cruzamento e de mutação em uma população P_{t+1}.

3.3.6 SPEA2

O *Strength Pareto Evolutionary Algorithm* 2 (SPEA2) é um algoritmo evolucionário que foi desenvolvido por Zitzler e Thiele (1999) para problemas de otimização multi-

Algoritmo 7: NSGA-II

Entrada: pop, t_c, t_m, *critério de parada*

$ND \leftarrow$ Solução Inicial();
$P_0 \leftarrow$ Solução Inicial Aleatória(*pop*, *ND*) ;
$Q_0 \leftarrow \emptyset$;
$t \leftarrow 0$;
enquanto *critério de parada* não satisfeito **faça**
  $R_t \leftarrow P_t \cup Q_t$;
  $\mathcal{F} \leftarrow$ Fast non-dominated sorting(R_t) ;
  $P_{t+1} \leftarrow \emptyset$;
  $i \leftarrow 1$;
  enquanto $|P_{t+1}| + |\mathcal{F}_i| \leq N$ **faça**
    Atribuir *Crowding Distance*($\mathcal{F}_i$) ;
    $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i$;
    $i \leftarrow i + 1$;
  fim
  se $|P_{t+1}| < N$ **então**
    Ordenar($\mathcal{F}_i, \prec_n$);
    $j = 1$;
    enquanto $|P_{t+1}| < N$ **faça**
      $P_{t+1} \leftarrow P_{t+1} \cup \mathcal{F}_i[j]$;
      $j \leftarrow j + 1$;
    fim
  fim
  $Q_{t+1} \leftarrow$ Gerar Indivíduos da Próxima Geração(P_{t+1}, t_c, t_m) ;
  $t \leftarrow t + 1$;
fim
$ND \leftarrow$ soluções não-dominadas obtidas de P_t ;
retorna *ND* ;

objetivo. Ele preserva algumas características do seu antecessor, o SPEA, como por exemplo, a manutenção de um arquivo externo que armazena o conjunto de soluções não-dominadas. Entretanto, em três aspectos principais o SPEA2 diferencia-se do SPEA:

- A estratégia de atribuição da função de aptidão no SPEA2 considera, para cada indivíduo, tanto o número de soluções que o dominam quanto o número de soluções dominadas por ele.
- O SPEA2 utiliza uma técnica de estimativa de densidade da vizinhança incorporada à função de aptidão dos indivíduos; dessa forma, o processo de busca se torna mais preciso.
- No SPEA2 o algoritmo de agrupamento foi substituído por um novo método de truncamento, que previne a exclusão de soluções localizadas nas extremidades.

O pseudocódigo do SPEA2 é apresentado no Algoritmo 8.

Algoritmo 8: SPEA2

Entrada: pop, t_c, t_m, *critério de parada*

```
1  ND ← Solução Inicial();
2  P_0 ← Solução Inicial Aleatória(pop, ND) ;
3  A_0 ← ∅ ;
4  t ← 0 ;
5  enquanto critério de parada não satisfeito faça
6  |  para cada i ∈ P_t ∪ A_t faça
7  |  |  para cada j ∈ P_t ∪ A_t faça
8  |  |  |  se (i ≻ j) então
9  |  |  |  |  S(i) = S(i) + 1 ;
10 |  |  |  fim
11 |  |  fim
12 |  |  para cada j ∈ P_t ∪ A_t faça
13 |  |  |  se (i ≻ j) então
14 |  |  |  |  R(i) = R(i) + S(j) ;
15 |  |  |  fim
16 |  |  fim
17 |  |  D(i) ← 1/(σ_i^k + 2) ;
18 |  |  F(i) ← R(i) + D(i) ;
19 |  fim
20 |  A_{t+1} ← Seleção Populacional(P_t ∪ A_t, N) ;
21 |  se critério de parada não satisfeito então
22 |  |  P_{t+1} ← Gerar Indivíduos da Próxima Geração(A_{t+1}, t_c, t_m) ;
23 |  fim
24 |  t ← t + 1 ;
25 fim
26 ND ← soluções não-dominadas obtidas de A_t ;
27 retorna ND ;
```

No Algoritmo 8, inicialmente, cria-se um população inicial P_0 (linha 2) de tamanho *pop* e uma arquivo externo A_0 vazio que será utilizado para guardar os melhores indivíduos. A cada iteração do laço principal (linha 5 – 25), é realizado o cálculo da função de aptidão para cada indivíduo (linha 6 – 19). Esse cálculo é detalhado na seção 3.3.6.2. Em seguida, realiza-se o processo de seleção populacional, linha 20, que é detalhado na seção 3.3.6.1. Caso o critério de parada ainda não esteja satisfeito, uma etapa de reprodução é feita (linha 22) para gerar novos indivíduos para a população.

3.3.6.1 Arquivo Externo

O arquivo externo da próxima geração A_{p+1} é formado pelos N melhores indivíduos da união entre a população atual P_t e o arquivo externo atual A_t. Inicialmente, selecionam-se todos os indivíduos não-dominados, que podem ser definidos pela Equação 3.13:

$$A_{t+1} = \{i \mid i \in (P_t \cup A_t) \wedge F(i) < 1\} \tag{3.13}$$

Como o valor de $R(i) = 0$ para todo indivíduo não-dominado, então o valor da função de avaliação desses indivíduos será sempre menor que 1.

Se o tamanho do conjunto A_{p+1} for exatamente igual a N, o processo está completo. Entretanto, existem outras duas situações possíveis: o tamanho do conjunto pode ser menor que N, ($|A_{p+1}| < N$), ou o tamanho pode maior que N, ($|A_{p+1}| > N$).

No primeiro caso, os $N - |A_{p+1}|$ melhores indivíduos dominados, são selecionados do conjunto formado pela união de P_t e A_t. Nesta etapa, ordenam-se os indivíduos de acordo com a função de aptidão.

No segundo caso, realiza-se um processo chamado de truncamento que remove iterativamente um indivíduo de cada vez do conjunto A_{p+1} até que $|A_{p+1}| = N$. O indivíduo escolhido em cada iteração é aquele que tem a menor distância em relação aos outros indivíduos; dessa forma, um indivíduo localizado em uma região mais populosa é preferencialmente escolhido para ser removido da população.

3.3.6.2 Função de Aptidão

Para cada indivíduo i presente no arquivo externo A e na população P, atribui-se um valor de *strenght* $S(i)$, calculado conforma a Equação 3.14:

$$S(i) = |\{j \mid j \in P_t \cup A_t \wedge i \preceq j\}| \tag{3.14}$$

O valor de $S(i)$ representa o número de soluções que são dominadas pela solução i. Calcula-se ainda o valor de *raw fitness* $R(i)$ para cada solução de A e de P, pela Equação 3.15:

$$R(i) = \sum_{j \in P_t \cup A_t, j \preceq i} S(j) \tag{3.15}$$

O valor da *raw fitness* de i é calculado pelo somatório dos valores de $S(j)$ para todo indivíduo j que domina i. Assim, os indivíduos não-dominados terão $R(i) = 0$, enquanto que um alto valor para $R(i)$ indica que i é dominado por muitos outros indivíduos.

Para completar o cálculo da função de aptidão, é necessário encontrar o valor da densidade para cada solução. Esse cálculo é feito pelo Equação 3.16:

$$D(i) = \frac{1}{dist_{ij}^{k} + 2} \tag{3.16}$$

Para encontrar o valor de $dist$, deve-se calcular para cada solução i as distâncias euclidianas em relação às outras soluções $j \in P_t \cup A_t, j \neq i$, em seguida ordenar essas distâncias em ordem crescente, de modo que a k-ésima solução mais próxima é escolhida para que seja feito o cálculo da densidade. Recomenda-se o uso de $k = \sqrt{|P_t| + |A_t|}$.

Por fim, o valor da função de avaliação $F(i)$ é calculado pela Equação 3.17:

$$F(i) = R(i) + D(i) \tag{3.17}$$

O cálculo da função de aptidão tem complexidade computacional da ordem de $O(M^2 \log M)$, em que $M = |P_t| + |A_t|$.

3.4 Trabalhos Relacionados

Não foram encontrados na literatura trabalhos que resolvam o problema abordado com as mesmas características descritas. Em vista disso, as seções a seguir relatam abordagens correlatas encontradas na literatura, em que se consideram parte dessas características.

3.4.1 *Makespan*

van der Veen et al. (1998) abordam o problema de sequenciamento de tarefas em uma máquina em que as tarefas são agrupadas de acordo com suas características e um tempo de preparação é necessário apenas quando duas tarefas consecutivas são de grupos diferentes. Neste problema o objetivo é minimizar o *makespan*. Os autores mostram que o problema pode ser modelado como o problema do caixeiro viajante assimétrico. Esse trabalho apresenta uma solução para o problema do caixeiro viajante assimétrico utilizando uma estrutura especial de matriz de distâncias.

Em Yuan et al. (2006) apresenta-se o problema de sequenciamento de tarefas em uma máquina com tempo de preparação dependente da família em que as tarefas estão disponíveis para serem executadas a partir de certo instante. O objetivo é minimizar o *makespan* e neste trabalho mostra-se que este problema é fortemente NP-difícil. Os autores apresentam ainda um algoritmo de programação dinâmica para solucionar o problema.

O trabalho de Montoya Torres et al. (2009) apresenta uma heurística baseada em busca randômica para resolver o problema de sequenciamento em uma máquina com o objetivo de minimizar o *makespan*. Neste trabalho considera-se que cada tarefa possui um tempo de preparação que depende da sequência das tarefas e, além disso, cada tarefa está disponível a partir de um certo instante.

No trabalho de Tavakkoli-Moghaddam et al. (2010) é abordado o problema de sequenciamento de tarefas em uma máquina em que os objetivos são minimizar o *makespan* e o atraso total ponderado. Na formulação é considerado que existem n tarefas para serem processadas a partir do instante R_i, cada tarefa tem um tempo de processamento, uma data de entrega e um peso por atraso. Os autores propõem um método de programação linear multiobjetivo baseado em lógica *fuzzy* para resolver o problema. Um exemplo numérico mostra a viabilidade do método proposto.

3.4.2 Atraso Total

Pinedo (2008) demonstra que o problema 3-*partition* pode ser reduzido ao problema de sequenciamento em uma máquina com o objetivo de minimizar o atraso total ponderado. Portanto, este problema também é um problema pertencente à classe $\mathcal{NP}$-difícil.

Schaller e Gupta (2008) consideram o problema de sequenciar um conjunto de tarefas em uma máquina com o objetivo de minimizar o atraso e o adiantamento

total. Nesta abordagem as tarefas foram agrupadas em famílias. Os autores propuseram um procedimento *branch-and-bound* para o problema com *group technology assumption* e outro para o problema sem *group technology assumption*. Além destes, um algoritmo heurístico chamado Algoritmo H também foi proposto. O trabalho considera impraticável a utilização do método *branch-and-bound* para grandes instâncias, pois o tempo de CPU requerido cresce muito rapidamente em relação ao tamanho da instância. A partir dos resultados empíricos foi possível concluir que o sequenciamento de tarefas com a remoção do *group technology assumption* pode reduzir significativamente o atraso e adiantamento total.

Chantaravarapan (2002) aborda dois problemas de sequenciamento em máquina simples com o objetivo de minimizar o atraso total ponderado: o primeiro com tempo de preparação dependente da sequência e outro com tempo de preparação independente da sequência. Para o primeiro problema o autor propõe três heurísticas construtivas e cinco heurísticas de melhoramento. Para o segundo problema, foi proposto um algoritmo genético hibrido. Utilizou-se duas estratégias para comparar os resultados. Para as instâncias menores, a comparação foi feita em relação ao resultado do resolvedor CPLEX. Por outro lado, para as instâncias maiores, os resultados foram comparadas com uma outra heurística chamada Combo, que combina três outras heurísticas de melhoramento. Além disso, foram aplicados os testes estatísticos Friedman e Kruskal-Wallis para verificar a existência de diferença estatisticamente significativa entre os resultados. Os experimentos mostraram que o algoritmo genético híbrido conseguiu melhor desempenho do que a heurística combo, a exceção ocorreu para instâncias maiores com intervalo de data de entrega muito pequeno.

Tanaka e Araki (2012) estudaram o problema de sequenciamento de tarefas em uma máquina. Este trabalho considerou a existência de tempo de preparação entre a execução de duas tarefas e o objetivo é minimizar o atraso total ponderado. O estudo propôs um algoritmo exato de programação dinâmica para resolver o problema. Os experimentos foram realizados em três conjuntos de instâncias: o primeiro com 120 instâncias de 60 tarefas; o segundo com 32 instâncias com 15, 25, 35 e 45 tarefas; e o terceiro com 32 instâncias de 55, 65, 75 e 85 tarefas. O algoritmo foi implementado na linguagem C, e conseguiu encontrar a solução ótima para quase todas as instâncias analisadas.

3.4.3 Famílias de Tarefas

Em Baker e Magazine (2000) é tratado o problema de sequenciamento de tarefas em uma máquina em que o tempo de preparação depende das famílias das tarefas. Considerou-se o objetivo de minimizar o *lateness* máximo. Foi proposto um algoritmo *branch and bound* combinado com algumas propriedades deste problema que permitem reduzir o número de sequencias a serem consideradas para encontrar a solução ótima. Observou-se que o tempo necessário para a solução do problema está relacionado a vários fatores como número de famílias, número de tarefas por família, tempo de preparação e data de entrega do problema. Os experimentos foram realizados em instâncias com tamanho de 30 a 100 tarefas, e conclui-se que o método executa em tempo aceitável para instâncias de até 60 tarefas.

Jin et al. (2009) consideram o problema de sequenciamento de tarefas em uma máquina com tempo de preparação dependente da família com o objetivo de minimizar o *lateness* máximo. Os autores propuseram o algoritmo denominado *batch-based simulated annealing* (BSA), ele combina a metaheurística *simulated annealing* e as propriedades apresentadas em Baker e Magazine (2000). Os experimentos foram realizado em instâncias com tamanho entre 60 e 100 tarefas. Os resultado obtidos pelo BSA foram comparados com os resultado do algoritmo *simulated annealing*, a partir dos resultados foi possível concluir que o algoritmo BSA obteve os melhores resultados.

3.4.4 Problemas de Otimização Multiobjetivo

Choobineh et al. (2006) abordam o problema de sequenciamento em uma máquina com tempo de preparação dependente da sequência das tarefas. A função objetivo foi formada por uma função ponderada com os seguintes objetivos: minimizar o *makespan*, minimizar o número de tarefas atrasadas e minimizar o atraso total. Os autores apresentaram uma formulação de programação inteira mista multiobjetivo e um algoritmo busca tabu multiobjetivo para o problema. Os experimentos foram realizados com instâncias de 10 à 20 tarefas. Os testes realizados indicaram que, em geral, o algoritmo de busca tabu foi capaz de encontrar boas soluções e, em alguns casos, encontrou a solução ótima.

Li et al. (2012) consideraram o problema de sequenciamento de tarefas em máquinas paralelas, em que existem n tarefas que devem ser processadas por m máquinas idênticas. Todas as máquinas estão disponíveis a partir do instante zero e cada tarefa está disponível a partir do instante r_j. Um tempo de preparação é necessário entre a execução de duas tarefas consecutivas. Cada tarefa deve ser processada por um única máquina. Esse problema possui os seguintes objetivos: minimizar o *makespan* e minimizar o atraso total. Os autores apresentaram um modelo matemático de programação inteira mista, que não foi considerado nos experimentos. Uma nova versão do NSGA-II baseada em lógica *fuzzy* foi proposta denominado FLC-NSGA-II. Os experimentos foram feitos em 960 instâncias divididas em dois conjuntos, o primeiro de 20 tarefas e 3 máquinas e o segundo de 50 tarefas e 5 máquinas. Os resultados dos algoritmo FLC-NSGA-II foram comparados com os resultados do NSGA-II, segundo as métricas de cardinalidade e distância média. O algoritmo proposto obteve melhor resultado que o NSGA-II em relação as métrica avaliadas.

Capítulo 4

Metodologia

Neste capítulo são apresentados sete algoritmos multiobjetivos para resolver o problema abordado, sendo cinco baseados em busca local e dois em busca populacional. Nas seções a seguir são apresentadas as componentes dos algoritmos, a saber, representação e geração de soluções iniciais, estruturas de vizinhança, perturbação, seleção, cruzamento e mutação. Na seção 4.7 são detalhados os algoritmos implementados. Finalmente, na seção 4.9 são apresentados e discutidos valores para os parâmetros dos algoritmos.

4.1 Representação da Solução

Uma solução é representada por uma sequência $\pi = \{\pi_1, \pi_2, \ldots, \pi_k, \ldots, \pi_n\}$, em que π_k indica a k-ésima tarefa a ser executada.

Um exemplo de sequenciamento, não necessariamente ótimo, para o problema apresentado no capítulo 2 é a sequência $\pi = \{3, 4, 5, 2, 1\}$. A representação dessa solução é feita por um vetor conforme a Figura 4.1.

Figura 4.1: Representação da Solução

4.2 Geração da Solução Inicial

Os algoritmos propostos partem de um conjunto de soluções não-dominadas geradas por quatro diferentes heurísticas baseadas em regras de prioridade (Valente, 2006): *Earliest Due Date* (EDD), *Shortest Processing Time* (SPT), *Longest Processing Time* (LPT) e *Minimum Slack Time* (MST).

A regra EDD foi proposta por Jackson (1955), e é bastante conhecida por fornecer a solução ótima para o problema de minimização do atraso máximo em uma máquina. Para construir uma solução a partir da regra EDD, deve-se sequenciar as

tarefas em ordem não-crescente de suas datas de entrega. Esse método tem complexidade computacional de $O(n \log n)$. Uma solução gerada pela regra EDD possui a seguinte característica: $d_{\pi(1)} \leq d_{\pi(2)} \leq \cdots \leq d_{\pi(n)}$.

A regra SPT foi desenvolvida por Smith (1956), e é conhecida por produzir um sequenciamento ótimo para o problema de minimização de fluxo total em *flowshop*. Essa regra constrói uma solução a partir do sequenciamento das tarefas em ordem não-decrescente dos tempos de processamento. Esse método tem complexidade computacional de $O(n \log n)$. Uma solução gerada pela regra SPT possui a seguinte característica: $p_{\pi(1)} \leq p_{\pi(2)} \leq \cdots \leq p_{\pi(n)}$.

A regra MST é frequentemente empregada para minimizar o atraso total. Uma solução construída a partir desta regra é gerada a partir do sequenciamento das tarefas em ordem não-decrescente da diferença entre a data de entrega e o tempo de processamento. Esse método tem complexidade computacional de $O(n \log n)$. A solução gerada pela regra MST possui a seguinte característica: $d_{\pi(1)} - p_{\pi(1)} \leq d_{\pi(2)} - p_{\pi(2)} \leq \cdots \leq d_{\pi(n)} - p_{\pi(n)}$.

A regra LPT gera uma solução a partir do sequenciamento das tarefas em ordem não-crescente dos tempos de processamento. Esse método tem complexidade computacional de $O(n \log n)$. Uma solução gerada pela regra LPT possui a seguinte característica: $p_{\pi(1)} \geq p_{\pi(2)} \geq \ldots p_{\pi(n)}$.

Caso essas soluções geradas sejam não-dominadas, o conjunto inicial é formado por quatro soluções. No pior caso, uma solução domina todas as outras e, assim, o conjunto inicial será formado apenas por essa solução.

4.3 Estruturas de Vizinhança

Para explorar o espaço de soluções dos problemas, são aplicados movimentos de inserção e troca no sequenciamento das tarefas, conforme descrito a seguir.

4.3.1 Vizinhança de Inserção

Dada uma sequência $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, o movimento de inserção de uma tarefa π_x consiste em deslocá-la para a posição y ($y \neq x$ e $y \neq x - 1$). O conjunto de movimentos de inserção para uma sequência π define a vizinhança $N^I(\pi)$, a qual é composta por $(n-1)^2$ soluções.

Na Figura 4.2 é ilustrado um movimento de inserção em que o valor de x é menor que y.

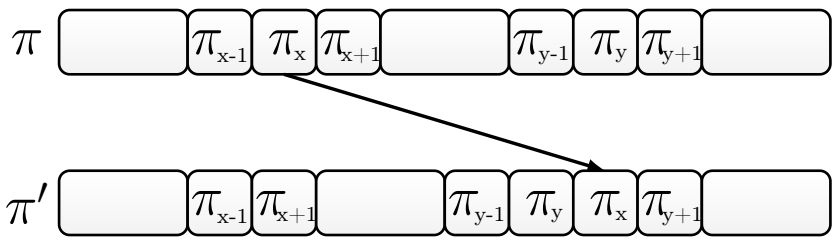

Figura 4.2: Movimento de inserção para $x < y$

Na Figura 4.3 pode-se observar um movimento de inserção em que o valor de y é menor que x.

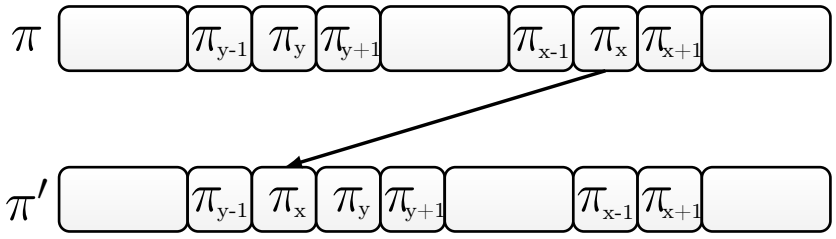

Figura 4.3: Movimento de inserção para $x > y$

4.3.2 Vizinhança de Troca

Dada uma sequência $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, o movimento de troca entre duas tarefas π_x e π_y consiste em deslocar a tarefa π_x para a posição y e a tarefa π_y para a posição x. O conjunto de movimentos de troca para uma sequência π define a vizinhança $N^T(\pi)$, formada por $\dfrac{n(n-1)}{2}$ soluções.

Na Figura 4.4 pode-se ver um movimento de troca entre duas tarefas π_x e π_y.

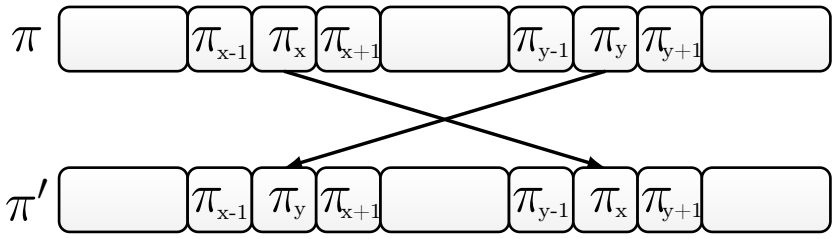

Figura 4.4: Movimento de troca

4.4 Perturbação

O procedimento de perturbação de uma solução (linha 24 do Algoritmo 4) foi modificado em relação à proposta de Geiger (2009), a qual está descrita na seção 3.3.4. A perturbação é aplicada em níveis, variando de 1 a $(n/2 - 1)$. Para cada nível p são realizadas $p + 1$ modificações (perturbações) na solução. Assim, no nível mais baixo da perturbação, são realizadas duas trocas enquanto que no nível mais alto são realizadas $n/2$ trocas. O nível p de perturbação aumenta à medida que a perturbação não é capaz de gerar uma solução não-dominada em relação ao conjunto *ND*. O aumento é feito somando-se uma unidade ao valor atual do nível da perturbação. No momento em que alguma solução não-dominada em relação ao conjunto *ND* é encontrada, o nível de perturbação retorna ao seu valor mínimo, no caso, 1. Caso o nível de perturbação alcance seu valor máximo, isto é, $(n/2 - 1)$ e ainda assim não se consiga gerar uma solução não-dominada em relação ao conjunto *ND*, o nível

de perturbação retorna ao seu valor mínimo. O procedimento proposto funciona, então, como segue. Uma solução π do conjunto *ND* é selecionada aleatoriamente. A seguir, escolhe-se, também aleatoriamente, um subconjunto de tarefas consecutivas de π nas posições $j, j+1, \ldots, j+2p+1$. Aplicam-se, então, as trocas entre os pares de tarefas $(j, j+2p+1), (j+1, j+2p), \ldots, (j+p, j+p+1)$. Dessa forma, o procedimento realiza $p+1$ movimentos de troca a cada chamada ao procedimento de perturbação. O algoritmo PILS com esta modificação será denominado PILS1.

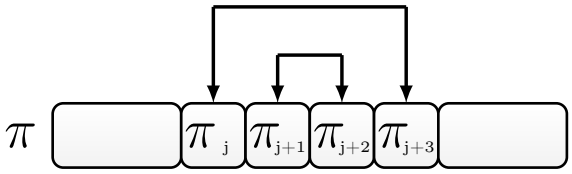

Figura 4.5: Nível 1 da Perturbação Proposta

A Figura 4.5 ilustra o primeiro nível de perturbação do método proposto. Neste nível são selecionadas quatro tarefas consecutivas $j, j+1, j+2, j+3$ para a realização de duas trocas. A primeira troca ocorre entre as tarefas j e $j+3$ e a segunda entre as tarefas $j+1$ e $j+2$.

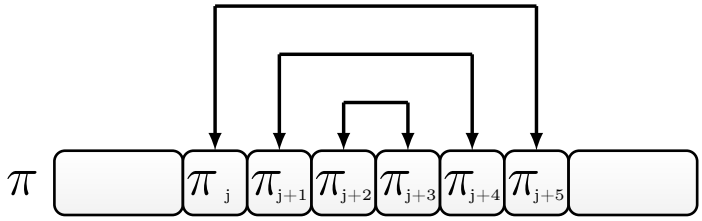

Figura 4.6: Nível 2 da Perturbação Proposta

A Figura 4.6 ilustra o próximo nível de perturbação do método proposto. Esse nível tem funcionamento parecido com o do nível anterior, diferenciando-se da anterior pela quantidade de trocas que são realizadas. Nesse nível são selecionadas 6 tarefas consecutivas e são feitas 3 trocas.

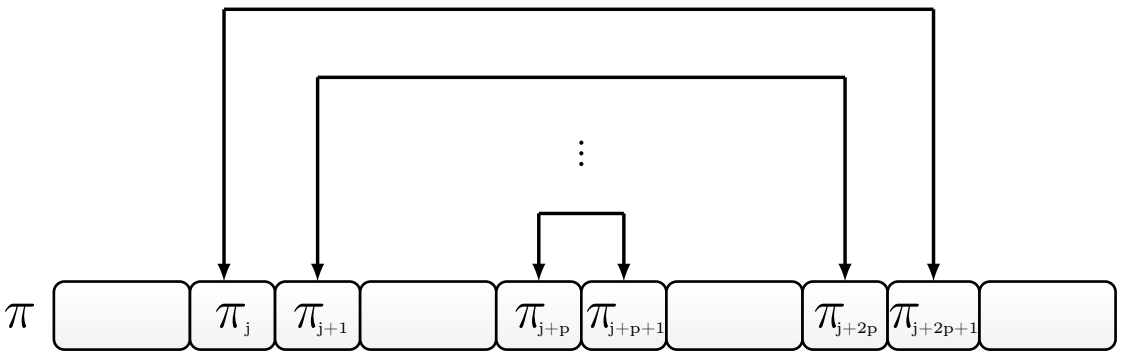

Figura 4.7: Nível p da Perturbação Proposta

A Figura 4.6 ilustra o procedimento em um nível p qualquer de perturbação, que consiste em realizar $p - 1$ trocas.

4.5 Seleção

Pode-se classificar os operadores genéticos de seleção em dois principais tipos: seleção para reprodução e seleção populacional.

4.5.1 Seleção para Reprodução

Em um algoritmo genético, normalmente, apenas alguns indivíduos participam da reprodução. Dessa forma, é necessário implementar um mecanismo que realize a seleção dos indivíduos da população atual que sofrerão cruzamento ou mutação. Os dois algoritmos desenvolvidos neste trabalho realizam essa tarefa utilizando o operador de seleção por torneio binário. Nesse operador, dois pares de indivíduos são selecionados aleatoriamente, mas apenas o melhor indivíduo de cada par é escolhido para fazer a reprodução. No NSGA-II escolhe-se o indivíduo que domina o outro; caso eles sejam não-dominados, escolhe-se o que tiver maior valor para a *crowding distance*. No SPEA2, seleciona-se o indivíduo que apresenta o menor valor para a função de aptidão. Caso não seja possível diferenciar os indivíduos, um deles é escolhido aleatoriamente.

4.5.2 Seleção Populacional

Um outro operador de seleção deve ser aplicado para escolher os indivíduos que estarão presentes na próxima geração. Neste trabalho a escolha é feita através de uma seleção elitista. Dessa forma, os indivíduos com maior aptidão são escolhidos para formarem a próxima geração, respeitando, sempre, o tamanho da população. As funções de aptidão variam de acordo com o algoritmo, e estão descritas nas seções 3.3.5 e 3.3.6.

4.6 Reprodução

Nos Algoritmos Genéticos, cada cromossomo está associado a uma solução do problema e cada gene está associado a uma componente da solução.

O processo de reprodução consiste em gerar uma população de novos indivíduos a partir da aplicação dos operadores de cruzamento e mutação.

4.6.1 Cruzamento

O operador de cruzamento atua em dois indivíduos de cada vez, gerando descendentes através da combinação dos cromossomos dos dois indivíduos. Esses operadores possuem diferentes variações, dependendo do problema tratado. Diversas pesquisas têm sido desenvolvidas com o objetivo de criar operadores genéticos para serem utilizados em problemas de sequenciamento. (Cheng et al., 1999; Dahal et al., 2007)

discutem os principais operadores que podem ser utilizados nessa classe de problemas. No presente trabalho foram utilizados três operadores de cruzamento: o *Order Crossover* (OX), o *Linear Order Crossover* (LOX) e o *Partial-Mapped Crossover* (PMX), a seguir descritos.

4.6.1.1 Operador OX

O operador genético *Order Crossover* OX foi proposto por Goldberg (1989). Ele pode ser implementado de acordo com os seguintes passos:

1. Dois pontos de corte são definidos nos cromossomos do pai 1 (P_1) e do pai 2 (P_2).
2. Os genes localizados entre os dois cortes de P_1 são herdados integralmente pelo filho 1 (O_1) e os genes de P_2 são herdados pelo filho 2 (O_2). Preservando, dessa forma, a ordem das tarefas nos descendentes.
3. O restante do cromossomo de O_1 é herdado de P_2, iniciando pelo corte 2 e desconsiderando as tarefas que já estão presentes em O_1. De maneria semelhante, o cromossomo de O_2 herda as características de P_1.

Para exemplificar a aplicação do operador de cruzamento OX, considere dois indivíduos:

$$P_1 = (\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$
$$P_2 = (\ 5\ 4\ 6\ 9\ 2\ 1\ 7\ 8\ 3)$$

para gerar dois descendentes O_1 e O_2 a partir da aplicação do operador OX. Inicialmente, defini-se dois pontos de corte em P_1 e P_2. Neste exemplo, o primeiro ponto foi definido após o gene que está na segunda posição e o segundo ponto de corte foi definido após o gene da sexta posição.

$$P_1 = (\ 1\ 2\ |\ 3\ 4\ 5\ 6\ |\ 7\ 8\ 9)$$
$$P_2 = (\ 5\ 4\ |\ 6\ 9\ 2\ 1\ |\ 7\ 8\ 3)$$

Em seguida, O_1 e O_2 herdam a sequência entre os dois pontos de corte de P_1 e P_2, respectivamente:

$$O_1 = (\ \mathrm{X}\ \mathrm{X}\ |\ 3\ 4\ 5\ 6\ |\ \mathrm{X}\ \mathrm{X}\ \mathrm{X})$$
$$O_2 = (\ \mathrm{X}\ \mathrm{X}\ |\ 6\ 9\ 2\ 1\ |\ \mathrm{X}\ \mathrm{X}\ \mathrm{X})$$

Os demais genes de O_1 são herdados de P_2. Para isso, é necessário definir uma sequência de tarefas, começando no segundo corte de P_2:

$$7 - 8 - 3 - 5 - 4 - 6 - 9 - 2 - 1$$

A seguir, devem ser removidas desta lista os genes que já estão presentes em O_1, resultando na sequência:

$$7 - 8 - 9 - 2 - 1$$

Esta sequência deve ser inserida em O_1 a partir da posição do segundo corte. Com este procedimento, o cromossomo de O_1 será:

$$O_1 = (\ 2\ 1 \mid 3\ 4\ 5\ 6 \mid 7\ 8\ 9)$$

Um processo semelhante gera O_2 a partir do pai P_1, e gera o seguinte cromossomo:

$$O_2 = (\ 4\ 5 \mid 6\ 9\ 2\ 1 \mid 7\ 8\ 3)$$

Na Figura 4.8 apresenta-se o resultado da aplicação do operador de cruzamento OX a dois indivíduos P_1 e P_2 e a consequente formação de dois descendentes O_1 e O_2.

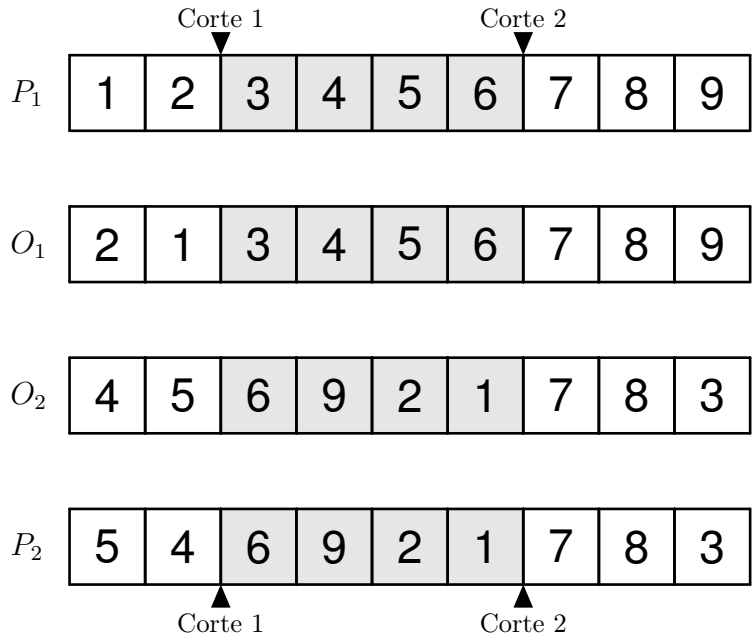

Figura 4.8: Operador OX

4.6.1.2 Operador LOX

O operador genético *Linear Order Crossover* LOX foi proposto por Falkenauer e Bouffouix (1991). Ele consiste em gerar novos descendentes a partir dos seguintes passos:

1. Dois pontos de corte são definidos nos cromossomos do pai 1 (P_1) e do pai 2 (P_2).
2. Os genes localizados entre os dois cortes de P_1 são herdados integralmente pelo filho 1 (O_1) e os genes de P_2 são herdados pelo filho 2 (O_2).
3. O restante do cromossomo de O_1 é herdado de P_2, iniciando pelo gene da primeira posição de P_2 e desconsiderando os genes que já estão presentes em O_1. De maneira semelhante, o cromossomo de O_2 herda as características de P_1.

Para mostrar aplicação do operador de cruzamento LOX, considere dois indivíduos:

$$P_1 = (\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$
$$P_2 = (\ 5\ 4\ 6\ 9\ 2\ 1\ 7\ 8\ 3)$$

Esses indivíduos são usados para gerar dois descendentes O_1 e O_2 a partir da aplicação do operador LOX. Inicialmente, define-se dois pontos de corte em P_1 e P_2. Neste exemplo, o primeiro ponto foi definido após o gene que está na segunda posição e o segundo corte foi definido após o gene da sexta posição.

$$P_1 = (\ 1\ 2\ |\ 3\ 4\ 5\ 6\ |\ 7\ 8\ 9)$$
$$P_2 = (\ 5\ 4\ |\ 6\ 9\ 2\ 1\ |\ 7\ 8\ 3)$$

Em seguida, O_1 e O_2 herdam a sequência entre os dois pontos de corte de P_1 e P_2, respectivamente:

$$O_1 = (\ \mathrm{X}\ \mathrm{X}\ |\ 3\ 4\ 5\ 6\ |\ \mathrm{X}\ \mathrm{X}\ \mathrm{X})$$
$$O_2 = (\ \mathrm{X}\ \mathrm{X}\ |\ 6\ 9\ 2\ 1\ |\ \mathrm{X}\ \mathrm{X}\ \mathrm{X})$$

Os demais genes de O_1 são herdados de P_2. Para isso, é necessário definir uma sequência de tarefas, começando do primeiro elemento de P_2:

$$5 - 4 - 6 - 9 - 2 - 1 - 7 - 8 - 3$$

No entanto, devem ser removidas desta lista os genes que já estão presentes em O_1; assim a nova lista será:

$$9 - 2 - 1 - 7 - 8$$

Essa sequência deve ser inserida em O_1 a partir da primeira posição. Por fim, o cromossomo de O_1 será:

$$O_1 = (\ 9\ 2\ |\ 3\ 4\ 5\ 6\ |\ 1\ 7\ 8)$$

Um processo semelhante gera O_2 a partir do pai P_1, resultando no seguinte cromossomo:

$$O_2 = (\ 3\ 4\ |\ 6\ 9\ 2\ 1\ |\ 5\ 7\ 8)$$

Na Figura 4.9 estão representados os pontos de corte dos pais P_1 e P_2, e os filhos O_1 e O_2 gerados a partir da aplicação do operador LOX.

A principal diferença entre os operadores OX e LOX é a escolha da posição para completar a formação do cromossomo filho. No operador OX escolhe-se o gene na posição após o segundo ponto de corte, enquanto que no operador LOX escolhe-se o gene na primeira posição do cromossomo do pai.

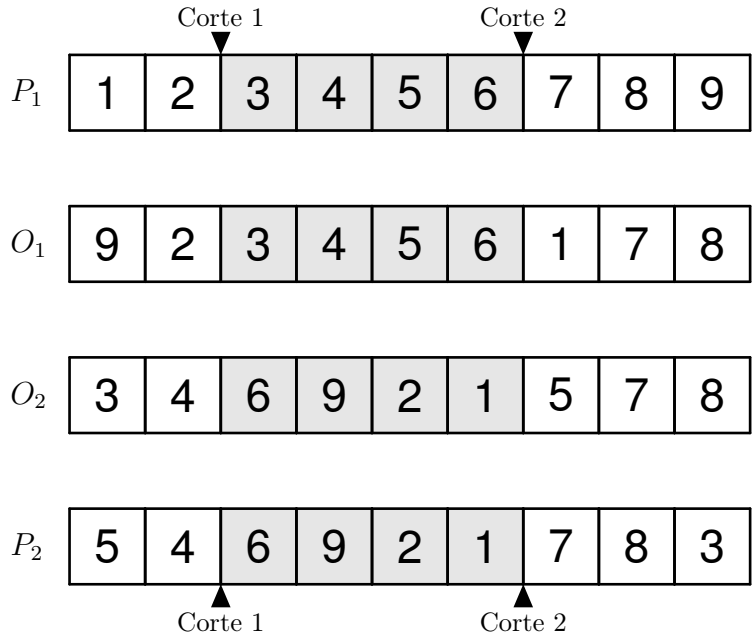

Figura 4.9: Operador LOX

4.6.1.3 Operador PMX

O operador *Partial Mapped Crossover* PMX foi proposto por Goldberg e Lingle (1985). O seu funcionamento segue os seguintes passos:

1. Dois pontos de corte são definidos nos cromossomos do pai 1 (P_1) e do pai 2 (P_2).

2. Os genes localizados entre os dois cortes de P_1 são herdados integralmente pelo filho 2 (O_2) e os genes de P_2 são herdados pelo filho 1 (O_1).

3. Os genes de P_1 localizados a esquerda do primeiro ponto de corte juntamente com os genes da direita do segundo ponto de corte que ainda não estão presentes em O_1, também são herdados pelo filho respeitando a posição no pai. O mesmo acontece com O_2 que herda os genes de P_2.

4. O restante do cromossomo de O_1 é composto pelo resultado do mapeamento dos genes de P_2 em relação aos genes de P_1, desconsiderando os genes que já estão presentes em O_1. Um processo semelhante é realizado para gerar O_2.

Para exemplificar o funcionamento do operador PMX, considere dois indivíduos:

$$P_1 = (\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$
$$P_2 = (\ 4\ 2\ 6\ 1\ 8\ 5\ 9\ 3\ 7)$$

para gerar dois descendentes O_1 e O_2 a partir da aplicação do operador PMX. Inicialmente, define-se dois pontos de corte em P_1 e P_2. Neste exemplo, o primeiro ponto foi definido após o elemento que está na terceira posição e o segundo ponto de corte foi definido após o elemento da sétima posição.

$$P_1 = (\ 1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$$
$$P_2 = (\ 4\ 2\ 6 \mid 1\ 8\ 5\ 9 \mid 3\ 7)$$

Em seguida, O_1 e O_2 herdam a sequência entre os dois pontos de corte de P_2 e P_1, respectivamente:

$$O_1 = (\ X\ X\ X \mid 1\ 8\ 5\ 9 \mid X\ X)$$
$$O_2 = (\ X\ X\ X \mid 4\ 5\ 6\ 7 \mid X\ X)$$

Na próxima etapa, O_1 herda os genes de P_1 localizados a esquerda do primeiro ponto de corte juntamente com os genes que estão à direita do segundo ponto de corte, esse gene é inserido em O_1 respeitando a posição do gene no pai. Nessa etapa são desconsiderados os genes que já estão presentes em O_1. Um processo semelhante é aplicado para O_2:

$$O_1 = (\ X\ 2\ 3 \mid 1\ 8\ 5\ 9 \mid X\ X)$$
$$O_2 = (\ X\ 2\ X \mid 4\ 5\ 6\ 7 \mid 3\ X)$$

Os demais genes de O_1 são preenchidos através do mapeamento dos elementos de P_1 em relação à P_2. O mapeamento é realizado em relação à posição dos elementos. A Figura 4.10 ilustra o mapeamento do primeiro elemento de O_1.

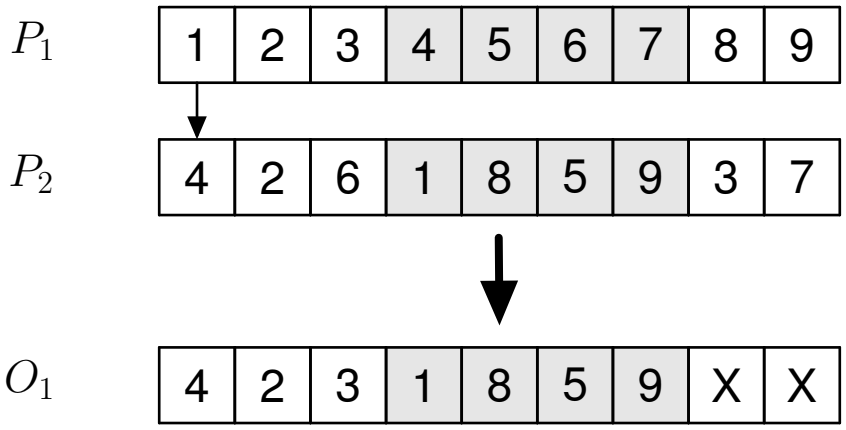

Figura 4.10: Mapeamento do primeiro elemento

De acordo com a Figura 4.10, para definir o primeiro elemento de O_1, deve-se encontrar a tarefa que está localizada na primeira posição de P_2, no caso, a tarefa 4. Como O_1 ainda não tem essa tarefa, o mapeamento é finalizado. Dessa forma, a tarefa 4 é inserida na primeira posição de O_1.

Repete-se esse procedimento para o próximo elemento que está localizado na oitava posição de P_1. A Figura 4.11 ilustra o mapeamento para o segundo elemento de O_1.

Conforme a Figura 4.11, o oitavo elemento de P_1 é mapeado em relação ao elemento que está na mesma posição de P_2, e o valor obtido é 3. Como esse valor já está presente em O_1, um novo mapeamento é feito. Para realizar esse novo mapeamento deve-se encontrar a posição do elemento 3 em P_1 e, assim, obter o

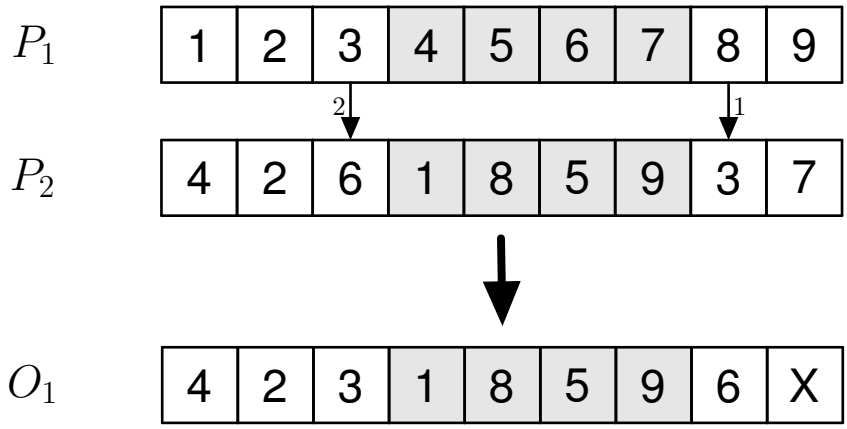

Figura 4.11: Mapeamento do segundo elemento

elemento que está na mesma posição em P_2. Para o exemplo dado, a posição é 3 e o elemento nesta posição em P_2 tem valor igual a 6. Tendo em vista que O_1 ainda não tem o elemento com esse valor, o mapeamento está completo e esse elemento é inserido em O_1 na oitava posição.

Por fim, o elemento da nona posição é mapeado, e o valor obtido é 7. Esse valor é inserido na nona posição de O_1. Após esse processo O_1 terá a seguinte composição:

$$O_1 = (4\ 2\ 3 \mid 1\ 8\ 5\ 9 \mid 6\ 7)$$

Um filho O_2 é gerado a partir do pai P_1 com as seguintes características:

$$O_2 = (\ 1\ 2\ 8 \mid 4\ 5\ 6\ 7 \mid 3\ 9)$$

Considerando que foram criados três operadores (OX, LOX ou PMX), a cada cruzamento, escolhe-se aleatoriamente um operador para realizar o processo de reprodução. O cruzamento ocorre com uma probabilidade igual a t_c.

4.6.2 Mutação

A mutação é uma mudança na estrutura genética de um indivíduo. Nos algoritmos genéticos os operadores de mutação são importantes instrumentos que permitem introduzir características novas ao indivíduo. Com isso, é possível manter a diversidade genética da população, diminuindo as chances de que o algoritmo fique preso a um ótimo local. Foram utilizados dois operadores de mutação: troca e inserção.

Nos algoritmos implementados a mutação ocorre com uma probabilidade t_m.

4.6.2.1 Operador de Troca

O operador de troca consiste em selecionar dois genes do indivíduo e trocá-los de posição. Por exemplo, considere o seguinte indivíduo:

$$P = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$$

Para aplicar o operador de troca deve-se selecionar dois genes (sublinhados no exemplo abaixo):

$$P = (4\ 1\ \underline{2}\ 8\ \underline{7}\ 6\ 9\ 3\ 5)$$

Em seguida realiza-se a troca de posição entre os genes selecionados. No exemplo foram selecionados os genes 2 e 7, e o resultado será um indivíduo com uma nova formação genética:

$$P' = (4\ 1\ 7\ 8\ 2\ 6\ 9\ 3\ 5)$$

4.6.2.2 Operador de Inserção

O Operador de inserção remove um gene de uma posição e o insere em outra posição diferente. Para exemplificar, considere o seguinte indivíduo:

$$P = (9\ 3\ 8\ 1\ 5\ 7\ 2\ 6\ 4)$$

Para aplicar o operador de inserção deve-se selecionar um gene:

$$P = (9\ 3\ 8\ 1\ 5\ 7\ \underline{2}\ 6\ 4)$$

Em seguida seleciona-se uma nova posição para inserir o gene selecionado, por exemplo 5, como resultado tem-se um indivíduo com uma nova formação genética:

$$P' = (9\ 3\ 8\ 1\ 2\ 5\ 7\ 6\ 4)$$

A mutação acontece com uma probabilidade igual a t_m, e a cada mutação escolhe-se aleatoriamente um entre os dois operadores (troca ou inserção) para ser utilizado.

4.7 Algoritmos Implementados

Foram implementados sete algoritmos, a seguir descritos.

4.7.1 MOVNS

O algoritmo MOVNS implementado é aquele descrito no Algoritmo 1 com as seguintes adaptações:

- Na linha 1 é determinada a solução inicial com base no procedimento descrito na seção 4.2;
- Na linha 2 adota-se como critério de parada aquele descrito na seção 4.9;
- Na linha 5 é feita a escolha de uma vizinhança dentre aquelas descritas na seção 4.3.

4.7.2 MOVNS_Ottoni

O algoritmo MOVNS_Ottoni consiste em adicionar um procedimento de intensificação entre as linhas 12 e 13 do Algoritmo 1. Para implementar este método, além das mesmas adaptações feitas no MOVNS, foi definido um valor para o parâmetro n_r, apresentado na seção 4.9.

4.7.3 MOVNS_Arroyo

Um procedimento de intensificação diferente, apresentado no Algoritmo 3, é incluído entre as linhas 12 e 13 do Algoritmo 1, originando o MOVNS_Arroyo. Para implementar esse método, além das mesmas adaptações feitas no MOVNS, também foi necessário definir um valor para o parâmetro n_r. Essa definição é dada na seção 4.9.

4.7.4 PILS

O algoritmo PILS foi desenvolvido baseado no Algoritmo 4, adotando as seguintes modificações:

- A solução inicial da linha 1 é determinada pelo método descrito na seção 4.2;
- O critério de parada adotado pelo algoritmo na linha 3 é aquele descrito na seção 4.9;
- Na linha 6 é feita a escolha de uma vizinhança dentre aquelas apresentadas na seção 4.3.

4.7.5 PILS1

O PILS1 é um método baseado no algoritmo PILS. Apresenta-se no Algoritmo 9 o pseudocódigo básico deste algoritmo.

No Algoritmo 9, obtém-se, inicialmente, um conjunto de soluções não-dominadas (*ND*) (linha 1), de acordo com o procedimento descrito na seção 4.2. Em seguida, inicializa-se o nível com 1 (linha 2) e seleciona-se aleatoriamente uma das soluções do conjunto *ND* (linha 3) para ser a solução corrente s. A cada iteração do laço (linhas 4 – 35), explora-se toda a vizinhança de s (linhas 6 – 18). Se alguma solução vizinha s' dominar s (linha 10), então s' passa a ser a nova solução corrente (linha 11). Além disso, as vizinhanças são reordenadas aleatoriamente (linha 12) e o procedimento continua a partir da primeira vizinhança na nova ordem gerada. Depois de explorar toda a vizinhança da solução s, ela é marcada como visitada (linha 19). Em seguida, escolhe-se aleatoriamente uma solução não-visitada do conjunto *ND* para ser a nova solução corrente (linha 21) e atribui-se 1 ao nível de perturbação (linha 22). Caso todas as soluções do conjunto *ND* já tenham sido visitadas, seleciona-se aleatoriamente uma solução do conjunto *ND* (linha 25) e aplica-se uma perturbação (linha 26) com o nível n, gerando uma solução perturbada s'', que passa a ser a nova solução corrente (linha 27). Em seguida, se o nível de perturbação for igual a $n/2 - 1$, ele recebe 1 (linha 29), senão ele é incrementado em uma unidade (linha 32). O laço mais externo (linhas 4 – 35) é repetido enquanto o critério de parada não for satisfeito.

Algoritmo 9: PILS1

Entrada: critério de parada

```
1  ND ← SolucaoInicial();
2  nivel ← 1 ;
3  Selecione uma solução s ∈ ND ;
4  enquanto critério de parada não satisfeito faça
5  |  i ← 1 ;
6  |  enquanto i < k ∧ critério de parada não satisfeito faça
7  |  |  para cada s' ∈ N^i(s) faça
8  |  |  |  ND ← soluções não-dominadas obtidas de ND ∪ {s'} ;
9  |  |  fim
10 |  |  se ∃s' ∈ N^i(s) | s' domina s então
11 |  |  |  s ← s' ;
12 |  |  |  Reordene as vizinhanças N^1, ..., N^k, em uma ordem aleatória ;
13 |  |  |  i ← 1 ;
14 |  |  fim
15 |  |  senão
16 |  |  |  i++;
17 |  |  fim
18 |  fim
19 |  Marque s como visitada ;
20 |  se ∃s' ∈ ND | s'ainda não foi visitada então
21 |  |  s ← s';
22 |  |  nível ← 1 ;
23 |  fim
24 |  senão
25 |  |  Selecione uma solução s' ∈ ND ;
26 |  |  s'' ← Perturbacao(s',nível) ;
27 |  |  s ← s'' ;
28 |  |  se nível == n/2 − 1 então
29 |  |  |  nível ← 1 ;
30 |  |  fim
31 |  |  senão
32 |  |  |  nível ← nível +1 ;
33 |  |  fim
34 |  fim
35 fim
36 retorna ND ;
```

4.7.6 NSGA-II

O algoritmo NSGA-II que foi implementado está descrito no Algoritmo 7 com as seguintes adaptações:

- Na linha 1 é determinada a solução inicial com base no método descrito na seção 4.2 combinado com um procedimento que gera soluções aleatoriamente,

uma vez que é necessário que a população inicial do NSGA-II tenha tamanho *pop*;

- Na linha 5 o critério de parada adotado está descrito na seção 4.9;
- A geração de novos indivíduos da linha 23 é realizada por mutação e cruzamento conforme descrito nas seções 4.6.2 e 4.6.1 respectivamente. A mutação acontece com uma probabilidade t_m e o cruzamento acontece com uma probabilidade t_c. Esses parâmetros foram definidos conforme descrito na seção 4.9;
- Cada indivíduo da população é uma solução do problema, sendo representado conforme descrito na seção 4.1.

4.7.7 SPEA2

O algoritmo SPEA2 que foi implementado está descrito no Algoritmo 8 com as seguintes adaptações:

- A solução inicial da linha 1 é obtida a partir do procedimento descrito na seção 4.2 combinado com um método que gera solução aleatoriamente, pois é necessário ter uma população inicial de tamanho *pop*;
- O critério de parada adotado pelo algoritmo na linha 5 é aquele descrito na seção 4.9;
- A etapa de reprodução da linha 22 é realizada por mutação e cruzamento conforme descrito nas seções 4.6.2 e 4.6.1 respectivamente. A mutação acontece com uma probabilidade t_m e o cruzamento acontece com uma probabilidade t_c, esses parâmetros foram definidos conforme descrito na seção 4.9;
- Cada indivíduo da população é uma solução para o problema, sendo representado conforme descrito na seção 4.1.

4.8 Normalização

Para utilizar as métricas de avalização de desempenho, os valores das funções objetivo foram normalizados segundo a Equação (4.1):

$$f_i^*(s) = \left(\frac{f_i(s) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \times 100 \right) + 1 \tag{4.1}$$

Nesta equação, $f_i^{\min}$ e $f_i^{\max}$ são, respectivamente, o menor e o maior valor encontrado para o i-ésimo objetivo. Utiliza-se o valor 1 na segunda parcela da Equação (4.1) pois o atraso total ponderado (objetivo f_2) poderia assumir valor nulo, o que impediria a utilização da métrica *epsilon*. Desta forma, os valores da função objetivo normalizada $f_i^*(s)$ calculada pela Equação (4.1) estão no intervalo $[1, 101]$.

4.9 Parâmetros

Nesta seção é descrito o procedimento utilizado para determinar os valores dos parâmetros dos algoritmos. Optou-se por experimentar alguns valores para os parâmetros a fim de escolher um que seja mais adequado. Esses testes consistiram em executar os algoritmos em 15 instâncias escolhidas aleatoriamente do conjunto de instâncias disponíveis, e escolher o valor do parâmetro que fornecesse o melhor resultado médio nas métricas testadas.

Os algoritmos MOVNS_Ottoni e MOVNS_Arroyo têm, em comum, o parâmetro n_r, que é necessário para a etapa de intensificação destes algoritmos. Neste caso, foram testados os valores $n_r \in \{2, 4, 6, 8\}$, os mesmos valores que foram testados pelos autores que propuseram esses algoritmos.

Os algoritmos NSGA-II e SPEA2 possuem três parâmetros: tamanho da população *pop*, taxa de cruzamento t_c e taxa de mutação t_m. Para o tamanho da população foram testados os valores $pop \in \{50, 100, 150, 200\}$. Para a taxa de cruzamento, os valores $t_c \in \{70\%, 75\%, 80\%, 85\%\}$, enquanto que para a taxa de mutação foram testados os valores $t_m \in \{0,2\%, 0,4\%, 0,6\%, 0,8\%\}$.

Na Tabela 4.1 são apresentados os parâmetros testados para os algoritmos e os melhores valores para os mesmos segundo as cinco métricas avaliadas:

Tabela 4.1: Valores dos melhores parâmetros

Métrica	n_r	*pop*	t_c (%)	t_m (%)
Distância Média	6	50	80	0,6
Cardinalidade	8	50	70	0,4
Métrica *epsilon*	6	50	75	0,6
Hipervolume	8	50	80	0,6
Distância Máxima	6	50	80	0,6

Na Tabela 4.1 estão os valores dos parâmetros que apresentaram o melhor resultado de acordo com cada métrica. O valor de cada parâmetro é calculado tendo em vista os resultados apresentados pelos algoritmos nos testes preliminares. O cálculo do parâmetro n_r é feito para os algoritmos MOVNS_Ottoni e MOVNS_Arroyo, enquanto que os parâmetros *pop*, t_c e t_m são calculados para os algoritmos NSGA-II e SPEA2.

Pelos resultados apresentados na Tabela 4.1, pode-se verificar que não houve um mesmo valor de parâmetro que produzisse os melhores resultados para todas as métricas avaliadas. Sendo assim, convencionou-se que o valor que obteve o melhor resultado no maior número de métricas seria o escolhido. Desta forma, a escolha recaiu nos seguintes valores de parâmetros: $n_r = 6$, $pop = 50$, $t_c = 80\%$ e $t_m = 0,6\%$.

Todos os algoritmos implementados têm um parâmetro em comum, que é o critério de parada. Para esse parâmetro adotou-se $1000 \cdot n$ milissegundos como limite de tempo para a execução de cada algoritmo, sendo n o número de tarefas da instância.

Capítulo 5

Resultados Computacionais

Todos os algoritmos foram codificados na linguagem C++ e os testes foram realizados em um microcomputador com as seguintes configurações: processador Intel(R) Core(TM) 2 Quad com 2,4 GHz de frequência de *clock*, 6 GB de memória RAM e com um sistema operacional Ubuntu de 64 bits instalado.

O critério de parada de cada algoritmo é o tempo de CPU, e está relacionado com o tamanho da instância. Na literatura é comum a utilização deste critério em algoritmos de otimização. Adotou-se $1000 \cdot n$ milissegundos como limite de tempo de execução de cada algoritmo, sendo n o número de tarefas da instância. Para cada instância foram realizadas 30 execuções, com diferentes sementes geradas aleatoriamente. Assim, cada algoritmo demorou 96 horas de CPU para ser executado, e ao todo foram necessárias 672 horas de CPU para executar os sete algoritmos.

5.1 Instâncias

Para testar os algoritmos foram geradas instâncias de forma aleatória e com distribuição uniforme. Assim como em Jin et al. (2010), o número de tarefas é um número inteiro $n \in \{60, 80, 100\}$, o número de famílias $f \in \{2, 3, 4, 5\}$ e o tempo de processamento é um número inteiro no intervalo $[1, 99]$. Já a data de entrega das tarefas foi gerada tal como em Baker e Magazine (2000), sendo definida no intervalo $(0, h\sum p_j)$, com $h \in \{0,5; 1,5; 2,5; 3,5\}$. Finalmente, o tempo de preparação entre as famílias de tarefas são números inteiros cujos valores pertencem a três classes de intervalos:

- Classe S: $[10, 20]$
- Classe M: $[51, 100]$
- Classe L: $[101, 200]$

A formação de tais intervalos de tempos de preparação é uma sugestão proposta em Hariri e Potts (1997). Os tempos de preparação da classe S são relativamente menores do que o tempo de processamento médio. Os tempos de preparação da classe M são relativamente próximos ao tempo de processamento médio, enquanto o tempo de preparação da classe L são relativamente maiores do que o tempo de processamento médio.

Combinando-se o número de tarefas, número de famílias, número de intervalos para as datas de entrega e o número de classes de intervalos para os tempos de preparação, um total de 144 instâncias diferentes foram geradas para a realização dos testes. Note que, para cada n, foram geradas 48 instâncias.

5.2 Experimentos Computacionais

Apresentam-se nas tabelas a seguir os resultados obtidos pelos algoritmos desenvolvidos considerando a aplicação de diferentes métricas. Para cada métrica são apresentados o resultado médio e a média dos melhores resultados. O valor do resultado médio de cada instância é obtido pela média aritmética entre as 30 execuções. Analogamente, o valor para o melhor resultado é obtido considerando apenas o melhor resultado para cada instância, em relação a cada métrica. Listam-se ainda os valores médios da métrica em análise considerando todas as 144 instâncias e o desvio padrão de cada algoritmo.

Na Tabela 5.1 estão os resultados obtidos pelos algoritmos implementados (MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS, PILS1, NSGA-II e SPEA2) com relação ao número de soluções geradas por cada um deles. Na primeira coluna indica-se o conjunto de instâncias agrupadas de acordo com o número de tarefas n. Na segunda coluna é apresentada a soma do número de soluções distintas que compõem o conjunto de referência R de todas as instâncias de mesmo número de tarefas em 30 execuções de cada algoritmo. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias de mesmo número de tarefas, o número total de soluções não-dominadas geradas pelo respectivo algoritmo em 30 execuções. Na última linha totalizam-se os resultados de cada algoritmo.

Tabela 5.1: Número de soluções não-dominadas

n	R	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
60	683	106	199	228	114	549	97	34
80	635	21	68	90	44	400	116	29
100	536	14	63	19	32	220	186	31
Total	1854	141	330	337	190	1169	399	94

Considerando que, ao todo, 1169 soluções do conjunto de referência foram geradas pelo PILS1, de um total de 1854, pode-se afirmar que esse algoritmo conseguiu encontrar o maior número de soluções pertencentes ao conjunto de referência. O segundo melhor resultado foi do algoritmo NSGA-II que encontrou 399, e o pior foi do SPEA2, que conseguiu encontrar apenas 94 soluções do conjunto de referência. O algoritmo NSGA-II obteve o seu melhor resultado para as instâncias maiores, com 100 tarefas. Por outro lado, os algoritmos de busca local (MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS e PILS1) apresentaram seus melhores resultado para as instâncias menores, com 60 tarefas.

Na Tabela 5.2 apresentam-se os resultados obtidos dos algoritmos implementados com relação à métrica de cardinalidade $C1_R(A)$. Na primeira coluna indica-se o

tipo de resultado: médio e melhor. Na segunda coluna, o conjunto de instâncias agrupadas de acordo com o número de tarefas n. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias, o resultado para a métrica de cardinalidade $C1_R(A)$, calculada de acordo com a Equação (3.6). Abaixo do conjunto de resultados estão as médias para cada algoritmo. Na última linha estão registrados os valores para o desvio padrão de cada algoritmo.

Observa-se que cada algoritmo foi executado 30 vezes para uma determinada instância do problema e o conjunto de soluções não-dominadas obtidas nessas execuções é considerado no cálculo da medida de desempenho do algoritmo.

Tabela 5.2: Comparação dos algoritmos com relação à métrica de Cardinalidade ($C1_R$)

	n	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
Resultado Médio	60	2,13	3,56	4,00	2,68	32,15	2,18	0,29
	80	0,19	0,73	0,72	0,29	10,57	1,22	0,16
	100	0,05	0,28	0,10	0,09	2,89	2,32	0,16
	Média	0,79	1,52	1,60	1,02	15,20	1,91	0,20
Melhores Resultados	60	12,42	18,44	22,66	9,53	58,47	15,65	5,32
	80	2,45	7,19	10,48	3,78	43,68	14,71	3,72
	100	1,41	6,30	2,42	1,85	32,64	29,85	4,60
	Média	5,43	10,64	11,85	5,05	44,93	20,07	4,55
Desvio Padrão		4,41	6,40	6,43	4,74	25,76	8,21	2,48

Considerando o resultado médio observado na Tabela 5.2, o algoritmo PILS1 é capaz de gerar um número muito superior de soluções não-dominadas em relação aos demais algoritmos. Além disso, o número de soluções de referência geradas por esse algoritmo é, pelo menos, sete vezes maior que aquele gerado por qualquer outro algoritmo. O algoritmo PILS1 obteve seu melhor desempenho nas instâncias menores, de tamanho 60, e o seu pior resultado foi para as instâncias maiores, com 100 tarefas. Esse diferença contribui para o valor elevado do desvio padrão, que é o maior entre todos os algoritmos testados. Quanto menor o valor do desvio padrão, mais robusto é considerado o algoritmo. O SPEA2 apresentou o menor desvio padrão, entretanto, obteve o pior desempenho em todas as comparações, ou seja, encontra resultados parecidos, geralmente ruins.

Em relação aos melhores resultados, o algoritmo PILS1, também conseguiu o melhor desempenho. Entretanto, a diferença para o NSGA-II foi menor.

Na Tabela 5.3 estão os resultado obtidos para a métrica distância média $D1_R$, calculados de acordo com a Equação (3.7). Existem dois grupos: os resultados médios e os melhores resultados. Na última linha da tabela, estão os valores do desvio padrão para cada algoritmo.

Analisando o resultado médio da métrica distância média $D1_R$, apresentado pela Tabela 5.3, é possível afirmar que o algoritmo PILS1 produziu o conjunto de soluções não-dominadas que estão mais próximas do conjunto de referência. Além disso, o algoritmo MOVNS_Arroyo obteve o segundo melhor desempenho, enquanto que o SPEA2 ficou com o pior resultado.

Considerando apenas os melhores resultados, o PILS1 conseguiu o melhor valor

Tabela 5.3: Comparação dos algoritmos com relação à métrica Distância Média ($D1_R$)

	n	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
Resultado Médio	60	3,54	2,92	2,74	4,05	0,90	3,18	4,61
	80	5,52	4,56	4,47	6,21	2,04	5,16	7,81
	100	7,92	6,43	6,41	8,19	4,32	8,34	11,86
	Média	5,66	4,63	4,54	6,15	2,42	5,56	8,09
Melhores Resultados	60	1,90	1,63	1,48	2,67	0,26	0,69	1,30
	80	2,81	2,33	2,26	4,05	0,94	1,02	2,13
	100	4,66	3,68	3,57	5,67	2,16	1,66	3,68
	Média	3,12	2,55	2,43	4,13	1,12	1,13	2,37
Desvio Padrão		4,59	3,79	3,84	4,60	2,34	6,60	7,51

a métrica $D1_R$, seguido pelo NSGA-II. O pior resultado foi obtido pelo algoritmo PILS.

O PILS1 obteve o menor desvio padrão, enquanto que o SPEA2 ficou o maior valor. Observou-se, para essa métrica, que todos os algoritmos conseguiram melhor desempenho nas instâncias menores, com 60 tarefas, e pior desempenho nas instâncias maiores, de 100 tarefas.

A Tabela 5.4 apresenta os resultados obtidos pelos algoritmos implementados com relação à métrica de distância máxima $D_{\max}$, calculada de acordo com a Equação (3.8). Nessa tabela apresenta-se o resultado médio e os melhores resultados encontrados. Na segunda coluna indica-se o conjunto de instâncias agrupadas de acordo com o número de tarefas n. Nas demais colunas, mostra-se, para cada algoritmo e para cada grupo de instâncias, o resultado para a métrica $D_{\max}$, que foi calculada de acordo com a Equação (3.8). Na última linha são apresentados os valores do desvio padrão para cada algoritmo.

Tabela 5.4: Comparação dos algoritmos com relação à métrica Distância Máxima ($D_{\max}$)

	n	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
Resultado Médio	60	7,74	7,19	6,98	8,48	3,88	5,90	7,42
	80	10,75	9,93	9,80	11,54	7,20	8,67	11,23
	100	13,50	12,39	12,34	13,97	10,70	12,45	15,84
	Média	10,66	9,84	9,71	11,33	7,26	9,01	11,50
Melhores Resultados	60	5,61	5,52	5,21	6,95	1,13	1,70	2,87
	80	7,91	7,47	7,40	9,42	4,15	2,29	4,77
	100	10,23	9,55	9,32	11,68	7,80	3,27	7,18
	Média	7,92	7,51	7,31	9,35	4,36	2,42	4,94
Desvio Padrão		6,47	6,32	6,34	6,74	6,32	7,57	8,23

Pelo resultado médio da Tabela 5.4, verifica-se que o algoritmo PILS1 apresentou o melhor resultado na métrica distância máxima $D_{\max}$, enquanto que o pior resultado foi o do algoritmo SPEA2.

Para os melhores resultados, nota-se que os algoritmos evolutivos obtiveram um bom desempenho nessa métrica, sendo que o primeiro e o terceiro melhor resultado foram dos algoritmos NSGA-II e SPEA2, respectivamente. O PILS1 apresentou o segundo menor valor para a métrica $D_{\max}$.

O menor desvio padrão para essa métrica foi apresentado pelos algoritmos PILS1 e MOVNS_Ottoni, enquanto que o maior desvio padrão foi o do algoritmo SPEA2.

Apresenta-se, na Tabela 5.5, os resultados obtidos pelos algoritmos implementados com relação à métrica diferença de hipervolume $H^-(.)$. Na primeira coluna estão os tipos de resultados: resultado médio e melhores resultado. Na segunda coluna dessa tabela, indica-se o conjunto de instâncias agrupadas de acordo com o número de tarefas n. Nas demais colunas, para cada algoritmo, estão os valores da diferença de hipervolume H^- obtidos sobre 48 instâncias. Na última linha estão os valores do desvio padrão de cada algoritmo para a métrica $H^-(.)$.

Tabela 5.5: Comparação dos algoritmos com relação à métrica Diferença de Hipervolume (H^-)

	n	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
Resultado Médio	60	1.050,74	908,04	856,52	1.190,03	317,95	1.016,82	1.420,06
	80	1.491,81	1.271,56	1.253,95	1.656,59	628,39	1.434,97	2.078,97
	100	1.917,98	1.576,87	1.575,02	1.980,08	1.086,05	1.982,22	2.762,23
	Média	1.486,84	1.252,16	1.228,50	1.608,90	677,46	1.478,00	2.087,08
Melhores Resultados	60	591,22	518,78	474,68	829,40	78,22	149,41	385,92
	80	840,09	681,96	658,10	1.134,62	251,70	202,60	582,71
	100	1.149,47	865,35	875,68	1385,38	477,43	325,18	903,10
	Média	860,26	688,70	669,49	1.116,47	269,12	225,73	623,91
Desvio Padrão		1.123,57	990,37	992,91	1.192,67	751,04	1.627,51	1.787,41

Pelos valores dos resultados médios da Tabela 5.5, verifica-se que o algoritmo PILS1 produz uma melhor cobertura para o conjunto de referência R, comparado com os demais algoritmos. Isso significa que a área formada entre as pontos do conjunto R e os ponto do conjunto não-dominado do algoritmo PILS1 apresenta os menores valores. Os algoritmos MOVNS_Arroyo e MOVNS_Ottoni apresentaram o segundo e o terceiro melhor resultado respectivamente.

Considerando apenas os melhores resultados de cada instância, o NSGA-II conseguiu o menor valor para H^- entre todos os algoritmos. O segundo melhor desempenho foi do PILS1 e o maior valor para a métrica H^- foi do PILS.

O menor valor para o desvio padrão foi do algoritmo PILS1, enquanto que o maior foi o do SPEA2.

A Tabela 5.6 mostra os resultados obtidos pelos algoritmos com relação à métrica *epsilon* $I^1_\epsilon(.)$. Na primeira coluna dessa tabela, são exibidos os tipos de resultados. A segunda coluna apresenta o conjunto de instâncias agrupadas de acordo com o número de tarefas n e nas demais colunas, estão, para cada algoritmo e para cada grupo de instâncias, o valores para essa métrica.

De acordo com os resultados médios da Tabela 5.6 verifica-se que o algoritmo PILS1 é o que produz os menores valores para a métrica *epsilon*, indicando que as soluções não-dominadas geradas por este algoritmo estão mais próximas do conjunto de

Tabela 5.6: Comparação dos algoritmos com relação à métrica *Epsilon*($I_\epsilon^1(.)$)

	n	Algoritmo						
		MOVNS	**MOVNS_Ottoni**	**MOVNS_Arroyo**	**PILS**	**PILS1**	**NSGA-II**	**SPEA2**
Resultado Médio	60	1,43	1,37	1,35	1,48	1,13	1,40	1,58
	80	1,23	1,19	1,19	1,25	1,10	1,38	1,63
	100	1,18	1,15	1,15	1,18	1,10	1,63	2,01
	Média	1,28	1,24	1,23	1,31	1,11	1,47	1,74
Melhores Resultados	60	1,24	1,21	1,19	1,33	1,04	1,11	1,18
	80	1,13	1,10	1,10	1,17	1,05	1,10	1,19
	100	1,11	1,09	1,09	1,13	1,05	1,16	1,33
	Média	1,16	1,13	1,13	1,21	1,05	1,13	1,24
Desvio Padrão		0,27	0,22	0,21	0,28	0,11	0,85	1,37

referência R. Além disso, os algoritmos de busca local (MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS e PILS1) obtiveram melhor resultado se comparados com os algoritmos evolutivos (NSGA-II e SPEA2).

Em relação ao melhores resultados, o PILS1 também obteve o menor valor para a métrica I_ϵ^1. Três algoritmos ficaram empatados com o segundo melhor resultado: MOVNS_Ottoni, MOVNS_Arroyo e NSGA-II.

O desvio padrão do algoritmo PILS1 é o menor entre todos os algoritmos. Os algoritmos de busca local conseguiram o seu melhor desempenho nessa métrica nas instâncias maiores com 100 tarefas.

Na Tabela 5.7 apresenta-se um resumo dos resultados de todos algoritmos para todas as métricas. Na primeira coluna são listados os algoritmos e nas demais colunas, mostra-se apenas a média total para cada métrica. O melhor resultado de cada métrica está destacado em negrito na tabela.

Tabela 5.7: Resumo da comparação dos algoritmos por métrica em relação aos resultados médios

Algoritmo	Métrica				
	$C1_R$	$D1_R$	$D_{\max}$	H^-	I_ϵ^1
MOVNS	0,79	5,66	10,66	1.486,84	1,28
MOVNS_Ottoni	1,52	4,63	9,84	1.252,16	1,24
MOVNS_Arroyo	1,60	4,54	9,71	1.228,50	1,23
PILS	1,02	6,15	11,33	1.608,90	1,31
PILS1	**15,20**	**2,42**	**7,26**	**677,46**	**1,11**
NSGAII	1,91	5,56	9,01	1.478,00	1,47
SPEA2	0,20	8,09	11,50	2.087,08	1,74

De acordo com a Tabela 5.7, pode-se afirmar que o algoritmo PILS1 conseguiu o melhor resultado médio entre todos os outros algoritmos em relação a todas as métricas avaliadas.

A Tabela 5.8 mostra uma comparação dos algoritmos em relação a cada uma das métricas tendo em vista a média dos melhores resultados encontrados por cada algo-

ritmo em cada métrica e em 30 execuções de cada instância. Na primeira coluna são listados os algoritmos e nas demais colunas, mostra-se o resultado de cada algoritmo em relação à cada métrica. Destacou-se na tabela o melhor resultado apresentado em cada métrica.

Tabela 5.8: Resumo da comparação dos algoritmos por métrica em relação aos melhores resultados encontrados

Algoritmo	Métrica				
	$C1_R$	$D1_R$	$D_{\max}$	H^-	I_ϵ^1
MOVNS	5,43	3,12	7,92	860,26	1,16
MOVNS_Ottoni	10,64	2,55	7,51	688,70	1,13
MOVNS_Arroyo	11,85	2,43	7,31	669,49	1,13
PILS	5,05	4,13	9,35	1.116,47	1,21
PILS1	**44,93**	**1,12**	4,36	269,12	**1,05**
NSGAII	20,07	1,13	**2,42**	**225,73**	1,13
SPEA2	4,55	2,37	4,94	623,9	1,24

De acordo com a Tabela 5.8, o algoritmo PILS1 tem melhor desempenho com relação aos melhores valores nas métricas cardinalidade, distância média e *epsilon*, enquanto que o NSGA-II foi melhor nas métricas distância máxima e diferença de hipervolume.

5.3 Análise Estatística

Os experimentos a seguir têm por objetivo verificar se existe diferença estatisticamente significativa entre os resultados médios dos algoritmos implementados. Para a realização desta tarefa utilizou-se o teste não-paramétrico Kruskal-Wallis (Field, 2009). Os testes estatísticos foram executados com o auxílio da ferramenta R (R Development Core Team, 2011).

O teste não-paramétrico Kruskal-Wallis é realizado para cada par de algoritmo. Esse teste determina se as diferenças amostrais observadas sugerem que as amostras são de populações diferentes ou se são apenas variações casuais que podem ser esperadas entre amostras aleatórias da mesma população.

Na Tabela 5.9 mostra-se o resultado do teste Kruskal-Wallis de comparação múltipla para a métrica de cardinalidade com a finalidade de verificar se existe diferença significativa entre os resultados dos algoritmos. Nesta tabela, a primeira coluna indica o nome algoritmo e nas demais estão os valores do teste em relação a cada par de algoritmo. Destacou-se nessa tabela os resultados que apresentaram diferença superior a $104,24$, que foi a diferença crítica para esse teste. As células destacadas indicam os pares de algoritmos que o resultado apresenta diferença estatisticamente significativa entre si.

A partir dos resultados da Tabela 5.9, pode-se afirmar que existem evidências estatísticas de que o resultado da métrica de cardinalidade é significativamente diferente entre os pares de algoritmos: MOVNS × PILS1, MOVNS × NSGA-II,

Tabela 5.9: Teste Kruskal-Wallis: cardinalidade ($C1_R$)

	MOVNS	MOVNS_Ottoni	MOVNS_Arroyo	PILS	PILS1	NSGAII	SPEA2
MOVNS	-	99,50	96,88	16,39	**448,40**	**193,06**	62,72
MOVNS_Ottoni	99,50	-	2,63	**115,90**	**348,90**	93,55	**162,23**
MOVNS_Arroyo	96,88	2,63	-	**113,27**	**351,53**	96,18	**159,60**
PILS	16,39	**115,90**	**113,27**	-	**464,80**	**209,45**	46,33
PILS1	**448,40**	**348,90**	**351,53**	**464,80**	-	**255,35**	**511,13**
NSGAII	**193,06**	93,55	96,18	**209,45**	**255,35**	-	**255,78**
SPEA2	62,72	**162,23**	**159,60**	46,33	**511,13**	**255,78**	-

MOVNS_Ottoni × PILS, MOVNS_Ottoni × PILS1, MOVNS_Ottoni × SPEA2, MOVNS_Arroyo × PILS, MOVNS_Arroyo × PILS1, MOVNS_Arroyo × SPEA2, PILS × PILS1, PILS × NSGA-II, PILS1 × NSGA-II, PILS1 × SPEA2 e NSGA-II × SPEA2.

Assim, de acordo com a Tabela 5.9, há evidências estatísticas de que os valores médios relativos à métrica de cardinalidade ($C1_R$) do algoritmo PILS1 são significativamente diferentes de todos os outros algoritmos.

O resultado do teste Kruskal-Wallis de comparação múltipla para a métrica de distância média é apresentado pela Tabela 5.10. Destacou-se em negrito nessa tabela os resultados com diferença superior a $104,24$, que foi a diferença crítica para esse teste.

Tabela 5.10: Teste Kruskal-Wallis: distância média ($D1_R$)

	MOVNS	MOVNS_Ottoni	MOVNS_Arroyo	PILS	PILS1	NSGAII	SPEA2
MOVNS	-	59,95	69,03	31,13	**253,70**	31,31	**113,13**
MOVNS_Ottoni	59,95	-	9,08	91,08	**193,75**	28,64	**173,08**
MOVNS_Arroyo	69,03	9,08	-	100,17	**184,67**	37,72	**182,16**
PILS	31,13	91,08	100,17	-	**284,83**	62,44	81,99
PILS1	**253,70**	**193,75**	**184,67**	**284,83**	-	**222,39**	**366,83**
NSGAII	31,31	28,64	37,72	62,44	**222,39**	-	**144,44**
SPEA2	**113,13**	**173,08**	**182,16**	81,99	**366,83**	**144,44**	-

A partir do resultado apresentado na Tabela 5.10, é possível afirmar que existem evidências estatísticas de que o resultado para a métrica distância média é significativamente diferente entre os pares de algoritmos: MOVNS × PILS1, MOVNS × SPEA2, MOVNS_Ottoni × PILS1, MOVNS_Ottoni × SPEA2, MOVNS_Arroyo × PILS1, MOVNS_Arroyo × SPEA2, PILS × PILS1, PILS1 × NSGA-II, PILS1 × SPEA2 e NSGA-II × SPEA2.

Os resultados da Tabela 5.10 mostram, portanto, que o algoritmo PILS1 produz valores médios da métrica distância média ($D1_R$) significativamente diferentes de todos os demais algoritmos. Esse fato confirma a superioridade do algoritmo PILS1 nessa métrica.

Apresenta-se na Tabela 5.11 o resultado do teste Kruskal-Wallis de comparação múltipla para a métrica de distância máxima com a finalidade de verificar se existe diferença significativa entre os resultados dos algoritmos.

De acordo com os resultados da Tabela 5.11, é possível afirmar que existem

Tabela 5.11: Teste Kruskal-Wallis: distância máxima ($D_{\max}$)

	MOVNS	MOVNS_Ottoni	MOVNS_Arroyo	PILS	PILS1	NSGAII	SPEA2
MOVNS	-	42,22	49,19	25,96	**175,86**	88,71	24,78
MOVNS_Ottoni	42,22	-	6,97	68,17	**133,65**	46,49	67,00
MOVNS_Arroyo	49,19	6,97	-	75,15	**126,67**	39,52	73,97
PILS	25,96	68,17	75,15	-	**201,82**	**114,67**	1,17
PILS1	**175,86**	**133,65**	**126,67**	**201,82**	-	87,15	**200,65**
NSGAII	88,71	46,49	39,52	**114,67**	87,15	-	**113,49**
SPEA2	24,78	67,00	73,97	1,17	**200,65**	**113,49**	-

evidências estatísticas de que o resultado para a métrica de distância máxima é diferente entre os pares de algoritmos: MOVNS × PILS1, MOVNS_Ottoni × PILS1, MOVNS_Arroyo × PILS1, PILS × PILS1, PILS × NSGA-II e PILS1 × SPEA2.

Considerando que para essa métrica o algoritmo PILS1 apresentou resultado médio maior que todos os outros algoritmos, o teste de Kruskal-Wallis evidencia que essa diferença é estatisticamente significativa em relação aos algoritmos MOVNS, MOVNS_Ottoni, MOVNS_Arroyo, PILS, PILS1 e SPEA2. Entretanto, apesar de existir diferença entre os resultados médios dos algoritmos PILS1 e NSGA-II, não é possível afirmar que essa diferença é estatisticamente significativa nessa métrica.

Na Tabela 5.12 estão os resultados do teste Kruskal-Wallis de comparação múltipla para a métrica diferença de hipervolume. Este teste tem a finalidade de verificar a existência de diferença significativa entre os resultados produzidos entre cada par de algoritmos.

Tabela 5.12: Teste Kruskal-Wallis: diferença de hipervolume (H^-)

	MOVNS	MOVNS_Ottoni	MOVNS_Arroyo	PILS	PILS1	NSGAII	SPEA2
MOVNS	-	62,28	69,67	23,29	**237,27**	35,59	101,38
MOVNS_Ottoni	62,28	-	7,40	85,57	**174,99**	26,69	**163,65**
MOVNS_Arroyo	69,67	7,40	-	92,97	**167,60**	34,08	**171,05**
PILS	23,29	85,57	92,97	-	**260,56**	58,88	78,08
PILS1	**237,27**	**174,99**	**167,60**	**260,56**	-	**201,68**	**338,65**
NSGAII	35,59	26,69	34,08	58,88	**201,68**	-	**136,97**
SPEA2	101,38	**163,65**	**171,05**	78,08	**338,65**	**136,97**	-

Analisando a Tabela 5.12, pode-se afirmar que há evidências estatísticas de que os valores médios relativos à métrica diferença de hipervolume são estatisticamente significativos entre os pares de algoritmos: MOVNS × PILS1, MOVNS_Ottoni × PILS1, MOVNS_Ottoni × SPEA2, MOVNS_Arroyo × PILS1, MOVNS_Arroyo × SPEA2, PILS × PILS1, PILS1 × NSGA-II, PILS1 × SPEA2 e NSGA-II × SPEA2.

Os resultados da Tabela 5.12 evidenciam que o algoritmo PILS1 é superior estatisticamente aos demais algoritmos em relação à métrica diferença de hipervolume (H^-).

O teste Kruskal-Wallis de comparação múltipla também foi aplicado para a métrica *epsilon* com a finalidade de verificar se existe diferença significativa entre os resultados dos algoritmos nessa métrica. Na Tabela 5.13 são apresentados os resultados encontrados considerando os valores médios de cada algoritmo nessa métrica.

Tabela 5.13: Teste Kruskal-Wallis: métrica *epsilon* ($I_\epsilon^1(.)$)

	MOVNS	MOVNS_Ottoni	MOVNS_Arroyo	PILS	PILS1	NSGAII	SPEA2
MOVNS	-	47,10	55,03	19,83	**226,90**	33,24	**137,95**
MOVNS_Ottoni	47,10	-	7,93	66,93	**179,80**	80,34	**185,05**
MOVNS_Arroyo	55,03	7,93	-	74,86	**171,86**	88,27	**192,98**
PILS	19,83	66,93	74,86	-	**246,72**	13,42	**118,13**
PILS1	**226,90**	**179,80**	**171,86**	**246,72**	-	**260,14**	**364,85**
NSGAII	33,24	80,34	88,27	13,42	**260,14**	-	**104,71**
SPEA2	**137,95**	**185,05**	**192,98**	**118,13**	**364,85**	**104,71**	-

Conforme pode ser observado na Tabela 5.13, há evidências estatísticas de que os valores médios da métrica *epsilon* são diferentes entre os seguintes pares de algoritmos: MOVNS × PILS1, MOVNS × SPEA2, MOVNS_Ottoni × PILS1, MOVNS_Ottoni × SPEA2, MOVNS_Arroyo × PILS1, MOVNS_Arroyo × SPEA2, PILS × PILS1, PILS × SPEA2, PILS1 × NSGA-II, PILS1 × SPEA2 e NSGA-II × SPEA2.

Assim, de acordo com essa tabela, pode-se afirmar que os valores médios da métrica *epsilon* ($I_\epsilon^1(.)$) produzidos pelo algoritmo PILS1 são significativamente diferentes de todos os outros, confirmando a superioridade do algoritmo PILS1 frente aos demais. Por outro lado, pode-se afirmar que o algoritmo SPEA2 é estatisticamente inferior aos demais, tendo em vista que seus resultados médios são piores do que de todos os outros nessa métrica.

A Tabela 5.14 mostra um resumo do teste Kruskal-Wallis com todas as métricas e todos os pares de algoritmos. Na primeira coluna, são listados os pares de algoritmos e as demais colunas assumem valor `V` caso exista diferença estatística entre os algoritmos comparados, e valor `F`, caso contrário.

Em vista dos resultados médios apresentados pelos algoritmos e dos resultados da Tabela 5.14, pode-se concluir que o algoritmo PILS1 é estatisticamente superior a todos os outros algoritmos em todas as métricas. A exceção ocorre apenas na métrica distância máxima, situação na qual não é possível definir se a diferença existente entre o PILS1 e o NSGA-II é estatisticamente significativa.

Tabela 5.14: Teste Kruskal-Wallis de comparação múltipla

Par de Algoritmos	$C1_R$	$D1_R$	$D_{\max}$	H^-	I_ϵ^1
MOVNS × MOVNS_Ottoni	F	F	F	F	F
MOVNS × MOVNS_Arroyo	F	F	F	F	F
MOVNS × PILS	F	F	F	F	F
MOVNS × PILS1	V	V	V	V	V
MOVNS × NSGA-II	V	F	F	F	F
MOVNS × SPEA2	F	V	F	F	V
MOVNS_Ottoni × MOVNS_Arroyo	F	F	F	F	F
MOVNS_Ottoni × PILS	V	F	F	F	F
MOVNS_Ottoni × PILS1	V	V	V	V	V
MOVNS_Ottoni × NSGA-II	F	F	F	F	F
MOVNS_Ottoni × SPEA2	V	V	F	V	V
MOVNS_Arroyo × PILS	V	F	F	F	F
MOVNS_Arroyo × PILS1	V	V	V	V	V
MOVNS_Arroyo × NSGA-II	F	F	F	F	F
MOVNS_Arroyo × SPEA2	V	V	F	V	V
PILS × PILS1	V	V	V	V	V
PILS × NSGA-II	V	F	V	F	F
PILS × SPEA2	F	F	F	F	V
PILS1 × NSGA-II	V	V	F	V	V
PILS1 × SPEA2	V	V	V	V	V
NSGA-II × SPEA2	V	V	V	V	V

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho tratou o problema de sequenciamento em uma máquina em que o tempo de preparação das tarefas depende da sequência e da família, tendo-se dois critérios de otimização a serem atendidos: a minimização do *makespan* e a minimização do atraso total ponderado.

Para resolvê-lo, foram adaptados sete algoritmos multiobjetivos: *Multi-objective Variable Neighborhood Search* (MOVNS), MOVNS_Ottoni, MOVNS_Arroyo, *Pareto Iterated Local Search* (PILS), PILS1, *Nondominated Sorting Genetic Algorithm* II (NSGA-II) e *Strength Pareto Evolutionary Algorithm* 2 (SPEA2). Os cinco primeiros são baseados em busca local, enquanto os dois últimos são baseados em busca populacional. Os algoritmos MOVNS_Ottoni e MOVNS_Arroyo são variantes do algoritmo MOVNS encontradas na literatura. O PILS1 é uma variante do algoritmo PILS proposta neste trabalho, diferindo deste com relação à forma como são feitas as perturbações.

Os algoritmos foram comparados entre si em relação a cinco métricas: cardinalidade, distância média, distância máxima, diferença de hipervolume e *epsilon*.

Os resultados computacionais realizados em instâncias-teste geradas para o problema, validados por análises estatísticas, mostraram que a variante PILS1 é superior, em termos de resultado médio, aos demais algoritmos em relação às métricas de cardinalidade, distância média, hipervolume e *epsilon*. Em relação à métrica distância máxima, o PILS1 foi melhor a todos os outros algoritmos; no entanto, de acordo com teste Kruskal-Wallis, apenas não foi possível afirmar que a diferença existente entre o PILS1 e o NSGA-II foi estatisticamente significativa.

Em termos de melhores resultados, mostrou-se que o algoritmo PILS1 teve melhor desempenho nas métricas cardinalidade, distância média e *epsilon*, enquanto que o NSGA-II foi melhor nas métricas distância máxima e diferença de hipervolume.

O bom desempenho do PILS1 evidencia a contribuição do procedimento de perturbação proposto neste trabalho. Além disso, esse trabalho permitiu a comparação de algoritmos pertencentes a duas classes de problemas, de um lado os algoritmos baseados em busca local e, do outro, os de busca populacional. Acrescenta-se, ainda, como vantagem adicional do PILS1 é que não há necessidade de parâmetros para serem calibrados.

Como trabalhos futuros, são feitas as seguintes sugestões:

- Criar novas estruturas de vizinhanças, como por exemplo, troca em bloco de

tarefas que são da mesma família;

- Utilizar outras heurísticas construtivas, como por exemplo, a heurísticas NEH;
- Aplicar o algoritmo PILS1 em outros problemas de otimização multiobjetivo;
- Comparar os resultados com outros algoritmos multiobjetivos;
- Utilizar outros operadores para os algoritmos evolutivos;
- Aplicar um algoritmo capaz de gerar a fronteira Pareto com a finalidade de avaliar melhor a qualidade dos resultados desses algoritmos para o problema em estudo.

Apêndice A

Publicações

A seguir são listados os trabalhos oriundos desta dissertação que foram apresentados e publicados em anais de eventos:

1. **Título**: Multi-objective algorithms for the single machine scheduling problems with setup time dependent on the sequence and the job family
   Autores: Marcelo Ferreira Rego, Marcone Jamilson Freitas Souza e José Elias Claudio Arroyo
   Evento: XXXVIII Conferencia Latinoamericana En Informatica (CLEI 2012)
   Data: 01 a 05 de Outubro de 2012
   Local: Medellin, Colômbia
   Digital Object Identifier: 10.1109/CLEI.2012.6427172

2. **Título**: Multi-objective Algorithms for the Single Machine Scheduling Problem with Sequence-dependent Family Setups
   Autores: Marcelo Ferreira Rego, Marcone Jamilson Freitas Souza e José Elias Claudio Arroyo
   Evento: XXXI International Conference of the Chilean Computer Science Society (2012)
   Data: 14 a 16 de Novembro de 2012
   Local: Valparaíso, Chile

3. **Título**: Multi-objective Algorithms for the Single Machine Scheduling Problem with Sequence-dependent Family Setups
   Autores: Marcelo Ferreira Rego, Marcone Jamilson Freitas Souza, Igor Machado e José Elias Claudio Arroyo
   Evento: 17th Online World Conference on Soft Computing in Industrial Applications (2012)
   Data: 03 a 14 de Dezembro de 2012
   Local: Anywhere on Earth
   Link: http://dap.vsb.cz/wsc17conf/Media/Default/Page/online_wsc17_submission_37.pdf

Apêndice B

Resultados Completos

Os resultados completos estão disponíveis em:
http://www.decom.ufop.br/prof/marcone/projects/smsp-sdfst/

Referências Bibliográficas

Arroyo, J. E. C. *Heurística e Metaheurística para Otimização Combinatória Multi-objetivo*. Tese de doutorado, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, (2002).

Arroyo, J. E. C.; Ottoni, R. S. e Oliveira, A. P. (2011). Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. *Electronic Notes in Theoretical Computer Science*, v. 281, p. 5–19.

Baker, K. R. e Magazine, M. J. (2000). Minimizing maximum lateness with job families. *European Journal of Operational Research*, v. 127, n. 1, p. 126–139.

Brucker, P. (2007). *Scheduling algorithms*. Springer Verlag.

Bustamante, L. M. (2007). Minimização do custo de antecipação e atraso para o problema de sequenciamento de uma máquina com tempo de preparação dependente da sequência: aplicação em uma usina siderúrgica. Dissertação de mestrado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Minas Gerais, Belo Horizonte.

Chantaravarapan, S. *Heuristics for the family scheduling problems to minimize total tardiness*. PhD thesis, Texas Tech University, (2002).

Cheng, R.; Gen, M. e Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: hybrid genetic search strategies. *Computers & Industrial Engineering*, v. 36, n. 2, p. 343–364.

Choobineh, F. F.; Mohebbi, E. e Khoo, H. (2006). A multi-objective tabu search for a single-machine scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, v. 175, n. 1, p. 318–337.

Coello, C. A. C. e Lamont, G. B. (2004). *Applications of multi-objective evolutionary algorithms*, volume 1. World Scientific Publishing Company Incorporated.

Coello, C.A.C.; Lamont, G.B. e Van Veldhuizen, D.A. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

Czyżżak, P. e Jaszkiewicz, A. (1998). Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, v. 7, n. 1, p. 34–47.

Dahal, K.; Tan, K.C. e Cowling, P.I. (2007). *Evolutionary Scheduling*. Studies in Computational Intelligence. Springer. ISBN 9783540485827.

Deb, K. e Jain, S. (2002). Running performance metrics for evolutionary multi-objective optimization. Relatório técnico. DOI: 10.1.1.9.159.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 2, p. 182–197.

Falkenauer, E. e Bouffouix, S. (1991). A genetic algorithm for job shop. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, p. 824–829. IEEE, (1991).

Field, A. (2009). Descobrindo a estatística usando o spss. *Porto Alegre: Artmed*.

Fonseca, C. M.; Knowles, J. D.; Thiele, L. e Zitzler, E. (2005). A tutorial on the performance assessment of stochastic multiobjective optimizers. *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO)*, volume 216, (2005).

Geiger, M. J. (2004). Randomised variable neighbourhood search for multi objective optimisation. *4th EU/ME: Design and Evaluation of Advanced Hybrid Meta-Heuristics*, p. 34–42, (2004).

Geiger, M. J. (2006). The PILS metaheuristic and its application to multi-objective machine scheduling. Küfer, Karl-Heinz; Rommelfanger, Heinrich; Tammer, Christiane e Winkler, Kristin, editors, *Multicriteria Decision Making and Fuzzy Systems – Theory, Methods and Applications*, Industriemathematik und Angewandte Mathematik, p. 43–58. Shaker Verlag, Aachen. ISBN 3-8322-5540-0.

Geiger, M. J. (2009). Improvements for multi-objective flow shop scheduling by pareto iterated local search. *8th Metaheuristics International Conference (MIC)*, p. 195.1 – 195.10, (2009).

Gendreau, M.; Laporte, G. e Guimaraes, E. M. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 133, n. 1, p. 183–189.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning.

Goldberg, D. E. e Lingle, R. (1985). Alleles, loci, and the tsp. *Proceedings of the 1st International Conference on Genetic Algorithms*, p. 154–159. Lawrence Erlbaum Associates, (1985).

Haimes, Y.Y.; Lasdon, L.S. e Wismer, D.A. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 1, n. 3, p. 296–297.

Hansen, M. P. e Jaszkiewicz, A. (1998). *Evaluating the quality of approximations to the non-dominated set*. IMM, Department of Mathematical Modelling, Technical University of Denmark.

Hariri, A. M. A. e Potts, C. N. (1997). Single machine scheduling with batch set-up times to minimize maximum lateness. *Annals of Operations Research*, v. 70, p. 75–92.

Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness. Relatório técnico, DTIC Document.

Jin, F.; Gupta, J. N. D.; Song, S. e Wu, C. (2010). Single machine scheduling with sequence-dependent family setups to minimize maximum lateness. *Journal of the Operational Research Society*, v. 61, n. 7, p. 1181–1189. ISSN 0160-5682.

Jin, F.; Song, S. e Wu, C. (2009). A simulated annealing algorithm for single machine scheduling problems with family setups. *Computers & Operations Research*, v. 36, n. 7, p. 2133–2138.

Kim, D. W.; Kim, K. H.; Jang, W. e Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, v. 18, n. 3-4, p. 223–231.

Knowles, J. e Corne, D. (2002). On metrics for comparing nondominated sets. *Proceedings of the Congress on Evolutionary Computation (CEC)*, volume 1, p. 711–716. IEEE, (2002).

Leung, J. Y.-T. (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC.

Li, X.; Yalaoui, F.; Amodeo, L. e Chehade, H. (2012). Metaheuristics and exact methods to solve a multiobjective parallel machines scheduling problem. *Journal of Intelligent Manufacturing*, v. 23, n. 4, p. 1179.

Lourenço, H. R.; Martin, O. e Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics*, p. 320–353.

Mavrotas, G. (2009). Effective implementation of the< i> ε</i>-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, v. 213, n. 2, p. 455–465.

Minella, G.; Ruiz, R. e Ciavotta, M. (2010). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, v. 20, n. 3, p. 451.

Mladenovic, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100.

Montoya Torres, J. R.; Soto Ferrari, M.; Gonzalez Solano, F. e Alfonso Lizarazo, E. H. (2009). Machine scheduling with sequence-dependent setup times using a randomized search heuristic. *International Conference on Computers & Industrial Engineering, 2009 (CIE)*, p. 28–33. IEEE, (2009).

Ottoni, R. S.; Arroyo, J. E. C. e Santos, A. G. (2011). Algoritmo vns multi-objetivo para um problema de programação de tarefas em uma máquina com janelas de entrega. *Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, p. 1801 – 1812, (2011).

Pedroso, M.C. e Corrêa, H.L. (1996). Sistemas de programação da produção com capacidade finita: uma decisão estratégica? *Revista de Administração de Empresas*, v. 36, n. 4, p. 00–00.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems*. Springer Verlag.

R Development Core Team,. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Áustria, (2011). URL http://www.R-project.org/. ISBN 3-900051-07-0.

Schaller, J. E. e Gupta, J. N. D. (2008). Single machine scheduling with family setups to minimize total earliness and tardiness. *European Journal of Operational Research*, v. 187, n. 3, p. 1050–1068.

Smith, W.E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, v. 3, n. 1-2, p. 59–66.

Srinivas, Nidamarthi e Deb, Kalyanmoy. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, v. 2, n. 3, p. 221–248.

Takahashi, R. H. C. Otimização escalar e vetorial, (2004). Notas de Aula. Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.

Tanaka, S. e Araki, M. (2012). An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*.

Tang, L. e Wang, X. (2009). Simultaneously scheduling multiple turns for steel color-coating production. *European Journal of Operational Research*, v. 198, n. 3, p. 715–725.

Tavakkoli-Moghaddam, R.; Javadi, B.; Jolai, F. e Ghodratnama, Ali. (2010). The use of a fuzzy multi-objective linear programming for solving a multi-objective single-machine scheduling problem. *Applied Soft Computing*, v. 10, n. 3, p. 919 – 925. ISSN 1568-4946.

Valente, J. (2006). An analysis of the importance of appropriate tie breaking rules in dispatch heuristics. *Pesquisa Operacional*, v. 26, n. 1, p. 169–180.

van derVeen, J. A. A.; Woeginger, G. J. e Zhang, S. (1998). Sequencing jobs that require common resources on a single machine: a solvable case of the tsp. *Mathematical programming*, v. 82, n. 1, p. 235–254.

Yuan, JJ; Liu, ZH; Ng, CT e Cheng, T.C.E. (2006). Single machine batch scheduling problem with family setup times and release dates to minimize makespan. *Journal of scheduling*, v. 9, n. 6, p. 499–513.

Zitzler, E.; Deb, K. e Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, v. 8, n. 2, p. 173–195.

Zitzler, E. e Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms-a comparative case study. *Parallel problem solving from nature-PPSN V*, p. 292–301. Springer, (1998).

Zitzler, E. e Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 4, p. 257–271.

Zitzler, E.; Thiele, L.; Laumanns, M.; Fonseca, C. M. e da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, v. 7, n. 2, p. 117–132.