Heurística para sintonização de metaheurísticas aplicada ao Problema de FlowShop Permutacional

Thiago Henrique Lemos Fonseca

Heurística para sintonização de metaheurísticas aplicada ao Problema de FlowShop Permutacional

Thiago Henrique Lemos Fonseca

Orientador: **Prof. Dr. Alexandre Cesár Muniz de Oliveira**

Dissertação apresentada ao Programa de Pós-Graduação da Universidade Federal do Maranhão, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação

UFMA – São Luís
Março de 2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Lemos Fonseca, Thiago Henrique.
Heurística para Sintonização de Metaheurísticas aplicada ao Problema de FlowShop Permutacional / Thiago Henrique Lemos Fonseca. – 2018.
102 f.

Orientador(a): Alexandre César Muniz de Oliveira.
Dissertação (Mestrado) – Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, Auditório do NTI – UFMA, 2018.

1. Algoritmos de Corrida. 2. BRKGA. 3. Clustering Search. 4. Metaheurísticas. 5. Sintonização. I. Muniz de Oliveira, Alexandre César. II. Título.

Este trabalho é dedicado aos meus familiares,
pois é graças ao esforço deles que hoje estou realizando meus sonhos.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado força em cada dia da minha vida para que eu pudesse caminhar e estudar, assim chegando hoje onde estou. Agradeço aos meus familiares, por todo o suporte, atenção e compreensão nesses anos de muito trabalho e sacrifício. Agradeço a minha namorada pelo companheirismo e paciência durante as várias madrugadas acordado; aos meus amigos de longa data, por estarem sempre ao meu lado desde o primeiro projeto da escola até esse momento especial. Finalmente, Agradeço ao professor Alexandre César, responsável pela orientação desse trabalho, pela atenção e carinho em suas correções, sempre visando transformar meu esforço em excelência e aos docentes do PPGCC/UFMA que apoiaram cada etapa da pesquisa e contribuíram com seus ensinamentos e experiência.

“É preciso ter um caos dentro de si para dar à luz uma estrela cintilante.”
(Friedrich Nietzsche)

RESUMO

FONSECA, T. H. L.. **Heurística para sintonização de metaheurísticas aplicada ao Problema de FlowShop Permutacional**. 2018. 102 f. Dissertação (Mestrado em XXX) – Universidade Federal do Maranhão (UFMA), UFMA – MA.

Muitos problemas de Otimização Combinatória são considerados NP-Difíceis e portanto requerem um alto custo computacional para serem resolvidos por algoritmos exatos. Uma alternativa promissora é a utilização de metaheurísticas, modelos algorítmicos genéricos capazes de encontrar boas soluções para problemas de otimização complexos em tempo razoável. Contudo, para que as metaheurísticas obtenham soluções de qualidade, parâmetros de diversos tipos devem ser calibrados.

Ao problema de encontrar o melhor ajuste para esses parâmetros dá-se o nome de Sintonização. Usualmente, o processo de encontrar configurações ótimas em um espaço de busca de parâmetros possui dificuldade igual ou superior à busca de soluções ótimas no espaço de soluções do problema, tal obstáculo torna o estudo da sintonização pouco atrativo aos pesquisadores, que preferem abordagens mais baratas baseadas em tentativa e erro ou experiência de especialistas.

Como não existe um padrão na sintonização, pesquisadores tendem a ajustar parâmetros seguindo abordagens próprias que influenciam de diferentes modos a eficácia de seus algoritmos, dificultando a comparação e aperfeiçoamento dos mesmos devido a aspectos pouco mensuráveis.

Este trabalho apresenta um método de sintonização heurístico denominado *Cross-Validated Racing* (CVR) que agrega validação cruzada ao processo de sintonização por corrida buscando alcançar uma perspectiva de generalização por Aprendizagem de Máquina, obtendo assim soluções de qualidade para instâncias desconhecidas.

Para a validação do CVR, um algoritmo genético de chaves aleatórias e viciadas híbrido (BRKeCS) foi projetado e aplicado para resolver problemas do tipo *FlowShop Permutacional* com instâncias randômicas e realistas. Os resultados computacionais demonstraram que a CVR é robusto ao encontrar uma configuração de parâmetros efetiva com um erro residual médio de menos de 2.3% quando comparado com outras metaheurísticas desenvolvidas especialmente para o problema uma vez que requer processo de treinamento em apenas metade do conjunto total de instâncias. Também verificou-se a estabilidade da qualidade de soluções do CVR quando a topologia do espaço de busca é modificada pela alteração de instâncias aleatórias para as realísticas.

Palavras-chave: BRKGA, Clustering Search, Algoritmos de Corrida, Metaheurísticas, Otimização, Sintonização.

ABSTRACT

FONSECA, T. H. L.. **Heurística para sintonização de metaheurísticas aplicada ao Problema de FlowShop Permutacional**. 2018. 102 f. Dissertação (Mestrado em XXX) – Universidade Federal do Maranhão (UFMA), UFMA – MA.

Many combinatorial optimization problems are considered NP-Hard and therefore require a high computational cost to be solved by exact algorithms. A promising alternative is the use of metaheuristics, generic algorithmic models capable of finding great solutions to complex optimization problems in a reasonable time. However, for metaheuristics to obtain quality solutions, parameters of various types must be calibrated.

The problem of finding the best setting for these parameters is called Tuning. Usually, the process of finding optimal settings in a parameter search space has difficulty equal to or greater than the search for optimal solutions in the solution space of the problem, such an obstacle makes the study of tuning unattractive to researchers, who prefer cheaper approaches based in trial and error or expert experience.

As there is no standard in the tuning, researchers use to set parameters by following their own approaches that influence in different ways the effectiveness of their algorithms, making it difficult to compare and improve them due to aspects that are not very measurable.

This work presents a cross-validated Racing (CVR) heuristic tuning method that adds cross-validation to the tuning process by racing to achieve a generalization perspective by Machine Learning, thus obtaining quality solutions for unknown instances .

For CVR validation, a hybrid biased random-key genetic algorithm (BRKeCS) was designed and applied to solve Permutational FlowShop Problems with random and realistic instances. The computational results demonstrated that the CVR is robust to find an effective parameter configuration with an average residual error of less than 2.3% when compared to other metaheuristics specially developed for the problem since it requires training process in only half of the set total number of instances. We also verified the stability of the quality of CVR solutions when the search space topology is modified by changing from random to realistic instances.

Key-words: BRKGA, Clustering Search, Racing Algorithms, Metaheuristics, Otimization, Tuning.

LISTA DE ILUSTRAÇÕES

Figura 1 – Superfície de resposta tridimensional e o contorno correspondente para a força da idade inicial do cimento romano onde x_1 é a subtemperatura de calcinação (C) e x_2 é o tempo de permanência (min.) 38
Figura 2 – Um experimento 3^3 Fatorial Completo (27 pontos) 40
Figura 3 – 3 frações de $1/3$ de um experimento 3^3 fatorial 40
Figura 4 – Representação de uma solução de Flowshop Permutacional com 8 tarefas e duas máquinas. Independente do número de máquinas a ordem das tarefas deve ser a mesma. 44
Figura 5 – Gráfico de Gantt para o exemplo da FlowShop Permutacional com o melhor makespan 45
Figura 6 – Uma instância de $3/4/P/c_{max}$. Legenda: (...) Arcos para máquina 1, (- - -) arcos para máquina 2, (- . -) arcos para máquina 3 e (- . . -) arcos para máquina 4 60
Figura 7 – Projeto conceitual do BRKeCS 68
Figura 8 – Design conceitual do CVR 73
Figura 9 – Erro residual Médio para os 6 experimentos sob o maior grupo de instâncias 84
Figura 10 – Erro residual Médio para os 6 experimentos sob o maior grupo de instâncias do Taillard e realísticas 85
Figura 11 – Erro residual por número de tarefas 85
Figura 12 – Diferença entre os erros residuais do BRKeCS sintonizado e com ajuste manual para o maior grupo de instâncias 86

LISTA DE ALGORITMOS

Algoritmo 1 – Força Bruta 31
Algoritmo 2 – Algoritmo de Corrida genérico 36
Algoritmo 3 – Recozimento Simulado 55
Algoritmo 4 – Busca Tabu 57
Algoritmo 5 – Algoritmo Genético genérico 63
Algoritmo 6 – BRKeCS 67

LISTA DE TABELAS

Tabela 1 – Erro residual para instâncias 20x20 71
Tabela 2 – Comparação BRKeCS com ACO 71
Tabela 3 – Configuração de parâmetros gerada pelo CVR para a análise 1 82
Tabela 4 – Comparação entre o BRKeCS ajustado pelo CVR e outras metaheuristicas aplicadas ao problema 82
Tabela 5 – Configuração de parâmetros gerada pelo CVR 82
Tabela 6 – Erro residual médio do BRKeCS ajustado com o CVR para instâncias não utilizadas no treinamento 83

SUMÁRIO

1 INTRODUÇÃO 16
1.1 Contextualização 16
1.2 Objetivo Geral 17
1.3 Objetivos Específicos 17
1.4 Justificativa 18
1.5 Organização do Trabalho 18

2 SINTONIZAÇÃO DE METAHEURÍSTICAS 19
2.1 Metaheurísticas 19
2.2 Análise experimental de algoritmos 21
2.3 Definição formal da Sintonização 22
2.4 Complexidade da Sintonização 23
2.4.1 Análise de primeira ordem do estimador *25*
2.4.2 Análise de segunda ordem do estimador *26*
2.5 Técnicas em Sintonização de parâmetros para Metaheurísticas . . . 30
2.5.1 Abordagem baseada em modelo livre *32*
2.5.1.1 Revac *33*
2.5.1.2 ParamILS *33*
2.5.1.3 Algoritmo genético baseado em gênero *34*
2.5.2 Abordagem baseada em modelos *34*
2.5.2.1 EGO *34*
2.5.2.2 Abordagem por Corrida *35*
2.5.2.3 Abordagem por Metodologia de Superfície de Resposta *37*
2.5.2.4 Função Modelo Aproximada *37*
2.5.2.5 Projeto de Experimentos *39*
2.5.2.6 Experimento Fatorial *39*
2.5.2.7 Calibra *41*
2.5.2.8 SKO *41*
2.5.2.9 SPO *41*
2.5.2.10 SMAC *41*

3 PROBLEMAS DE PROGRAMAÇÃO DE TAREFAS 43
3.1 Problema FlowShop Permutacional 44

3.2 Métodos Exatos . . . 46
3.2.1 Programação Dinâmica . . . *46*
3.2.2 Regra de Johnson . . . *47*
3.3 Heurísticas Construtivas . . . 49
3.3.1 Algoritmo de Palmer . . . *49*
3.3.2 Algoritmo de Gupta . . . *50*
3.3.3 Algoritmo CDS . . . *50*
3.3.4 Algoritmo NEH . . . *50*
3.4 Metaheurísticas aplicadas ao problema FlowShop Permutacional . . . 51
3.4.1 Recozimento Simulado . . . *51*
3.4.2 Busca Tabu . . . *55*
3.4.3 Colônias de Formigas . . . *57*
3.4.4 Algoritmos Genéticos . . . *61*

4 ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS GUIADO POR AGRUPAMENTO . . . 64
4.1 Algoritmos Genéticos de chaves aleatórias e viciadas . . . 65
4.2 Evolutionary Clustering Search . . . 65
4.3 BRKeCS . . . 67
4.3.1 Cruzamento . . . *67*
4.3.2 Busca Local . . . *69*
4.3.3 Decodificador . . . *69*
4.3.4 Assimilação . . . *69*
4.3.5 Parâmetros de desempenho . . . *70*
4.4 BRKeCS aplicado ao problema de FlowShop Permutacional . . . 70

5 CROSS-VALIDATED RACING (CVR) . . . 72
5.1 Definição . . . 72
5.2 Design conceitual . . . 72
5.3 Pré-processamento . . . 73
5.4 Sintonia Cross-Validada . . . 74
5.4.1 Estimativa do erro de aprendizagem . . . *74*
5.4.2 Validação cruzada . . . *74*
5.4.3 Algoritmo de corrida . . . *75*
5.5 Testes estatísticos . . . 76

6 RESULTADOS COMPUTACIONAIS . . . 80
6.1 Instâncias e metaheurística . . . 80
6.2 Análise 1 . . . 82
6.3 Análise 2 . . . 83

6.4 Análise 3 84
6.5 Considerações finais 86

7 CONCLUSÃO 87
7.1 Resumo das Contribuições 87
7.2 Discussão 88
7.3 Trabalhos Futuros 88

REFERÊNCIAS 89

CAPÍTULO

1

INTRODUÇÃO

1.1 Contextualização

Maximizar benefícios ou minimizar perdas são diretivas presentes em todas as áreas. Naturalmente, o ser humano tem dedicado grande esforço no desenvolvimento e aperfeiçoamento de ferramentas e técnicas que o permitam alcançar resultados esperados com um mínimo de esforço (PAPADIMITRIOU; STEIGLITZ, 1998).

De um modo geral, pode-se dizer que a otimização é um processo inerente ao ser humano desde sempre. No mundo contemporâneo, o processo de otimização tem papel relevante em diversas áreas do conhecimento, com destaque para as diversas engenharias e, portanto, possui grande relevância econômica (KANTOROVICH, 1980) (KELLEY, 2010)

A história dos métodos de otimização tem sido dominada por duas preocupações principais: desenvolver um método de otimização eficiente, ou seja, capaz de encontrar soluções de alta qualidade em um curto intervalo de tempo. Por outro lado, um método de otimização deve ser gerenciável, ou seja, fácil de se usar e capaz de se adaptar à outros problemas de otimização com poucas mudanças (WOEGINGER, 2003).

Muitos problemas de otimização, em especial a otimização combinatória, possuem alta complexidade computacional, tornando as técnicas exatas utilizadas para resolvê-los pouco eficientes e bem específicas, prejudicando assim a capacidade do algoritmo de ser gerenciável. Metaheurísticas formam uma classe de algoritmos com potencial para superar esses obstáculos (WOEGINGER, 2003).

Uma metaheurística é um modelo algorítmico genérico que pode ser usado para encontrar soluções de alta qualidade de problemas difíceis de otimização. Para chegar a um algoritmo funcional, uma metaheurística precisa ser instanciada em termos de procedimentos específicos e sintonização de parâmetros (SAKA; DOGAN, 2012).

A importância da sintonização é reconhecida pela comunidade científica (BARR *et al.*, 1995): é evidente para quem tem alguma experiência direta com esse tipo de algoritmo de otimização que esses métodos são bastante sensíveis ao valor de seus parâmetros e que uma sintonia cuidadosa geralmente melhora o desempenho de forma significativa. Contudo, apenas alguns trabalhos promissores como Coy *et al.* (2001) e Adenso-Diaz e Laguna (2006) têm dado a devida atenção para o problema.

Como evidenciado no decorrer desta dissertação, o problema de sintonização tem características de um problema de aprendizado de máquina, consistindo em encontrar a melhor configuração possível para se produzir os melhores resultados sobre um conjunto de instâncias de problema.

Em particular, o problema de sintonia se baseia no conceito de classe de instâncias, que considera uma medida de probabilidade sobre o espaço das instâncias do problema de otimização em questão. Embora com objetivos diferentes, a abordagem probabilística já foi usada na literatura por pesquisadores como Wolpert e Macready (1997) em seus trabalhos sobre a inexistência de almoço grátis Birattari (2009a), Radcliffe e Surry (1995), Schumacher, Vose e Whitley (2001), Igel e Toussaint (2003) e Corne e Knowles (2003).

1.2 Objetivo Geral

Neste trabalho, propõe-se uma metodologia de sintonização de metaheurísticas denominada *Cross-Validated Racing* (CVR) que agrega validação cruzada ao processo de sintonização por corrida buscando alcançar uma perspectiva de generalização por Aprendizagem de Máquina. A capacidade de generalização deve permitir que configurações de alta qualidade sejam encontradas a partir de um processo de treino e validação sobre um subconjunto das instâncias totais, possibilitando diminuir consideravelmente o tempo de sintonização.

1.3 Objetivos Específicos

- Implementar a metaheurística híbrida BRKeCS (*Biased Random-Key Evolutionary Clustering Search*), construída a partir do *algoritmo genético de chaves aleatórias e viciadas* e da *busca guiada por agrupamentos* que realiza otimização por agrupamento em um espaço de busca codificado em vez do espaço de busca original, para ser sintonizada pela técnica proposta no Objetivo Geral.

- Analisar a eficiência do BRKeCS com sintonização e sem sintonização frente à problemas do tipo *FlowShop Permutacional*.

- Comparar a eficiência do BRKeCS sintonizado pelo método CVR com as principais metaherísticas da literatura implementadas para problemas do tipo *FlowShop Permutacional*.

- Realizar testes de acurácia da técnica CVR sobre a base de dados.
- Analisar a capacidade de generalização do CVR a partir do experimento das instâncias mínimas, em que se diminui gradativamente a quantidade de instâncias de treino visando verificar até qual momento a técnica consegue manter soluções de boa qualidade.
- Analisar a robustez da técnica através da modificação da topologia do espaço de busca feita pela modificação de instâncias aleatórias do Taillard (1990) por instâncias realísticas do Watson *et al.* (2002).

1.4 Justificativa

A Sintonização é um processo computacionalmente custoso e pontual, ou seja, o ajuste é geralmente refeito para cada classe de instâncias. Um método que analise um conjunto de instâncias de diferentes tipos de maneira a prover configurações de parâmetros para instâncias dentro e fora do conjunto de treinamento tem grande potencial de aceitação por parte da comunidade científica.

A Sintonização de metaheurísticas possui grande influência sob a qualidade das soluções encontradas. Por este motivo, grandes esforços têm sido empreendidos para a elaboração de técnicas que permitam o ajuste fino dos parâmetros do algoritmo. No entanto, uma variedade de implementações e tipos de metaheurísticas resultam em ajustes dependentes de um especialista ou por técnicas de tentativa e erro, o que é ineficiente e dificulta a validação dos resultados.

Embora a sintonização de metaheuristicas não seja um campo desconhecido na literatura, poucos trabalhos deram a devida profundidade a esse tema uma vez que o problema de encontrar a melhor configuração de parâmetros é tão complexo quanto o problema original ao qual a sintonização se destina.

1.5 Organização do Trabalho

Para que a explicação do método seja clara, este trabalho iniciou o Capítulo 2 com a definição formal do processo de sintonização, evidenciando as principais pesquisas na área, a formalização matemática do problema e as técnicas mais utilizadas na atualidade. O Capítulo 3 abordou as características dos problemas *FlowShop*, sua complexidade e as técnicas conhecidas para resolvê-lo baseadas em métodos exatos e heurísticos, bem como suas limitações. O Capítulo 4 expõe a metaheurística híbrida denominada BRKeCS que foi implementada para solucionar o problema abordado no capítulo anterior. O Capítulo 5 introduz o método proposto *Cross-Validated Racing*. O Capitulo 6 mostra os resultados computacionais da Sintonização do BRKeCS com CVR para problemas do tipo *FlowShop*. Finalmente, no Capítulo 7, destacam-se as principais conclusões e expectativa de trabalhos futuros.

CAPÍTULO

2

SINTONIZAÇÃO DE METAHEURÍSTICAS

2.1 Metaheurísticas

As técnicas de otimização podem ser amplamente classificadas em métodos exatos e aproximados (TALBI, 2009). Os métodos exatos obtêm soluções ótimas em qualquer execução. Esta categoria inclui métodos como algoritmos *branch and bound*, programação dinâmica, algoritmos de busca Bayesiana e métodos de aproximação sucessivos. Métodos aproximados destinam-se a fornecer uma solução de boa qualidade (que nem sempre é a melhor solução possível) em um período razoável de tempo. Métodos aproximados podem ser divididos em algoritmos de aproximação e métodos heurísticos (TALBI, 2009). Algoritmos de aproximação fornecem qualidade de solução provável e limites de tempo de execução prováveis, enquanto heurísticas envolvem uma solução razoavelmente boa em um tempo razoável. Os algoritmos heurísticos são altamente dependentes do problema, o que gera um obstáculo importante. As metaheurísticas formam uma classe de algoritmos que atuam como mecanismo de orientação para as heurísticas subjacentes que não são dependentes de problema ou domínio específico podendo ser aplicadas a qualquer problema de otimização. O termo metaheurísticas foi introduzido por Glover (1986).

Uma metaheurística é formalmente definida como um processo de geração de soluções candidatas iterativo que orienta uma heurística subordinada, combinando conceitos diferentes para intensificar e explorar o espaço de busca; As estratégias de aprendizagem são usadas para estruturar informações, a fim de encontrar soluções eficientemente que sejam ótimas ou próximas às ótimas (OSMAN; LAPORTE, 1996). Os algoritmos metaheurísticos criam um equilíbrio entre intensificação e diversificação do espaço de busca e podem ser usados para resolver de forma eficiente problemas NP-Difíceis com grande número de variáveis e funções objetivas não-lineares.

O desenvolvimento no campo das metaheurísticas decorre principalmente da importância

de problemas complexos de otimização para a área industrial e científica. Nas últimas duas décadas, muitos algoritmos metaheurísticos foram propostos como alternativa para obtenção de soluções de qualidade em tempo computacional exequível: algoritmos genéticos (HOLLAND, 1992), algoritmo memético (MOSCATO *et al.*, 1989), sistema imunológico artificial (FARMER; PACKARD; PERELSON, 1986), recozimento simulado (KIRKPATRICK; GELATT; VECCHI, 1983), busca tabu (GLOVER; LAGUNA, 2013), otimização por colônias de formigas (DORIGO, 1992), otimização por enxames de partículas (EBERHART; KENNEDY, 1995) e evolução diferencial (PRICE; STORN; LAMPINEN, 2006), entre outros.

Os Teorema da não existência de almoço grátis (*NFL -No free lunch theorems*) possuem implicações significantes no campo da Otimização, em especial para as metaheurísticas. Um dos teoremas afirma que se um algoritmo **A** supera o algoritmo **B** para algumas funções de otimização, então **B** será superior a **A** para outras funções. Em outras palavras, se considerarmos todo o espaço de funções possíveis, **A** e **B** serão executados, na média, igualmente bem. Ou seja, não há algoritmos universalmente melhores. Um ponto de vista alternativo é que não há necessidade de se encontrar a média sobre todas as funções possíveis para um problema de otimização. Neste caso, a principal tarefa é encontrar as melhores soluções, que não tem nada a ver com a média de todo o espaço de funções possível. Outros pesquisadores acreditam que não existe uma ferramenta universal e, com base em experiências, alguns algoritmos superam outros para determinados tipos de problemas de otimização. Assim, o objetivo principal seria escolher o algoritmo mais adequado para um determinado problema ou projetar algoritmos melhores para a maioria dos tipos de problemas, não necessariamente para todos os problemas (WOLPERT; MACREADY, 1997). Graças à esses teoremas, uma grande quantidade de metaheurísticas com diferentes características e para diferentes fins foram e estão sendo desenvolvidas, sendo que algumas são mais adequadas que outras para determinados problemas.

Metaheurísticas encontram aplicação em áreas como telecomunicações, engenharia ,logística e planejamento de negócios. Um ponto de interesse no estudo desses algoritmos se encontra na necessidade de se ajustar parâmetros que são específicos para cada tipo de metaheurística. Cada conjunto de parâmetros escolhido atua de maneira diferente na intensificação e exploração do espaço de busca, logo, um bom ajuste pode melhorar significativamente a eficiência da metaheuristica em relação a qualidade da solução encontrada e ao custo computacional para encontrá-la. O ajuste de algoritmos com o objetivo de melhorar a performance é estudado há décadas, O Capítulo 4 explica o funcionamento de algumas das principais metaheurísticas descritas, bem como sua aplicação em problemas *FlowShop*, a seção seguinte resume as principais contribuições no campo da análise de algoritmos, precursora para o estudo de sintonização abordado na Seção 2.3 .

2.2 Análise experimental de algoritmos

O desenvolvimento de técnicas para análise de algoritmos iniciaram nos anos 80 com os trabalhos de Golden, Stewart e J.K. (1985) e Golden *et al.* (1986). Com a evolução da área e capacidade computacional, técnicas mais robustas puderam ser propostas, como a utilização de redução de variança (MCGEOCH, 1992) e algoritmos de otimização estocásticos (BARR *et al.*, 1995). Contudo, apenas nos anos 90 a preocupação com uma análise experimental, ou seja, com um objetivo mais prático que teórico foi colocada em discussão. Em Hooker (1995) levantaram-se ideias de como considerar uma distribuição de probabilidade sobre um conjunto de instâncias e foi defendida uma prática experimental que deve descrever algoritmos em vez de compará-los. Em particular, o autor insistiu na necessidade de obter um modelo de algoritmos que devesse destacar o efeito de um ou mais fatores sobre o desempenho do próprio algoritmo. Para isso, um fator específico deve ser identificado e dois casos diferentes de implementação devem ser comparados: com a presença e com a ausência do fator.

Em muitos casos, uma metaheurística é sintonizada manualmente na base da tentativa e erro. Essa abordagem é comum em vários artigos na pesquisa na área. De fato, alguns pesquisadores declaram explicitamente a limitação dessa abordagem (BREEDAM, 1995) (GENDREAU; HERTZ; LAPORTE, 1994). O principal problema com a abordagem se refere a produção industrial em larga escala de metaheurísticas que requerem grande quantidade de tempo, trabalho intensivo e atenção de um especialista. Em nível acadêmico, a adoção dessa abordagem incorre no risco de invalidar conclusões para comparações experimentais de diferentes algoritmos.

Alguns autores, influenciados por Hooker (1995), adotaram a metodologia baseada em projeto fatorial que é característico de uma análise descritiva. Contudo, antes de resolver o problema de sintonização propriamente dito, o algoritmo passa pelo problema intermediário complexo de definir a importância relativa dos parâmetros. Por exemplo, em Xu e Kelly (1996) tentaram identificar a contribuição relativa de cinco diferentes componentes de uma Busca Tabu, desabilitando um componente de cada vez, executando o algoritmo para sete instâncias e comparando os resultados do grau de efetividade de cada componente. Essa abordagem, possui desvantagens relacionadas ao custo computacional e complexidade dos experimentos.

No início dos anos 2000, uma nova técnica denominada metodologia de superfície de resposta começou a ser utilizada com ótimos resultados (DEAN; VOSS, 1999) (MYERS; MONTGOMERY, 1995). Este método, que é semelhante ao método de Descida do Gradiente, consiste em uma busca iterativa no espaço da parâmetros que podem ser descrito da seguinte forma: uma métrica é definida no espaço dos parâmetros que atribui uma distância entre cada par de configurações da metaheurística a serem sintonizadas. A pesquisa começa em algum ponto do espaço do parâmetros, ou seja, para alguns valores dados dos parâmetros, e se move iterativamente no espaço do parâmetros, considerando as sequências de pontos. Na iteração genérica do processo de busca, a metaheurística é testada para o valor dos parâmetros correspondentes ao ponto atual e para aqueles que correspondem a pontos vizinhos. Se o resultado observados no ponto atual

é melhor que o observado na vizinhança considerada, a pesquisa é interrompida e os valores dos parâmetros correspondentes ao ponto atual são retornados. Caso contrário, o ponto atual é substituído pelo melhor ponto na vizinhança considerada e o processo é iterado. Apesar do método não garantir a configuração ótima global, geralmente encontra soluções de qualidade.

A principal desvantagem da abordagem inspirada na metodologia de superfície de resposta é a necessidade de que uma métrica seja definida no espaço de parâmetros. Embora esta suposição não represente nenhum problema se os parâmetros são variáveis ordinais, discretas ou contínuas, não se mantém no caso de variáveis categóricas, isto é, para variáveis cujos valores possíveis não podem ser ordenados de forma significativa.

Hutter, Hoos e Stützle (2007) propuseram uma abordagem baseada em uma busca local iterativa no espaço de parâmetros. A validação dessa abordagem tem sido demonstrada com sucesso em vários estudos em que o objetivo é minimizar o tempo de execução de algoritmos que resolvem Problemas de Decisão de Satisfatibilidade (SCHAEFER, 1978).

Em geral, a principal limitação de muitos dos trabalhos acima mencionados é a ausência de uma definição clara do próprio problema de sintonização.

2.3 Definição formal da Sintonização

Metaheurísticas não são propriamente algoritmos, mas sim arcabouços que servem como um guia para geração de um algoritmo específico para um dado problema previamente definido e modelado. É conveniente olhar para a metaheurística como um modelo algorítmico que precisa ser instanciado para produzir um algoritmo totalmente funcional. Em todos os casos, vários parâmetros, numéricos ou categóricos ou ambos, precisam ser ajustados.

Birattari (2009a) define o problema de sintonização como: dado um conjunto finito de configurações candidatas Θ fornecido juntamente com uma classe de instâncias I. Cada $\theta \in \Theta$ escolhido para executar durante um tempo $t \in T$ sobre uma instância $i \in I$ com probabilidade $P_I(i)$ gera um custo c. Seja C a coleção de c com os possíveis valores de custo da melhor solução encontrada na execução de θ em I. Seja $P_C\ (c \mid \theta, i)$ um indicador da probabilidade de c ser o custo da melhor solução encontrada com tempo $t(i)$ de aplicação da configuração θ na instância i. O problema de sintonia consiste em encontrar, dentro de um tempo T máximo, a melhor configuração ($\min_\theta$), de acordo com um critério C, quando as medidas P_I e P_C são desconhecidas. Assim, o problema de sintonia, enfim, pode ser descrito pelos seguintes componentes:

- Θ : conjunto de configurações candidatas;
- I: conjunto de instâncias;
- P_I: probabilidade da instância i ser selecionada para ser resolvida;
- $t : I \rightarrow R$: função que relaciona o tempo computacional para cada instância;

- c: variável aleatória que representa o custo da melhor solução encontrada executando a configuração θ na instância i por $t(i)$ segundos;
- $C \subset \mathbb{R}$: intervalo de valores possíveis de c, ou seja, o custo da melhor solução encontrada da execução da configuração θ na instância i;
- $P_C(c|\theta,i)$: Probabilidade de que c seja o custo da melhor solução encontrada ao executar a configuração θ por $t(i)$ segundos na instância i;
- $C(\theta) = C(\theta|\Theta,I,P_i,P_C,t)$: critério a ser otimizado em relação à θ;
- T:Tempo máximo para o experimento, dado um conjunto de configurações candidatas em um conjunto de instâncias.

Com base nesses conceitos, o Problema de Ajuste pode ser descrito formalmente como o 7-Tuple $\langle \Theta, I, PI, PC, t, C, T \rangle$ onde o objetivo é dado por (BIRATTARI, 2009a)

$$\overline{\theta} = arg\min_{\theta} C(\theta) \quad . \tag{2.1}$$

Espera-se encontrar o custo $C(\theta)$ expresso pela integral:

$$C(\theta) = \int c dP_C(c|\theta,i) dP_I(i) \quad . \tag{2.2}$$

A expressão acima não pode ser calculada analiticamente uma vez que os valores de P_C e P_I não são conhecidos. No entanto, as amostras podem ser analisadas e, de acordo com essas medidas, as quantidades $C(\theta)$ podem ser estimadas usando o método de Monte Carlo (RUBINSTEIN; KROESE, 2016).

Usando a perspectiva de aprendizado de máquina, é possível dar uma definição formal para o problema de sintonização e desenvolver um algoritmo genérico para sintonização de metaheurísticas. Além disso, a partir da perspectiva de aprendizado de máquina, torna-se possível realçar algumas falhas na metodologia de pesquisa atual e propor orientações para futuras pesquisas na área de Metaheurísticas (BIRATTARI, 2009b).

2.4 Complexidade da Sintonização

Um problema de sintonização pode ser descrito pela tupla (Θ,I,P_i,P_C,t,C,T), um conjunto finito Θ de configurações candidatas e uma classe de instâncias I. Em cada passo uma instância i é escolhida com probabilidade definida pela medida P_I. O custo da melhor solução de i encontrada por um candidato θ em um tempo $t(i)$ é uma quantidade estocástica descrita pela medida P_C. O problema de sintonização consiste em encontrar, dentro de um tempo total T, a melhor configuração de acordo com o critério C, quando as as medidas P_I e P_C são desconhecidas.

Uma amostra de instâncias pode ser obtida para testar as configurações candidatas. Como é assumido que o conjunto θ é finito, parâmetros contínuos devem ser discretizados para que o método proposto seja aplicado efetivamente.

No tratamento do problema de sintonização, restringi-se a atenção para o caso em que o critério C, a ser minimizado, é o custo esperado μ da solução encontrada por um candidato θ em uma instância, expresso como a integral dada pela Equação 2.3 que não pode ser calculada analiticamente.

$$\mu(\theta) = \int c dP_C(c|\theta,i) dP_I(i) \quad . \tag{2.3}$$

De fato, qualquer instância escolhida com probabilidade P_I e qualquer execução de uma dada configuração θ nessa instância i, é uma amostra aleatória da medida $P_C = P_C(c|\theta,i)$. Com base nessas amostras, pode-se imediatamente obter uma estimativa de Monte Carlo μ do valor esperado.

A seguir, apresentam-se as definições matemáticas necessárias para o cálculo do estimador μ contidas em Birattari (2009b).

Definição 2.1. *Denomina-se um cenário para a estimativa do comportamento esperado da configuração θ, a medida de probabilidade conjunta $P(c,i) = P_C(c|i)P_I(i)$.*

Exemplo 2.1. A estimativa, com base em N execuções de uma configuração θ, do comportamento médio na classe I, ou seja, o valor esperado do custo c em relação ao cenário $P(c,i) = P_C(c|i)P_I(i)$, é dado por:

$$\mu = E[c] = \int c dP_C(c|\theta,i) dP_I(i), \tag{2.4}$$

no qual o operador E representa a esperança matemática em relação à probabilidade $P(c,i)$.

Para esse objetivo, executam-se um conjunto de experiências, com $|J| = N$. Para cada experimento $j \in J$, observa-se um custo c_j. A quantidade μ pode ser estimada por meio de $\widehat{\mu}$:

$$\widehat{\mu} = \frac{1}{N} \sum_{j \in J} c_j \tag{2.5}$$

Por exemplo, seja K instâncias distintas $i_1, i_2, ..., i_K$, com $K \leq N$ e a configuração θ sendo executada n_1 vezes na instância i_1, n_2 vezes na instância i_2, e assim sucessivamente. Isso equivale a considerar um conjunto de experiências J que é particionado em subconjuntos $J_1, J_2, ..., J_K$, em que $|J_k| = n_k$ e $\sum_k n_k = N$. Cada elemento j no subconjunto genérico J_k é um experimento consistindo em executar θ uma vez na instância i_k.

Definição 2.2. *Chama-se uma configuração experimental, ou simplesmente uma configuração, a sequência de números naturais $S_N = (K, n_1, n_2, ..., n_K)$, isto é, a especificação de quantas instâncias devem ser consideradas, juntamente com o número de execuções em cada um deles*

Definição 2.3. *Se K divide N, denota-se com $H_{K|N/K}$ a configuração homogênea, isto é, $S_N = (K, n_1, n_2, ..., n_K)$, onde $n_k = N/K$ para todos os k. Em particular, $H_{N|1} = (N, n_1, n_2, ..., n_N)$, com $n_k = 1$ para todos k, é a configuração N instâncias, uma execução por instância. Da mesma forma, $H_{1|N} = (1, N)$ é a configuração Uma instância, N execuções.*

Definição 2.4. *Em um determinado cenário $P(c,i) = P_C(c|i)P_I(i)$, e para uma dada configuração experimental $S_N = (K, n_1, n_2, ...n_k)$, o estimador $\widehat{\mu}_{SN}$ do valor esperado μ do custo c é dado por:*

$$\widehat{\mu}_{SN} = \frac{1}{N} \sum_{k=1}^{K} \sum_{j \in J_k} c_j, \tag{2.6}$$

em que $N = \sum_{k=1}^{K} |J_k|$, $|J_k| = n_k$ e as instâncias i_k e custos c_j são extraídos de acordo com $P(c,i)$.

Definição 2.5. *O valor esperado para o custo c na instância i é definida por:*

$$\mu_i = E[c|i] = \int c dP_C(c|i). \tag{2.7}$$

Definição 2.6. *A variança do custo c na instância i é dada por:*

$$\sigma_i^2 = E[(c - \mu_i)^2 | i] = \int (c - \mu_i)^2 dP_C(c|i) \tag{2.8}$$

Definição 2.7. *A variança esperada na instância é:*

$$\overline{\sigma}_{WI}^2 = \int \sigma_i^2 dP_I(i), \tag{2.9}$$

representando o valor esperado com respeito a distribuição de instâncias da variança de c dentro de uma mesma instância.

Definição 2.8. *A variança entre instâncias é:*

$$\sigma_{AI}^2 = \int (\mu_i - \mu)^2 dP_I(i), \tag{2.10}$$

representando a variança entre as instâncias do valor esperado do custo para cada instância.

2.4.1 Análise de primeira ordem do estimador

O estimador é calculado a partir do comportamento esperado de uma dada configuração θ atuando com uma medida de probabilidade $P(c,i)$ explicada anteriormente. Para a análise de primeira ordem do estimador μ são necessárias as seguintes definições de Birattari (2009b)

Lema 2.1. *Em um dado cenário $P(c,i) = P_C(c|,i)P_I(i)$, e para um dado configuração experimental $S_N = (K, n_1, n_2, ..., n_K)$, a probabilidade de obter as instâncias específicas $i_1, i_2, ..., i_K$ e os resultados específicos $c_1, c_2, ..., c_N$ em que o $\widehat{\mu}_{SN}$ é baseado e dado por:*

$$P(i_1, i_2, ..., i_K, c_1, c_2, ..., c_N) = \prod_{k=1}^{K} P_I(i_k) \prod_{j \in J_k} P_C(c_j|i_k), \tag{2.11}$$

em que $P_I(i_k)$ é a probabilidade da instância i_k e $P_C(c_j|i_k)$ é a probabilidade de se obter o custo c_j como melhor resultado na execução da configuração θ na instância i_k.

Demonstração. As K instâncias são amostradas independentemente de acordo com a medida de probabilidade $P_I(i)$. Similarmente, o custo c obtido em uma dada instância i é amostrado independentemente de acordo com a medida $P_C(c|i)$.A probabilidade conjunta é portanto o produto dos termos. □

Teorema 2.1. *Em todos os cenários, independentemente da configuração S_N, ou seja, de como K e n_k com $k = 1...K$ são selecionados, $\widehat{u}_S N$ é um estimador imparcial de μ*

Demonstração. A prova é dada apenas por uma questão de integridade:

$$\int \widehat{\mu}_{S_N} = \int \frac{1}{N} \sum_{k=1}^{K} \sum_{j \in J_k} c_j \prod_{k=1}^{K} dP_I(i_k) \prod_{j \in J_k} dP_C(c_j|i_k) = \frac{1}{N} \sum_{k=1}^{K} \sum_{j \in J_k} \int c_j dP_C(c_j|i_k) dP_I(i_k) = \mu \tag{2.12}$$

□

Em particular, $\widehat{\mu}_{H_{1|1}}$, baseado em uma única execução sobre uma única instância, é uma estimador imparcial de μ, independentemente de qual instância é considerada, desde que seja selecionada de acordo com a probabilidade desconhecida $P_I(i)$. Da mesma forma, o estimador $\widehat{\mu}_{H_{1|N}}$ baseado em N execuções em uma única instância é imparcial, assim como $\widehat{\mu}_{H_{N/10|10}}$ que considera $N/10$ instâncias, sendo 10 execuções por instância.

2.4.2 Análise de segunda ordem do estimador

Todos os estimadores que podem ser escritos como a Equação 2.6 são, portanto, equivalentes no que diz respeito ao comportamento esperado. No entanto, eles diferem quanto às estatísticas de segunda ordem. Tem-se, portanto, interesse em encontrar o melhor estimador de variância mínima quando o número total N de experimentos é fixado. Em outras palavras, pretende-se responder à questão: *Se eu executar 100 experimentos, deve-se considerar (i) 1 instância e 100 execuções; (ii) 10 instâncias e 10 execuções para cada; (iii) 100 instâncias e 1 execução para cada ?*

Lema 2.2. *Em um dado cenário $P(c,i) = P_C(c|i)P_I(i)$ e para uma dada configuração experimental $S_N = (K, n_1, n_2, ..., n_K)$, a variança do estimador $\widehat{\mu}_{S_N}$ é dado por:*

$$\int (\widehat{\mu}_{S_N} - \mu)^2 dP(\widehat{\mu}_{S_N}) = \frac{1}{N} \overline{\sigma}^2_{WI} + \frac{\Sigma_{k=1}^{K} n_k^2}{N^2} \sigma^2_{AI}$$

Demonstração.

$$\int \widehat{\mu}_{S_N} dP(\widehat{\mu}_{S_N}) = \int (\frac{1}{N} \sum_{k=1}^{K} \sum_{j \in J_k} c_j - \mu) \prod_{k=1}^{K} dP_I(i_k) \prod_{j \in j_k} dP_C(c_j|i_k) = \tag{2.13}$$

$$= \int (\frac{1}{N} \sum_{k=1}^{K} \sum_{j \in J_k} (c_j - \mu_{i_k} + \mu_{i_k} - \mu))^2 \prod_{k=1}^{K} dP_I(i_k) \prod_{j \in j_k} dP_C(c_j|i_k) = \tag{2.14}$$

$$\frac{1}{N^2} \sum_{k,k'=1}^{K} \sum_{j \in J_k, j' \in J'_k} \int (c_j - \mu_{i_k})(c_{j'} - \mu_{i_{k'}}) dP_C(c_j|i_k) dP_I(i_k) dP_C(c_{j'}|i_{k'}) dP_I(i_{k'}) + \tag{2.15}$$

$$+ \frac{1}{N^2} \sum_{k,k'=1}^{K} \sum_{j \in J_k, j' \in J'_k} \int (\mu_{i_k} - \mu)(\mu_{i_{k'}} - \mu) dP_I(i_k) dP_I(i_{k'}) + \tag{2.16}$$

$$+ \frac{1}{N^2} \sum_{k,k'=1}^{K} \sum_{j \in J_k, j' \in J'_k} 2 \int (c_j - \mu_{i_k})(\mu_{i_{k'}} - \mu) dP_C(c_j|i_k) dP_I(i_k) dP_I(i_{k'}) \tag{2.17}$$

Resolvendo a etapa (2.15) acima tem-se, para $k \neq k'_k$:

$$\int (c_j - \mu_{i_k})(c_{j'} - \mu_{i_{k'}}) dP_C(c_j|i_k) dP_I(i_k) dP_C(c_{j'}|i_{k'}) dP_I(i_{k'}) =$$
$$= \int (c_j - \mu_{i_k}) dP_C(c_j|i_k) dP_I(i_k) \int (c_{j'} - \mu_{i_{k'}}) dP_C(c_{j'}|i_{k'}) dP_I(i_{k'}) = 0$$

Similarmente, se $k = k'$, mas $j = j'$, tem-se o resultado para a primeira parte igual:

$$\int (\int (c_j - \mu_{i_k}) dP_C(c_j|i_k) \int (c_{j'} - \mu_{i_{k'}}) dP_C(c_{j'}|i_{k'})) dP_I(i_k) = 0$$

Se $k = k'$ e $j = j'$, tem-se:

$$\int (c_j - \mu_{i_k})(c_{j'} - \mu_{i_{k'}}) dP_C(c_j|i_k) dP_I(i_k) dP_C(c_{j'}|i_{k'}) dP_I(i_{k'}) =$$
$$= \int (c_j - \mu_{i_k})^2 dP_C(c_j|i_k) dP_I(i_k)$$

Logo, a (2.15) resulta em :

$$\frac{1}{N^2} \sum_{k,k'=1}^{K} \sum_{j \in J_k, j'} \int (c_j - \mu_{i_k})^2 dP_C(c_j|i_k) dP_I(i_k)$$

Na etapa (2.16), uma vez que a integral é independente de j e j', tem-se:

$$\frac{1}{N^2} \sum_{k,k'=1}^{K} \sum_{j \in J_k, j' \in J'_k} \int (\mu_{i_k} - \mu)(\mu_{i_{k'}} - \mu) dP_I(i_k) dP_I(i_{k'}) =$$
$$= \frac{1}{N^2} \sum_{k,k'=1}^{K} |J_k| \cdot |J_{k'}| \int (\mu_{i_k} - \mu)(\mu_{i_{k'}} - \mu) dP_I(i_k) dP_I(i_{k'})$$

Se $k \neq k^{'}$, resulta:

$$\int (\mu_{i_k} - \mu)(\mu_{i_{k'}} - \mu) dP_I(i_{k'} = 0$$

Se Se $k = k^{'}$, resulta:

$$\int (\mu_{i_k} - \mu)(\mu_{i_{k'}} - \mu) dP_I(i_{k'} = \int (\mu_{i_k} - \mu)^2 dP_I(i_k)$$

Assim, 2.16 equivale a

$$\frac{1}{N^2} \sum_{k=1}^{K} |J_k|^2 \int (\mu_{i_k} - \mu)^2 dP_I(i_k).$$

Adicionado (2.17):

$$\int (c_j - \mu_{i_k})(\mu_{i_{k'}} - \mu) dP_C(c_j|i_k) dP_I(i_k) dP_I(i_{k'}) = 0$$

Isso resulta em :

$$\int (\widehat{\mu}_{S_N} - \mu)^2 dP(\widehat{\mu}_{S_N}) = \frac{1}{N^2} \sum_{k=1}^{K} \sum_{j \in J_k} \int (c_j - \mu_{i_k})^2 dP_C(c_j|i_k) dP_I(i_k) + \\ + \frac{1}{N^2} \sum_{k=1}^{K} |J_k|^2 \int (\mu_{i_k} - \mu)^2 dP_I(i_k)$$

Segundo as definições 2.7 e 2.8, pode-se escrever :

$$\int (\widehat{\mu}_{S_N} - \mu)^2 dP(\widehat{\mu}_{S_N}) = \frac{1}{N^2} \sum_{k=1}^{K} |J_k| \overline{\sigma}^2_{WI} + \frac{1}{N^2} \sum k = 1^K |J_k|^2 \sigma^2_{AI} \tag{2.18}$$

Dado que $\sum_{k=1}^{K} |J_k| = N$ e que $|J_k| = n_k$, tem-se:

$$\int (\widehat{\mu}_{S_N} - \mu)^2 dP(\widehat{\mu}_{S_N}) = \frac{1}{N} \overline{\sigma}^2_{WI} + \frac{\sum_{k=1}^{K} n_k^2}{N^2} \sigma^2_{AI} \tag{2.19}$$

□

Voltando à pergunta principal: Com as restrição de que o número de execução deve ser N, qual o número ótimo de instâncias para considerar e quantas execuções devem ser realizadas em cada uma?

Teorema 2.2. *A variança de $\widehat{\mu}_{S_N}$ é minimizada pela configuração experimental $\overline{S}_N = H_{N|1}$, ou seja, N instâncias com* 1 *execução por instância.*

Demonstração. De acordo com Lema 2.2, a variança de $\widehat{\mu}_{S_N}$ é dada por:

$$\int (\widehat{\mu}_{S_N} - \mu)^2 dP(\widehat{\mu}_{S_N}) = \frac{1}{N}\overline{\sigma}^2_{WI} + \frac{\sum_{k=1}^{K} n_k^2}{N^2}\sigma^2_{AI} \tag{2.20}$$

Como o primeiro termo não depende de S_N e dado que N é fixado e σ^2_{AI} não pode ser controlado, pode-se focar na minimização de $\sum_{k=1}^{K} n_k^2$ com a restrição de $\sum_{k=1}^{K} n_k = N$. □

Colorário 2.1. *A variança do melhor estimador $\widehat{\mu}_{H_{N|1}}$ é:*

$$E[(\widehat{\mu}_{H_{N|1}} - \mu)^2] = \frac{1}{N}(\overline{\sigma}^2_{WI} + \sigma^2_{AI})$$

Demonstração. Deduzível a partir do Lema 2.2 □

Colorário 2.2. *$\widehat{\mu}_{H_{N|1}}$ é um estimador consistente de μ, isto é, a probabilidade converge para μ:*

$$\lim_{N\to\infty} \quad Prob|\widehat{\mu}_{H_{N|1}} - \mu| > \varepsilon = 0, \quad \forall \varepsilon > 0$$

Demonstração. A prova deriva diretamente do Corolário 2.1. De fato, $\widehat{\mu}_{H_{N|1}}$ converge para μ no sentido do quadrado médio: Desde que $\overline{\sigma}^2_{WI}$ e σ^2_{AI} são finitos, como N tende ao infinito, $E[(\widehat{\mu}_{N|1} - \mu)^2]$ converge para zero. A afirmação segue, uma vez que a convergência no quadrado médio implica convergência na probabilidade (Papoulis; Pillai, 2002)

□

É interessante considerar um exemplo numérico explicado por Birattari (2009b) que compara o melhor estimador $\widehat{\mu}_{H_{N|1}}$ com outro possível estimador μ. Assuma que o número total de execuções é fixado em $N = 100$, a variança do estimador é obtida nos seguintes experimentos: (i) N instâncias com 1 execução por instância, (ii) 10 instâncias com 10 execuções por instância, (iii) 1 instância com 100 execuções. A partir do Lema 2.2, tem-se:

Experimento (i): 100 instâncias, 1 execução por instância

$$E[(\widehat{\mu}_{H_{100|1}} - \mu)^2] = \frac{1}{100}\overline{\sigma}^2_{WI} + \frac{1}{100}\sigma^2_{AI}.$$

Experimento (ii): 10 instâncias, 10 execuções por instância.

$$E[(\widehat{\mu}_{H_{10|10}} - \mu)^2] = \frac{1}{100}\overline{\sigma}^2_{WI} + \frac{1}{10}\sigma^2_{AI}$$

Experimento (iii): 1 instância, 100 execuções por instância.

$$E[(\widehat{\mu}_{H_{1|100}} - \mu)^2] = \frac{1}{100}\overline{\sigma}^2_{WI} + \sigma^2_{AI}$$

Enquanto os três experimento agem da mesma maneira sobre o coeficiente do primeiro termo, surge uma diferença quanto ao coeficiente do segundo termo: os experimentos 2 e 3 não conseguem reduzir de forma eficiente a contribuição da variância de interferência. A variança produzida pelas três configurações é igual somente no caso trivial em que a variância entre instâncias é nula, ou seja, quando todas as instâncias compartilham o mesmo custo esperado $\mu = \mu_i, \forall i \in I$.

Embora o estimador $\widehat{\mu}_{H_{10|10}}$ considerado no experimento 2 seja menos eficiente que o melhor $\widehat{\mu}_{H_{100|1}}$ considerado na configuração 1, o mesmo também é consistente. Por outro lado, o estimador $\widehat{\mu}_{H_{1|100}}$ do experimento 3 não é consistente para o caso trivial em que $\sigma^2_{AI} = 0$.

Também deve notar-se que não existe nenhum cenário no qual o estimador $\widehat{\mu}_{H_{N|1}}$ produz uma variação maior que qualquer estimador $\widehat{\mu}_{S_N}$, Ou seja, não existe uma melhor configuração que *N instâncias, uma execução por instância* de forma independente das medidas P_I e P_C .

Colorário 2.3. *A variância do custo c obtido pela configuração dada θ em toda a classe I de instâncias pode ser decomposta em dois termos, a expectativa de variação dentro da instância e variância entre instâncias:*

$$\sigma^2 = E[(c - \mu)^2] = \overline{\sigma}^2_{WI} + \sigma^2_{AI}$$

Demonstração. O resultado segue imediatamente o Corolário 2.1, uma vez percebido que a variância do custo c é igual a variância de um estimador $\mu_{H_{1|1}} = c$, baseado em uma única amostra. □

A variação esperada dentro da instância $\overline{\sigma}_{WI}$ mede o quão diferente pode ser os custos c obtidos pela configuração θ da metaheurística em diferentes execuções na mesma instância; Essa quantidade é calculada de forma média em todas as instâncias em I. Por outro lado, a variação entre instâncias σ^2_{AI} mede o quão diferente as instâncias são uma da outra para o que diz respeito ao valor esperado do custo obtido pela configuração dada θ.

2.5 Técnicas em Sintonização de parâmetros para Metaheurísticas

Como dito anteriormente, se houvesse uma configuração ideal para um algoritmo de otimização, convergindo para o melhor em qualquer problema de otimização, a discussão sobre

o Ajuste de Parâmetros seria sem sentido. Infelizmente, o teorema de otimização da inexistência de almoço grátis revela que não existe um melhor algoritmo global para todos os problemas, pelo menos em teoria (WOLPERT; MACREADY, 1997). Uma das consequências dessa afirmação, tendo um grande impacto no ajuste de parâmetros como um todo, é o fato de que não há melhor configuração de parâmetro inicial para uma metaheurística para todos os problemas de otimização. Portanto, as configurações ideais para os parâmetros iniciais podem variar significativamente de problema para problema.

Dentro dessa diversidade de possibilidades, pode-se imaginar várias formas de prover um ajuste para uma metaheurística. Uma abordagem por força bruta é uma maneira bem simples de fazer isso. Se o tempo total disponível para sintonizar é T e cada experimento único é executado por um tempo t, o número total de experimentos que podem ser realizados é $M = T/t$. Na abordagem por força bruta, o poder computacional é uniformemente alocado para os diferentes candidatos em Θ, cada um deles é portanto, testado $N = M/\theta$ vezes para obter os estimadores $\widehat{\mu}(\theta_1), \widehat{\mu}(\theta_2), \widehat{\mu}(\theta_{|\Theta|})$ de performance esperada na classe I de instâncias. A alocação ótima dos N experimentos realizados em cada configuração consiste em executar cada configuração uma vez em N instâncias diferentes amostradas de acordo com a medida desconhecida P_I.

Pode-se definir um algoritmo por força bruta em que as N instâncias são aleatoriamente amostradas de acordo com a medida P_I. Todas as configurações candidatas são testadas uma vez nas instâncias selecionadas. Com base nos resultados observados, para cada candidato θ, um estimador $\widehat{\mu}(\theta)$ da performance esperada $\mu(\theta)$ sob a classe I com respeito à medida P_I é computada. O candidato $\tilde{\theta}$, considerado o de melhor performance, é selecionado. O algoritmo 1 descreve o funcionamento do processo.

Algoritmo 1: Força Bruta

```
Input: M
N = floor(M/|Θ|);
A = alocaVetor(|Θ|);
for k ← 1 to N do
    i ← escolheInstancia(i);
    for θ ∈ Θ do
        s = executa(θ, i);
        c = avalia(s);
        A[θ] = updateMedia(A[θ], c, k)
    θ̄ = min(A);
    return θ̄;
```

Nota-se que os requisitos de memória de uma abordagem por força bruta são particularmente limitados. Como a única informação que se deve reter sobre cada configuração de candidato é apenas uma estimativa de seu desempenho esperado - isto é, a média empírica dos resultados observados em N instâncias diferentes - e uma vez que a média dos valores N pode ser calculada de forma incremental, é necessário simplesmente armazenar a estimativa atual para

cada candidato. Uma matriz de comprimento $|\Theta|$ contém, portanto, toda a informação sobre a computação em andamento.

No contexto dos Algoritmos Evolutivos (EA), o ajuste de parâmetros é uma abordagem comumente praticada que equivale a encontrar bons valores para os parâmetros antes da execução do algoritmo e depois executar o algoritmo usando esses valores, que permanecem fixos durante a execução (EIBEN; HINTERDING; MICHALEWICZ, 1999).

Eiben, Hinterding e Michalewicz (1999) definem ainda o controle de parâmetros como uma alternativa ao ajuste de parâmetros, onde os valores dos parâmetros iniciais estão mudando durante a execução do algoritmo. O controle de parâmetro é particularmente relevante quando se trata de técnicas de pesquisa locais, como Pesquisa Tabu (TS), Mapas Auto-organizáveis (SOM) ou Recozimento Simulado (SA), onde a alteração dinâmica dos parâmetros de controle é incorporada ao design algorítmico . Mesmo que o controle de parâmetro seja um problema mais complexo, as diferenças são um pouco confusas. A maior parte do comportamento dinâmico de um algoritmo pode ser extraída para chegar os parâmetros iniciais do algoritmo.

O foco deste trabalho está no ajuste de parâmetros. A razão para isto é que muitas contribuições interessantes foram publicadas recentemente somente dentro do campo de ajuste de parâmetros.

O objetivo da metaheurística é encontrar boas soluções em tempo razoável. O envolvimento de aleatoriedade e a inspiração baseada em fenômenos biológicos ou físicos conduzem a um comportamento algorítmico que é difícil de rastrear e analisar para tirar conclusões sólidas e gerais sobre as propriedades dos algoritmos. A convergência, por exemplo, é difícil de provar pois não é possível prever as interações dos parâmetros antes da execução de uma metaheurística em um problema de otimização sem tentar uma abordagem experimental para avaliar a qualidade dos ajustes.

Existem duas distinções entre os métodos de ajuste de parâmetros automatizados: modelo livre e baseados em modelo (HUTTER; HOOS; LEYTON-BROWN, 2011). A principal diferença entre as duas é que as abordagens baseadas em modelo constroem um modelo interpretando a relação entre o algoritmo e seus valores de parâmetro das observações obtidas executando o algoritmo de destino com várias configurações enquanto na abordagem de modelo livre isso não ocorre.

2.5.1 Abordagem baseada em modelo livre

Sintonizadores de modelo livre tiram conclusões implícitas com base em regras heurísticas. Escolhas para vetores de parâmetros interessantes a serem investigados são frequentemente guiadas por aleatoriedade ou um projeto experimental simples. Geralmente eles são mais leves e mais rápidos na execução do que os sintonizadores baseados em modelo. Como desvantagem, os sintonizadores de modelo livre têm um potencial de extrapolação muito limitado.

Encontrar bons valores de parâmetro para problema de otimização combinatória é altamente complexo. Isso pode ser atribuído ao fato de que nenhuma suposição sobre a paisagem do problema (dimensionalidade, multimodalidade, interações de parâmetro) pode ser feita. Para este tipo de problemas, um método baseado em população bem conhecido, Algoritmo Evolutivo, é comumente aplicado. Como os EAs são metaheurísticas, sua utilização como sintonizadores pode ser vista como ineficiente uma vez que os mesmos têm parâmetros para serem sintonizados. No entanto, eles mostram eficácia ao melhorar o desempenho do algoritmo de destino.

Meta EAs, como introduzido pela primeira vez em Mercer e Sampson (1978), são EAs com o objetivo de otimizar metaheurísticas. O genes dos EAs codifica os parâmetros dos otimizadores. Eles são chamados de meta EAs, já que sua atividade pode ser considerada meta-otimização. A abordagem evolutiva intensifica boas regiões de parâmetros e substitui as más. Todos os EAs que podem lidar com a codificação de valor real podem ser usados como meta EA para parâmetros numéricos. Uma escolha muito comum entre os EAs para essa tarefa é o CMA-ES (Estratégias Evolutivas de Adaptação Matricial de Covariância (HANSEN, 2006)); ES provou ser um bom otimizador numérico. Um meta EA relevante, o Meta-GA, foi apresentado em Grefenstette (1986).

2.5.1.1 Revac

Nannen e Eiben (2006) apresentou a Estimativa de Relevância e Calibração de Valor de Parâmetros (REVAC) que estima a distribuição de vetores de parâmetros promissores por meio da teoria da informação. REVAC pode ser classificado como um Algoritmo de Estimativa de Distribuição (EDA) e está limitado a domínios de parâmetros contínuos.

2.5.1.2 ParamILS

ParamILS é uma descrição algorítmica abstrata, apresentada em Hutter *et al.* (2009b). Os autores apresentam duas implementações: BasicILS e FocussedILS. O BasicILS compara estimativas simples para as estatísticas de custo de execuções subsequentes, enquanto FocussedILS tenta superar o excesso de confiança baseado nas instâncias de treinamento de forma adaptativa, escolhendo o número de instâncias de treinamento a serem usadas para a avaliação de cada configuração de parâmetros (HUTTER *et al.*, 2009b). Ambos utilizam técnicas de busca local iterativa (ILS) para orientar a busca por áreas promissoras no espaço de busca. Em suma, ambos os algoritmos começam com uma configuração de parâmetro inicial definida pelo usuário. Sequencialmente, as combinações são avaliadas quanto ao desempenho, e a pesquisa local é conduzida até que um limiar de adequação, ou um limite iminente, seja atingido. Para cada nova experiência, apenas as alterações de parâmetros individuais são realizadas. ParamILS requer que o usuário discretize todos os intervalos de parâmetros.

2.5.1.3 Algoritmo genético baseado em gênero

Ansótegui, Sellmann e Tierney (2009) apresentaram o Algoritmo genético baseado em gênero (GGA). Segundo os autores, a escolha do parceiro provavelmente terá um alto impacto na qualidade dos resultados do que, por exemplo, a seleção natural. Diferentes tipos de pressão de seleção são utilizados para os dois gêneros. Ao avaliar apenas um dos gêneros, é relatado a diminuição de metade do tempo de processamento, enquanto produzem uma perda insignificante no desempenho.

O algoritmo permite que o usuário insira árvores de variáveis ou parâmetros para especificar o parentesco, se conhecido antecipadamente. Essas árvores servem como estrutura do genes dos indivíduos. O método pode lidar com variáveis contínuas e inteiras. Os autores mencionam variáveis categóricas, mas o algoritmo, em sua versão atual, não as suporta. Um conjunto de treinamento de instâncias de problemas pode ser especificado para o algoritmo. O GGA seleciona aleatoriamente os subconjuntos desses e executa os indivíduos (vetores de parâmetros) em cada iteração entre si. A implementação faz uso de cálculos paralelos.

2.5.2 Abordagem baseada em modelos

O objetivo essencial para sintonizadores baseados em modelo é determinar pontos de amostra interessantes a serem investigados, melhorando o modelo. Uma ferramenta comum é a interpretação gráfica dos vetores de parâmetros com relação à adequação do alvo atingido. As técnicas relacionadas são referidas como métodos de superfície de resposta (RSM); o termo genérico para esse tipo de otimização é a modelagem substituta. Em outras palavras, as abordagens baseadas em modelos constroem modelos com a capacidade de :

- Interpolar para a escolha de novas configurações de parâmetro a serem investigadas
- Extrapolar (recomendar) bons vetores de parâmetros para novos problemas ou instâncias de problemas.

Os métodos de otimização de parâmetros baseados em modelo têm suas raízes em projeto experimental tradicional e otimização global de caixa preta. Alguns exemplos de abordagens baseadas em modelo são citadas nas subseções seguintes.

2.5.2.1 EGO

Em Jones, Schonlau e Welch (1998), os autores propõem um algoritmo para a criação de modelo de superfície esponjosa para funções de caixa preta. Este algoritmo faz uso de Design e Análise de Experimentos Computacionais (SACKS *et al.*, 1989) para a diversificação e uma fase baseada em *Branch and Bound* para intensificação das áreas mais promissoras. O algoritmo é chamado *Efficient Global Optimization* (EGO).

Os autores demonstram a sua qualidade em aproximar o panorama do problema da função Branin, considerando apenas alguns poucos pontos de amostragem. A pista com este método é que ele usa a chamada função de melhoria esperada com o objetivo de melhorar globalmente o modelo. O cálculo dessa função é barato e leva a sugestões proficientes para pontos de amostra adicionais e promissores a serem avaliados.

O EGO é restrito a algoritmos ou simulações determinísticas. Atraves da investigação minuciosa da modelagem substituta foi elaborado uma regra geral: para receber um bom modelo, a quantidade de pontos de amostra deve ser dez vezes maior que a quantidade de dimensões do espaço do problema.

2.5.2.2 Abordagem por Corrida

Os algoritmos de corrida foram projetados para fornecer uma melhor alocação de recursos computacionais entre as configurações candidatas e, portanto, superior ao método de força bruta. Ao mesmo tempo, a corrida, indiretamente permite uma solução limpa para problemas de corrigir o número de instâncias e o número de execuções a ser considerado.

Para fazer isso, os algoritmos de corrida avaliam sequencialmente configurações candidatas e descartam as de pior qualidade logo que evidências estatísticas suficientes são reunidas contra os mesmos. A eliminação de candidatos inferiores acelera o procedimento e permite avaliar as configurações promissoras em mais instâncias, assim, obter estimativas mais confiáveis do seu comportamento.

Como exemplo, dado que uma sequência randômica de instâncias de treino i, em que o termo i_k pertencente a I de acordo com a medida P_I é escolhido. $c^k(\theta|i)$ representa o vetor de k termos em que cada elemento $c(\theta|i_l)$ é a melhor solução encontrada para a configuração θ executando sobre a instância i_l no tempo $t(i_l)$. Por definição, o vetor c^k pode ser obtido pelo vetor c^{k-1} acrescentando ao vetor o último custo. O objetivo é que a cada passo da corrida, o espaço de configurações a serem avaliadas Θ diminua até que todas as configurações sejam descartadas ou quando um tempo T predefinido for alcançado. Um algoritmo de corrida genérico pode ser descrito pelo pseudocódigo a seguir:

A abordagem por corrida que utiliza o teste estatístico *Friedman two-way analysis of variance ranks* é denominado F-Race (CONOVER; CONOVER, 1980). Uma vez que o algoritmo atingiu o passo k e $n = |\Theta_{k-1}|$ configurações ainda estão no processo de corrida, o teste de *Friedman* assume que o custo observado são as variáveis aleatórias $c^k(\theta_1, i_l), c^k(\theta_2, i_l), ..., c^k(\theta_n, i_l)$ denominadas blocos (DEAN; VOSS, 1999), em que cada bloco corresponde ao resultado computacional na instância i_l para cada configuração na corrida no passo k. Dentro de cada bloco as quantidades $c^k(\theta|i_l)$ são enfileiradas da menor para a maior e para cada configuração $\theta_j \in \Theta_{k-1}$ e $R_{l,j}$ é o rank de θ_j dentro do bloco l, $R_j = \sum_{l=1}^{k} R_{l,j}$ a soma dos *ranks* sobre todas instâncias i_l com $1 \leq l \leq k$. O teste de *Friedman* considera a seguinte estatística (CONOVER; CONOVER,

Algoritmo 2: Algoritmo de Corrida genérico

Input: M

$numExperimentos \leftarrow 0$;
$numInstancias \leftarrow 0$;
$C \leftarrow alocarMatrix(tam(base), \Theta)$;
$sobreviventes \leftarrow \Theta$;
repeat
  $i \leftarrow escolheInstancia(base)$;
  $numInstancias \leftarrow numInstancias + 1$;
  for $\theta \in sobreviventes$ **do**
    $s \leftarrow executa(\theta, i)$;
    $numExperimentos \leftarrow numExperimentos + 1$;
    $C[numInstancias, \theta \leftarrow avalia(s)$;
  $sobreviventes \leftarrow eliminaCandidatos(sobreviventes, C, Friedman)$;
until *numExperimentos+|sobreviventes|>M e numInstancias+1>tam(base)*;
$\overline{\theta} \leftarrow selecionaMelhores(sobreviventes, C)$;
return $\overline{\theta}$;

1980):

$$T = \frac{(n-1)\sum_{j=1}^{n}(R_j - \frac{k(n+1)}{2})^2}{\sum_{l=1}^{k}\sum_{j=1}^{n}R_{l,j}^2 - \frac{kn(n+1)^2}{4}} \tag{2.21}$$

Sob a hipótese nula de que todos os rankings possíveis dos candidatos dentro de cada bloco são igualmente prováveis, T é X^2 distribuído com $n-1$ graus de liberdade. Se o T observado exceder o $(1-\alpha)$ de tal distribuição, o nulo é rejeitado, no nível aproximado, em favor da hipótese de que pelo menos um candidato tende a produzir um melhor desempenho que pelo menos um outro.

Se o nulo for rejeitado, realizam-se comparações entre pares de candidatos individuais. Os candidatos θ_j e θ_h são considerados diferentes se:

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k(1-\frac{T}{k(n-1)})(\sum_{l=1}^{k}\sum_{j=1}^{n}R_{l,j}^2 - \frac{kn(n+1)^2}{4})}{(k-1)(n-1)}}} > t_{1-\alpha/2}$$

onde $t_{1-\alpha/2}$ é o quantil $1-\alpha/2$ da distribuição t de Student (CONOVER; CONOVER, 1980).

No F-Race, se no passo k o nulo da comparação agregada não é rejeitado, todos os candidatos em Θ_{k-1} passam para Θ_k. Por outro lado, se o nulo for rejeitado, as comparaçsubsubões em pares são executadas entre o melhor candidato e os outros. Todos os candidatos que possuem qualidade significativamente pior do que os melhores são descartados e não aparecerão no Θ_k.

2.5.2.3 Abordagem por Metodologia de Superfície de Resposta

A metodologia de superfície de resposta (RSM) é uma coleção de técnicas matemáticas e estatísticas para construção de modelos empíricos. Através de planejamento de experimentos, o objetivo é otimizar a resposta (variável de saída) que é influenciada por variáveis independentes (variáveis de entrada). Um experimento é uma série de testes em que mudanças são feitas nas variáveis de entrada para identificar as razões de mudanças nas variáveis de saída.

Originalmente, RSM foi desenvolvido para modelos experimentais de resposta (BOX; DRAPER, 1987) e então migrado para a modelagem de experimentos numéricos. A diferença das duas propostas está no tipo de erro gerado pela resposta. Em experimentos físicos, a imprecisão pode ser devida, por exemplo, a erros de medição, enquanto, em experimentos computacionais, o ruído numérico é resultado da convergência incompleta de processos iterativos, erros de arredondamento ou a representação discreta de fenômenos físicos contínuos. No RSM , os erros são aleatórios.

A aplicação do RSM para a otimização visa reduzir o custo de métodos de análise caros (por exemplo, método de elementos finitos ou análise CFD) e seus ruídos numéricos associados. O problema pode ser aproximado com suavização de funções que melhoram a convergência da otimização reduzindo os efeitos do ruído e permitindo o uso de algoritmos baseados em derivadas. Venter, Haftka e Starnes Jr (1996) discutiram as vantagens da utilização de RSM para aplicações de otimização em projetos.

Por exemplo, no caso da otimização da calcinação de cimento romano, o engenheiro quer encontrar os níveis de temperatura x_1 e tempo x_2 que maximizam a força de idade precoce y do cimento. A força da idade precoce é uma função dos níveis de temperatura e tempo, da seguinte forma:

$$y = f(x_1, x_2) + \varepsilon \tag{2.22}$$

em que ε representa o ruído ou erro observado na resposta y. A superfície representada por $f(x_1, x_2)$ é denominada superfície de resposta.

A resposta pode ser representada graficamente, seja no espaço tridimensional ou como tramas de contorno que ajudam a visualizar a forma da superfície de resposta. Os contornos são curvas de resposta constantes desenhadas no plano x_i, x_j, mantendo todos as outras variáveis fixadas. Cada contorno corresponde a uma altura particular da superfície de resposta, como mostrado na Figura 1.

2.5.2.4 Função Modelo Aproximada

Geralmente, a estrutura do relacionamento entre a variável de resposta e as variáveis independentes são desconhecidas. O primeiro passo no RSM é encontrar a aproximação adequada para o relacionamento. As formas mais comuns são polinômios de baixa ordem (primeira ou

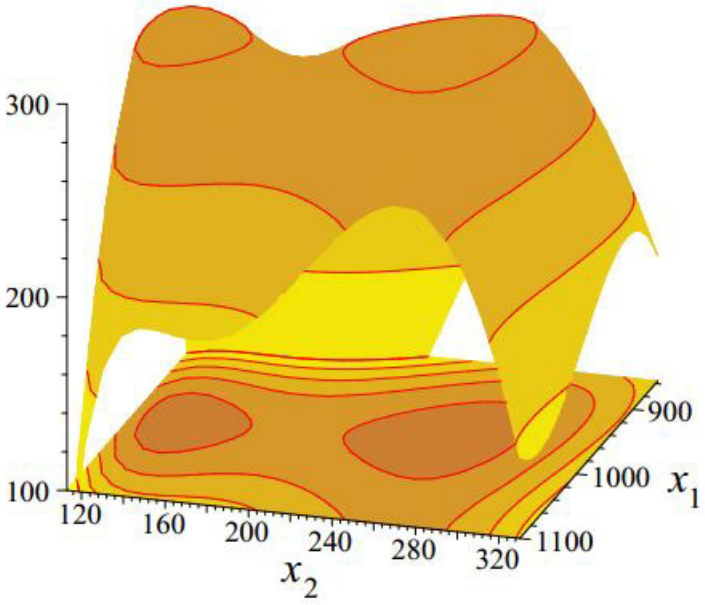

Figura 1 – Superfície de resposta tridimensional e o contorno correspondente para a força da idade inicial do cimento romano onde x_1 é a subtemperatura de calcinação (C) e x_2 é o tempo de permanência (min.)

Fonte: (ALVAREZ, 2000)

segunda ordem), Além disso, a complexidade da função não está limitada a um polinômio, mas pode ser generalizada com a inclusão de qualquer operador matemático (funções trigonométricas, etc). dependendo da compreensão de engenharia do problema. Os coeficientes de regressão incluídos no modelo de aproximação são chamados parâmetros de ajuste e são estimados minimizando a soma dos quadrados do erros (BOX; DRAPER, 1987):

$$G(a) = \sum_{p=1}^{P} (w_p(F_p - \tilde{F}_p(a))^2) \tag{2.23}$$

onde w_p é um coeficiente de peso que caracteriza a contribuição relativa da informação de uma função original nos pontos $p, p = 1, ..., P$. A construção de modelos de superfície de resposta é um processo iterativo. Uma vez que o modelo aproximado é obtido, um modelo estatístico determina se a solução é satisfatória.

Para reduzir o número de análises em simulações computacionais, sensibilidade de dados pode ser usadas na construção do modelo, embora esta informação nem sempre esteja disponível com um baixo custo. Se além dos valores da função original $F_p = F(x_p)$ sua derivada de primeira ordem no ponto p é $F_{p,i} = \frac{\partial F_p}{\partial x_i}$, com $(i = 1, ..., N, p = 1, ..., P)$ são conhecidos, a Equação (2.23) pode ser substituído por (TOROPOV; FILATOV; POLYNKIN, 1993):

$$G(a) = \sum_{p=1}^{P} (w_p((F_p - \tilde{F}_p(a))^2 + \gamma \frac{\sum_{i=1}^{N} (F_p - \tilde{F}_p(a))^2}{\sum_{i=1}^{N} F_{p,i}^2})) \tag{2.24}$$

onde $\gamma > 0$ é o parâmetro que caracteriza um grau de desigualdade da contribuição da resposta e

dos dados de sensibilidade.

2.5.2.5 Projeto de Experimentos

Um importante aspecto da RSM é o design ou projeto de experimentos (BOX; DRAPER, 1987), usualmente abreviado como *DoE*. Essas estratégias foram originalmente desenvolvidas para modelagem de experimentos físicos, mas também pode ser aplicado em experimentos numéricos. O objetivo do DoE é a seleção dos pontos onde a resposta deve ser avaliada.

A maioria dos critérios para o design ideal de experimentos estão associados com o modelo matemático do processo. Geralmente, esses modelos matemáticos são polinômios com uma estrutura desconhecida, de modo que as experiências correspondentes são concebidas apenas para cada problema particular. A escolha do DoE pode ter uma larga influência na acurácia da aproximação e no custo da construção da superfície de resposta.

Em um DoE tradicional, as experiências de triagem são realizadas nos estágios iniciais do processo, quando é provável que muitas das variáveis de projeto inicialmente consideradas tenham pouco ou nenhum efeito sobre a resposta. O objetivo é identificar variáveis que têm grandes efeitos para uma investigação mais aprofundada.

Uma descrição detalhada sobre DoE pode ser encontrada em (BOX; DRAPER, 1987), (MONTGOMERY, 1997), (SACKS *et al.*, 1989), entre outros.

2.5.2.6 Experimento Fatorial

Para construir um modelo de aproximação que capture interações entre as N variáveis, uma aproximação baseada em experimento fatorial completo (MONTGOMERY, 1997) pode ser utilizada para investigar todas as possíveis combinações. Um experimento fatorial é uma estratégia experimental em que variáveis do projeto variam em conjunto em vez de uma de cada vez.

Os limites inferior e superior de cada uma das N variáveis de projeto no problema de otimização precisam ser definidos. O intervalo permitido é então discretizado em diferentes níveis, se cada uma das variáveis for definida apenas nos limites inferior e superior (dois níveis), o experimento é denominado 2^N Fatorial Completo. Da mesma forma, se os pontos médios estiverem incluídos, o experimento é denominado 3^N Fatorial Completo como mostra a Figura 2.

Experimentos fatoriais podem ser usados para construir modelos de segunda ordem. Um modelo de segunda ordem pode melhorar significativamente o processo de otimização quando um modelo de primeira ordem sofre falta de ajuste devido à interação entre variáveis e curvatura da superfície. Um modelo geral de segunda ordem é definido como:

$$y = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} a_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{i=1}^{n} a_{ij} x_i x_j, i < j \tag{2.25}$$

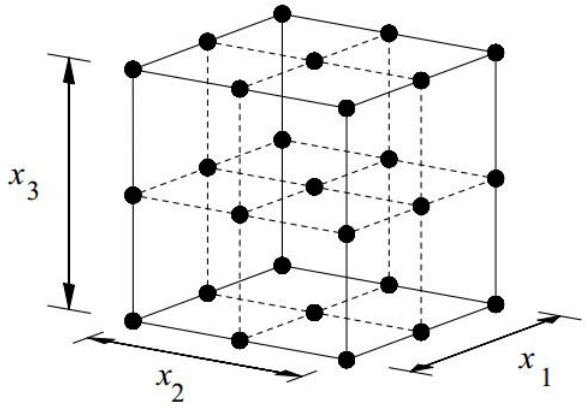

Figura 2 – Um experimento 3^3 Fatorial Completo (27 pontos)

Fonte: (MONTGOMERY, 1997)

onde x_i e x_j são variáveis de projeto e a são parâmetros de sintonização.

A construção de um modelo de superfície de resposta quadrática em variáveis N requer o estudo em três níveis para que os parâmetros de ajuste possam ser estimados. Portanto, pelo menos $(N+1)(N+2)/2$ avaliações de função são necessárias. Geralmente, para um número de variáveis grande, o número de experimentos cresce exponencialmente (3^N para um fatorial completo) e torna-se impraticável, logo, o fatorial completo é utilizado para cinco variáveis ou menos.

Se o número de variáveis de projeto se tornar grande, uma fração de um fatorial completo pode ser usada ao custo de estimar apenas algumas combinações entre variáveis. Essa técnica é denominada experimento fatorial fracionado e geralmente é usada para rastrear variáveis importantes. Para um 3^N fatorial, um $(\frac{1}{3})^p$ fração pode ser construída, resultando em 3^{N-p} pontos. Por exemplo, para $p = 1$ em um 3^3 fatorial, o resultado é o experimento 3^{3-1} fatorial, como mostrado na Figura 3 (MONTGOMERY, 1997).

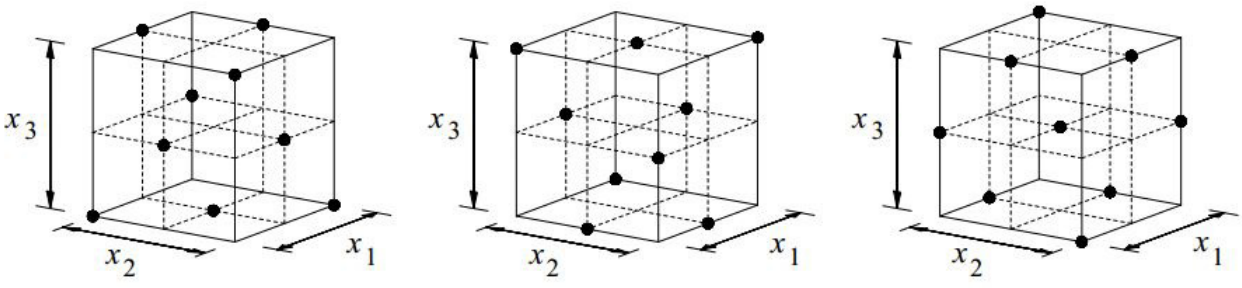

Figura 3 – 3 frações de $1/3$ de um experimento 3^3 fatorial

Fonte: (MONTGOMERY, 1997)

2.5.2.7 Calibra

O algoritmo *Calibra* combina projeto experimental (*exploration*) e pesquisa local (*exploitation*) visando encontrar bons ajustes de parâmetros (ADENSO-DIAZ; LAGUNA, 2006). Baseado em Design de Experimentos (DoE), o Calibra herda seus problemas de escalabilidade. Um espaço de parâmetro de crescimento linear permite que a quantidade de experimentos necessários cresça quadraticamente. A ferramenta *Calibra 2* só pode ser usada para 5 parâmetros discreto, exigindo que os parâmetros contínuos sejam discretizados. A abordagem pode ser aplicada a todo um conjunto de instâncias de treinamento, permitindo encontrar uma configuração robusta de parâmetros.

2.5.2.8 SKO

Sequential Kriging Optimization (SKO, (HUANG *et al.*, 2006)) é um algoritmo que estende o projeto inicial de EGOs para algoritmos aleatórios ou funções com ruido. O algoritmo baseia-se no projeto experimental e *Krigagem*, uma coleção de técnicas de interpolação para a criação de um modelo de superfície de resposta. O *SKO* restringe o usuário a otimizar os parâmetros contínuos através de uma abordagem sequencial na qual o modelo vai sendo atualizado a cada iteração até que uma certa qualidade ou limite de tempo seja atingido.

2.5.2.9 SPO

Outra extensão sequencial do EGO foi apresentada em Bartz-Beielstein, Lasarczyk e Preuß (2005), adicionando-se distribuição gaussiana com média zero de variância que representa o erro do modelo experimental acrescido do ruído experimental. A Otimização Sequencial de Parâmetros (SPO) não é um algoritmo, mas uma metodologia que enfatiza o sintonizador a fazer as perguntas certas a fim de reduzir a quantidade de experimentos para a criação de um modelo adequado do espaço de parâmetros. A versão original do SPO pode manipular somente parâmetros contínuos. Hutter *et al.* (2009a) ampliou a estrutura de SPO por um projeto experimental inicial baseado em *Latin hypercube sampling* (LHS).

2.5.2.10 SMAC

Em Hutter, Hoos e Leyton-Brown (2011), define-se um modelo geral para ajuste de parâmetros chamado de Otimização baseada em Modelo Sequencial (*Sequential Model based Optimization - SMBO*), consistindo de três etapas repetidas:

- O ajuste do modelo;
- A seleção de novas configurações a serem investigadas;

- Uma fase de intensificação limitada pelo tempo, na qual as configurações são testadas em um subconjunto selecionado aleatoriamente de um usuário especificado conjunto de treinamento.

A Configuração Sequencial do Algoritmo Baseado em Modelos é uma implementação do SMBO, na qual o treinamento consiste em avaliação de aptidão com base em configurações de parâmetros distribuídas aleatoriamente bem como na melhoria esperada. É o primeiro algoritmo que aborda a otimização de várias instâncias de problemas, sendo capaz de lidar com parâmetros contínuos, discretos e categóricos da mesma forma. A interpolação de modelo utiliza uma métrica de distância, baseada na distância de *Hamming* ponderada e Florestas Aleatórias, para parâmetros categóricos. Uma Análise de Componente Principal é executada para rastrear o espaço do problema. Os autores consideram a abordagem como um algoritmo de corrida agressivo, porque rejeita as combinações *online*.

CAPÍTULO

3

PROBLEMAS DE PROGRAMAÇÃO DE TAREFAS

Em um modelo programação de tarefas, cada trabalho deve ser processado em um conjunto de máquinas em ordem especificada. O objetivo da programação é determinar a sequência de tarefas para otimizar uma dada função objetivo. Em qualquer momento, cada máquina pode processar no máximo uma tarefa de cada vez, e cada tarefa pode ser processada por uma máquina de cada vez e em ordem, ou seja, uma tarefa não pode ser antecipada por outras.

Problemas de *FlowShop* (como são denominados) ocorrem amplamente na produção industrial e fabricação mecânica como observamos em Jiang *et al.* (2015), Jun e Park (2015) e Tang e Wang (2010) e são considerados NP-Difíceis (RUDEK; RUDEK, 2013). Por exemplo, em um processo de fabricação de aço, o aço fundido é moldado em lajes semi-acabadas, Depois de serem aquecidas pelo forno térmico, as lajes são enroladas em produtos no laminador.

Desde de que o primeiro algoritmo foi apresentado por Johnson (1954) para um problema *FlowShop* com duas máquinas com o objetivo de minimizar o tempo para a última tarefas ser processada pela última máquina (objetivo conhecido como *makespan*), muitos trabalhos têm sido conduzidos nessa área. Uma pesquisa abrangente sobre problemas de *FlowShop* até 2010 pode ser encontrada em Potts e Strusevich (2009) e Bai e Ren (2013).

Em Rudek e Rudek (2013), provaram a complexidade NP-difícil para o problema de minimização de *makepan* em problemas *FlowShop* de duas máquinas quando os tempos de processamento do trabalho são descritos por funções dependentes da posição não decrescentes (efeito de envelhecimento) em pelo menos uma máquina e indicaram complexidade NP-Difícil se os tempos de processamento do trabalho variam em ambas as máquinas. Em Aydilek e Allahverdi (2013), foi apresentado um algoritmo heurístico de tempo polinomial para o problema *FlowShop* com *makespan* de duas máquinas com datas de lançamento. Para facilitar a minimização de

FlowShop de máquinas com problema de considerações de aprendizado, Chung e Tong (2011) propuseram um teorema de dominância e um limite inferior para acelerar o algoritmo de *branch-and-bound* para buscar a solução ideal. Para o critério de *makespan*, uma heurística construtiva de alto desempenho com uma estratégia eficaz de desempate foi proposta por Ying e Lin (2013) visando melhorar a qualidade das soluções. Da mesma forma, Gupta, Nailwal e Sharma (2014) propuseram um algoritmo heurístico alternativo que é comparado com os algoritmos de *benchmark*, *Palmer*, *CDS* e *NEH* (explicados na Seção 3.2), para resolver o problema de programação de tarefas com minimização de *makepan*. Para o resultado do critério relacionado às tarefas, Bai (2015) apresentou a otimização assintótica dos algoritmos baseados no tempo de processamento mais curto visando otimizar o tempo de conclusão quadrática total com datas de lançamento.

3.1 Problema FlowShop Permutacional

Um problema de sequenciamento é definido como, dado um conjunto de tarefas F que precisam ser concluídas, cada uma das $n \in F$ tarefas requer um conjunto M de máquinas com cardinalidade m. O objetivo é prover uma ordem de alocação das máquinas às tarefas de forma a minimizar o tempo de execução de todas as tarefas.

O *FlowShop permutacional* representa um caso particular de problema *FlowShop* em que a sequência de máquina deve ser a mesma para todas as tarefas e a sequência de tarefas também deve ser a mesma. Esse problema é considerado intratável, pois o espaço de soluções possíveis tem tamanho $n!$

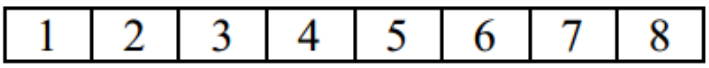

Figura 4 – Representação de uma solução de Flowshop Permutacional com 8 tarefas e duas máquinas. Independente do número de máquinas a ordem das tarefas deve ser a mesma.

Fonte: AUTOR

Um problema *FlowShop permutacional* é frequentemente descrito pelos símbolos $n|m|permi|C_{max}$, em que n representa o número de tarefas, m representa o número de máquinas, *prmu* define o tipo de problema *FlowShop* (esse caso é especificamente permutacional), C_{max} é o *makespan*. Dado que $t_{i,j}(1 \leq i \leq n, 1 \leq j \leq m)$ são os tempos de processamento da tarefa i na máquina j, assumindo que o tempo de preparação para cada tarefa é zero ou está incluído no tempo de processamento $t_{i,j}$, $\pi = (j_1, j_2, ..., j_n)$ é uma sequência de processamento das tarefas. $\prod$ é o conjunto de todas as sequências possíveis de tarefas, $C(j_i, k)$ é o tempo necessário para que a tarefa j_i seja completada na máquina k e toda tarefa será processada da máquina 1 para

máquina m ordenadamente. O objetivo de achar a sequência de tarefas que tenha o tempo mínimo para que a ultima tarefa seja processada na última máquina é dado por:

$$C(j_i,1) = t_{j_{1,1}} \tag{3.1}$$

$$C(j_i,1) = C(j_{i-1},1) + t_{j_i,1}, i = 2,3,...,n, \tag{3.2}$$

$$C(j_1,k) = C(j_1,k-1) + t_{j_i,k}, k = 2,3,...,m, \tag{3.3}$$

$$C(j_i,k) = max \quad [C(j_{i-1},k), C(j_i,k-1)] + t_{j_i,k}, i = 2,3,...,n \quad k = 2,3,...,m, \tag{3.4}$$

$$\pi_* = arg[C_{max}(\pi) = C(j_n,m)] \rightarrow min, \forall \pi \in \prod \tag{3.5}$$

em que π_* é a sequência de tarefas mais otimizada que gera o *makespan* mínimo $C_{max}(\pi_*)$.

O gráfico de Gantt representado na Figura 5 ilustra visualmente a sequencia de processamento de uma solução hipotética ao longo de um conjunto de 4 máquinas.

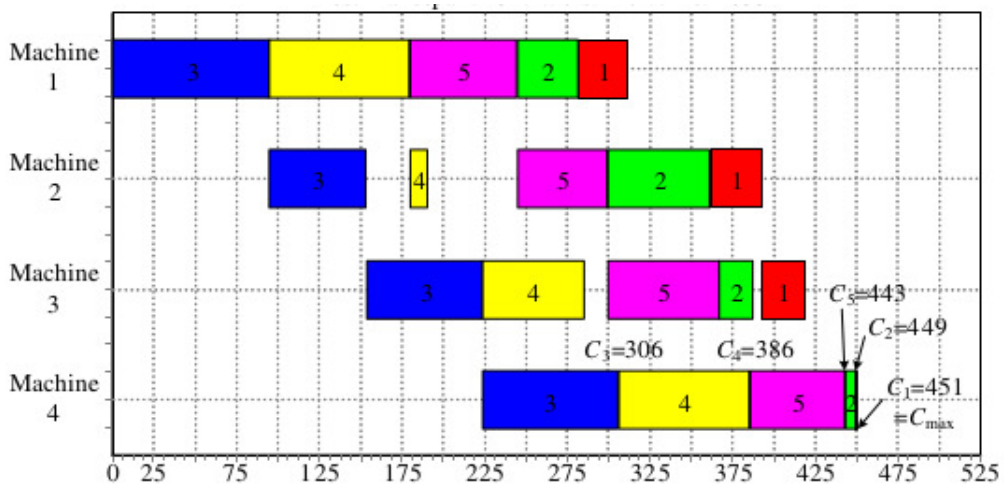

Figura 5 – Gráfico de Gantt para o exemplo da FlowShop Permutacional com o melhor makespan

Fonte: (RUIZ, 2015)

Existem muitos estudos sobre como resolver o problema de maneira exata. Pode-se citar o algoritmo baseado em *branch-and-bound* que emprega distribuição de tarefas entre processos viáveis(KOUKI; JEMNI; LADHARI, 2011), cujos resultados se mostraram encorajadores para resolver os *benchmarks* de Taillard (TAILLARD, 2013a).

Contudo, estas técnicas podem ser computacionalmente custosas, mesmo para instâncias médias, e assim, não podem ser aplicadas para problemas reais . Para contornar essa situação, métodos heurísticos que garantem a obtenção de soluções aceitáveis com recursos computacionais razoáveis possuem vantagem em eficiência. Interesses recentes em desenvolvimento eficiente de algoritmos têm resultado na atenção considerável para metaheurísticas que são particularmente atrativas para instâncias em larga escala (TSAI *et al.*, 2009).

Exemplos de contribuições significantes para a pesquisa de metaheurísticas na solução de problemas *FlowShop permutacional* são: algoritmo MOACSA de Yagmahan e Yenisey (2010),

algoritmo CR(MC) de Rajendran (1995a), algoritmo ACO Lamoudan *et al.* (2012) e algoritmo HAMC (HAMC1, HAMC2, HAMC3) de Ravindran e Selvakumar (2005). Em seguida foram explicados métodos exatos e heurísticos conhecidos utilizados para resolução de problemas *FlowShop*.

3.2 Métodos Exatos

Conforme mencionado acima, os problemas *FlowShop* pertencem a uma classe de problemas NP-difíceis. Portanto, métodos exatos são apenas adequados para alguns problemas de tamanho pequeno, por isso, para se familiarizar mais com as contribuições iniciais, uma abordagem por Programação Dinâmica e também, o conhecido algoritmo na literatura que implementa a regra de Johnson foram descritos nas Seções 3.2.1 e 3.2.2. Ambos podem ser usados para problemas de minimização de *makespan* em que $m = 2$ ou formalmente descrito por Conway e Maxwell (1967) como $n/2/F/C_{max}$ e para problema permutacionais do tipo $F/prmu/C_{max}$. Além disso, outras abordagens foram propostas, como exemplo temos: abordagens *branch-and-bound* ((IGNALL; SCHRAGE, 1965),(LOMNICKI, 1965), (DANIELS; MAZZOLA, 1994)), Regras de Eliminação (BAKER, 1974), Algoritmos de Geração de Linhas (FRIEZE; YADEGAR, 1989) e até mesmo Modelos MILP para SDST (tempos de configuração dependentes da sequência) e problemas SSIST (tempos de configuração independentes da sequência separáveis) (JR; TSENG, 1990) .

3.2.1 Programação Dinâmica

Programação Dinâmica é utilizada para resolver problemas *FlowShop* de tamanho pequeno (HELD; KARP, 1962). Dado que a_i seja representativo para t_{1i} (tempo de processamento da tarefa i na máquina 1) e b_i para t_{2i}. Suponha que $\sigma_1, \sigma_2, ..., \sigma_k$ é a permutação para as tarefas $T_1, T_2, ..., T_k$. Para esta sequência, $f1$ e $f2$ são o momentos em que o processamento dos trabalhos $T_1, T_2, ..., T_k$ são completados nos processadores (máquinas) 1 e 2, respectivamente. $t = f2 - f1$, O estado dos processadores (máquinas) na sequência das decisões é completamente caracterizado por t.

Seja $g(s,t)$ o comprimento de uma sequência ideal para o subconjunto de trabalhos s sob o pressuposto de que a máquina 2 não está disponível até o tempo t. o Comprimento de uma programação ideal para o conjunto de trabalho $(1,2,...,n)$ é $g((1,2,...,n),0)$. Como o princípio é valido, obtém-se:

$$g((1,2,...,n),0) = \min_{1 \leq i \leq n} a_i + g((1,2,...,n) - i, b_i) \tag{3.6}$$

Esta equação pode ser simplesmente generalizada para s e t da seguinte maneira:

$$g(s,t) = \min_{i \in S} a_i + g((s) - i, b_i + \max(t - a_i, 0)) \tag{3.7}$$

O termo $\max(t - a_i, 0)$ participa da equação pois a tarefa t_{2i} não pode começar até $\max(a_i, t)$. Consequentemente $f_2 - f_1 = b_i + \max(a_i, t) - a_i = b_i + \max(t - a_i, 0)$. A equação acima leva à regra de Johnson explicada na próxima secção (BAKER, 1974).

3.2.2 Regra de Johnson

O campo de problemas de sequenciamento sofreu grande influência dos estudos realizados por Johnson (1954). A regra de Johnson forneceu uma base para o desenvolvimento de uma abordagem heurística para problemas maiores (como em procedimentos Gupta e CDS) com grande sucesso.

A regra de Johnson que é a regra ótima mais conhecida aplicável a classes de problemas *FlowShop* diz que a tarefa i precede a tarefa j em uma sequência ótima se: $\min(t_{i1}, t_{j2}) \leq \min(t_{i2}, t_{j1})$. Implementando a regra de Johnson, o problema do *FlowShop* para $m = 2$ e *makespan* como critério de desempenho ou $n/2/F/c_{max}$ podem ser resolvidos de forma otimizada pelo seguinte procedimento:

1. Encontre $\min_i(t_{i1}, t_{i2})$
2. Se o tempo de processamento mínimo requer:
   a) Máquina 1: coloque a tarefa associado na primeira posição disponível em sequência e vá para o passo 3
   b) Máquina 2: coloque a tarefa associado na última posição disponível em sequência e vá para o passo 3
3. Remova a tarefa atribuída da consideração e volte para o Passo 1 até que todos as posições em sequência sejam preenchidas.

O procedimento também pode ser descrito da seguinte forma:

1. Dados $U = (j | t_{ji} \leq t_{j2})$ e $V = (j | t_{j1} \geq t_{j2})$
2. Organize os membros do conjunto U em ordem não decrescente de t_{j1} e membros do conjunto V na ordem não crescente de t_{j2}.
3. Uma sequência ótima é o conjunto ordenado U seguido pelo conjunto ordenado V.

Além disso, existem algumas extensões da regra de Johnson, uma é para $m = 3$ e critério *makespan*. essa extensão define que uma generalização é possível quando a segunda máquina é dominada (ou seja, quando nenhum gargalo poderia ocorrer na segunda máquina). Esta extensão é descrita a seguir:

1. Se $\min_k(t_{k1}) \geq \max_k(t_{k2})$, então a tarefa i precede a tarefa j em uma sequência ótima se :

$$\min(t_{i1} + t_{i2}, t_{j2} + t_{j3}) \leq \min(t_{i2} + t_{i3}, t_{j1} + t_{j2})$$

2. Se $\min_k(t_{k3}) \geq \max_k(t_{k2})$, então a tarefa i precede a tarefa j em uma sequência ótima se :

$$\min(t_{i1} + t_{i2}, t_{j2} + t_{j3}) \leq \min(t_{i2} + t_{i3}, t_{j1} + t_{j2})$$

Para aplicar esses resultados em um algoritmo, é possível usar um componente principal implementando a regra de Johnson, com o primeiro passo para procurar um mínimo na forma de $\min(t_{i1} + t_{i2}, t_{i2} + t_{i3})$ ao invés de procurar o tempo de processamento mínimo. Além disso, se não houver presente dominância, também é sabido que, o componente principal produz a mesma sequência ideal para os sub-problemas de 2 máquinas representados pelo conjunto (t_{i1}, t_{i2}) e (t_{i2}, t_{i3}). Então essa sequência é ótima também para o problema de três máquinas.

Existem outros tipos de generalização para o caso de duas máquinas, como análises de Mitten (1959) para um problema de duas máquinas envolvendo a adição de atrasos. Naquele trabalho, em muitas situações práticas, é possível iniciar a operação 2 antes da operação 1 estar totalmente completa.

Esta estrutura pode aparecer quando as tarefas consistem em grandes partes que podem ser divididas em subpartes e onde as subpartes concluídas podem prosseguir para uma operação subsequente sem esperar o tarefa toda ser processada. Essa estrutura, é chamada *fase de colapso*. Nessa fase, há um intervalo específico a_j, chamado de início de atraso, tal que A operação j_2 pode ser iniciada assim que as unidades de tempo a_j, após a operação j_1, forem iniciadas. Em particular, o rigoroso requisito de precedência do modelo *FlowShop* requer $j = t_{j1}$, mas uma fase de colapso, permite $a_j < t_{j1}$.

O problema do *FlowShop* de duas máquinas tem sido considerado em muitas pesquisas com diferentes objetivos e premissas. Allahverdi e Al-Anzi (2002) usaram uma Problema do *FlowShop* de 2 máquinas com atraso máximo para modelar objetos de dados para aplicações Web entre outras aplicações (ALLAHVERDI, 2004). Cheng, Gupta e Wang (2000) consideraram um *FlowShop* de duas máquinas sem espera com restrições de disponibilidade; Sen, Sulek e Dileepan (2003) também se concentraram em um *FlowShop* de duas máquinas sem espera; T'kindt, Gupta e Billaut (2003) trabalharam em duas máquinas com critério secundário; Allahverdi (1995) concentrou-se em um problema de *FlowShop* de duas máquinas com tempos de configuração independentes de sequência para minimizar o tempo médio de fluxo. No entanto, embora exista

muitas pesquisas sobre este tipo de problema, as mesmas não demonstram aplicações no mundo real, a falta dessas aplicações e estudos de caso no campo industrial são importantes para projeto de soluções de maior qualidade. Neste trabalho, experimentos foram realizados com instâncias aleatórias e realísticas para suprir essa necessidade.

3.3 Heurísticas Construtivas

Heurísticas construtiva constroem uma solução passo a passo, introduzindo um elemento da solução a cada passo. Conforme mencionado acima, abordagens baseadas em métodos exatos como Programação Dinâmica, *Branch-and-Bound*, Algoritmos de Geração de Colunas, Programação Inteira , entre outros, não conseguem resolver de maneira eficiente problemas de *FlowShop* por necessitarem uma grande quantidade de recursos computacionais, sendo assim, soluções pouco escaláveis. Consequentemente, abordagens heurísticas tem sido propostas em várias pesquisas tais como: Dudek e Jr (1964), Palmer (1965), Campbell, Dudek e Smith (1970), Gupta (1971), Bonney e Gundry (1976), Dannenbring (1977), King e Spachis (1980), Nawaz, Jr e Ham (1983), Hundal e Rajgopal (1988), Koulamas (1998), Sarin e Lefoka (1993), Pour (2001), Framinan, Leisten e Rajendran (2003). Além disso, para tais heurísticas construtivas, existem algumas heurísticas de melhoria, incluindo: Dannenbring (1977) e Ho e Chang (1991). Nesse sentido, as Seções seguintes descreveram alguns algoritmos heurísticos construtivos iniciais que foram pistas para os pesquisadores desenvolverem muitas outras técnicas conhecidas na literatura.

3.3.1 Algoritmo de Palmer

Palmer (1965) sugeriu um algoritmo usando o conceito de um *índice de inclinação* para cada tarefa que é uma medida de se uma tarefa passa de um tempo de processamento mais curto para um mais longo na sequência. A sequência é então construída com índices de inclinação descendente, com a ideia de que tarefas que tendem a ir de tempos de processamento mais curto para tempos de processamento mais longo na sequência de operações são processadas primeiro. Embora possa haver várias maneiras de implementar este preceito, Palmer propôs um índice de inclinação SI_i para a tarefa i como:

$$SI_i = -\sum_{j=1}^{m} [m-(2j-1)]t_{ij}/2 \tag{3.8}$$

em que t_{ij} é o tempo de processamento para a tarefa i na máquina j e m é o número de máquinas. Assim, a permutação é determinada usando a ordenação :

$$SI_{[1]} \geq SI_{[2]} \geq ... \geq SI_{[n]}$$

para $m = 2$, o algoritmo se transforma no algoritmo de Johnson

3.3.2 Algoritmo de Gupta

Gupta (1971) propôs que para $m = 2$, o *index* da tarefa pode ser calculado como:

$$SI_i = e_i / \min_{1 \leq k \leq m-1} (t_{ik} + t_{i(k+1)}) \tag{3.9}$$

em que

$$e_i = \begin{cases} 1 & se \quad t_{i1} < t_{im} \\ -1 & se \quad t_{i1} \geq t_{im} \end{cases}$$

então a ordem da permutação

$$SI_{[1]} \geq SI_{[2]} \geq ... \geq SI_{[n]}$$

faz uma boa permutação. Ele também observou que quando a regra de Johnson é otimizada no caso das três máquinas, é na forma dos cálculos acima mencionados para $m = 3$ e então comparou essa heurística com a de Palmer descobrindo que ela gera melhores resultados do que em uma maioria substancial de casos. Além disso, Gupta investigou um conjunto de outras heurísticas que também são baseadas em construção via regras transitivas.

3.3.3 Algoritmo CDS

Campbell, Dudek e Smith (1970) propuseram um algoritmo para problemas de *makepan* denominado algoritmo CDS. Usando dois princípios principais, esse procedimento encontra boas soluções usando a regra de Johnson de forma heurística e geralmente criando várias sequências, sendo a melhor, escolhida.

O algoritmo CDS cria $m-1$ problemas artificiais de duas máquinas e, em seguida, resolve-os implementando o algoritmo de Johnson. Então, a melhor $m-1$ solução obtida torna-se a melhor solução para o problema principal. Geralmente, os tempos de processamento para os subproblemas para a tarefa i na máquina j no estágio k devem ser calculados como:

$$t_{i1}^{'} = \sum_{j=1}^{k} t_{ij} \quad e \quad t_{i2}^{'} = \sum_{j=m-k+1}^{m} t_{ij} \tag{3.10}$$

Campbell, Dudek e Smith (1970) testaram e compararam os resultados com o a heurística de Palmer descobrindo que o algoritmo CDS é geralmente mais eficaz para pequenos e grandes problemas com os tempos de computação da mesma magnitude para $n \leq 20$.

3.3.4 Algoritmo NEH

Em Nawaz, Jr e Ham (1983), foi proposta uma heurística com base no pressuposto que uma tarefas com maior tempo de processamento total em todas as máquinas devem ter

mais alta prioridade do que tarefas com menor tempo de processamento total em todas as máquinas. Portanto, propõe-se inicialmente arranjar as tarefas em ordem decrescente do tempo de processamento total (calculado simplesmente por $\sum_{j=1}^{m} t_{ij}$).

A heurística NEH, em homenagem aos seus proponentes NAWAZ, ENSCORE e HAM, usa a ideia de sequenciamento parcial, a sequência é construída através da introdução de uma tarefa por vez na sequência parcial, se existir k tarefas na sequência parcial anterior, em cada inclusão de tarefa é escolhida a melhor sequência parcial entre $k+1$ sequências parciais possíveis, ou seja, a nova tarefa pode ser colocada em um dos $k+1$ lugares possíveis na sequência parcial e depois de escolher o melhor lugar desta tarefa, em relação ao *makespan* obtido, esta sequência parcial é fixada para o restante do procedimento (inclusão de uma nova tarefa).

Esta inclusão é baseada na ordem decrescente dos tempos de processamento totais em todas as máquinas. Por enumeração total $n(n-1)/2-1$, a sequência heurística e consequentemente a solução heurística para o problema de *makespan* pode ser obtida. Estudos computacionais, como Nawaz, Jr e Ham (1983) e Turner e Booth (1987), mostraram que a heurística NEH é a melhor das várias heurísticas construtivas conhecidas. No entanto, o procedimento de Koulamas (1998),Sarin e Lefoka (1993) e Pour (2001) são reivindicados como de desempenho superior a NEH.

3.4 Metaheurísticas aplicadas ao problema FlowShop Permutacional

Heurísticas modernas têm sido implementadas como algoritmos de melhoria que conseguem orientar a busca de modo a superar procedimentos típicos de otimização local em termos de qualidade das soluções encontradas. Existem muitos desses algoritmos que foram implementados em problemas *FlowShop*, incluindo: Recozimento Simulado (SA) (OSMAN; POTTS, 1989), Algoritmos Genéticos (GA) (MURATA; ISHIBUCHI; TANAKA, 1996), Busca Tabu (TS) (REEVES, 1993), abordagens Gulosas (RUIZ; STÜTZLE, 2007), abordagem de Pesquisa de Profundidade Variável (NEGENMAN, 2001), métodos Piloto (VOSSS; FINK; DUIN, 2005), Procedimentos de Subida da Encosta (ISHUBUCHI; MASAKI; TANAKA, 1995) e métodos baseados em otimização por Colônias de Formigas (ACO) (T'KINDT; GUPTA; BILLAUT, 2003). Algorítimos híbridos, que combinam algoritmos de várias formas também foram propostos com sucesso (ZOBOLAS; TARANTILIS; IOANNOU, 2009).

3.4.1 Recozimento Simulado

Kirkpatrick, Gelatt e Vecchi (1983) introduziram o conceito de Recozimento Simulado (*Simulated Annealing - SA*) considerando a analogia entre o processo de recozimento de sólidos e o processo de resolução de problemas de otimização combinatória. No entanto, foi originalmente

desenvolvido como modelo de simulação para um processo de recozimento físico de matéria condensada (METROPOLIS *et al.*, 1953). Laarhoven e Aarts (1987) iniciaram uma discussão abrangente sobre a teoria e revisão de várias aplicações mostrando que o processo de recozimento simulado converge para o conjunto de soluções globais ótimas sob certas condições.

Muitas pesquisas foram feitas sobre a aplicação de SA para problemas de sequenciamento incluindo Osman e Potts (1989), Ogbu e Smith (1990), Ishubuchi, Masaki e Tanaka (1995), Zegordi, Itoh e Enkawa (1995), Liu (1999), Wodecki e Bożzejko (2001) e Zheng e Wang (2003a). O procedimento principal da SA pode ser descrito a partir de uma solução inicial para o problema e depois gerando uma nova solução candidata na vizinhança na solução atual. Se a nova solução for melhor do que a solução atual, ela é aceita como a nova solução atual. Caso contrário, pode ser aceita ou rejeitado dependendo de uma função de probabilidade determinada pela diferença entre a função objetivo das duas soluções e por um parâmetro de controle chamado temperatura, seguindo a convenção em termodinâmica.

Inicialmente, a temperatura é definida em alto nível, como num processo de recozimento, de modo que quase todos os movimentos tendem a serem aceitos enquanto nesse patamar de entropia. Em seguida, reduz-se gradativamente durante a procedimento até que quase nenhum movimento ser aceito. O procedimento SA pode ser descrito da seguinte forma:

1. Inicialização: parâmetros do recozimento
2. Selecione um mecanismo de iteração: uma receita simples para gerar uma transição do estado atual para outro estado por uma pequena perturbação.
3. Avalie o novo estado, calcula ΔE (valor do estado atual menos valor de novo estado).
4. Se o novo estado for melhor, torne-o estado atual, de outra forma, probabilisticamente aceita ou rejeita-o (com uma determinada função de probabilidade geralmente chamada função de probabilidade de aceitação)
5. Com base nas regras de parada, pare ou continue as iterações no Passo 2.

Para implementar uma SA em aplicações específicas, como por exemplo, para problemas *FlowShop*, itens como formas de gerar soluções de vizinhança, denominado de esquema perturbação, programação de resfriamento, que é uma temperatura decrescente, bem como a relação entre a qualidade da solução e o tempo de computação, são muito importantes e devem ser sintonizados de maneira apropriada. Os seguintes parâmetros devem ser determinados:

1. Valor inicial da temperatura
2. Função que determina como a temperatura é modificada

3. Condição do equilíbrio Metropolis (METROPOLIS *et al.*, 1953) sob cada temperatura, isto é, o número de iterações a serem realizadas a cada temperatura

4. Critério de parada para terminar o algoritmo, isto é, o valor final do parâmetro de temperatura.

Para um problema de *FlowShop*, diferentes esquemas de perturbação foram considerados. Sridhar e Rajendran (1994) utilizaram três esquemas de perturbação (intercâmbios adjacentes, esquema de inserção e esquema de intercâmbio aleatório). Osman e Potts (1989) adotaram intercambio de vizinhança. Ogbu e Smith (1990) escolheram a inserção e a troca entre pares como o esquema de perturbação. Liu (1999) trabalhou no tamanho da vizinhança e seu efeito sobre o recozimento simulado para um problema de *FlowShop* usando uma vizinhança de tamanho variável para seu algoritmo SA proposto e projetando que variasse em função do número de testes até agora: $tamanho = 1 + (n/2-1)(1-t/T)^{0.4}$, em que t é o número de testes até o momento e T é o número total de testes para todo o processo. O tamanho da vizinhança para um método de troca pode ser caracterizado pela distância entre a posição de duas tarefas sendo trocadas na sequencia. A partir de qualquer um dessas duas tarefas, é possível contar para esquerda ou direita. A distância entre as duas tarefas na sequência pode ser considerada pelo número menor. Tian, Ma e Zhang (1999) usaram seis esquemas de perturbação do seguinte modo:

- troca de duas tarefas adjacentes.
- troca de duas tarefas.
- Movendo uma única tarefa.
- Mover uma tarefa subsequente.
- Inverter uma tarefa subsequente.
- Inverter e / ou mover uma subsequencia de tarefas.

A escolha do esquema de resfriamento apropriado ou do cronograma de recozimento também está relacionada à qualidade da solução e o tempo de computação. Normalmente, o resfriamento ocorre por um fator de redução α e uma função recursiva $\beta_n = \alpha\beta_{n-1}$, em que n é o estágio no qual o horário de resfriamento é colocado. A probabilidade de aceitação geralmente é determinada pela função $e^{-\delta/k\beta_n}$, em que δ é a diferença entre a nova solução S_0 e a solução anterior S, isto é, no que diz respeito ao *makespan* como critério objetivo $\delta = c_{max}(S) - c_{max}(S_0)$. Além disso, é sabido que se c_{ij} denota o tempo de conclusão da ith tarefa na sequencia de máquinas $1,2,...,j$, para diferentes problemas *FlowShop* com diferentes políticas de armazenamento intermediário, c_{max} é c_{nm}, em que c_{ij} pode ser definido como:

- Para armazenamento intermediário ilimitado (UIS): $c_{ij} = \max(c_{(i-1)j}, c_{i(j-1)}) + t_{ij}$
- Para armazenamento intermediário finito (FIS): $c_{ij} = \max(c_{(i-1)j}, c_{i(j-1)}, c_{(i-z-1)(j+1)} - t_{ij}) + t_{ij}$
- Para nenhum armazenamento intermediário (NIS):$c_{ij} = \max(c_{(i-1)j}, c_{i(j-1)}, c_{(i-1)(j+1)} - t_{ij}) + t_{ij}$

Portanto, pode-se alcançar δ para diferentes problemas de *FlowShop* com *makespan*. k na função $e^{-\delta/k\beta_n}$ é uma constante de Boltzman que no início da SA pode ser estimado por $k = [c_{max}(S) - c_{max}(S_0)]/lnp_0/\beta_0$, em que p_0 é o valor inicial da probabilidade de aceitação.

No entanto, Ogbu e Smith (1990) mostraram que um SA com uma função de probabilidade de aceitação que é independente da mudança no valor da função objetivo ,δ, proveu soluções com tão boa qualidade quanto o esquema convencional (Metropolis). Portanto, a função de probabilidade de aceitação, para $\delta > 0$, ou seja, aceitar uma solução pior, também pode ser dada por algumas funções, como $AP(k) = p_0(rf)^{k-1}$, em que $k = 1$ e $rf < 1$ é o fator de redução.

Para comparar diferentes SA's, há algumas medidas de desempenho propostas. Por exemplo, uma é a otimização da solução final: $(c_{max}(T) - c^*_{max})/c^*_{max}$, em que T é o número total de testes e $c_{max}(T)$ é o *makespan* da solução final. Outra medida de performance pode ser proposta para a eficácia de todo o processo, tal como:

$$\int_0^T [(c_{max}(t) - c^*_{max})/(c_{max}(0) - c^*_{max})]dt \tag{3.11}$$

em que $c_{max}(t)$ é o *makespan* da melhor solução encontrada por t testes. A eficácia dada acima é a área sob a curva de otimidade contra o número de ensaios entre 0 e T testes.

O algoritmo 3 mostra o SA para o problema de *FlowShop* com critério *makespan* com os parâmetros e detalhes descritos.

Algoritmo 3: Recozimento Simulado

Input: Conjunto de parâmetros
Selecione uma solução inicial S_0;
Configure $\beta_0, \beta_{fim}, p_0, N_0$;
Configure α de acordo com o tempo computacional permitido;
Estime $c_{max}(S) - c_{max}(S_0)$ inicial e calcule k;
repeat
    repeat
        Selecione aleatoriamente uma solução S da vizinhança de S_0;
        $\delta \leftarrow c_{max}(S) - c_{max}(S_0)$;
        if $\delta < 0$ **then**
            $S_0 \leftarrow s$;
        else
            Gere uma distribuição uniforme aleatória entre [0,1];
            if $x < e^{-\delta/k\beta}$ **then**
                $S_0 \leftarrow S$;
    until *o número de testes em* β *atinja* N_0;
    $\beta \leftarrow \alpha\beta$;
until $\beta < \beta_{fim}$;
return a melhor solução S gravada no processo;

3.4.2 Busca Tabu

A metaheurística Busca Tabu (*Tabu Search - TS*) foi proposta e bastante estudada por Glover (1989) e Glover e Laguna (2013). Esta abordagem, a partir de uma solução inicial, iterativamente aplica um mecanismo de movimento para pesquisar a vizinhança da solução atual e escolher o movimento mais apropriado. A solução vizinha é admissível se não for considerada *tabu* (proibida) ou se um critério de aspiração (como permitir que todos os movimentos que levem a um vizinho com melhor função objetiva do que encontrado até agora) seja cumprido.

Para usar as informações sobre o histórico de pesquisas, os movimentos selecionados são armazenados em uma estrutura de dados chamada lista tabu. Esta lista contém l elementos de cada vez e uma vez que um movimento é inserido, o mais antigo é excluído, ou seja, o movimento selecionado é colocado na lista de tabu e permanece lá para as próximas iterações. Esse parâmetro, l, é chamado de tamanho da lista de tabu. Para determinar este parâmetro, existem atualmente três alternativas (SARMADY, 2012): definir um valor constante, escolher aleatoriamente de um determinado intervalo e mudar dinamicamente através de determinados ajustes.

Apesar da simplicidade da TS, faz-se necessário a definição do mecanismo de vizinhança, tamanho da lista tabu etc. A construção de tais componentes influencia o desempenho do algoritmo, a velocidade de convergência, o tempo de execução, etc. Várias abordagens baseadas em TS foram propostas para um problema *FlowShop*, incluindo Taillard (1990), Reeves (1993),

Moccellin (1995), Nowicki e Smutnicki (1996),Ben-Daya e Al-Fawzan (1998) e o famoso algoritmo SPIRIT proposto por Widmer e Hertz (1989).

Ben-Daya e Al-Fawzan (1998) implementaram um algoritmo de Busca Tabu com alguns recursos extras, como esquemas de intensificação e diversificação que proporcionam melhores movimentos no processo de busca. O algoritmo proposto fornece resultados semelhantes ao TS de Taillard (1990) e resultados ligeiramente melhores que a SA de Ogbu e Smith (1990).

Implementando propriedades de bloco, Solimanpur, Vrat e Shankar (2004) propuseram um neuro-busca tabu e Grabowski e Wodecki (2004), uma abordagem rápida de Busca Tabu para problemas *FlowShop Permutacional* com critério de *makespan*.

Bem conhecido na literatura, o $c_{max}(\pi)$ (*makespan*) é dado pela melhor sequencia de tarefas π e pode ser definido como:

$$c_{max}(\pi) = \max_{1\leq t_1\leq t_2\leq \ldots\leq t_{m-1}\leq n} (\sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \ldots + \sum_{j=t_m-1}^{n} p_{\pi(j)m}) \tag{3.12}$$

em que $\pi(j)$ é o jth elemento de permutação π. Em outras palavras, o *makespan* associado à π pode ser considerado como o melhor caminho do nó $(1,1)$ para o nó (m,n) é um grafo orientado. Cada sub-caminho do grafo horizontal é chamado de bloco, em outras palavras, se considerarmos o caminho $u = (u_1, u_2, \ldots, u_{m-1})$, tal que :

$$c_{max}(\pi) = c(\pi, u) = (\sum_{j=1}^{u_1} p_{\pi(j)1} + \sum_{j=u_1}^{u_2} p_{\pi(j)2} + \ldots + \sum_{j=u_m-1}^{n} p_{\pi(j)m}).$$

bem como:

$$B_k = (\pi(u_{k-1}), \pi(u_{k-1}+1), \ldots, \pi(u_k))$$

que é um caminho crítico em relação a π, uma sequência de tarefas é chamado de bloco k em π, que é dependente de π, mas por simplicidade, é representado como B_k.

Pode-se considerar muitos tipos de operadores de reordenação, como trocar duas tarefas adjacentes ou não adjacentes ou remover uma tarefa de seu lugar na sequência e inseri-la em outra posição. Taillard (1990) experimentalmente sugeriu que o último, mover $v = (x, y)$, remover a tarefa colocada na posição x e inseri-la na posição y, é mais eficiente e eficaz. como o número total dos movimentos a serem avaliados em cada iteração é $(N-1)^2$ (alto para grandes problemas), para reduzir o número de movimentos de algumas políticas de redução com base nas propriedades do bloco e observações podem ser usadas.

Por exemplo, uma propriedade elementar em termos aproximados mostra que as tarefas em movimento em blocos com os operadores de movimentos discutidos não conseguem fazer

uma melhor configuração, consequentemente, eles não são movimentos interessantes e o número de movimentos pode ser reduzido. Além disso, os achados experimentais de Nowicki e Smutnicki (1996) mostraram que poucas inserções nos blocos adjacentes podem ser mais promissoras do que outras posições, que podem ser implementadas para redução adicional de movimentos, como o que Solimanpur, Vrat e Shankar (2004) usaram em sua heurística de busca de neuro-tabu para um problema de *FlowShop*.

Grabowski e Wodecki (2004), para reduzir o gasto computacional, propuseram a utilização de limites inferiores no *makespan* para selecionar a melhor solução e construir uma TS muito rápida para um problema *FlowShop* com critério de *makespan*. O algoritmo 4 exemplifica o funcionamento da Busca Tabu.

Algoritmo 4: Busca Tabu

Input: S,Memória Tabu

Inicialize a memória Tabu;
while *Critério de parada não for satisfeito* **do**
    $s^{'} \leftarrow MelhorVizinho(s, Vizinhana, MemriaTAbu)$; $MemriaTabu.add(move(s, s^{'}))$;
    $s \leftarrow s^{'}$; $MemriaTabu.add(s)$;
    if *MemóriaTabu acionar a flag de escape* **then**
        promova movimento de diversidade;
return a melhor solução S gravada no processo;

3.4.3 Colônias de Formigas

O sistema de colônias de formigas (ACS) proposto pela primeira vez por Dorigo e Gambardella (1997) é uma das metaheurísticas mais recentes e promissoras para problemas de otimização combinatória. A otimização por colônias de formigas (ACO) simula os hábitos coletivos de forrageamento de formigas, se arriscando por comida e trazendo de volta ao ninho. Formigas reais são capazes de encontrar o caminho mais curto de uma fonte de alimento para o seu ninho sem usar pistas visuais pois as mesmas possuem pouca visão.

As formigas comunicam informações sobre fontes alimentares via uma essência aromática. Esta substância química depositada enquanto viajam é chamada feromônio. Uma maior quantidade de feromônio no caminho dá a uma formiga um estímulo mais forte e, portanto, uma maior probabilidade de seguir aquele caminho. Formigas, essencialmente, se movem aleatoriamente, mas quando encontram uma trilha de feromônio, decidem (dependendo da quantidade de feromônio no caminho potencial), segui-lo ou não, e, se o fizerem, depositam seu próprio feromônio na trilha, o que reforça o caminho. Uma vez que as formigas que passam por uma fonte de alimento por um caminho mais curto voltarão ao ninho mais cedo do que as formigas através de caminhos mais longos, o caminho mais curto terá uma maior densidade de tráfego e, portanto, a quantidade de feromônio depositada no caminho mais curto crescerá mais rápido. Além disso, com o tempo, o feromônio evapora e como as formigas que tomam o caminho mais

curto voltarão ao ninho primeiro com comida, o caminho mais curto possui mais feromônio. Consequentemente, o caminho mais curto torna-se uma alternativa mais atraente para outras formigas. No entanto, há sempre uma probabilidade de uma formiga não seguir uma trilha de feromônio bem marcada, essa probabilidade (embora talvez pequena) permite a exploração de outras trilhas.

O comportamento de forrageamento descrito de colônias reais de formigas pode ser usado para resolver problemas de otimização combinatória por simulação: o valor objetivo correspondente à qualidade da fonte de alimento, formigas artificiais buscando no espaço de soluções simula formigas reais que buscam em seu ambiente e uma memória adaptativa correspondente à trilha de feromônios. As formigas artificiais estão equipadas com uma função heurística local para orientar sua pesquisa através do conjunto de soluções viáveis.

O ACO foi aplicado para resolver diferentes tipos de problemas de otimização combinatória incluindo o TSP (DORIGO; GAMBARDELLA, 1997), QAP (Problema de atribuição quadrática) (GAMBARDELLA; TAILLARD; DORIGO, 1999), o Problemas de sequenciamento (COLORNI *et al.*, 1994), VRP (*Vehicle Routing Problem*) (BULLNHEIMER; HARTL; STRAUSS, 1999), o problema de coloração de grafos (COSTA; HERTZ, 1997), o Problema de Partição (KUNTZ; LAYZELL; SNYERS, 1997) e o problema das redes telecomunicação (SCHOONDERWOERD *et al.*, 1997).

Existem algumas pesquisas sobre a implementação de ACO para o problemas *FlowShop*. T'kindt, Gupta e Billaut (2003) propuseram um algoritmo ACO para minimizar o tempo de conclusão total. Rajendran e Ziegler (2004) consideraram o problema *FlowShop Permutacional* com o objetivo de minimizar o *makespan*, seguido pela consideração da minimização do tempo total de fluxo de trabalho. Eles propuseram dois algoritmos de colônia de formigas. O primeiro algoritmo amplia as ideias da colônia de formigas do chamado sistema de colônia max-min (MMAS) de Stützle e Hoos (1998), incorporando a regra de soma sugerida por Merkle e Middendorf (2000) e uma técnica de busca local recentemente proposta. Os primeiros artigos aplicando ACO (ainda denominado *Ant Colony System - ACS*) para $n/m/p/c_{max}$ ou $F/prmu/c_{max}$ foram feitos por Stützle e Hoos (1998) e Ying e Liao (2004), que, conduzidos no bem conhecido *benchmark* de Taillard (1990), mostraram que a abordagem ACS é uma metaheurística efetiva para problemas *FlowShop*.

Como exemplo, pode-se descrever a representação do problema *FlowShop Permutacional* com critério de*makespan* no ACO. Este problema pode ser representado por um grafo disjuntivo $G = (O, C, D)$, em que O é um conjunto de nós, C é um conjunto de arcos conjuntivos direcionados e D é um conjunto de arcos disjuntivos não direcionados.

A Figura 6 mostra uma instância de $3/4/p/c_{max}$ (YING; LIAO, 2004)). Os arcos disjuntivos não direcionados D de m saltos, um para cada máquina, podem determinar a ordem de processamento da operação na máquina; e enquanto em problemas de permutação todos os trabalhos tem a mesma sequência de pedidos em cada máquina , basta encontrar a primeira

sequencia do salto.

Em uma abordagem baseada em ACS, primeiro um conjunto de formigas artificiais está inicialmente posicionado no nó inicial de acordo com alguma regra de inicialização (por exemplo, aleatoriamente). Cada formiga constrói um caminho que é uma solução viável para o problema $(n/m/p/c_{max})$, isso é feito aplicando repetidamente uma regra estocástica gulosa, que é chamada de regra de transição de estado. De acordo com Dorigo e Gambardella (1997) e Ying e Liao (2004), na construção de um caminho no ACS, uma formiga na posição atual do nó i escolhe o próximo nó j para mover-se aplicando a regra de transição de estado dada pela seguinte equação:

$$j = \begin{cases} argmax[[\tau(i,u)][\eta(i,u)]^{\beta}] & se \quad q \leq q_0 \\ J \quad contrario \end{cases} \tag{3.13}$$

onde $\tau(i,u)$ é a trilha de feromônio do arco (i,u). A heurística $\eta(i,u) = 1/\delta(i,u)$ é o inverso do comprimento do nó i ao nó $u(\delta(i,u))$, $S_k(i)$ é o conjunto de nós que permanecem para serem visitados pela formiga k posicionada no nó i (para fazer uma solução factível).

A constante β é um parâmetro que determina a importância relativa do feromônio versus distância ($\beta > 0$) sendo um número aleatório uniformemente distribuído em $(0,1)$, q_o é um parâmetro $(0 \leq q_0 \leq 1)$ que determina a importância relativa de intensificação versus exploração. J é uma variável aleatória que dá a probabilidade com que a formiga k no nó i escolha mudar para o nó j que está selecionado de acordo com probabilidade de distribuição:

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)][\eta(i,j)]^{\beta}}{\sum_{u \in S_k(i)}[\tau(i,j)][\eta(i,j)]^{\beta}} & se \quad j \in S_k(i) \\ 0 \quad contrario \end{cases} \tag{3.14}$$

Ao construir seus caminhos, os arcos escolhidos, as formigas são guiadas por ambas as informações heurísticas e informação de feromônio. A regra de transição do estado resultante das Equações 3.13 e 3.14 favorecem a escolha de nós conectados por arcos mais curtos com uma maior quantidade de feromônio.

Toda vez que uma formiga no nó i tem que escolher um nó j para se mover, é amostrado um número aleatório q, se $q \leq q_0$, então o melhor arco (de acordo com a Equação 3.13) é escolhido (intensificação), caso contrário, um arco é escolhido de acordo com a Equação 3.14 (exploração tendenciosa).

Ying e Liao (2004) definiram o comprimento entre o nó i e o nó u do primeiro salto como:

$$\delta(i,u) = 1/\eta(i,u)$$

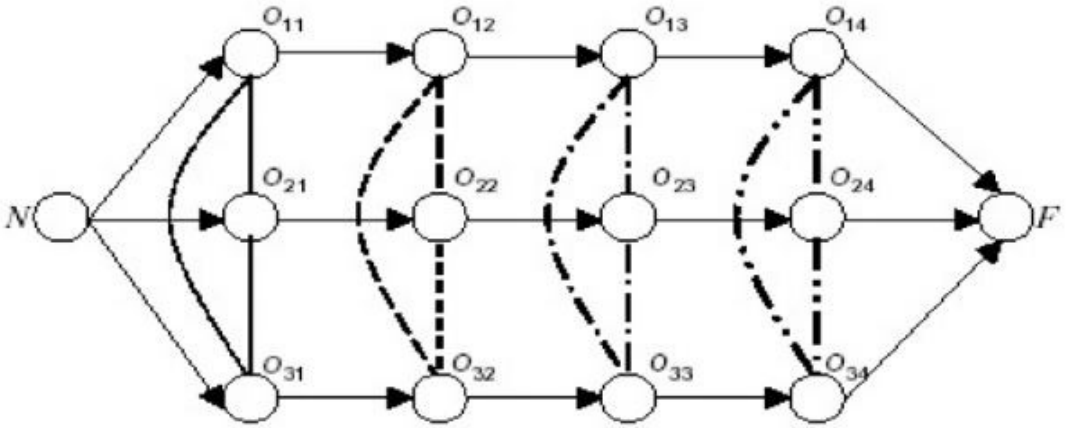

Figura 6 – Uma instância de $3/4/P/c_{max}$. Legenda: (...) Arcos para máquina 1, (- - -) arcos para máquina 2, (- . -) arcos para máquina 3 e (- . . -) arcos para máquina 4

Fonte: (HEJAZI*; SAGHAFIAN, 2005)

em que

$$\eta(i,u) = (m-1)p_{u,m} + (m-3)p_{u,m-1} + \ldots - (m-3)p_{u,2} - (m-1)p_{u,1} - \min_{u}(\eta(i,u)) + 1 \quad (i \neq u)$$

Ao construir seu caminho, uma formiga também atualiza a quantidade de feromônio ao aplicar a regra de atualização local da seguinte maneira:

$$\tau(i,j) = (1-\rho)\tau(i,j) + \rho\tau_0 \tag{3.15}$$

no qual τ_0 é o nível inicial de feromônio e $0 < \rho < 1$ é a evaporação do parâmetro feromônio. O efeito da regra de atualização local é fazer a escolha de arcos mude dinamicamente para embaralhar o caminho. Se as formigas exploram diferentes caminhos, então há uma maior probabilidade de que uma delas encontre uma solução melhorada do que se todas procurarem em uma vizinhança de um melhor caminho anterior. Toda vez que uma formiga constrói um caminho, a regra de atualização local fará o feromônio de arcos visitados diminuir e tornar-se menos atraente, assim, os nós em um caminho de formigas serão escolhidos com menor probabilidade na construção de outros caminhos de formigas. Consequentemente, formigas irão favorecer a exploração de arcos ainda não visitados e evitar a convergência para um caminho comum.

Finalmente, uma vez que todas as formigas terminaram suas trajetórias, trilhas de feromônio nos arcos são modificados novamente aplicando a regra de atualização global. Para fazer a busca se mover direcionado, esta regra destina-se a fornecer uma maior quantidade de feromônio a mais curta trajetória e reforça-la. Portanto, apenas a melhor formiga que encontrou a melhor solução (caminho mais curto) até as iterações atuais do algoritmo é permitido depositar feromônio. O nível de feromônio pode ser modificado por:

$$\tau(i,j) = (1-\alpha)\tau(i,j) + \alpha\Delta\tau(i,j). \tag{3.16}$$

, sendo que:

$$\Delta\tau(i,j) = \begin{cases} (L_{gb}^{-1}) \quad se \quad (i,j) \in Melhor \quad caminho \quad global \\ 0 \quad contrrio \end{cases} \tag{3.17}$$

Na Equação 3.16, $\alpha(0 \leq \alpha \leq 1)$ é o parâmetro de evaporação de feromônio. Na Equação 3.17 , L_{gb} é o tamanho do melhor caminho encontrado até a iteração atual, assim, a equação 3.16 modifica o nível de feromônio para fazer a busca se mover em direção aos caminhos mais curtos reforçados.

3.4.4 Algoritmos Genéticos

Algoritmos genéticos (*Genetic Algorithm - GA*) foram descritos por Holland (1992) baseados no processo de evolução biológica com o duplo objetivo de: melhorar a compreensão do processo de adaptação natural e projetar sistemas artificiais com propriedades semelhantes aos sistemas naturais (GOLDBERG; HOLLAND, 1988), durante a última década, os GAs foram amplamente aplicados em muitos campos de otimização, como problemas de otimização combinatória (por exemplo, no TSP e problemas de sequenciamento e agendamento: Jog (1989),Whitley e Fuquay (1991), Ulder *et al.* (1990), Cleveland e Smith (1989), Gabbert (1991), Nakano e Yamada (1991), Fox e McMahon (1991), Syswerda (1991). A implementação de GA para um problema *FlowShop* é vista em muitos artigos (por exemplo, Syswerda (1991), Reeves (1995), Neppalli, Chen e Gupta (1996), Murata, Ishibuchi e Tanaka (1996), Reeves e Yamada (1998), Ponnambalam, Aravindan e Rao (2001), Zheng e Wang (2003b)).

SA e GA tiveram seus desempenhos comparados para problemas *FlowShop* que variam de 20 tarefas e cinco máquinas para 500 tarefas e 20 máquinas (REEVES, 1993). A principal conclusão desse trabalho foi que o SA possui melhor performance que o GA na maioria dos testes, onde o número de tarefas é de 50 ou menos; Por outro lado, o GA forneceu a soluções superiores às obtidas pelo SA para grandes problemas.

Murata, Ishibuchi e Tanaka (1996) apresentaram resultados semelhantes comparando vários operadores de cruzamento e mutação e mostraram que o cruzamento de dois pontos e operadores de mutação são eficazes para um problema *FlowShop*. Eles também consideraram duas variantes híbridas do GA, nomeadamente Busca Genética Local e algoritmos de Recozimento Simulado Genético que apresentaram alto desempenho. Lee e Shaw (2000) usaram a GA com operador de cruzamento de recombinação de borda (como se viu em Whitley e Fuquay ()) obtendo soluções de boa qualidade. Eles também usaram uma rede neural hibridizada com GA que apresentou ótimo desempenho. Algoritmos Genéticos para problemas *FlowShop* geralmente usam representação de números inteiros, em que os cromossomos são vetores de índices de tarefas, que representam sequências das mesmas. Isso significa que, se uma tarefa está na posição i do cromossomo, então também está na posição i dessa sequência.

O espaço de busca é composto de todos os cromossomos atuais em uma população com um O número fixo de cromossomos P. P é referido como o tamanho da população. A população inicial pode consistir em P sequências geradas aleatoriamente ou podem ser obtidas por heurísticas construtivas descritas anteriormente. A função de aptidão de um cromossomo reflete a eficácia do *makespan* que corresponde à sequência de tarefas que ele representa (para problemas de *makespan*). Simplesmente, um valor de aptidão atribuído a um cromossomo i pode ser proporcional a $f_i = r_i / \sum_j r_j$, em que r_i é o recíproco do *makespan* do cromossomo i na população. Ao selecionar aleatoriamente cromossomos da população atual (geralmente dois pais) com uma probabilidade proporcional ao seu valor de aptidão , uma prole que herda características genéticas dos pais pode ser construída. O tamanho da população é mantido constante pela remoção de um cromossomo sempre que uma nova prole é introduzida na população. Este cromossomo removido pode ser selecionado aleatoriamente ou com base na sua aptidão. Isso simula um cenário real em que os cromossomos que representam soluções com melhores *makespans* têm uma maior probabilidade de permanecer na população e de ser selecionados mais vezes para passarem suas características adiante. O ato de passar as características é feito por operadores de cruzamento e mutação para formar a próxima geração.

O objetivo principal do cruzamento é trocar informações entre cromossomos parentais selecionados aleatoriamente com o objetivo de produzir melhores descendentes e direcionar a busca por melhores genes. Dois pais selecionados para o cruzamento são copiados com probabilidade $1 - p_c$ e com probabilidade p_c chamadas probabilidades de cruzamento, um ponto aleatório (ou pontos) é selecionado e seções de pais diferentes são unidas para criar um novo cromossomo. A dificuldade na aplicação de operadores de cruzamento para cromossomos que não são binários, como na representação inteira do problema *FlowShop*, é que a recombinação geralmente resulta em uma cadeia inviável em que as tarefas aparecem duas vezes ou não na prole. Por exemplo, um GA padrão com cruzamento de um ponto (chamado operador 1X) entre dois pais é realizado escolhendo um número k aleatório entre 1 e $l - 1$, em que l é o tamanho do cromossomo. Dois novos cromossomos são criados trocando todos os elementos da posição $k + 1$ para l. Esse cruzamento pode resultar em cromossomos inviáveis. No entanto, muitas modificações e operadores de cruzamento são sugeridos e demonstraram ser apropriados para um problema *FlowShop* em muitas pesquisas, incluindo cruzamento de dois pontos (2X), cruzamento de ordem linear (LOX), cruzamento parcialmente mapeado (PMX), cruzamento de ciclo (CX), operador C1, operador NABEL, cruzamento multi-parental (MPX) entre outros. Alguns desses operadores foram descritos a seguir:

- **2X**: No caso de um cruzamento de dois pontos, os cromossomos parentais são cortados em dois pontos aleatórios. As tarefas nos extremos dos dois cortes são herdados pelos filhos nas localizações exatas e na ordem em que aparecem no cromossomo.
- **LOX**: Primeiro, dois locais de corte dos cromossomos parentais, exemplo (264735891) e (452187693), são escolhidos aleatoriamente, exemplo 2 e 5. Em segundo lugar, os símbo-

los que aparecem na seção transversal do primeiro pai (a área situada entre os dois pontos de corte) são removidos do segundo pai, deixando alguns "buracos", ou seja, $(H5|218|H69H)$ e $(H6|473|5H9H)$. Então os buracos são deslizados das extremidades para o centro até chegarem à seção de cruzamento, isto é $(52|HHH|1869)$ e $(64|HHH|7359)$. Finalmente, a seção transversal é substituída com o do pai correspondente para obter os filhos, isto é $(52|473|1869)$ e $(64|218|7359)$. Considerou-se que a LOX poderia preservar o máximo possível das posições relativas entre os genes e as posições absolutas em relação às extremidades do cromossomo.

- **PMX**:O operador PMX pode ser considerado o operador de cruzamento mais popular. Ele escolhe os dois primeiros pontos de cruzamento e troca a subseção dos pais entre os dois pontos e, em seguida, preenche os cromossomos por mapeamento parcial. Considerando os dois pais acima, se os dois pontos de cruzamento forem 3 e 7, então os filhos serão $(234|1876|95)$ e $(412|7358|96)$.

Além do cruzamento, operadores de mutação são implementados em GAs para melhorar o desempenho. A mutação é uma forma de ampliar o espaço de busca. Atua para prevenir a seleção e cruzamento focada em uma área estreita do espaço de busca,ficando presa em um ótimo local. Para cada filho obtido a partir do cruzamento, o operador de mutação é aplicado de forma independente com uma pequena probabilidade, chamada probabilidade de mutação.

A Figura 5 mostra o algoritmo GA de uma maneira geral.

Algoritmo 5: Algoritmo Genético genérico

Input: P,MaxGen

$t \leftarrow 0$;
Inicializar uma população de tamanho $P(t)$;
$Avalia(P(t))$;
while *gt*<*MaxGen* **do**
    $P_{best}(t) \leftarrow P(t).melhores$;
    $P_{cross}(t) \leftarrow Reproducao(P_{best}(t))$;
    $Mutacao(P_{cross}(t))$;
    $Avalia(P_{cross}(t))$;
    $P(t+1) \leftarrow NovaGeracao(P_{cross}(t), P(t))$;
    $t \leftarrow t+1$;
return a melhor solução em P ;

CAPÍTULO

4

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS GUIADO POR AGRUPAMENTO

Com intuito de validar a metodologia de sintonização proposta, está sendo proposto neste trabalho um algoritmo genético híbrido para solucionar instâncias de *FlowShop Permutacional*. O algoritmo proposto, nomeado neste trabalho como *Biased Random-Key Evolutionary Clustering Search* (*BRKeCS*), é baseado no algoritmo genético de chaves aleatórias viciadas (acrônimo em inglês *BRKGA*) e na busca guiada por agrupamentos (acrônimo em inglês *CS*).

A híbridização *BRKGA+CS* foi inicialmente proposta para resolver problemas de minimização de trocas de ferramentas, agregando o objetivo de reduzir a complexidade de desenvolvimento de aplicações baseadas em *Clustering Search* que comumente incorporam uma série de componentes específicos destinados a resolução do problema em questão (CHAVES *et al.*, 2016). Portanto, o *BRKeCS* realiza busca guiada por agrupamentos em um espaço de soluções candidatas codificado em vez do espaço de busca original do problema.

Apesar do número de aplicações envolvendo algoritmos baseados em busca guiada por agrupamento tanto no campo de combinatória quanto otimização contínua (OLIVEIRA; LORENA, 2007; LIU Y.AND OUYANG; SHENG; ZHANG, 2008; OLIVEIRA; COSTA, 2014a; OLIVEIRA; COSTA, 2014b), uma certa dificuldade reside na necessidade de desenvolvimento de procedimentos específicos para cálculo da métrica de distância, operação de assimilação e busca local, além de operadores heurísticos nativos (melhor explicados a seguir). Uma aplicação CS, ao final, oferece um desafio a mais que é o ajuste de muitos parâmetros de desempenho.

A metaheurística híbrida desenvolvida neste trabalho, além da expectativa de competitividade que venha a favorecer o processo de validação da sintonização, também vem ao encontro da necessidade premente de reduzir a complexidade desses procedimentos específicos do *framework CS*.

4.1 Algoritmos Genéticos de chaves aleatórias e viciadas

Um algoritmo genético de chaves aleatórias e viciadas (*Biased random-key genetic algorithms*-BRKGA) é uma metaheurística de otimização baseada nas chaves aleatórias de *Bean* (BEAN, 1994). A solução de um problema de otimização é codificada como um vetor de números reais mantidos sempre no intervalo $[0,1]$. O algoritmo realiza uma busca através do espaço de soluções do problema de otimização combinatória indiretamente, explorando o hipercubo de unidade contínua e mapeando soluções contínuas para soluções discretas do problema usando uma heurística chamada decodificador.

A população de chaves aleatórias, p, é evoluída durante uma série de iterações chamadas gerações. Cada componente de um indivíduo é gerado na faixa $[0,1]$ e cada indivíduo tem sua aptidão calculada pelo método *decoder*, durante a geração k. A população é particionada em dois grupos: uma pequena p_e de soluções de elite, composta dos indivíduos com as melhores aptidões, e os $p - p_e$ indivíduos denominados como grupo não-elite. Na geração seguinte, uma nova população é produzida, todos os indivíduos do grupo elite da geração k são copiados sem modificações para a população da geração $k+1$. Um pequeno número de indivíduos mutantes também é gerado, a partir de modificações pontuais nos indivíduos da população.

Com a elite p_e e os mutantes p_m, é necessário gerar os outros $p - p_e - p_m$ indivíduos, produzidos pelo processo de cruzamento. Durante a geração $k+1$, a população é novamente decodificada, avaliada e separada em elite e não-elite para iniciar uma nova geração. O processo é repetido até ser alcançado o critério de parada.

4.2 Evolutionary Clustering Search

Clustering Search (CS) é uma forma genérica de combinar metaheurísticas de otimização com algoritmos de agrupamento visando detectar regiões promissoras do espaço de busca para que essas regiões sejam interativamente exploradas por heurísticas específicas do problema (OLIVEIRA; LORENA, 2007). CS divide dinamicamente o espaço de busca em *clusters* a partir das soluções candidatas amostradas pela metaheurística de suporte.

Evolutionary Clustering Search (ECS) é um algoritmo evolutivo híbrido que emprega a estrutura CS para explorar o espaço de busca, possibilitando controlar melhor tanto a intensificação por busca local (OLIVEIRA; LORENA, 2004) quanto a velocidade de convergência (COSTA; OLIVEIRA; LORENA, 2010).

O ECS tenta localizar áreas de pesquisa promissoras enquadrando-as por agrupamentos que são representados por *centros* c. O número de máximo de agrupamentos pode ser fixado ou determinado dinamicamente de acordo com a largura das áreas de busca (tamanho do problema em questão) (OLIVEIRA; LORENA, 2004). A cobertura do *cluster* é determinada por uma métrica de distância que calcula a semelhança entre uma dada solução e o centro do *cluster*, c.

Distância de *Hamming* e *Euclidiana* são métricas populares em aplicações CS (OLIVEIRA; LORENA, 2004).

O ECS pode ser melhor compreendido considerando suas quatro partes conceitualmente independentes. O componente *Algoritmos Evolutivo* funciona como um gerador contínuo de soluções candidatas, evoluindo uma população independentemente dos demais componentes da estrutura CS. Indivíduos selecionados são apresentados ao componente *Agrupador Iterativo* para que o mesmo promova o agrupamento em um dos grupos já identificados ou crie um novo *cluster*, ativando ou criando um *cluster*, de acordo com a medida de similaridade empregada. Cada *cluster* ativado recebe proporções de votos e pode ser melhor explorado posteriormente. Um *cluster* não ativado pode ser removido e a respectiva área de pesquisa, é esquecida (OLIVEIRA; LORENA, 2004).

Considerando $\mathscr{G}_{j\ (j=1,2,\cdots)}$ como todos os *clusters* correntes, a seguinte regra define quando um novo *cluster* deve ser criado:

$$c_{new} = s_k \text{ if } \wp(s_k, c_j) > r_j, \forall \mathscr{G}_j, \text{ ou} \tag{4.1}$$

Quando um indivíduo é bastante semelhante a um *cluster* existente, a regra de assimilação é aplicada ao centro do *cluster* ativado, c_i e ao indivíduo semelhante, s_k, produzindo em um novo posicionamento do centro c_i':

$$c_i' = c_i \oplus \beta(s_k \ominus c_i), \text{ contrário.} \tag{4.2}$$

em que $\oplus$ e $\ominus$ são operações abstratas sobre c_i e s_k significando, respectivamente, adição e subtração de soluções. A operação $(s_k \ominus c_i)$ significa quão distantes são as soluções s_k e c_i, considerando a métrica de distância. Uma taxa de aprendizado, β, desta diferença é usada para atualizar c_i para c_i'.

A assimilação desempenha um papel importante no processo de agrupamento, uma vez que os *clusters* devem enquadrar e representar uma área de busca onde existe uma sobreamostragem de soluções candidatas, identificando provavelmente uma área de busca promissora.

O *Módulo Analisador* verifica os recebidos por cada *cluster* em intervalos de geração regulares, indicando quais são promissores. A densidade é o número de votos ocorridos recentemente no *cluster*. Em algumas aplicações, o analisador também é responsável pela remoção de *clusters* de baixa densidade (OLIVEIRA; LORENA, 2004). Finalmente, o componente de *Busca Local* fornece um mecanismo de exploração em supostas áreas promissoras, enquadradas pelos *clusters* mais votados.

4.3 BRKeCS

O BRKeCS segue o mesmo princípio de implementações anteriores do *BRKGA+CS*, porém com modificações nos operadores de cruzamento e busca local para melhor adequá-lo ao problema em questão. O pseudocódigo usado para implementação do BRKeCS é apresentado por meio do Algoritmo 6.

Algoritmo 6: BRKeCS

Input: Conjunto de parâmetros

$f \leftarrow \infty$; `// inicialize o valor da melhor solução encontrada`
repeat
  Gere uma população P com vetores de n chaves aleatórias;
  Decodifica($p \in P$); `// Avalie o custo de cada solução em P`
  Particione P em dois grupos, um com as soluções mais bem avaliadas e outro com o restante: P_e e P_{ne};
  $P^+ \leftarrow P_e$; `// Inicialize a próxima geração com a elite`
  Gere um conjunto de mutantes P_m;
  $P^+ \leftarrow P^+ \bigcup P_m$; `// Adicione` P_m `à população da próxima geração`
  for $i \leftarrow 1$ **to** $(|P| - |P_e| - |P_m|)$ **do**
    Escolha um pai a aleatoriamente de P_e;
    Escolha um pai b aleatoriamente de P_{ne};
    $a \leftarrow -\alpha$;
    $b \leftarrow 1 + \alpha$;
    for $j \leftarrow 1$ **to** n **do**
      Gere um número aleatório ρ no intervalo $[0,1]$;
      $r \leftarrow \rho * (b - a)$;
      $c[j] \leftarrow a[j] + r * (b[j] - a[j])$;
    $P^+ \leftarrow P^+ \bigcup c$; `// Adicione` c `à população da próxima geração`
    for $i \leftarrow P_e$ **to** *Tamanho(P)* **do**
      $ECS(P^+[i])$; `// Adiciona` $P^+[i]$ `ao cluster`
  $P \leftarrow P^+$; `// Atualiza a população`
  Ache a melhor solução X^+ em P;
  if $Decodifica(X^+) < Decodifica(f)$ **then**
    $f \leftarrow X^+$;
until *Critério de parada for satisfeito*;
return f;

O BRKGA possibilita a simplificação de alguns componentes do CS, necessitando apenas implementar o decodificador e a heurística de Busca Local, conforme mostrado na Figura 7.

4.3.1 Cruzamento

O operador de cruzamento é o principal operador evolutivo da metaheurística, combinando basicamente as subestruturas de dois cromossomos para produzir novas estruturas. Um

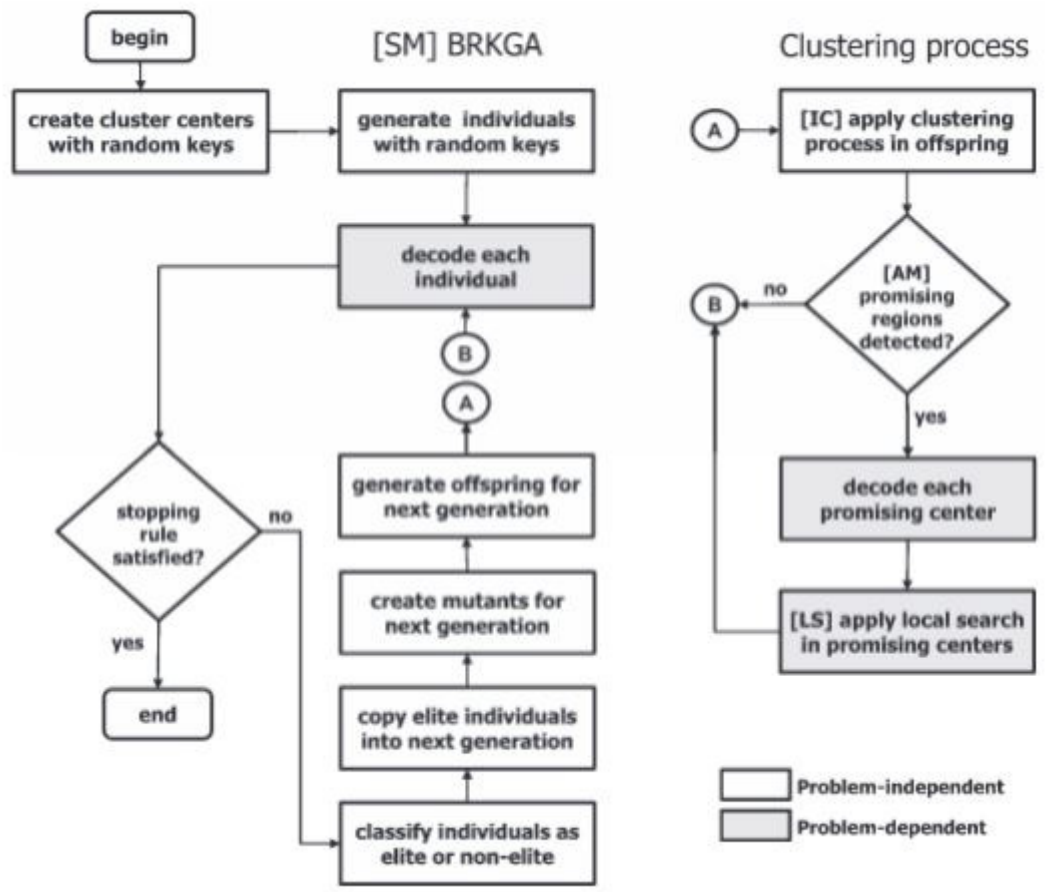

Figura 7 – Projeto conceitual do BRKeCS

Fonte: (CHAVES *et al.*, 2016)

cromossomo tem associado um nível de aptidão que está correlacionado com o valor da função objetivo correspondente à solução candidata que ele codifica.

Ao contrário do BRKGA regular, o BRKeCS emprega o *Blender Crossover* (BLX-α) (ESHELMAN; SHAFFER,) para o qual dado dois cromossomos p_1 e p_2, o cromossomo q é produzido por $q = p_1 + \alpha(p_2 - p_1)$ onde $\alpha \simeq U(-\alpha, 1+\alpha)$, com U representando uma distribuição uniforme. Normalmente, os valores α são de $0,5$ ou $0,25$ imprimindo um comportamento mais ou menos diversificado, respectivamente.

O BLX-α tende a gerar indivíduos,com probabilidade uniforme, no interior do hiperplano convexo centrado no segmento de reta traçado entre os dois indivíduos pais, admitindo uma expansão proporcional ao α usado. Se ambos os pais estão próximos um do outro, sucessivos cruzamentos dão um comportamento mais local ao algoritmo. Por outro lado, se os pais estão distantes entre si, a busca passa a ser mais global (ESHELMAN; SHAFFER,).

4.3.2 Busca Local

A busca local também desempenha um papel importante para obtenção de soluções de qualidade pela metaheurística híbrida. A presente proposta usa a heurística 2-*opt* para estruturar e explorar a vizinhança em torno dos indivíduos promissores encontrados pelo BRKeCS.

A heurística 2-*opt* foi proposta por Croes (1958) para resolver o problema do Caixeiro Viajante (HOFFMAN; PADBERG; RINALDI, 2013). O objetivo da heurística é melhorar a solução considerada promissora avaliando uma vizinhança da solução candidata. Uma Busca Local completa de 2-*opt*, a partir de uma solução candidata, pode avaliar $O(n^2)$ (sendo n o número de variáveis codificadas) possíveis soluções vizinhas obtidas pela remoção e reinserção de arestas. Esta técnica pode ser aplicada a qualquer problema de sequenciamento de padrões (JACKSON J.AND GIRARD, 2010; OLIVEIRA; LORENA, 2007).

4.3.3 Decodificador

o decodificador, juntamente com a busca local, formam os dois únicos operadores específicos para o problema de *FlowShop Permutacional*. Todos os demais operadores do BRKeCS, como seleção, cruzamento e mutação, são genéricos e trabalham no espaço de busca de chaves viciadas. A busca local, bem como a avaliação de aptidão de soluções candidatas promovem a busca no espaço real do problema, que este caso é o espaço $O(n!)$ formado por todas as possíveis permutações de tamanho n.

O *Decoder* é projetado para garantir que, dado como entrada um vetor de chaves aleatórias (vetor de números reais), seja produzida uma solução discreta (permutação) viável para o *PFSP*. A Eq. 4.3 representa o procedimento de decodificação de soluções candidatas.

$$\Phi(\mathbb{R}^n) \rightarrow \mathbb{N} \tag{4.3}$$

onde n é uma sequencia de tarefas (tamanho do indivíduo).

4.3.4 Assimilação

A operação de assimilação ocorre sempre que um *cluster* é ativado por um indivíduo gerado pela metaheurística, causando uma perturbação na localização do centro, em geral, proporcional â distância deste para o indivíduo assimilado (OLIVEIRA; LORENA, 2004).

No presente trabalho, foi implementada a assimilação baseada em *Path-Relinking* (LAGUNA; MARTI, 2012) que gera várias soluções candidatas uniformemente espaçadas no caminho entre indivíduo assimilado e centro do *cluster*. O processo de assimilação corresponde a uma busca local interna ao *clusters* (intensificação), usando parâmetros de início e fim de busca, e retornando, ao final, a melhor solução encontrada a ser assumida como novo centro.

A implementação do BRKeCS, especificamente a codificação em chaves viciadas, permite que a amostragem de soluções candidatas gerada pelo *path-relinking* ocorra no espaço codificado em $\mathbb{R}$, onde é mais intuitivo traçar um caminho formado por pontos a serem avaliados pela função objetivo. Todavia, a avaliação de aptidão de cada ponto candidato evidentemente precisa passar pelo decodificador.

4.3.5 Parâmetros de desempenho

O conjunto de parâmetros de desempenho do BRKeCS pode ser dividido em 3 grupos: parâmetros de busca local, parâmetros de metaheurística e parâmetros de agrupamento.

1. Parâmetros da busca local 2-opt:
   - $altura$: esse parâmetro é responsável por definir um número de trocas realizadas por busca local 2-OPT dentro de um esmo segmento de largura;
   - $largura$: responsável por definir o segmento de tarefas dentro uma solução de PFSP a qual seriam realizadas as trocas;
   - $rmax$: define o número de vezes que a busca local é realizada em um mesmo *cluster* dado como promissor.
2. Parâmetros do BRKeCS:
   - p_e: porcentagem da população seguinte que será preenchida com os melhores indivíduos da população anterior;
   - p_m: porcentagem de indivíduos da geração seguinte que será formada por modificações (mutações) nos indivíduos da população anterior que não pertencem a elite;
   - p: número de indivíduos amostrados em cada geração.
3. Parâmetros do CS:
   - $numcl$: quantidade de *clusters* amostrados em cada geração;
   - λ: porcentagem de indivíduos em um *cluster* com relação ao tamanho da população para que o mesmo seja considerado promissor.

4.4 BRKeCS aplicado ao problema de FlowShop Permutacional

O algoritmo BRKeCS foi implementado usando a linguagem C e avaliado em um notebook com CPU Core I3 (2,4 GHz) e memória de 3,8 GiB. Foram realizados testes iniciais com intuito de avaliar a competitividade do BRKeCS em instâncias propostas por Taillard

(1990). Os resultados obtidos foram comparados com os obtidos pelos algoritmos MOACSA (B. YAGMAHAN, 2010), CR (MC) (RAJENDRAN, 1995b), ACO e HAMC (HAMC1, HAMC2, HAMC3) (D. NOORUL HAQ A.; R., 2005)

Os parâmetros de desempenho do BRKeCS foram sintonizados conforme encontrado na literatura e equiparado aos parâmetros dos algoritmos competidores (execução de tempo, números de repetições, etc.). A Tabela 1 mostra a erro percentual dos algoritmos para instâncias 20x20. Como pode ser observado o BRKeCS tem desempenho equivalente ou superior aos demais algoritmos.

Tabela 1 – Erro residual para instâncias 20x20

Instância	HAMC1	HAMC2	HAMC3	CR(MC)	MOACSA	ACOC	GA	BRKeCS
Ta021	2,51	4,49	4,77	7,41	0	5,44	0,45	0,13
Ta022	11,96	11,96	16,22	3,42	2,56	5,43	0	0
Ta023	0,46	0,91	3,94	0	4,31	6,06	7,13	0,08
Ta024	11,41	16,88	18,14	0,29	0,30	6,70	0	0,04
Ta025	0	1,36	3,02	5,16	2,19	7,90	0,21	0,21

Para as instâncias maiores, o BRKeCS mostrou desempenho satisfatório quando comparado com o ACO. A Tabela 2 apresenta todos os resultados das instâncias consideradas de dificuldade alta e média.

Tabela 2 – Comparação BRKeCS com ACO

Instances	Nº de tarefas	Nº de máquinas	Ótimo conhecido	ACO	BRKeCS
Ta041	50	10	3025	4036	3109
Ta051	50	20	3875	7080	3923
Ta061	100	5	5493	6245	5493
Ta071	100	10	5779	7563	5782
Ta081	100	20	6282	13270	6378

CAPÍTULO

5

CROSS-VALIDATED RACING (CVR)

Neste Capítulo, a proposta de sintonia de metaheurísticas por *Cross-Validated Racing* é apresentada, com ênfase em seu design conceitual e etapas.

5.1 Definição

O *Cross-Validated Racing* (CVR) é uma técnica de sintonização automática de metaheurísticas que emprega abordagem por corrida e validação cruzada, realizando avaliação de diferentes ajustes de parâmetros sobre diferentes conjuntos de instâncias de forma a possibilitar a racionalização do tempo de execução do experimento a partir de uma perspectiva de generalização por Aprendizado de Máquina.

5.2 Design conceitual

Dada uma base com diferentes instâncias do problema que se pretende resolver, o módulo de Pré-processamento do CVR analisa, formata e organiza as instâncias em k grupos ou *folds* de maneira aleatória para que possam ser usadas pelo módulo de Validação Cruzada e módulo de Corrida. No módulo de Validação Cruzada, $k-1$ grupos são separados para um processo de treinamento e 1 grupo para teste.

O treino é executado pelo módulo de corrida que analisa os $k-1$ grupos com o objetivo de gerar um conjunto de parâmetros ótimos que será performado no grupo de teste (que não participou do treinamento) para avaliar sua capacidade de generalização. Este processo em que participam o módulo de Validação Cruzada e o módulo de Sintonização ou Corrida é feito k vezes, ou seja, até que todos os grupos gerados pelo módulo de Pré-processamento sejam utilizados como grupo de teste uma vez. Ao final, serão geradas k configurações candidatas ótimas que passarão pelo módulo de Análise Estatística onde será comprovada sua relevância estatística e gerada a configuração ótima para o modelo geral. Essas etapas foram explicadas de

maneira detalhada ao longo do capítulo, a Figura 8 representa o processo completo em um nível de abstração mais geral.

Figura 8 – Design conceitual do CVR

Fonte: Autor

5.3 Pré-processamento

A etapa de pré-processamento no CVR compreende a captação, organização, tratamento e a preparação dos dados. É uma etapa que possui fundamental relevância no processo de sintonização que considera desde a correção de dados até a formatação dos dados para as etapas seguintes.

As instâncias são altamente suscetíveis ao ruído e inconsistências, principalmente as realísticas. A fim de ajudar a melhorar a qualidade dos dados e, consequentemente, melhorar a eficiência e facilidade do processo de sintonização, o pré-processamento de dados é utilizado na preparação e transformação do conjunto de instâncias.

5.4 Sintonia Cross-Validada

Após as instâncias estarem no formato correto, devem ser organizadas em *folds* destinados ao experimento de validação cruzada para avaliar o *erro do método de aprendizado* gerado pela etapa de sintonia por método de corrida. O CVR estima o erro do método de aprendizado para instâncias não incluídas no processo de treino e validação, ou seja, o CVR estima o erro residual para novas instâncias.

5.4.1 Estimativa do erro de aprendizagem

De fato, a teoria da generalização diz que uma máquina M é capaz de generalizar se, no conjunto de treinamento $T = [x_0, x_1, \ldots, x_n; y_0, y_1, \ldots, y_n]$ contendo as respostas $y_0, y_1, \ldots, y_n$ dadas por um supervisor para as entradas de valores $x_0, x_1, \ldots, x_n$, a máquina é capaz de prever as respostas para as variáveis de entrada que não estão contidas em T (VAPNIK, 1995).

Para analisar a estimação de erro do CVR, é necessário considerar os conceitos de **bias** e **variância**. O bias de um estimador é definido como o valor do erro real menos o valor do erro esperado: $\varepsilon - E[\widehat{\varepsilon}]$. A variância é determinada por $E[(\varepsilon - E[\widehat{\varepsilon})^2]$, ou seja, o bias mensura a média de precisão do erro estimado, enquanto a variância mensura a variabilidade do erro estimado.

5.4.2 Validação cruzada

No processo de validação cruzada *k-fold*, a base de instâncias S_n é aleatoriamente particionada em k grupos de instâncias mutualmente exclusivas de mesmo tamanho $P = P_1, P_2, ..., P_k$ em que $k-1$ grupos são utilizados pelo modelo de estimação em um processo de treinamento e o grupo restante para teste. O processo é repetido até que todos os grupos tenham passado por um estado de *treino* e *validação*. O erro estimado do modelo depois da validação é dado por:

$$\widehat{\varepsilon}(S_n, P) = \frac{1}{n} \sum_{i=1}^{k} \sum_{j=1}^{P_k} RACE(P_j, teste) \tag{5.1}$$

onde $RACE(P_j, teste)$ é o erro resultante da aplicação do algoritmo de corrida treinado com o grupo P_j e testado no grupo de teste, ou seja, o erro estimado pela validação cruzada é uma média de erros do modelo treinados com as respectivas partições P_j. É esperado que as sucessivas repetições da validação cruzada estabilizem o erro de estimação, reduzindo a variança e aumentando a confiabilidade nos resultados adquiridos.

5.4.3 Algoritmo de corrida

Com os conjuntos de instâncias geradas, o algoritmo de corrida é aplicado em um processo de treinamento e teste. No processo de treino, a avaliação do desempenho de uma configuração candidata θ é adquirida incrementalmente. De fato, a média epírica:

$$\widehat{\mu}_k(\theta) = \sum_{j=1}^{k} c_j^{\theta}, \tag{5.2}$$

dos resultados para k experimentos é uma estimativa do critério dado por

$$\mu(\theta) = \int c dP_C(c|\theta, i) dP_I(i) \tag{5.3}$$

desde que as instancias $i_1, i_2, ..., i_k$ sejam amostradas de acordo com a medida P_I e os custos $c_1^{\theta}, c_2^{\theta}, ..., c_k^{\theta}$ das melhores soluções encontradas em uma execução da configuração θ em um tempo t sejam a realização de variáveis estocásticas descritas pelas medidas desconhecidas $P_C(c|\theta, i_1), P_C(c|\theta, i_2), ..., P_C(c|\theta, i_3)$.

Baseado nessas premissas, pode-se concluir que uma sequência de estimadores $\widehat{\mu}_1(\theta), \widehat{\mu}_2(\theta), ...$ podem ser construídos onde $\widehat{\mu}_1(\theta) = c_1^{\theta}$ é simplesmente o custo obtido pela execução de uma configuração θ em um determinado tempo t sobre uma instância i_1 e o termo genérico $\widehat{\mu}_k(\theta)$ é dado por:

$$\widehat{\mu}_k(\theta) = \frac{(k-1)\widehat{\mu}_{k-1}(\theta) + c_k^{\theta}}{k} \tag{5.4}$$

onde c_k^{θ} é o custo obtido por executar a configuração θ em um tempo t sobre a $k-th$ instância i_k de acordo com o a medida P_I, em outras palavras, $\widehat{\mu}_k(\theta)$ é uma média empírica do vetor de observações dadas por:

$$c^k(\theta) = (c_1^{\theta}, c_2^{\theta}, ..., c_k^{\theta})$$

e esse vetor pode ser adquirido adicionando o termo c_k^{θ} ao vetor definido como:

$$c^{k-1}(\theta) = (c_1^{\theta}, c_2^{\theta}, ..., c_{k-1}^{\theta})$$

cuja média é a estimativa $\widehat{\mu}_{k-1}(\theta)$.

A variância desta sequência de estimativas diminui com $1/k$ e, portanto, a estimativa do desempenho do candidato θ fica mais nítida quando k aumenta e converge para a verdadeira expectativa $\mu(\theta)$.

Dada a possibilidade de construir a sequência de estimativas, $\mu_1(\theta), \mu_2(\theta), ...$ para cada candidato $\theta \in \Theta$, um algoritmo de corrida constrói incrementalmente em paralelo tais sequências para todos os candidatos em Θ e, logo que sejam obtidas provas suficientes que a estimativa $\mu(\theta^{'})$

para um determinado candidato θ' é melhor que a estimativa $\mu(\tilde{\theta})$ de algum outro candidato $\tilde{\theta}$, é descartado $\tilde{\theta}$ de avaliações adicionais.

O algoritmo, portanto, gera uma sequência de conjuntos aninhados de configurações candidatas:

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq ...,$$

iniciando com $\Theta_0 = \Theta$, sendo que o passo de um conjunto Θ_{k-1} para Θ_k é obtido descartando algumas configurações sub-ótimas com base em informações disponíveis no passo k.

No passo k, quando o conjunto de candidatos ainda na corrida é Θ_{k-1}, uma nova instância i_k é selecionada. Cada candidato $\theta \in \Theta_{k-1}$ é testado em i_k e cada custo observado c_k^{θ} é anexado ao respectivo vetor $c^{k-1}(\theta)$ para formar os diferentes vetores $c^k(\theta)$, um para cada $\theta \in \Theta_{k-1}$. O Passo k termina definindo o conjunto Θ_k eliminando de Θ_{k-1} as configurações sub-ótimas utilizando o teste estatístico de *Friedman* que compara os vetores $c_k(\theta)$ para todos $\theta \in \Theta_{k-1}$.

Esse processo guia a busca pela melhor estrutura do modelo, acelerada pelo descarte de candidatos inferiores logo que sejam reunidas provas estatísticas de sua baixa qualidade enquanto solução candidata. De fato, a avaliação da medida de $\mu(\theta)$ em relação ao genérico candidato θ, pode ser realizado de forma incremental: sendo a média de K erros, cada um com relação a um dos K exemplos no conjunto de dados, esta quantidade pode ser aproximada pela média $\mu_k(\theta)$ que é a configuração de parâmetros ótimos para cada passo da validação cruzada. Cada K teste de Validação Cruzada gera uma configuração ideal de parâmetros, contudo o objetivo é encontrar a configuração de parâmetros que tenha ótima performance sob toda base de instâncias e instâncias desconhecidas. Deste modo, o CVR utiliza um método estatístico baseado no estimador $\mu = \int c dP_C(c|\theta,i) dP_I(i)$ que permite encontrar uma configuração de parâmetros ótima com relevância estatística que possua menor tempo computacional entre os parâmetros escolhidos sob toda a base de instâncias.

5.5 Testes estatísticos

Se os grupos de instâncias gerados na etapa de pré-processamento forem representativas, ou seja, cada grupo for uma amostra representativa da base total de instâncias, as etapas seguintes retornarão k configurações igualmente eficazes e com significância estatística para a resolução do problema em questão, contudo, como não há como garantir a regularidade da base de instância que será analisada pelo modelo (podem existir grupos de instâncias maiores ou menores, valores errados e etc) , existe a possibilidade de no processo de validação cruzada, alguns grupos apresentem configurações ótimas de baixa qualidade com relação aos demais, logo, é necessário um meio de verificar esses casos e tratá-los caso aconteçam.

O teste estatístico utilizado para garantir a significância do resultado do CVR é o de

Kruskal-Wallis (ou Anova não-paramétrico). Diferentemente do que ocorre com a Análise de Variância de Um Critério (ou ANOVA de Fisher, teste paramétrico), *Kruskal-Wallis* não exige as suposições de normalidade da variável, nem homogeneidade de variâncias entre os tratamentos. É caracterizado como teste livre de distribuição, ou seja, a distribuição teórica populacional dos dados não precisa ser estimada pelas médias ou variâncias amostrais para sua correta aplicação. Quando se detecta diferenças significativa entre os tratamentos pelo teste de *Kruskal-Wallis*, usualmente são efetuadas comparações múltiplas envolvendo todos os pares de tratamentos. No final dessa etapa, tem-se uma configuração otimizada para todo o conjunto de instâncias e capaz de performar bem em instâncias novas.

Uma vez que os dados provem da observação do erro residual médio da aplicação do Algoritmo de Corrida considerando cada grupo (sendo um grupo representado um *fold k*) como um grupo de teste uma vez e k amostras aleatórias (considera-se agora a amostra aleatória como um grupo de instâncias k e não mais apenas uma instância) e independentes com tamanhos amostrais $n_1, n_2, ..., n_k$ sendo $N = n_1 + n_2 + ... + n_k$ o número total de elementos considerados em todas as amostras. Dessa forma. tem-se que os elementos de cada amostra j, com $j \in 1, 2..., k$ formam uma distribuição contínua com função de distribuição F_j que se relacionam atraves da relação:

$$F_j(t) = F(t - \tau_j), -\infty < \infty,$$

para $j = 1, 2, ..., k$ em que F é uma função de distribuição para uma distribuição contínua com mediana desconhecida e τ_j é o efeito do processo de treinamento com o grupo j como teste. Neste caso, a hipótese nula H_0 é que não há diferença entre os efeitos $\tau_1, ..., \tau_k$, isto é:

$$\tau_1 = \tau_2 = ... = \tau_k$$

Está hipótese garante que $F_1 = F_2 = ... = F_k$. Para aplicar o método de *Kruskal-Wallis*, primeiramente ordena-se todas as N observações das k amostras da menor para a maior observação e considera-se r_{ij} como sendo o posto dos custos finais X_{ij}:

$$R_i = \sum_{j=1}^{n_i} r_{ij} \quad e \quad R_i = \frac{R_i}{n_i}, i = 1, ..., k.$$

Deste modo, $R1$ é a soma dos postos dos elementos da amostra 1 e R_i é o posto médio destas mesmas observações. A estatística de *Kruskal-Wallis* H, é dada por:

$$H = \frac{\frac{12}{N(N+1)} \sum_{i=1}^{k} n_i (R_i - \frac{N+1}{2})^2}{1 - \frac{\sum_{j=1}^{a} t_j^3 - t_j}{N^3 - N}} = \frac{(\frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i}) - 3(N+1)}{1 - \frac{\sum_{j=1}^{a} t_j^3 - t_j}{N^3 - N}}$$

em que t_j é o tamanho do grupo de elementos repetidos j. Uma observação que não se repete é considerada como um grupo de tamanho 1. Esta estatística tem, aproximadamente, uma distribuição qui-quadrado com $k - 1$ graus de liberdade.

Pode-se resumir esse processo da seguinte maneira, dada as hipóteses:

$$\begin{cases} H_0 : \tau_1 = \tau_2 = ... = \tau_k \\ H-1 : \quad contrario \end{cases}$$

Ordena-se de forma crescente de magnitude os valores deste novo conjunto de dados e associa-se a cada valor seu posto correspondente, tendo cada posto o mesmo sinal do valor que este representa. Posteriormente é calculado o valor da estatística H. Em seguida, fixa-se o nível de significância α e se verifica os valores críticos referentes ao nível de significância fixado. Neste caso, calcula-se os valores Q_α de modo que $P[H > Q_\alpha] = \alpha$ (sob H_0). Se $H_{obs} > Q_\alpha$, a hipótese nula de que as amostras provém de grupos igualmente distribuídos é rejeitada. O p-valor é calculado da seguinte forma:

$$P - valor = P[X^2_{k-1} \geq H|H_0]$$

Quando rejeita-se a hipótese nula H_0 no teste de *Kruskal-Wallis*, indica que ao menos um dos grupos é diferente dos demais. Porém, não se tem a informação de quais são diferentes. Neste sentido, procedimento de comparações múltiplas nos permite determinar quais grupos são diferentes. Existe um procedimento simples para determinar quais os pares de grupos são diferentes.

Inicialmente, testa-se as diferenças para todos os pares de grupos. Quando o tamanho da amostra é grande, estas diferenças tem distribuição assintótica normal padrão. No entanto, quando tem-se um grande número de diferenças e se essas diferenças não são independentes, o procedimento de comparação múltipla deve ser ajustada de forma apropriada. Suponha que a hipótese de não haver diferença entre os k grupos foi testada e rejeitada ao nível de significância α. Uma alternativa é testar a significância dos pares de diferenças através da seguinte desigualdade:

$$|R_i - R_j| \geq Z(\frac{\alpha}{k(k-1)}) \sqrt[2]{\frac{N(N+1)}{12}(\frac{1}{n_i} + \frac{1}{n_j})} \tag{5.5}$$

em que:

- n_i e n_j são os tamanhos das amostras do grupo i e j respectivamente;
- $N = n_1 + n_2 + ... + n_k$ o número total de elementos considerados em todas as amostras;
- R_i e R_j é o efeito dos postos (ranks) dos grupos i e j respectivamente;
- $|R_i - R_j|$ é a diferença observada;
- $Z(\frac{\alpha}{k(k-1)}) \sqrt[2]{\frac{N(N+1)}{12}(\frac{1}{n_i} + \frac{1}{n_j})}$ é a diferença crítica.

Assim, se a Equação 5.5 ocorre, pode-se rejeitar a hipótese $\tau_i = \tau_j$ e concluir que $\tau_i \neq \tau_j$. Para k grupos, então o número de comparações é de $\frac{k(k-1)}{2}$.

Como dito anteriormente, o processo de geração dos k folds é aleatório, logo, determinados folds e grupos de folds podem não serem bons representantes da base total de instâncias, tal disposição das instâncias podem gerar configurações que dentro de seus respectivos testes são ótimas, contudo são de baixa qualidade para todo o conjunto de instâncias. O teste estatístico de *Kruskal-Wallis* identifica se existe tal configuração de baixa qualidade e, caso exista, a mesma é eliminada e a configuração ótima média é re-calculada.

CAPÍTULO

6

RESULTADOS COMPUTACIONAIS

Neste Capítulo, descrevem-se os experimentos computacionais realizados visando a validação da proposta de *Cross-Validated Racing* (CVR). Nos experimentos, a metaheurística híbrida BRKeCS foi utilizada devido a mesma ser considerada bastante competitiva e apresentar um número razoável de parâmetros de desempenho a serem ajustados.

Metaheurísticas desenvolvidas especialmente para a resolução de problemas *FlowShop* foram comparadas com o BRKeCS sintonizado pelo CVR para medir a diminuição do erro residual médio (distância para o ótimo conhecido). Experimentos de generalização também foram realizados para analisar a capacidade do CVR de inferir boas configurações para instâncias que não participaram do processo de treinamento.

Para corroborar a eficiência do CVR em diferentes topologias de espaços de busca, duas bases de instâncias foram utilizadas, uma base aleatória (TAILLARD, 2013b) e uma base realística e portanto estruturada(WATSON *et al.*, 2002).

6.1 Instâncias e metaheurística

Para os testes, foi utilizado um *benchmark* composto de instâncias propostas por Taillard (TAILLARD, 2013b) e instâncias realísticas geradas aleatoriamente com grau de dificuldade majorado através do correlacionamento de tarefa e máquinas, conforme proposto por (WATSON *et al.*, 2002).

O uso de *benchmarks* randômicos, como o do Taillard (2013b), para avaliar o desempenho de algoritmos de sequenciamento levanta uma importante questão: os resultados encontrados podem ser generalizados para instâncias reais ?. Pesquisadores geralmente assumem que algoritmos com performance superior em instâncias com características aleatórias também terão uma boa performance em instâncias realísticas, ou seja, mais estruturadas e com um padrão de construção definido. Contudo, pequenas mudanças na topologia do espaço de busca podem

afetar a performance de certos algoritmos tornado necessária uma investigação mais profunda para esses casos. Com o objetivo de analisar a robustez do CVR para um conjunto de instâncias com topologia do espaço de busca diferente, um segundo conjunto de experimentos foi realizado posteriormente com as instâncias realísticas propostas por Watson *et al.* (2002).

Usualmente, as instâncias utilizadas para comparar o desempenho de algoritmos são geradas aleatoriamente seguindo uma distribuição simples e geralmente não correspondem a nenhum caso real. A principal crítica com relação a essa abordagem é o risco de sobreajuste dos algoritmos para instâncias específicas. De fato, diferenças na topologia do espaço de busca podem gerar mudanças significativas na eficiência do algoritmo testado (WATSON *et al.*, 2002)

Kan e George (1976) introduziram métodos para criar dois tipos de problemas *FlowShop Permutacional*: Correlação de tarefa e gradiente de tempo. Em correlação de tarefas, um pequeno número de distribuições sem sobreposição são definidas e todos os tempos de processamento para uma dada tarefa são amostrados de uma dessas distribuições. No gradiente de tempo, impõe-se uma estrutura sintática não aleatória criando uma tendência nos tempos de processamento como uma função da máquina que processam as tarefas. Em problemas com gradiente de tempo positivo ou negativo, as tarefas requerem menos ou mais tempo nas primeiras máquinas que nas últimas.

As instâncias realísticas utilizadas neste trabalho são uma mistura de correlação de tarefa e correlação com máquinas. Nesse tipo de problema, os *ranks* relativos de tempo de processamento das tarefas são consistentes em todas as máquinas, ou seja, se uma tarefa tem um tempo maior de processamento na máquina 1, então certamente terá um tempo de processamento maior nas máquinas restantes (PANWALKAR; DUDEK; SMITH, 1973).

Instâncias consideradas de grande porte (500 tarefas e 20 máquinas) foram removidas do conjunto de treinamento e validação como parte da estratégia de avaliação do método, acrescentando-se uma etapa posterior, chamada de *precisão final*, na qual o desempenho da metaheurística sintonizada pelo CVR é comparado a outras formas de sintonização nessas instâncias. As instâncias menores, incluídas no processo de treinamento e validação, foram divididas em 5 *folds* mutuamente exclusivos (sem repetições de instâncias).

Nos experimentos realizados, cada execução do BRKeCS assume limites de 2.000.000 para chamadas da função objetivo (CFO) e 1.000 para tentativas para garantia de significância estatística. As soluções ótimas de cada instância foram usadas como referência para cálculo da acurácia, dada por:

$$Ac = \frac{1}{v}\sum_{i=1}^{v} \varepsilon_{y_i,\overline{y}_i} \tag{6.1}$$

onde v é o número de dados de validação e $\varepsilon_{y_i,\overline{y}_i}$ é o erro residual dado por: $\varepsilon_{y_i,\overline{y}_i} = \frac{y_i - \overline{y}_i}{y_i}$, sendo que y_i e $\overline{y_i}$ são o ótimo conhecido e ótimo encontrado, respectivamente.

6.2 Análise 1

Foram realizados testes com a comparação da metaheurística sintonizada pelo método proposto com outras metaheurísticas projetadas para o problema *FlowShop Permutacional*. A Tabela 4 apresenta os resultados dos erros residuais das metaheurísticas para as instâncias de Taillard (TAILLARD, 2013b). Pode-se observar que o BRKeCS ajustado pelo CVR apresentou os melhores resultados com um erro residual médio de menos de $2,3\%$.

A Tabela 3 apresenta os parâmetros encontrados pelo CVR para a análise 1.

Tabela 3 – Configuração de parâmetros gerada pelo CVR para a análise 1

altura	*largura*	p_e	p_m	*rmax*	λ	*p*	*numcl*
5	5	0,3003	0,2818	10	0,598	733	13

Tabela 4 – Comparação entre o BRKeCS ajustado pelo CVR e outras metaheuristicas aplicadas ao problema

Instâncias	$\varepsilon_{y_i,\bar{y}_i}CDS$	$\varepsilon_{y_i,\bar{y}_i}SA$	$\varepsilon_{y_i,\bar{y}_i}PAL$	$\varepsilon_{y_i,\bar{y}_i}CVR$
20x5	0,0949	0,0940	0,01082	-0,00016
20x10	0,1213	0,1860	0,01528	0,00012
20x20	0,0964	0,3260	0,1634	0,001
50x5	0,0610	0,03	0,0534	0,00098
50x10	0,1298	0,17	0,1403	0,00914
50x20	0,1577	0,28	0,1794	0,1506
100x5	0,0513	0,04	0,0251	0,00202
100x10	0,0915	0,11	0,0913	0,0135
100x20	0,1419	0,15	0,1555	0,02302
$\varepsilon_{y_i,\bar{y}_i}$	0,1050	0,1540	0,09272	0,0222

Na Tabela 5, estão as configurações geradas pelos 5 experimentos de ajuste de parâmetros realizados. Na Tabela 6, são apresentados 5 testes do BRKeCS sintonizado com as configurações ótimas achadas pelo CVR. O erro residual médio de CVR sobre instancias não usadas no treinamento é $0,0089$ ou $0,89\%$, ou seja, o CVR é capaz de generalizar com eficiência. No teste de validação, o conjunto de instâncias $20x5$ apresentou um erro residual médio menor que o ótimo conhecido na literatura.

Tabela 5 – Configuração de parâmetros gerada pelo CVR

Teste	*altura*	*largura*	p_e	p_m	*rmax*	λ	*p*	*numcl*
CVR1	3	4	0,3368	0,3314	3	0,2341	840	4
CVR2	4	5	0,3322	0,1792	3	0,4943	708	17
CVR3	5	5	0,3003	0,2818	5	0,5424	722	15
CVR4	5	5	0,3673	0,1411	2	0,3498	921	4
CVR5	5	5	0,3143	0,2821	3	0,598	689	22

Tabela 6 – Erro residual médio do BRKeCS ajustado com o CVR para instâncias não utilizadas no treinamento

Instâncias	$\varepsilon_{y_i,\bar{y}_i}CVR1$	$\varepsilon_{y_i,\bar{y}_i}CVR2$	$\varepsilon_{y_i,\bar{y}_i}CVR3$	$\varepsilon_{y_i,\bar{y}_i}CVR4$	$\varepsilon_{y_i,\bar{y}_i}CVR5$
20x5	0	0	0	0	**-0,0008***
20x10	0,0006	0	0	0	0
20x20	0	0,0004	0,0025	0	0,0021
50x5	0	0,0014	0,0003	0,0029	0,0003
50x10	0,0085	0,0107	0,0069	0,0022	0,01741
50x20	0,0147	0,0188	0,0199	0,0079	0,0140
100x5	0	0,0041	0,0034	0,0017	0,0009
100x10	0,0046	0,0132	0,0107	0,0208	0,0182
100x20	0,0171	0,0283	0,0316	0,0204	0,0177
200x10	0,0091	0,0116	0,0112	0,0165	0,0056
200x20	0,0222	0,0193	0,0256	0,0253	0,0179
$\varepsilon_{y_i,\bar{y}_i}$	0,0070	0,0098	0,0102	0,0089	0,0085
$Ac = 0,0089 == 99,1\%$					

6.3 Análise 2

Para analisar a estabilidade dos resultados do CVR, realizou-se o teste das instâncias mínimas que consiste em especificar o menor número de instâncias necessárias para que a CVR execute de maneira efetiva, ou seja, sem comprometer a qualidade do treinamento.

Os testes foram realizados por meio de 6 experimentos. O primeiro consiste em usar os melhores parâmetros encontrados pelo sintonizador treinado para instâncias do mesmo grupo de validação. O problema com esta abordagem é o custo computacional necessário para resolver problemas de tamanho maior. Por exemplo, para encontrar a solução de uma instância de 200x200, exigiria treinamento em grupos de instâncias 200x200, ou seja, um processo de alto custo computacional.

Os próximos 4 experimentos usam a CVR com 46%, 42%, 25%, 8% do total das instâncias. O último usa as melhores configurações de parâmetros encontrados na literatura para o algoritmo em diferentes problemas (D. NOORUL HAQ A.; R., 2005) (MORO *et al.*, 2017), (MAINIERI, 2014), (POMARI; CHAVES, 2014).

A Figura 9 representa as configurações ótimas encontradas para a maior instância (500x20), que não participou do processo de treinamento e validação, visando avaliar a capacidade de generalização da CVR em instâncias de maior custo computacional sem a necessidade de incluí-las no processo de aprendizagem.

O teste preliminar demonstra uma predisposição CVR para manter o erro residual baixo quando usado entre 42% e 25% das instâncias totais.

Figura 9 – Erro residual Médio para os 6 experimentos sob o maior grupo de instâncias

6.4 Análise 3

Com o objetivo de corroborar os resultados em instâncias realísticas, um novo conjunto de experiências foi realizado com uma base do tipo correlação mista em instâncias realísticas. A Figura 10 representa um erro residual médio para os dois experimentos em conjunto.

Como esperado, a sintonização CVR se comportou de maneira semelhante no grupo de instâncias realísticas aleatórias do tipo correlação mista, indicando satisfatória robustez do método com relação a instâncias sintetizadas de diferentes formas.

A Figura 11 representa o grau de melhoria no erro residual de acordo com o número de tarefas para o BRKeCS. Para pequenas instâncias, o ganho foi de 5%, seguido de instâncias médias com ganho de 20%. Usando o modelo validado, um experimento final foi realizado no maior conjunto de instâncias $(500x20)$ com ganho de $32,78\%$ como mostrado pela Figura 12.

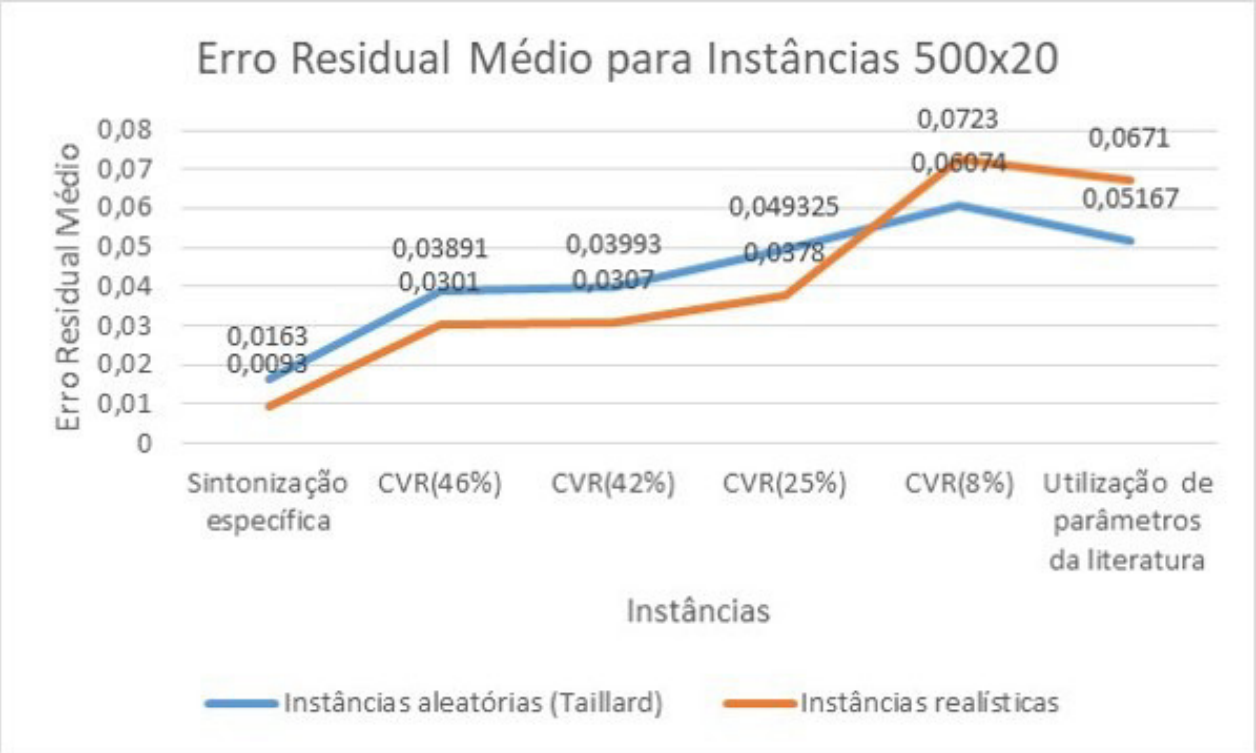

Figura 10 – Erro residual Médio para os 6 experimentos sob o maior grupo de instâncias do Taillard e realísticas

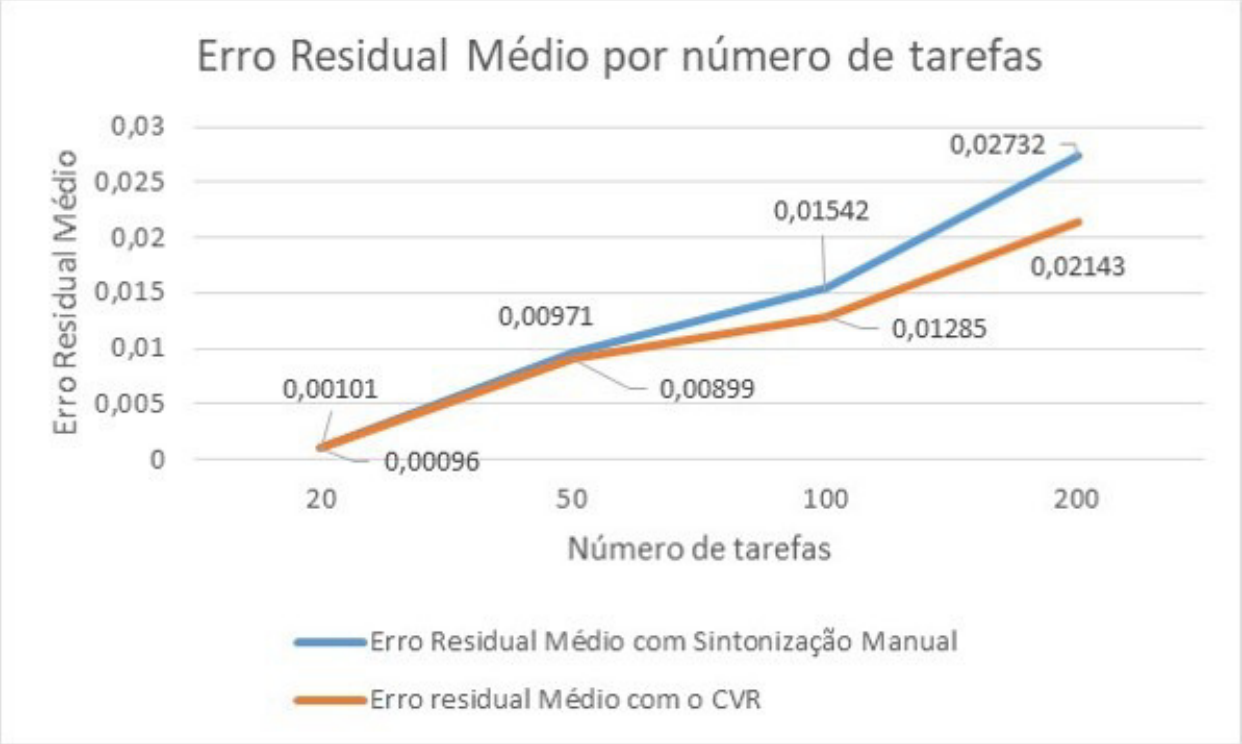

Figura 11 – Erro residual por número de tarefas

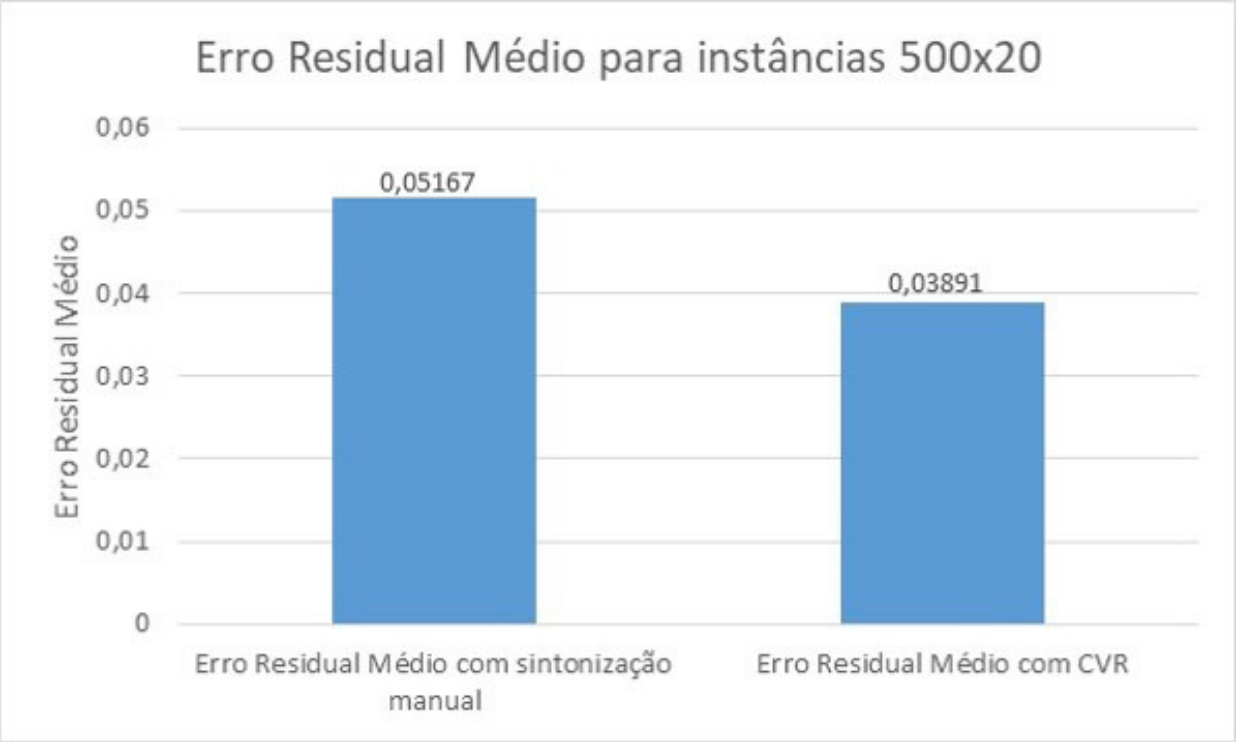

Figura 12 – Diferença entre os erros residuais do BRKeCS sintonizado e com ajuste manual para o maior grupo de instâncias

6.5 Considerações finais

O padrão de desempenho do método proposto se mantem satisfatório desde que mantenha-se o *dataset* de treinamento com tamanho compatível com a complexidade da sintonização que, neste trabalho, foi apurado ser entre 25% e 42% do total de instâncias disponível para o problema. O resultado também demonstra que o CVR, até onde foi testado neste trabalho, mantém robustez satisfatória com relação à mudança na natureza das instâncias de um mesmo problema.

CAPÍTULO

7

CONCLUSÃO

Metaheurísticas desempenham um importante papel para o desenvolvimento de *solvers* eficientes aplicáveis a problemas de otimização combinatória que apresentam alto custo computacional para sua solução ótima. Geralmente, uma metaheurística é sintonizada manualmente na base da tentativa e erro a partir do conhecimento do especialista. Tal abordagem é comum em várias pesquisas na área, sendo uma alternativa fácil e de baixo custo, apesar de sabidamente limitada.

O principal problema com a abordagem se refere a produção industrial em larga escala de metaheurísticas, que requer grande quantidade de tempo, trabalho intensivo e atenção de um especialista (geralmente quem implementou o algoritmo). Em nível acadêmico, a adoção dessa abordagem incorre no risco de invalidar conclusões para comparações experimentais de diferentes algoritmos. Portanto, o estudo da melhor sintonização dos parâmetros livres é de grande relevância para a construção de algoritmos mais eficientes.

7.1 Resumo das Contribuições

As principais contribuições deste trabalho estão relacionadas ao projeto do *Cross-Validated Racing*, que propõe uma técnica de baixo custo computacional para sintonização de metaheurísticas que utiliza uma perspectiva de aprendizado de máquina para prever configurações de parâmetros ótimas para instâncias que não fizerem parte do processo de treinamento (FONSECA; OLIVEIRA, 2017). Essa característica permite que muitas instâncias de grande porte possam ser retiradas do processo de treinamento, economizando tempo sem perda de qualidade das soluções.

A implementação do algoritmo híbrido BRKeCS também é uma importante contribuição. Modificações feitas no módulo de Busca Local e no operador de Cruzamento contribuíram para tornar o algoritmo competitivo para resolução de problemas *FlowShop* com resultados superiores

aos encontrados na literatura recente, considerando as mesmas circunstâncias experimentais.

7.2 Discussão

Devido a variedade de tipos de parâmetros que podem ser sintonizados, existe a possibilidade de mais de uma configuração candidata ser responsável pela geração de uma mesma solução de qualidade. Tal característica ocorre pela existência de famílias de configurações ótimas (uma vez que todas as configurações estão sob o mesmo grau de confiabilidade do teste estatístico). Essa característica abre a possibilidade de pesquisadores diferentes, projetando metaheurísticas variadas e que estejam utilizando o mesmo modelo de sintonização (CVR), possam comparar seus algoritmos de maneira mais confiável, pois apesar de seus modelos gerarem configurações ótimas diferentes, as configurações encontradas possuem o mesmo grau de confiabilidade estatística, sendo portanto equivalentes em nível de sintonização.

Em outras palavras, a superioridade da metaheurística **A** em relação à metaheurística **B** se dá unicamente pela qualidade do projeto do algoritmo **A**, uma vez seguida a mesma metodologia de sintonia. Tal comparação torna-se facilitada devido a diminuição da influência do ajuste de parâmetros no desempenho dos algoritmos em teste.

7.3 Trabalhos Futuros

Como trabalhos futuros, pretende-se ampliar os experimentos utilizando o CVR como metodologia de sintonia, favorecendo sua consolidação e reforçando-o como contribuição científica. Neste trabalho, o CVR foi testado para uma única metaheurística para diferentes conjuntos de instâncias de um mesmo problema.

Pretende-se futuramente aplicar o processo de sintonização em um modelo online, ou seja, que consiga realizar a sintonização a medida que tenta encontrar a melhor solução para o problema.

Também é importante investigar mais profundamente a capacidade do CVR de padronizar a sintonização e permitir a comparação de diferentes metaheurísticas feitas por diferentes pesquisadores. Apesar de ser uma consequência lógica do CVR, não foi testada a comparação de duas metaheurísticas diferentes sintonizadas pelo CVR usando diferentes *datasets* de treinamento. Também advoga-se a relevância de se reproduzir futuramente os experimentos com diferentes metaheurísticas e diferentes problemas.

REFERÊNCIAS

ADENSO-DIAZ, B.; LAGUNA, M. Fine-tuning of algorithms using fractional experimental designs and local search. **Operations research**, INFORMS, v. 54, n. 1, p. 99–114, 2006. Citado 2 vezes nas páginas 17 e 41.

ALLAHVERDI, A. Two-stage production scheduling with separated set-up times and stochastic breakdowns. **Journal of the Operational Research Society**, Taylor & Francis, v. 46, n. 7, p. 896–904, 1995. Citado na página 48.

____. A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. **Computers & Operations Research**, Elsevier, v. 31, n. 2, p. 157–180, 2004. Citado na página 48.

ALLAHVERDI, A.; AL-ANZI, F. S. Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for www applications. **Computers & Operations Research**, Elsevier, v. 29, n. 8, p. 971–994, 2002. Citado na página 48.

ALVAREZ, L. **Design Optimization based on Genetic Programming,, 2000**. Tese (Doutorado) — University of Bradford, 2000. Citado na página 38.

ANSÓTEGUI, C.; SELLMANN, M.; TIERNEY, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In: SPRINGER. **International Conference on Principles and Practice of Constraint Programming**. [S.l.], 2009. p. 142–157. Citado na página 34.

AYDILEK, H.; ALLAHVERDI, A. A polynomial time heuristic for the two-machine flowshop scheduling problem with setup times and random processing times. **Applied Mathematical Modelling**, Elsevier, v. 37, n. 12-13, p. 7164–7173, 2013. Citado na página 43.

B. YAGMAHAN, M. M. Y. A multi-objective ant colony system algorithm for flow shop scheduling problem. **Expert Systems with Applications**, v. 37, p. 1361–1368, 2010. Citado na página 71.

BAI, D. Asymptotic analysis of online algorithms and improved scheme for the flow shop scheduling problem with release dates. **International Journal of Systems Science**, Taylor & Francis, v. 46, n. 11, p. 1994–2005, 2015. Citado na página 44.

BAI, D.; REN, T. New approximation algorithms for flow shop total completion time problem. **Engineering Optimization**, Taylor & Francis, v. 45, n. 9, p. 1091–1105, 2013. Citado na página 43.

BAKER, K. R. **Introduction to sequencing and scheduling**. [S.l.]: John Wiley & Sons, 1974. Citado 2 vezes nas páginas 46 e 47.

BARR, R.; GOLDEN, B.; KELLY, J.; RESENDE, M.; STEWART, W. R. Designing and reporting on computational experiments with heuristic methods. v. 1, p. 9–32, 09 1995. Citado 2 vezes nas páginas 17 e 21.

BARTZ-BEIELSTEIN, T.; LASARCZYK, C. W.; PREUSS, M. Sequential parameter optimization. In: IEEE. **Evolutionary Computation, 2005. The 2005 IEEE Congress on**. [S.l.], 2005. v. 1, p. 773–780. Citado na página 41.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimizantion. **ORSA Journal on Computing**, v. 2, p. 154–160, 1994. Citado na página 65.

BEN-DAYA, M.; AL-FAWZAN, M. A tabu search approach for the flow shop scheduling problem. **European journal of operational research**, Elsevier, v. 109, n. 1, p. 88–95, 1998. Citado na página 56.

BIRATTARI, M. **Tuning Metaheuristics: A Machine Learning Perspective**. 1st ed. 2005. 2nd printing. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. 74 p. ISBN 3642004822, 9783642004827. Citado 3 vezes nas páginas 17, 22 e 23.

____. **Tuning Metaheuristics. A Machine Learning Perspective**. 2. ed. [S.l.]: Springer, 2009. ISBN 978-3-642-00482-7. Citado 4 vezes nas páginas 23, 24, 25 e 29.

BONNEY, M.; GUNDRY, S. Solutions to the constrained flowshop sequencing problem. **Journal of the Operational Research Society**, Taylor & Francis, v. 27, n. 4, p. 869–883, 1976. Citado na página 49.

BOX, G. E.; DRAPER, N. R. **Empirical model-building and response surfaces.** [S.l.]: John Wiley & Sons, 1987. Citado 3 vezes nas páginas 37, 38 e 39.

BREEDAM, A. V. Improvement heuristics for the vehicle routing problem based on simulated annealing. **European Journal of Operational Research**, v. 86, n. 3, p. 480 – 490, 1995. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/037722179400064J>. Citado na página 21.

BULLNHEIMER, B.; HARTL, R. F.; STRAUSS, C. Applying the ant system to the vehicle routing problem. In: **Meta-heuristics**. [S.l.]: Springer, 1999. p. 285–296. Citado na página 58.

CAMPBELL, H. G.; DUDEK, R. A.; SMITH, M. L. A heuristic algorithm for the n job, m machine sequencing problem. **Management science**, INFORMS, v. 16, n. 10, p. B–630, 1970. Citado 2 vezes nas páginas 49 e 50.

CHAVES, A. A.; LORENA, L. A. N.; SENNE, E. L. F.; RESENDE, M. G. Hybrid method with cs and brkga applied to the minimization of tool switches problem. **Computers & Operations Research**, Elsevier, v. 67, p. 174–183, 2016. Citado 2 vezes nas páginas 64 e 68.

CHENG, T. E.; GUPTA, J. N.; WANG, G. A review of flowshop scheduling research with setup times. **Production and operations management**, Wiley Online Library, v. 9, n. 3, p. 262–282, 2000. Citado na página 48.

CHUNG, Y.-H.; TONG, L.-I. Makespan minimization for m-machine permutation flowshop scheduling problem with learning considerations. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 56, n. 1-4, p. 355–367, 2011. Citado na página 44.

CLEVELAND, G. A.; SMITH, S. F. Using genetic algorithms to schedule flow shop releases. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the third international conference on Genetic algorithms**. [S.l.], 1989. p. 160–169. Citado na página 61.

COLORNI, A.; DORIGO, M.; MANIEZZO, V.; TRUBIAN, M. Ant system for job-shop scheduling. **Belgian Journal of Operations Research, Statistics and Computer Science**, v. 34, n. 1, p. 39–53, 1994. Citado na página 58.

CONOVER, W. J.; CONOVER, W. J. Practical nonparametric statistics. Wiley New York, 1980. Citado 2 vezes nas páginas 35 e 36.

CONWAY, R.; MAXWELL, W. **MLW Theory of scheduling**. [S.l.]: Addison-Wesley, 1967. Citado na página 46.

CORNE, D. W.; KNOWLES, J. D. No free lunch and free leftovers theorems for multiobjective optimisation problems. In: SPRINGER. **International Conference on Evolutionary Multi-Criterion Optimization**. [S.l.], 2003. p. 327–341. Citado na página 17.

COSTA, D.; HERTZ, A. Ants can colour graphs. **Journal of the operational research society**, Springer, v. 48, n. 3, p. 295–305, 1997. Citado na página 58.

COSTA, T. S.; OLIVEIRA, A. C. M. de; LORENA, L. A. N. Advances in clustering search. In: CORCHADO, E.; NOVAIS, P.; ANALIDE, C.; SEDANO, J. (Ed.). **Soft Computing Models in Industrial and Environmental Applications, 5th International Workshop (SOCO 2010)**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 227–235. ISBN 978-3-642-13161-5. Citado na página 65.

COY, S. P.; GOLDEN, B. L.; RUNGER, G. C.; WASIL, E. A. Using experimental design to find effective parameter settings for heuristics. **Journal of Heuristics**, Springer, v. 7, n. 1, p. 77–97, 2001. Citado na página 17.

CROES, G. : A method for solving traveling salesman problems. **Operations Research**, v. 6, p. 791–812, 1958. Citado na página 69.

D. NOORUL HAQ A., S. S. J. R.; R., S. Flow shop scheduling with multiple objective of minimizing makespan and total flow time. **International Journal of Advanced Manufacturing Technology**, v. 25, p. 1007–1012, 2005. Citado 2 vezes nas páginas 71 e 83.

DANIELS, R.; MAZZOLA, J. B. Flow shop scheduling with resource flexibility. v. 42, p. 504–522, 06 1994. Citado na página 46.

DANNENBRING, D. G. An evaluation of flow shop sequencing heuristics. **Management science**, INFORMS, v. 23, n. 11, p. 1174–1182, 1977. Citado na página 49.

DEAN, A.; VOSS, D. **Design and Analysis of Experiments**. New York, NY, USA: Springer Verlag, 1999. Citado 2 vezes nas páginas 21 e 35.

DORIGO, M. Optimization, learning and natural algorithms. **PhD Thesis, Politecnico di Milano**, 1992. Citado na página 20.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. **IEEE Transactions on evolutionary computation**, IEEE, v. 1, n. 1, p. 53–66, 1997. Citado 3 vezes nas páginas 57, 58 e 59.

DUDEK, R. A.; JR, O. F. T. Development of m-stage decision rule for scheduling n jobs through m machines. **Operations Research**, INFORMS, v. 12, n. 3, p. 471–497, 1964. Citado na página 49.

EBERHART, R.; KENNEDY, J. A new optimizer using particle swarm theory. In: IEEE. **Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on**. [S.l.], 1995. p. 39–43. Citado na página 20.

EIBEN, Á. E.; HINTERDING, R.; MICHALEWICZ, Z. Parameter control in evolutionary algorithms. **IEEE Transactions on evolutionary computation**, IEEE, v. 3, n. 2, p. 124–141, 1999. Citado na página 32.

ESHELMAN, L.; SHAFFER, J. Real-coded genetic algorithms and interval-schema. **Foundations of Genetic Algorithms**. Citado na página 68.

FARMER, J. D.; PACKARD, N. H.; PERELSON, A. S. The immune system, adaptation, and machine learning. **Physica D: Nonlinear Phenomena**, Elsevier, v. 22, n. 1-3, p. 187–204, 1986. Citado na página 20.

FONSECA, T. H. L.; OLIVEIRA, A. C. M. de. Tuning of clustering search based metaheuristic by cross-validated racing approach. In: SPRINGER. **International Work-Conference on Artificial Neural Networks**. [S.l.], 2017. p. 62–72. Citado na página 87.

FOX, B.; MCMAHON, M. Genetic operators for sequencing problems. In: **Foundations of genetic algorithms**. [S.l.]: Elsevier, 1991. v. 1, p. 284–300. Citado na página 61.

FRAMINAN, J.; LEISTEN, R.; RAJENDRAN, C. Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. **International Journal of Production Research**, Taylor & Francis, v. 41, n. 1, p. 121–148, 2003. Citado na página 49.

FRIEZE, A.; YADEGAR, J. A new integer programming formulation for the permutation flowshop problem. **European journal of operational research**, Elsevier, v. 40, n. 1, p. 90–98, 1989. Citado na página 46.

GABBERT, P. A system for learning routes and schedules with genetic algorithm. In: **Proc. of ICGA-91**. [S.l.: s.n.], 1991. Citado na página 61.

GAMBARDELLA, L. M.; TAILLARD, É. D.; DORIGO, M. Ant colonies for the quadratic assignment problem. **Journal of the operational research society**, Taylor & Francis, v. 50, n. 2, p. 167–176, 1999. Citado na página 58.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. A tabu search heuristic for the vehicle routing problem. **Management Science**, v. 40, n. 10, p. 1276–1290, 1994. Disponível em: <https://doi.org/10.1287/mnsc.40.10.1276>. Citado na página 21.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & operations research**, Elsevier, v. 13, n. 5, p. 533–549, 1986. Citado na página 19.

____. Tabu search—part i. **ORSA Journal on computing**, INFORMS, v. 1, n. 3, p. 190–206, 1989. Citado na página 55.

GLOVER, F.; LAGUNA, M. Tabu search. In: **Handbook of combinatorial optimization**. [S.l.]: Springer, 2013. p. 3261–3362. Citado 2 vezes nas páginas 20 e 55.

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. **Machine learning**, Springer, v. 3, n. 2, p. 95–99, 1988. Citado na página 61.

GOLDEN, B.; STEWART, W.; J.K., R. K. **Empirical analysis of heuristics**. [S.l.]: Wiley, 1985. Citado na página 21.

GOLDEN, B. L.; ASSAD, A. A.; WASIL, E. A.; BAKER, E. Experimentation in optimization. **European Journal of Operational Research**, v. 27, n. 1, p. 1 – 16, 1986. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221786800029>. Citado na página 21.

GRABOWSKI, J.; WODECKI, M. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. **Computers & Operations Research**, Elsevier, v. 31, n. 11, p. 1891–1909, 2004. Citado 2 vezes nas páginas 56 e 57.

GREFENSTETTE, J. J. Optimization of control parameters for genetic algorithms. **IEEE Transactions on systems, man, and cybernetics**, IEEE, v. 16, n. 1, p. 122–128, 1986. Citado na página 33.

GUPTA, D.; NAILWAL, K. K.; SHARMA, S. A heuristic for permutation flowshop scheduling to minimize makespan. In: SPRINGER. **Proceedings of the Third International Conference on Soft Computing for Problem Solving**. [S.l.], 2014. p. 423–432. Citado na página 44.

GUPTA, J. N. A functional heuristic algorithm for the flowshop scheduling problem. **Journal of the Operational Research Society**, Taylor & Francis, v. 22, n. 1, p. 39–47, 1971. Citado 2 vezes nas páginas 49 e 50.

HANSEN, N. The cma evolution strategy: a comparing review. In: **Towards a new evolutionary computation**. [S.l.]: Springer, 2006. p. 75–102. Citado na página 33.

HEJAZI*, S. R.; SAGHAFIAN, S. Flowshop-scheduling problems with makespan criterion: a review. **International Journal of Production Research**, Taylor & Francis, v. 43, n. 14, p. 2895–2929, 2005. Citado na página 60.

HELD, M.; KARP, R. M. A dynamic programming approach to sequencing problems. **Journal of the Society for Industrial and Applied Mathematics**, SIAM, v. 10, n. 1, p. 196–210, 1962. Citado na página 46.

HO, J. C.; CHANG, Y.-L. A new heuristic for the n-job, m-machine flow-shop problem. **European Journal of Operational Research**, Elsevier, v. 52, n. 2, p. 194–202, 1991. Citado na página 49.

HOFFMAN, K. L.; PADBERG, M.; RINALDI, G. Traveling salesman problem. In: **Encyclopedia of operations research and management science**. [S.l.]: Springer, 2013. p. 1573–1578. Citado na página 69.

HOLLAND, J. H. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [S.l.]: MIT press, 1992. Citado 2 vezes nas páginas 20 e 61.

HOOKER, J. Testing heuristics: We have it all wrong. journal of heuristics. **Journal of Heuristics**, v. 1, p. 33–42, 1995. Citado na página 21.

HUANG, D.; ALLEN, T. T.; NOTZ, W. I.; ZENG, N. Global optimization of stochastic black-box systems via sequential kriging meta-models. **Journal of global optimization**, Springer, v. 34, n. 3, p. 441–466, 2006. Citado na página 41.

HUNDAL, T. S.; RAJGOPAL, J. An extension of palmer's heuristic for the flow shop scheduling problem. **International Journal of Production Research**, Taylor & Francis, v. 26, n. 6, p. 1119–1124, 1988. Citado na página 49.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In: SPRINGER. **International Conference on Learning and Intelligent Optimization**. [S.l.], 2011. p. 507–523. Citado 2 vezes nas páginas 32 e 41.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; MURPHY, K. P. An experimental investigation of model-based parameter optimisation: Spo and beyond. In: ACM. **Proceedings of the 11th Annual conference on Genetic and evolutionary computation**. [S.l.], 2009. p. 271–278. Citado na página 41.

HUTTER, F.; HOOS, H. H.; LEYTON-BROWN, K.; STÜTZLE, T. Paramils: an automatic algorithm configuration framework. **Journal of Artificial Intelligence Research**, v. 36, p. 267–306, 2009. Citado na página 33.

HUTTER, F.; HOOS, H. H.; STüTZLE, T. Automatic algorithm configuration based on local search. In: **Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2**. AAAI Press, 2007. (AAAI'07), p. 1152–1157. ISBN 978-1-57735-323-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1619797.1619831>. Citado na página 22.

IGEL, C.; TOUSSAINT, M. On classes of functions for which no free lunch results hold. **Information processing letters**, Elsevier, v. 86, n. 6, p. 317–321, 2003. Citado na página 17.

IGNALL, E.; SCHRAGE, L. Application of the branch and bound technique to some flow-shop scheduling problems. **Operations research**, INFORMS, v. 13, n. 3, p. 400–412, 1965. Citado na página 46.

ISHUBUCHI, M.; MASAKI, S.; TANAKA, H. Modified simulated annealing for the flow shop sequencing problems. **European Journal of Operational Research**, v. 81, n. 2, p. 388–398, 1995. Citado 2 vezes nas páginas 51 e 52.

JACKSON J.AND GIRARD, A. R. S. A combined tabu search and 2-opt heuristic for multiple vehicle routing. **American Control Conference**, 2010. Citado na página 69.

JIANG, S.; LIU, M.; HAO, J.; QIAN, W. A bi-layer optimization approach for a hybrid flow shop scheduling problem involving controllable processing times in the steelmaking industry. **Computers & Industrial Engineering**, Elsevier, v. 87, p. 518–531, 2015. Citado na página 43.

JOG, P. The effects of population size, heuristic crossover and local improvement of a genetic algorithm for the traveling salesman problem. In: **Proc. 3rd ICGA**. [S.l.: s.n.], 1989. p. 110–115. Citado na página 61.

JOHNSON, S. Optimal two- and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly**, v. 1, p. 61–68, 1954. Citado 2 vezes nas páginas 43 e 47.

JONES, D. R.; SCHONLAU, M.; WELCH, W. J. Efficient global optimization of expensive black-box functions. **Journal of Global optimization**, Springer, v. 13, n. 4, p. 455–492, 1998. Citado na página 34.

JR, E. F. S.; TSENG, F. T. On the srikar-ghosh milp model for the ivx m sdst flowshop problem. **International Journal of Production Research**, Taylor and Francis, v. 28, n. 10, p. 1817–1830, 1990. Disponível em: <https://doi.org/10.1080/00207549008942836>. Citado na página 46.

JUN, S.; PARK, J. A hybrid genetic algorithm for the hybrid flow shop scheduling problem with nighttime work and simultaneous work constraints: A case study from the transformer industry. **Expert Systems with Applications**, Elsevier, v. 42, n. 15-16, p. 6196–6204, 2015. Citado na página 43.

KAN, R.; GEORGE, A. H. Machine scheduling problems : classification, complexity and computations /. 01 1976. Citado na página 81.

KANTOROVICH, L. The importance of mathematical optimization in economics. In: **Optimization Techniques**. [S.l.]: Springer, 1980. p. 135–136. Citado na página 16.

KELLEY, T. R. Optimization, an important stage of engineering design. **The Technology Teacher**, International Technology and Engineering Educators Association, v. 69, n. 5, p. 18, 2010. Citado na página 16.

KING, J.; SPACHIS, A. Heuristics for flow-shop scheduling. **International Journal of Production Research**, Taylor & Francis, v. 18, n. 3, p. 345–357, 1980. Citado na página 49.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. Citado 2 vezes nas páginas 20 e 51.

KOUKI, S.; JEMNI, M.; LADHARI, T. Solving the permutation flow shop problems with makespan citerion using grids. **International Journal of Grid and Distributed Computing**, v. 4, 2011. Citado na página 45.

KOULAMAS, C. A new constructive heuristic for the flowshop scheduling problem. **European Journal of Operational Research**, Elsevier, v. 105, n. 1, p. 66–71, 1998. Citado 2 vezes nas páginas 49 e 51.

KUNTZ, P.; LAYZELL, P.; SNYERS, D. A colony of ant-like agents for partitioning in vlsi technology. In: MIT PRESS, CAMBRIDGE, MA. **Proceedings of the Fourth European Conference on Artificial Life**. [S.l.], 1997. p. 417–424. Citado na página 58.

LAARHOVEN, P. J. V.; AARTS, E. H. Simulated annealing. In: **Simulated annealing: Theory and applications**. [S.l.]: Springer, 1987. p. 7–15. Citado na página 52.

LAGUNA, M.; MARTI, R. **Scatter search: methodology and implementations in C**. [S.l.]: Springer Science & Business Media, 2012. v. 24. Citado na página 69.

LAMOUDAN, T.; KHOUKHI, F. E.; ALAOUI, A. E. H.; BOUKACHOUR, J. Flow shop scheduling problem with transportation times, two-robots and inputs/outputs with limited capacity. **International Journal of Intelligent Computing Research**, v. 3, 2012. Citado na página 46.

LEE, I.; SHAW, M. J. A neural-net approach to real time flow-shop sequencing. **Computers & Industrial Engineering**, Elsevier, v. 38, n. 1, p. 125–147, 2000. Citado na página 61.

LIU, J. The impact of neighbourhood size on the process of simulated annealing: computational experiments on the flowshop scheduling problem. **Computers & Industrial Engineering**, Elsevier, v. 37, n. 1-2, p. 285–288, 1999. Citado 2 vezes nas páginas 52 e 53.

LIU Y.AND OUYANG, Y.; SHENG, H.; ZHANG, X. Artificial bee and differential evolution improved by clustering search on continuous domain optimization. **Signal Image Technology and Internet Based Systems**, 2008. Citado na página 64.

LOMNICKI, Z. A. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. **Journal of the Operational Research Society**, v. 16, n. 1, p. 89–100, Mar 1965. ISSN 1476-9360. Disponível em: <https://doi.org/10.1057/jors.1965.7>. Citado na página 46.

MAINIERI, G. B. **Meta-heurística BRKGA aplicada a um problema de programação de tarefas no ambiente flowshop híbrido.** Tese (Doutorado) — Universidade de São Paulo, 2014. Citado na página 83.

MCGEOCH, C. Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. **ACM Computing Surveys**, v. 2, p. 195–212, 1992. Citado na página 21.

MERCER, R. E.; SAMPSON, J. Adaptive search using a reproductive meta-plan. **Kybernetes**, MCB UP Ltd, v. 7, n. 3, p. 215–228, 1978. Citado na página 33.

MERKLE, D.; MIDDENDORF, M. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: SPRINGER. **Workshops on Real-World Applications of Evolutionary Computation**. [S.l.], 2000. p. 290–299. Citado na página 58.

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of state calculations by fast computing machines. **The journal of chemical physics**, AIP, v. 21, n. 6, p. 1087–1092, 1953. Citado 2 vezes nas páginas 52 e 53.

MITTEN, L. Sequencing n jobs on two machines with arbitrary time lags. **Management science**, INFORMS, v. 5, n. 3, p. 293–298, 1959. Citado na página 48.

MOCCELLIN, J. V. A new heuristic method for the permutation flow shop scheduling problem. **Journal of the Operational Research Society**, Springer, v. 46, n. 7, p. 883–886, 1995. Citado na página 56.

MONTGOMERY, D. C. Response surface methods and other approaches to process optimization. **Design and analysis of experiment**, Wiley-VCH, p. 427–510, 1997. Citado 2 vezes nas páginas 39 e 40.

MORO, M. A. *et al.* Meta-heurísticas grasp e brkga aplicadas ao problema da diversidade máxima. [sn], 2017. Citado na página 83.

MOSCATO, P. *et al.* On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. **Caltech concurrent computation program, C3P Report**, v. 826, p. 1989, 1989. Citado na página 20.

MURATA, T.; ISHIBUCHI, H.; TANAKA, H. Multi-objective genetic algorithm and its applications to flowshop scheduling. **Computers & Industrial Engineering**, Elsevier, v. 30, n. 4, p. 957–968, 1996. Citado 2 vezes nas páginas 51 e 61.

MYERS, R. H.; MONTGOMERY, D. C. **Response Surface Methodology: Process and Product in Optimization Using Designed Experiments**. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1995. ISBN 0471581003. Citado na página 21.

NAKANO, R.; YAMADA, T. Conventional genetic algorithm for job shop problems. In: **ICGA**. [S.l.: s.n.], 1991. v. 91, p. 474–479. Citado na página 61.

NANNEN, V.; EIBEN, A. E. A method for parameter calibration and relevance estimation in evolutionary algorithms. In: ACM. **Proceedings of the 8th annual conference on Genetic and evolutionary computation**. [S.l.], 2006. p. 183–190. Citado na página 33.

NAWAZ, M.; JR, E. E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, Elsevier, v. 11, n. 1, p. 91–95, 1983. Citado 3 vezes nas páginas 49, 50 e 51.

NEGENMAN, E. G. Local search algorithms for the multiprocessor flow shop scheduling problem. **European Journal of Operational Research**, Elsevier, v. 128, n. 1, p. 147–158, 2001. Citado na página 51.

NEPPALLI, V. R.; CHEN, C.-L.; GUPTA, J. N. Genetic algorithms for the two-stage bicriteria flowshop problem. **European journal of operational research**, Elsevier, v. 95, n. 2, p. 356–373, 1996. Citado na página 61.

NOWICKI, E.; SMUTNICKI, C. A fast tabu search algorithm for the permutation flow-shop problem. **European Journal of Operational Research**, Elsevier, v. 91, n. 1, p. 160–175, 1996. Citado 2 vezes nas páginas 56 e 57.

OGBU, F.; SMITH, D. K. The application of the simulated annealing algorithm to the solution of the n/m/cmax flowshop problem. **Computers & Operations Research**, Elsevier, v. 17, n. 3, p. 243–253, 1990. Citado 4 vezes nas páginas 52, 53, 54 e 56.

OLIVEIRA, A.; COSTA, T. S. Artificial bee and differential evolution improved by clustering search on continuous domain optimization. **Soft Computing**, p. 1–12, 2014. Citado na página 64.

____. ____. **Soft Computing**, p. 1–12, 2014. Citado na página 64.

OLIVEIRA, A.; LORENA, L. A. N. Hybrid evolutionary algorithms and clustering search. hybrid evolutionary system. **Studies in Computational Intelligence**, v. 75, p. 81–102, 2007. Citado na página 65.

OLIVEIRA, A. C. M.; LORENA, L. A. N. Advances in artificial intelligence – sbia 2004: 17th brazilian symposium on artificial intelligence, sao luis, maranhao, brazil, september 29-ocotber 1, 2004. proceedings. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. cap. Detecting Promising Areas by Evolutionary Clustering Search, p. 385–394. ISBN 978-3-540-28645-5. Citado 3 vezes nas páginas 65, 66 e 69.

OLIVEIRA, A. de; LORENA, L. Hybrid evolutionary algorithms and clustering search. In: ABRAHAM, A.; GROSAN, C.; ISHIBUCHI, H. (Ed.). **Hybrid Evolutionary Algorithms**. [S.l.]: Springer Berlin / Heidelberg, 2007, (Studies in Computational Intelligence, v. 75). p. 77–99. ISBN 978-3-540-73296-9. Citado 2 vezes nas páginas 64 e 69.

OSMAN, I. H.; LAPORTE, G. **Metaheuristics: A bibliography**. [S.l.]: Springer, 1996. Citado na página 19.

OSMAN, I. H.; POTTS, C. Simulated annealing for permutation flow-shop scheduling. **Omega**, Elsevier, v. 17, n. 6, p. 551–557, 1989. Citado 3 vezes nas páginas 51, 52 e 53.

PALMER, D. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. **Journal of the Operational Research Society**, Springer, v. 16, n. 1, p. 101–107, 1965. Citado na página 49.

PANWALKAR, S. S.; DUDEK, R. A.; SMITH, M. L. Sequencing research and the industrial scheduling problem. In: ELMAGHRABY, S. E. (Ed.). **Symposium on the Theory of Scheduling and Its Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1973. p. 29–38. Citado na página 81.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial optimization: algorithms and complexity**. [S.l.]: Courier Corporation, 1998. Citado na página 16.

Papoulis, A.; Pillai, S. U. **Probability, Random Variables, and Stochastic Processes**. 4. ed. [S.l.]: McGraw-Hill Higher Education, 2002. Citado na página 29.

POMARI, C. Z.; CHAVES, A. A. Algoritmo híbrido com cs e brkga aplicado ao problema de alocação de berços. **SBPO-Simpósio Brasileiro de Pesquisa Operacional. Salvador–BA**, 2014. Citado na página 83.

PONNAMBALAM, S.; ARAVINDAN, P.; RAO, P. S. Comparative evaluation of genetic algorithms for job-shop scheduling. **Production Planning & Control**, Taylor & Francis, v. 12, n. 6, p. 560–574, 2001. Citado na página 61.

POTTS, C. N.; STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. **Journal of the Operational Research Society**, Springer, v. 60, n. 1, p. S41–S68, 2009. Citado na página 43.

POUR, H. D. A new heuristic for the n-job, m-machine flow-shop problem. **Production Planning & Control**, Taylor & Francis, v. 12, n. 7, p. 648–653, 2001. Citado 2 vezes nas páginas 49 e 51.

PRICE, K.; STORN, R. M.; LAMPINEN, J. A. **Differential evolution: a practical approach to global optimization**. [S.l.]: Springer Science & Business Media, 2006. Citado na página 20.

RADCLIFFE, N. J.; SURRY, P. D. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In: **Computer Science Today**. [S.l.]: Springer, 1995. p. 275–291. Citado na página 17.

RAJENDRAN, C. Heuristics for scheduling in flowshop with multiple objectives. **European Journal of Operational Research**, v. 82, p. 540–555, 1995. Citado na página 46.

____. ____. **European Journal of Operational Research**, v. 82, p. 540–555, 1995. Citado na página 71.

RAJENDRAN, C.; ZIEGLER, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. **European Journal of Operational Research**, Elsevier, v. 155, n. 2, p. 426–438, 2004. Citado na página 58.

RAVINDRAN, D.; SELVAKUMAR. Flow shop scheduling with multiple objective of minimizing makespan and total flow time. **International Journal of Advanced Manufacturing Technology**, v. 25, p. 1007–1012, 2005. Citado na página 46.

REEVES, C. R. Improving the efficiency of tabu search for machine sequencing problems. **Journal of the Operational Research Society**, Springer, v. 44, n. 4, p. 375–382, 1993. Citado 3 vezes nas páginas 51, 55 e 61.

____. A genetic algorithm for flowshop sequencing. **Computers & operations research**, Elsevier, v. 22, n. 1, p. 5–13, 1995. Citado na página 61.

REEVES, C. R.; YAMADA, T. Genetic algorithms, path relinking, and the flowshop sequencing problem. **Evolutionary computation**, MIT Press, v. 6, n. 1, p. 45–60, 1998. Citado na página 61.

RUBINSTEIN, R. Y.; KROESE, D. P. **Simulation and the Monte Carlo Method (Wiley Series in Probability and Statistics)**. 2. ed. [S.l.: s.n.], 2016. ISBN 0470177942, 9780470177945. Citado na página 23.

RUDEK, A.; RUDEK, R. Makespan minimization flowshop with position dependent job processing times—computational complexity and solution algorithms. **Computers & Operations Research**, Elsevier, v. 40, n. 8, p. 2071–2082, 2013. Citado na página 43.

RUIZ, R. **Scheduling Heuristics**. 2015. 1-24 p. Citado na página 45.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, Elsevier, v. 177, n. 3, p. 2033–2049, 2007. Citado na página 51.

SACKS, J.; WELCH, W. J.; MITCHELL, T. J.; WYNN, H. P. Design and analysis of computer experiments. **Statistical science**, JSTOR, p. 409–423, 1989. Citado 2 vezes nas páginas 34 e 39.

SAKA, M.; DOGAN, E. Recent developments in metaheuristic algorithms: a review. **Comput Technol Rev**, v. 5, n. 4, p. 31–78, 2012. Citado na página 16.

SARIN, S.; LEFOKA, M. Scheduling heuristic for the n-jobm-machine flow shop. **Omega**, Elsevier, v. 21, n. 2, p. 229–234, 1993. Citado 2 vezes nas páginas 49 e 51.

SARMADY, S. An investigation on tabu search parameters. **School of Computer Sciences, Universiti Sains Malaysia**, v. 11800, 2012. Citado na página 55.

SCHAEFER, T. J. The complexity of satisfiability problems. In: **Proceedings of the Tenth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1978. (STOC '78), p. 216–226. Disponível em: <http://doi.acm.org/10.1145/800133.804350>. Citado na página 22.

SCHOONDERWOERD, R.; HOLLAND, O. E.; BRUTEN, J. L.; ROTHKRANTZ, L. J. Ant-based load balancing in telecommunications networks. **Adaptive behavior**, Sage Publications Sage UK: London, England, v. 5, n. 2, p. 169–207, 1997. Citado na página 58.

SCHUMACHER, C.; VOSE, M. D.; WHITLEY, L. D. The no free lunch and problem description length. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation**. [S.l.], 2001. p. 565–570. Citado na página 17.

SEN, T.; SULEK, J. M.; DILEEPAN, P. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. **International Journal of Production Economics**, Elsevier, v. 83, n. 1, p. 1–12, 2003. Citado na página 48.

SOLIMANPUR, M.; VRAT, P.; SHANKAR, R. A neuro-tabu search heuristic for the flow shop scheduling problem. **Computers & Operations Research**, Elsevier, v. 31, n. 13, p. 2151–2164, 2004. Citado 2 vezes nas páginas 56 e 57.

SRIDHAR, J.; RAJENDRAN, C. A genetic algorithm for family and job scheduling in a flowline-based manufacturing cell. **Computers & Industrial Engineering**, Elsevier, v. 27, n. 1-4, p. 469–472, 1994. Citado na página 53.

STÜTZLE, T.; HOOS, H. Improvements on the ant-system: Introducing the max-min ant system. In: SPRINGER. **Artificial Neural Nets and Genetic Algorithms**. [S.l.], 1998. p. 245–249. Citado na página 58.

SYSWERDA, G. The application of genetic algorithms to resource scheduling. In: **Proc. of ICGA-91**. [S.l.: s.n.], 1991. Citado na página 61.

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. **European journal of Operational research**, Elsevier, v. 47, n. 1, p. 65–74, 1990. Citado 5 vezes nas páginas 18, 55, 56, 58 e 71.

TAILLARD Éric. **Scheduling instances**. 2013. Urlhttp://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html. Citado na página 45.

____. ____. 2013. Urlhttp://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.htr
Citado 2 vezes nas páginas 80 e 82.

TALBI, E.-G. **Metaheuristics: from design to implementation**. [S.l.]: John Wiley & Sons, 2009. v. 74. Citado na página 19.

TANG, L.; WANG, X. An improved particle swarm optimization algorithm for the hybrid flowshop scheduling to minimize total weighted completion time in process industry. **IEEE transactions on control systems technology**, IEEE, v. 18, n. 6, p. 1303–1314, 2010. Citado na página 43.

TIAN, P.; MA, J.; ZHANG, D.-M. Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism. **European Journal of Operational Research**, Elsevier, v. 118, n. 1, p. 81–94, 1999. Citado na página 53.

T'KINDT, V.; GUPTA, J. N.; BILLAUT, J.-C. Two-machine flowshop scheduling with a secondary criterion. **Computers & Operations Research**, Elsevier, v. 30, n. 4, p. 505–526, 2003. Citado 3 vezes nas páginas 48, 51 e 58.

TOROPOV, V. V.; FILATOV, A. A.; POLYNKIN, A. A. Multiparameter structural optimization using fem and multipoint explicit approximations. **Structural optimization**, v. 6, n. 1, p. 7–14, Mar 1993. ISSN 1615-1488. Disponível em: <https://doi.org/10.1007/BF01743169>. Citado na página 38.

TSAI, P.-W.; PAN eng S.; LIAO, B.-Y.; CHU, S.-C. Enhanced artificial bee colony optimization. **International Journal of Innovative Computing, Information and Control**, v. 5, 2009. Citado na página 45.

TURNER, S.; BOOTH, D. Comparison of heuristics for flow shop sequencing. **Omega**, Elsevier, v. 15, n. 1, p. 75–78, 1987. Citado na página 51.

ULDER, N. L.; AARTS, E. H.; BANDELT, H.-J.; LAARHOVEN, P. J. V.; PESCH, E. Genetic local search algorithms for the traveling salesman problem. In: SPRINGER. **International Conference on Parallel Problem Solving from Nature**. [S.l.], 1990. p. 109–116. Citado na página 61.

VAPNIK, V. N. **The Nature of Statistical Learning Theory**. New York, NY, USA: Springer-Verlag New York, Inc., 1995. ISBN 0-387-94559-8. Citado na página 74.

VENTER, G.; HAFTKA, R.; STARNES JR, J. Construction of response surfaces for design optimization applications. In: **6th Symposium on Multidisciplinary Analysis and Optimization**. [S.l.: s.n.], 1996. p. 4040. Citado na página 37.

VOSSS, S.; FINK, A.; DUIN, C. Looking ahead with the pilot method. **Annals of Operations Research**, Springer, v. 136, n. 1, p. 285–302, 2005. Citado na página 51.

WATSON, J.-P.; BARBULESCU, L.; WHITLEY, L. D.; HOWE, A. E. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. **INFORMS Journal on Computing**, v. 14, n. 2, p. 98–123, 2002. Citado 3 vezes nas páginas 18, 80 e 81.

WHITLEY, T. S. D.; FUQUAY, D. Scheduling problems and traveling salesman: The genetic age recombination. In: **Proceedings of the Third ICGA**. [S.l.: s.n.]. Citado na página 61.

____. ____. In: **Proceedings of the Third ICGA**. [S.l.: s.n.], 1991. p. 133–140. Citado na página 61.

WIDMER, M.; HERTZ, A. A new heuristic method for the flow shop sequencing problem. **European Journal of Operational Research**, Elsevier, v. 41, n. 2, p. 186–193, 1989. Citado na página 56.

WODECKI, M.; BOŻZEJKO, W. Solving the flow shop problem by parallel simulated annealing. In: SPRINGER. **International Conference on Parallel Processing and Applied Mathematics**. [S.l.], 2001. p. 236–244. Citado na página 52.

WOEGINGER, G. J. Exact algorithms for np-hard problems: A survey. In: **Combinatorial Optimization—Eureka, You Shrink!** [S.l.]: Springer, 2003. p. 185–207. Citado na página 16.

WOLPERT, D. H.; MACREADY, W. G. No free lunch theorems for optimization. **IEEE transactions on evolutionary computation**, IEEE, v. 1, n. 1, p. 67–82, 1997. Citado 3 vezes nas páginas 17, 20 e 31.

XU, J.; KELLY, J. P. A network flow-based tabu search heuristic for the vehicle routing problem. **Transportation Science**, v. 30, n. 4, p. 379–393, 1996. Citado na página 21.

YAGMAHAN, B.; YENISEY, M. M. A multi-objective ant colony system algorithm for flow shop scheduling problem. **Expert Systems with Applications**, v. 37, p. 1361–1368, 2010. Citado na página 45.

YING, K.-C.; LIAO, C.-J. An ant colony system for permutation flow-shop sequencing. **Computers & Operations Research**, Elsevier, v. 31, n. 5, p. 791–801, 2004. Citado 2 vezes nas páginas 58 e 59.

YING, K.-C.; LIN, S.-W. A high-performing constructive heuristic for minimizing makespan in permutation flowshops. **Journal of Industrial and Production Engineering**, Taylor & Francis, v. 30, n. 6, p. 355–362, 2013. Citado na página 44.

ZEGORDI, S. H.; ITOH, K.; ENKAWA, T. Minimizing makespan for flow shop scheduling by combining simulated annealing with sequencing knowledge. **European Journal of Operational Research**, Elsevier, v. 85, n. 3, p. 515–531, 1995. Citado na página 52.

ZHENG, D.-Z.; WANG, L. An effective hybrid heuristic for flow shop scheduling. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 21, n. 1, p. 38–44, 2003. Citado na página 52.

____. ____. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 21, n. 1, p. 38–44, 2003. Citado na página 61.

ZOBOLAS, G.; TARANTILIS, C. D.; IOANNOU, G. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. **Computers & Operations Research**, Elsevier, v. 36, n. 4, p. 1249–1267, 2009. Citado na página 51.