UNIVERSIDADE FEDERAL DO MARANHÃO
Programa de Pós-Graduação em Ciência da Computação

CHRYSTIAN GUSTAVO MARTINS NASCIMENTO

META-APRENDIZAGEM PARA SELEÇÃO DE ALGORITMOS SINTONIZADOS APLICADA AO PROBLEMA DE FLOW SHOP PERMUTACIONAL

São Luis
2019

Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnológicas
Programa de Pós-Graduação em Ciência da Computação

CHRYSTIAN GUSTAVO MARTINS NASCIMENTO

META-APRENDIZAGEM PARA SELEÇÃO DE ALGORITMOS SINTONIZADOS APLICADA AO PROBLEMA DE FLOW SHOP PERMUTACIONAL

São Luís
2019

CHRYSTIAN GUSTAVO MARTINS NASCIMENTO

META-APRENDIZAGEM PARA SELEÇÃO DE ALGORITMOS SINTONIZADOS APLICADA AO PROBLEMA DE FLOW SHOP PERMUTACIONAL

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

Orientador: Profº Dr. Alexandre César Muniz de Oliveira

São Luís

2019

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

Nascimento, Chrystian Gustavo Martins.
META-APRENDIZAGEM PARA SELEÇÃO DE ALGORITMOS SINTONIZADOS APLICADA AO PROBLEMA DE FLOW SHOP PERMUTACIONAL / Chrystian Gustavo Martins Nascimento. - 2019.
71 f.

Orientador(a): Alexandre César Muniz de Oliveira.
Dissertação (Mestrado) - Programa de Pós-graduação em Ciência da Computação/ccet, Universidade Federal do Maranhão, São Luís, 2019.

1. Flow shop Permutacional. 2. Meta-aprendizagem. 3. Meta-heurística. 4. Método de corrida. 5. Representação baseada em grafos. I. Oliveira, Alexandre César Muniz de. II. Título.

Chrystian Gustavo Martins Nascimento

Meta-aprendizagem para seleção de algoritmos sintonizados aplicada ao Problema de Flow Shop Permutacional

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

Trabalho aprovado. São Luís, 18 de abril de 2019:

Prof. Dr. Alexandre César Muniz de Oliveira
Universidade Federal do Maranhão
Orientador

Prof. Dr. Luciano Reis Coutinho
Universidade Federal do Maranhão

Prof. Dr. Bruno Feres de Souza
Universidade Federal do Maranhão

Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho
Universidade de São Paulo

São Luís
2019

A Deus e à minha família.

Agradecimentos

Agradeço imensamente a minha família, por tudo que fazem por mim. Me considero muito abençoado por ter vocês na minha vida, me apoiando e guiando em todas as decisões que tive que tomar. Amo vocês.

Agradeço à todos os professores os quais tive a oportunidade de aprender e me desenvolver. Em especial ao meu orientador Prof. Prof. Dr. Alexandre César.

Agradeço a todos os meus amigos, sejam do LACMOR, da UFMA, da minha cidade ou da vida. Sintam-se todos abraçados. Nomeadamente aos amigos Thacyla, Ítalo, Matheus que desde o começo do curso me suportam. Aos companheiros de LACMOR e da UFMA, Ramon, Gilvan, Alex, André, Luann, Gleidson, Antonio, Rodrigo e o quase doutor Marcelo. Agradeço também ao pessoal do Lab 4 e a galera do DA de computação.

Agradeço à UFMA e à FAPEMA, bem como aos seus colaboradores. A estrutura, os incentivos e as condições que tive para desenvolver este trabalho foram de vital importância para o sucesso da pesquisa.

"O que prevemos raramente ocorre; o que menos esperamos geralmente acontece."
(Benjamin Disraeli)

Resumo

Meta-heurísticas são estratégias de busca de alto nível que orientam a busca para regiões mais promissoras do espaço da solução e tentam escapar das soluções ótimas locais. Porém, devido a heterogeneidade das instâncias dos problemas de otimização combinatória não é garantido que uma meta-heurística consiga obter sempre a melhor solução em um conjunto de meta-heurísticas, por isso a seleção de algoritmo, como a meta-aprendizagem, pode fornecer uma solução mais efetiva ao definir qual meta-heurística escolher de acordo com as características estruturais das instâncias. Neste trabalho propõe-se um *framework* de meta-aprendizagem para seleção de meta-heurísticas sintonizadas por método de corrida, que inclui meta-características extraídas a partir da representação baseada em grafo do problema e definição das meta-classes a partir das meta-heurísticas sintonizadas. Experimentos mostraram que a abordagem é eficaz para seleção de meta-heurísticas e de seus parâmetros para instancias *Flow Shop*. E que as meta-características escolhidas conseguem extrair informações estruturais relevantes para o problema abordado;

Palavras-chaves: Meta-aprendizagem. método de corrida. Meta-heurística. *Representação baseada em grafos. Problema de escalonamento Flow Shop Permutacional.*

Abstract

Meta-heuristics are high-level search strategies that guide the search for the most promising regions of the solution space and try to escape the optimal local solutions. However, due to the heterogeneity of instances of combinatorial optimization problems, it is not guaranteed that a meta-heuristic can always obtain the best solution in a set of metaheuristics, so algorithm selection, such as meta-learning, can provide a most effective solution when defining which meta-heuristic to choose according to the structural characteristics of the instances. In this work we propose a meta-learning framework for meta-heuristics selection tuned by the race method, which includes meta-features extracted from the graph-based representation of the problem and definition of the meta-classes from of the tuned meta-heuristics. Experiments have shown that the approach is effective for selecting meta-heuristics and their parameters for instances And that the chosen meta-features can extract structural information relevant to the problem addressed;

Palavras-chaves: Meta-learning. Racing. Metaheuristic. *Graph-based representation. Permutation Flow Shop Scheduling Problem.*

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MH	*Meta-heurística*
PFSP	*Problema Flow shop Permutacional*
AAEB	*Algoritmo Evolucionário Baseado em Blocos*
GRASP	*Greedy Randomized Adaptive Search Procedure*
EDH	*Evolução Diferencial Híbrida*
MSA	*Multi-start Simulated Anneling*
CDABC	*Discrete Artificial Bee Colony with Composite mutation strategies*
BRKeCS	*Biased Random-Key and Clustering Search*
CS	*Clustering Search*
Irace	*Iterated racing*
NFL	*(No Free Lunch*
MLT	*Machine Learning Toolbox*
MS	*medidas simples*
(ME	*edidas estatísticas*
MTI	*medidas de teoria da informação*
MCD	*medidas de complexidade de dados*
MD	*medidas discriminantes*
MM	*medidas de modelo*
MRC	*medidas de rede complexa*
LM	*Landmarkers*
AS	*Algorithm Selection*
CF	*Colaborative Filtering*
TSP	*Traveling Sallesman Problem*

DT	*Decision Tree*
RF	*Random Forest*
MLP	*Multilayer perceptron*
MVS	*Máquina de vetores de suporte*
NB	*Naives Bayes*
KNN	*K vizinhos mais próximos*
AP	*Affinity Propagation*

Sumário

1 Introdução

Otimização consiste em achar a melhor combinação dentre um conjunto de variáveis para maximizar ou minimizar uma função, geralmente chamada de função objetivo ou função custo. Aplicações de otimização são inúmeras. Todo processo tem um potencial para ser otimizado. Não há empresa que não esteja envolvida na solução de problemas de otimização. De fato, muitas aplicações desafiadoras na ciência e na indústria podem ser formuladas como problemas de otimização. A otimização ocorre na minimização do tempo, custo e risco ou na maximização do lucro, qualidade e eficiência. Por exemplo, existem muitas maneiras possíveis de projetar uma rede de telecomunicação para otimizar o custo e a qualidade do serviço; Há muitas maneiras de programar uma produção para otimizar o tempo; Há muitas maneiras de prever uma estrutura 3D de uma proteína para otimizar a energia potencial e assim por diante. Um grande número de problemas de otimização da vida real em ciência, engenharia, economia e negócios é complexo e difícil de resolver. Eles não podem ser resolvidos de maneira exata dentro de um período de tempo razoável. Visto que um método exato, em geral, tende a apresentar um crescimento assintótico do esforço computacional à medida que cresce o tamanho do problema. São exemplos de métodos exatos clássicos: métodos enumerativos, *branch and bound*, métodos baseados em programação matemática, programação dinâmica, etc. (ROTHLAUF, 2011).

Usar algoritmos aproximados é a principal alternativa para resolver essa classe de problemas. Os algoritmos aproximados podem ainda ser decompostos em duas classes: heurísticas especificas e meta-heurísticas. Heurísticas especificas são dependentes do problema; elas são projetadas e aplicáveis a um problema específico. Já as meta-heurísticas representam algoritmos aproximados mais gerais aplicáveis a uma grande variedade de problemas de otimização. Eles podem ser adaptados para resolver qualquer problema de otimização.(TALBI, 2009). As meta-heurísticas (MH) são estratégias de busca de alto nível que orientam a busca para regiões mais promissoras do espaço da solução e são equipadas com mecanismos capazes de escapar de soluções ótimas locais (GENDREAU; POTVIN et al., 2010). Podem ser aplicadas as instâncias de problemas de otimização, como o *Problema do Flowshop*(HEJAZI*; SAGHAFIAN, 2005), para gerar soluções viáveis com um custo computacional menor que métodos exatos clássicos. Porém, devido a sua natureza generalista, uma meta-heurística precisa ser instanciada especificamente para um dado problema, por meio de implementação de procedimentos específicos e sintonia de parâmetros de desempenho (SAKA; DOGAN, 2012).

A importância da sintonia é reconhecida pela comunidade cientifica (BARR et al., 1995): é evidente para quem tem alguma experiencia 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. O processo de sintonização consiste em encontrar a melhor configuração dentro um conjunto de configurações candidatas que melhore os resultados sobre um conjunto de instâncias de problema. Como método de sintonização para meta-heurísticas destaca-se o método de corrida que emprega uma estatística robusta para avaliar as configurações candidatas durante o processo de sintonização e descartar aquelas consideradas estatisticamente inferiores, assim que coletar evidências suficientes contra elas. A eliminação de candidatos inferiores acelera o procedimento e permite avaliar as configurações em mais instâncias e, assim, obter estimativas mais confiáveis do seu comportamento(BIRATTARI; KACPRZYK, 2009).

Porém, Devido a heterogeneidade das instâncias não é garantido que uma meta-heurística, mesmo sintonizada, consiga obter sempre a melhor solução dentre um conjunto de meta-heurísticas (WOLPERT; MACREADY et al., 1997). Uma forma de lidar com o desafio de identificar o melhor algoritmo para resolver um dado problema é a *seleção de algoritmo*, que pode empregar aprendizagem de máquina para produzir um modelo de decisão(RICE, 1976). Algoritmo de aprendizagem de máquina pode ser definido como um programa que consegue melhorar sua performance em uma tarefa com a aquisição de alguma experiência (MICHIE; SPIEGELHALTER; TAYLOR, 1994). Uma subárea de aprendizagem de máquina, a meta-aprendizagem estuda a seleção do melhor algoritmo para uma determinada tarefa (BRAZDIL et al., 2008). Ela estuda como os sistemas de aprendizagem podem aumentar em eficiência através da experiência, almejando ainda entender como o aprendizado em si pode se tornar flexível de acordo com o domínio ou a tarefa em estudo (VILALTA; DRISSI, 2002). Além disso, alguns dos classificadores induzem modelos que podem ser explicados e podem ser usados como ferramenta adicional para melhor entender a estrutura do problema (MIRANDA et al., 2018).

A meta-aprendizagem, com base nos dados de treinamentos que compreende características de entrada e a classe de saída que mostra qual algoritmo é o melhor, tem a tarefa de aprender um mapeamento para produzir uma previsão de qual algoritmo será melhor para instâncias de testes não vistas (SMITH-MILES, 2009). Um processo de meta-aprendizado aborda o problema de seleção de algoritmos de forma semelhante a um processo de aprendizagem tradicional, através da indução de um meta-modelo, que pode ser visto como o meta-conhecimento capaz de explicar por que certos algoritmos funcionam melhor do que outros em conjuntos de dados com características específicas. O meta-modelo também pode ser usado para prever o melhor algoritmo para um novo conjunto de dados / problema (SERBAN et al., 2013).

Portanto, meta-aprendizagem explora diferentes abordagens de aprendizagem de máquina que visam aprender a resolver problemas a partir da observação de experiência, de forma incremental, possibilitando generalizar para novos problemas rapida-

mente(VANSCHOREN, 2018). Isso não só acelera e aperfeiçoa drasticamente o design de pipelines de aprendizagem de máquina ou de arquiteturas neurais, mas também permite substituir algoritmos construídos manualmente por novas abordagens aprendidas de maneira orientada a dados (BRAZDIL et al., 2008).

Meta-aprendizagem pode ser aplicado para melhorar a recomendação das técnicas mais promissoras para uma determinada tarefa a partir de experiências anteriores com tarefas relacionadas (não necessariamente as mesmas). Algumas das aplicações de meta-aprendizagem para recomendação de algoritmos são abordadas em (DÔRES et al., 2016) que utiliza uma para recomendação de algoritmos na previsão de falha de software e (KANDA et al., 2016) que a utiliza para recomendar meta-heurísticas para o problema do caixeiro viajante.

1.1 Objetivo Geral

Este trabalho visa propor uma metodologia de meta-aprendizagem para seleção de meta-heurísticas sintonizadas por método de corrida, buscando auxiliar na escolha do algoritmo mais apropriado para cada instância de um problema de otimização. O método de corrida vem dar maior qualidade aos dados de experiência de desempenho, uma vez que o problema de ajuste de parâmetros é abordado de forma sistemática possibilitando uma maior capacidade de generalização da metodologia. Além disso, na etapa de caracterização das instâncias de problema de otimização, a metodologia contempla o uso de representação baseada em grafos, ao invés de representação específica do problema em questão. A proposta é validada em um conjunto de instâncias do Problema do *Flowshop* Permutacional (*Permutation Flowshop Problem - FFSP*). Esse problema trata sobre programação de tarefas que é um processo de tomada de decisão usada regularmente em muitas indústrias de manufatura e serviços. Esse processo lida com a alocação de recursos para tarefas em determinados períodos de tempo e sua meta é otimizar um ou mais objetivos (PINEDO, 2012).

1.2 Contribuições

A seguir, apresenta-se as etapas metodológicas que resultam em potenciais contribuições deste trabalho:

1. Extração de meta-características de instâncias, que compõem um importante *benchmarking* do *PFSP*, a partir de grafos comumente encontrados na literatura;

2. Implementação das principais meta-heurísticas encontradas na literatura para compor o conjunto de algoritmos candidatos;

3. Definição de metaclasses a partir de experiência prévia obtida por meio de sintonia sistemática das meta-heurísticas;

4. Definição de metadados para o *PFSP* a partir do agrupamento das configurações mais representativas encontradas no processo de sintonização de meta-heurísticas.

1.3 Organização do Trabalho

Esta dissertação está organizada em capítulos, sendo este o primeiro, que contextualiza a pesquisa e apresenta os principais objetivos do trabalho.

No Capítulo 2, são descrito os conceitos fundamentais relacionados ao problema *PFSP*, bem como os métodos de resolução do problema, com enfase em meta-heurísticas. No Capítulo 3, apresenta-se o conceito de configuração automática de algoritmo, seguido descrição do pacote *Irace* e seu funcionamento geral. Meta-aprendizagem tem seus principais conceitos apresentados no Capítulo 4, enfatizando-se a capacidade de ser empregado para seleção de algoritmos. No mesmo capítulo, caracterizam-se os principais classificadores utilizados nas abordagens de meta-aprendizagem. No Capítulo 5 define-se a abordagem utilizada, assim como a caracterização do *PFSP*, seleção dos algoritmos sintonizados e construção do modelo de meta-aprendizagem. No Capítulo 6, são fornecidos detalhes do ambiente computacional utilizado e discutidos os principais resultados encontrados.

2 Fundamentação Teórica

Neste capítulo, são descritos os conceitos fundamentais do problema de *flow shop* e meta-heurísticas de otimização a fim de fornecer um escopo geral do problema de *flow shop*.

2.1 Problema de programação de tarefas

A programação de tarefas é um processo de tomada de decisão que é usada regularmente em muitas indústrias de manufatura e serviços. Esse processo lida com a alocação de recursos para tarefas em determinados períodos de tempo e sua meta é otimizar um ou mais objetivos. Os recursos e tarefas de uma organização podem assumir muitas formas diferentes. Os recursos podem ser máquinas em uma oficina, pistas em um aeroporto, equipes em um canteiro de obras, unidades de processamento em um ambiente de computação e assim por diante. As tarefas podem ser operações em um processo de produção, decolagem e aterrissagem no aeroporto, etapas de um projeto de construção, execuções de programas de computador, e assim por diante. Cada tarefa pode ter um certo nível de prioridade, um determinado tempo de inicio e uma data de vencimento. Os objetivos também pode assumir muitas formas diferentes. Um objetivo pode ser a minimização do tempo de conclusão da última tarefa e outro pode ser a minimização do número de tarefas concluídas após seus respectivos vencimentos(PINEDO, 2012).

O problema de programação *flow shop* é uma classe de problemas de programação na qual o controle de fluxo deve permitir um sequenciamento apropriado para cada tarefa e para processamento em um conjunto de máquinas. Há máquinas em series. Cada tarefa tem que ser processada em cada uma das m máquinas. Todos as tarefas devem seguir a mesma rota, isto é, eles precisam ser processados primeiro na máquina 1, então na máquina 2, e assim por diante. Após o termino em uma máquina, a tarefa em seguida entra na fila da próxima máquina (PINEDO, 2012). Muitas aplicações nas industrias podem ser modeladas como problema de *flow shop* como, por exemplo, para industria de transformadores (JUN; PARK, 2015), industria siderúrgica (JIANG et al., 2015), as industrias 4.0 (IVANOV et al., 2016) e processo de controle de qualidade em uma empresa de roupas e vestuários (ERSEVEN et al., 2018).

Esse problema de programação da produção tem sido intensamente estudado na literatura, desde os resultados reportados por Johnson (JOHNSON, 1954) para o problema com somente duas máquinas. (RUIZ; MAROTO, 2005) apresenta uma revisão e avaliação comparativa de heurísticas e meta-heurísticas para o *PFSP* com o critério *makespan*. já em (HEJAZI*; SAGHAFIAN, 2005) é feito um levantamento completo do *PFSP* com

contribuição dos primeiros trabalhos de Johnson. Este trabalho pesquisa alguns métodos exatos (para problemas de tamanho pequeno), heurísticas construtivas e aprimoramento de abordagens meta-heurísticas e evolutivas, bem como algumas propriedades e regras bem conhecidas para este problema. Em (FERNANDEZ-VIAGAS; RUIZ; FRAMINAN, 2017) é apresentado uma revisão abrangente da literatura do problema *flow shop* referente aos últimos 10 anos. Nessa revisão são identificadas as melhores heurísticas e meta-heurísticas, sendo fornecido uma abrangente avaliação computacional e estatística e proposto um método para comparação de meta-heurísticas.

2.1.1 Problema do *Flow Shop* Permutacional

Um caso específico de programação *flow shop*, denominado permutacional(PFSP), é quando em cada máquina mantém-se a mesma ordem de processamento das tarefas. A solução do problema consiste em determinar dentre as $(n!)$ sequencias possíveis das tarefas, aquela que otimiza alguma medida de desempenho da programação, sendo que as usuais consistem na minimização da Duração Total da Programação (*makespan*), ou minimização do Tempo Médio de Fluxo. A primeira relaciona-se a uma utilização eficiente dos recursos (máquinas) enquanto que a segunda busca minimizar o estoque em processamento.

O problema flow shop permutacional é geralmente representado pela notação $Fm|prmu|Cmax$, Fm descreve máquina, onde são m máquinas em série, $prmu$ define o tipo de problema *flow shop* (no caso, permutacional), $Cmax$ é objetivo a ser minimizado, no caso *makespan* (PINEDO, 2012).

Uma definição matemática para o problema *flow shop* Permutacional pode ser descrita a partir dos seguintes elementos. Seja $t_{i,k}$ $(1 \leq i \leq m, 1 \leq k \leq n)$ os tempos de processamento da tarefa k na máquina i, assumindo que o tempo de preparação para cada tarefa é zero ou está incluído no tempo de processamento $t_{i,k}$, $\pi = (j_1, j_2, ..., j_n)$ é uma sequencia de processamento de tarefas. Π é o conjunto de todas as sequencias possíveis de tarefas, $C(i, j_k)$ é o tempo necessário para que a tarefa j_k seja completada na máquina i e toda tarefa deve ser processada da máquina 1 para a máquina m ordenadamente (PINEDO, 2012).

Em geral, o objetivo do problema é encontrar a sequencia de tarefas para a qual o tempo de processamento total seja mínimo, dado por:

$$C(j_{1,1}) = t_{j_{1,1}} \tag{2.1}$$

$$C(j_k, 1) = C(j_{k-1}, 1) + t_{j_{k,1}}, k = 2, 3, ..., n, \tag{2.2}$$

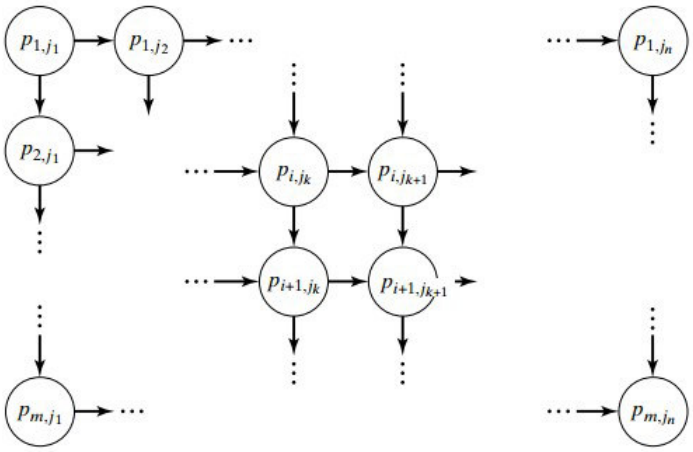
$$C(j_k, i) = C(j_1, i-1) + t_{j_k,i}, i = 2, 3, ..., m, \tag{2.3}$$

$$C(j_k, i) = max[C(j_{k-1}, i), C(j_k, i-1)] + t_{j_k,i),k=2,3,...,ni=2,3,...,m,} \tag{2.4}$$

$$\pi_* = arg[C_{max}(\pi) = C(j_n, m)] \rightarrow min, \forall\pi \in \Pi \tag{2.5}$$

O valor do *makespan* de uma determinada programação de permutação também pode ser calculado determinando o caminho crítico em um grafo direcionado que corresponde a programação. Para uma dada sequencia $j_1, ..., j_n$, este grafo direcionado é construído da seguinte forma: para cada operação, processamento da tarefa j_k na máquina i, existe um nó (i, j_k) com um peso que é igual ao tempo de processamento da tarefa j_k na máquina i. Nó $(i, jk), i = 1, ..., m-1$ e $k = 1, ..., n-1$, tem arestas saindo para nós $(i+1, j_k)$ e $(i, j_{k+1}$. Os nós correspondentes à maquina m têm apenas uma aresta de saída, assim como os nós correspondentes às tarefas j_n. O nó (m, j_n) não possui arestas de saída, como mostra a Figura 1. O peso total do caminho de peso máximo do nó $(1, j_1)$ para o nó $m, j_n)$ corresponde ao *makespan* da permutação $j_1, ..., j_n$ (PINEDO, 2012).

Figura 1 – Grafo direcional para Calculo do *makespan* em $Fm|prmu|C_{max}$ sob a sequencia $j_1, ..., j_n$.



Fonte – (PINEDO, 2012).

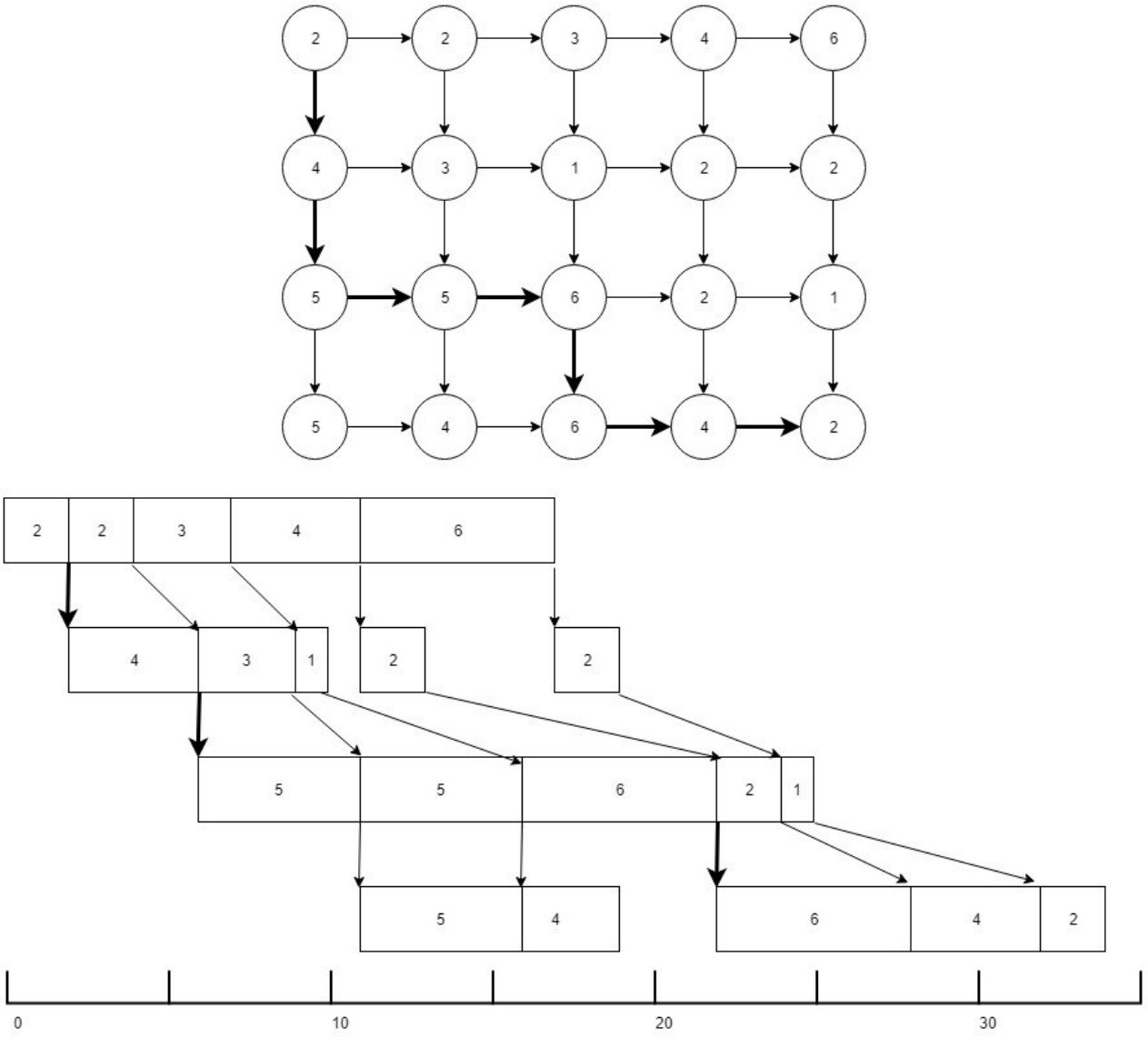
A seguir é apresentado um exemplo da representação em Grafo do *flow shop* permutacional. Esse exemplo possui 5 tarefas em 4 máquinas com o tempo de processamento apresentado na Tabela 1.

O grafo e diagrama de *Gannt* correspondente são retratados na Figura 2, ilustrando visualmente a sequencia de processamento de uma solução. A partir do grafo direcionado é calculado que o valor de *makespan* é 34. Este valor é determinado pelo caminho critico, destacado em negrito.

tarefas	t1	t2	t3	t4	t5
m1	2	2	3	4	6
m2	4	3	1	2	2
m3	5	5	6	2	1
m4	5	4	6	4	2

Tabela 1 – Problema *flow shop* em forma de matriz da permutação inicial.

Figura 2 – Grafo direto, caminho critico e diagrama de Gannt referente a Tabela 1 (As entradas numéricas representam o tempo de processamento das tarefas.)



Fonte – Acervo do autor.

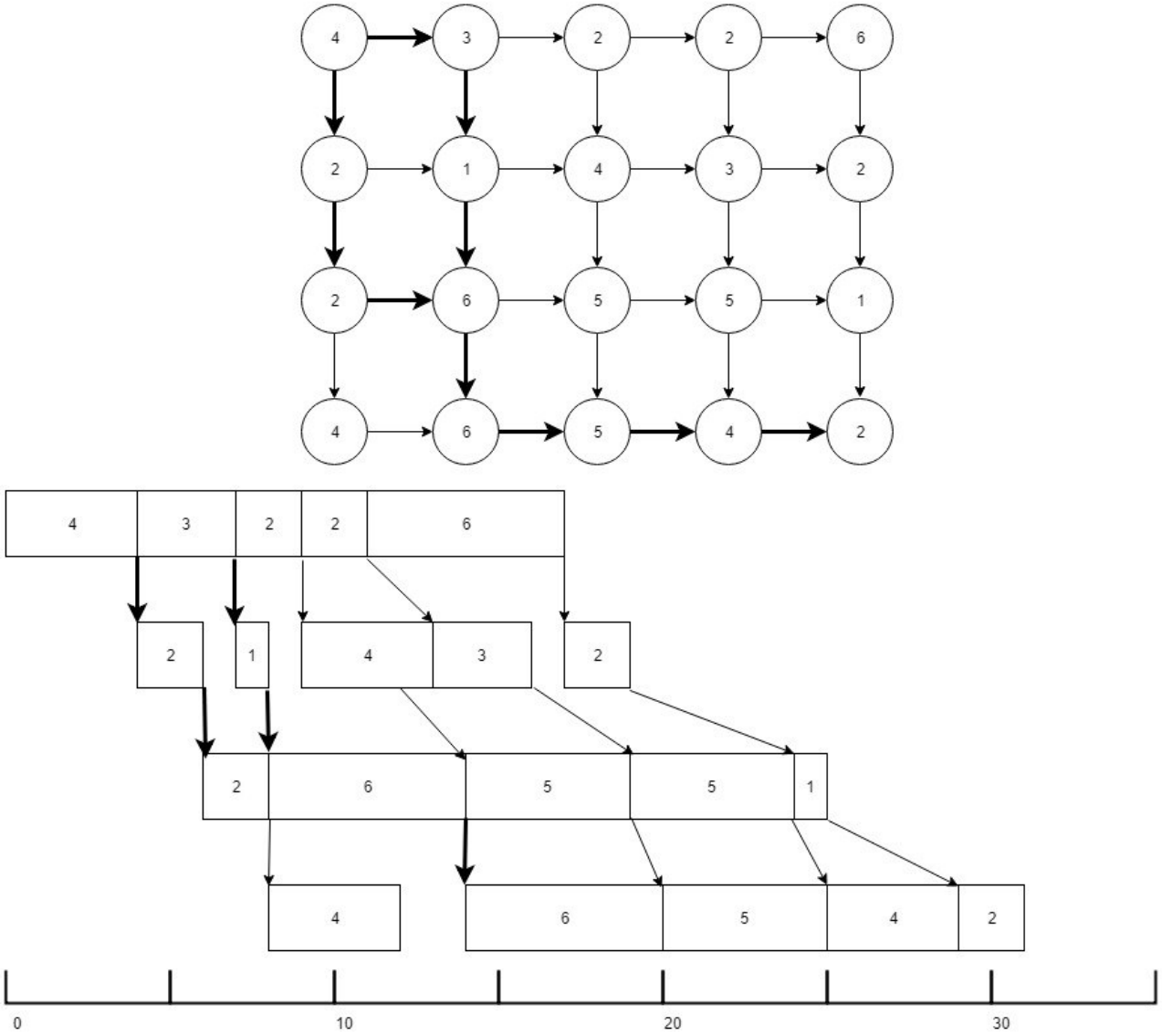
Trocando a permutação original, $1, 2, 3, 4, 5$ pela permutação $4, 3, 1, 2, 5$, ou seja, alterando a ordem das coluna, é obtido a Tabela 2.

Nessa nova permutação o valor de *makespan* passa a ser 31, como visto calculando os dois caminhos criticos mostrados no grafo da Figura 3.

tarefas	t4	t3	t1	t2	t5
m1	4	3	2	2	6
m2	2	1	4	3	2
m3	2	6	5	5	1
m4	4	6	5	4	2

Tabela 2 – Problema *flow shop* em forma de matriz da permutação final.

Figura 3 – Grafo direto, caminho critico e diagrama de Gantt referente a Tabela 2 (As entradas numéricas representam o tempo de processamento das tarefas.)



Fonte – Acervo do autor.

2.1.2 Métodos para resolução do problema *flow shop* permutacional

Métodos para resolução do problema *flow shop* permutacional podem ser dividido em duas categorias: Métodos exatos e métodos heurísticos. Como métodos exatos pode se citar a regra de Jonhson (JOHNSON, 1954), usada para resolver o problema pra duas

máquinas, e *Branch and Bound*. Porém, como o *PEFSP* é NP-Dificil, à medida que a escala do problema aumenta, é impraticável resolver o problema com os métodos exatos (GAREY; JOHNSON; SETHI, 1976) .Desde modo, heurísticas aproximativas são boas escolhas para resolução deste problemas.

Algoritmos heurísticos para resolução do *PFSP* podem ser categorizados como algoritmos construtivas e meta-heurísticas. AS heurísticas construtivas constroem uma solução passo a passo, introduzindo um elemento da solução a cada passo. Alguns métodos construtivos utilizadas para o *PFSP* são o Algoritmo de *Palmer* (PALMER, 1965), Algoritmo de *Gupta* (GUPTA, 1971), Algoritmo CDS (CAMPBELL; DUDEK; SMITH, 1970), Algoritmo *NEH(Nawas-Enscore-Ham)* (NAWAZ; JR; HAM, 1983).

Já as meta-heurísticas utilizam combinação de escolhas aleatórias e conhecimento histórico dos resultados anteriores adquiridos pelo método para se guiarem e realizar suas buscas pelo espaço de busca em vizinhanças dentro do espaço de pesquisa, o que reduz paradas prematuras em ótimos locais. Elas resolvem instâncias de problemas que se acredita serem difíceis em geral, explorando o espaço de pesquisa de solução geralmente grande dessas instâncias. Esses algoritmos conseguem isso reduzindo o tamanho efetivo do espaço e explorando esse espaço com eficiência. As meta-heurísticas servem a três finalidades principais: resolver problemas mais rapidamente, resolver grandes problemas e obter algoritmos robustos. Além disso, são simples de projetar e implementar, e são muito flexíveis (TALBI, 2009). Algumas abordagens de meta-heurísticas para o problema do *flow shop* são listadas a seguir.

(CHANG et al., 2013) propõe um algoritmo evolucionário baseado em blocos (AEBB) que conduzira operações em conjuntos de blocos em vez de genes. Os resultados do trabalho mostram que o *AEBB* compete com abordagens tradicionais como Algoritmo Genético Padrão e Algoritmo Genético com cromossomos artificiais e pode melhorar bastante o desempenho do algoritmo evolucionário e ainda ter uma economia de tempo.

Em (LIU; LIU, 2013) é apresentado um algoritmo hibrido de colonia de abelha discreta com *GRASP(Greedy Randomized Adaptive Search Procedure)* (MARQUES-SILVA; SAKALLAH, 1999) para minimizar o *makespan* no problema de escalonamento *flow shop* permutacional. Primeiro, a população inicial com certa qualidade e diversidade é gerada a partir do *GRASP*, baseado na heurística *NEH*. Segundo, os operadores e algoritmos discretos, como inserção, troca, religação de trajeto e *GRASP*, são aplicados para gerar nova solução para as abelhas, observadoras e exploradoras empregadas.

(LIU; YIN; GU, 2014) propõe uma evolução diferencial hibrida (EDH) chamada L-HDE para resolver o problema de escalonamento de *flow shop* que combina o DE com o esquema de melhoria Individual. Para tornar o DE adequado para o PEFSP é introduzida uma nova *Largest-Rank-rule (LRV)* baseada em chave aleatória é introduzida para converter a posição continua do DE para a permutação discreta para que o DE possa

ser usado para resolver *PFSP*.

(XIE et al., 2014) apresenta um algoritmo hibrido de Otimização Baseado em Ensino e Aprendizagem, que combina um novo algoritmo de otimização baseado em ensino-aprendizagem para evolução de soluções e busca de vizinhança variável (*VNS*) para melhoria rápida da solução. Um algoritmo de arrefecimento simulado é adotado como busca local do *VNS*.

2.2 Algoritmos de Otimização

Nesta seção é apresentado os algoritmos utilizados no trabalho. Os algoritmos considerados são *Multi-start Simulated Anneling (MSA)* (LIN et al., 2012), *(discrete artificial bee colony with composite mutation strategies, CDABC*) (LI; YIN, 2012) e um hibrido de AG e *Clustering Search* chamado de *Biased Random-Key and Clustering Search (BRKeCS)* (FONSECA; OLIVEIRA, 2017). São apresentados aspectos gerais dos algoritmos, pseudo-código e os parâmetros de desempenho.

2.2.1 Algoritmo *Multi-start Simulated Annealing (MSA)*

O algoritmo *Simulated Anneling* com multi-partida, proposto por (LIN et al., 2012), adota uma estrategia de escalada multi-partida no recozimento simulado. O uso da estrategia de escalada multi-partida é utilizada como uma maneira de alcançar diversificação e escapar de ótimos locais, pois essa estrategia executa o procedimento de busca com vários pontos de partida. A seguir é descrito o passo-a-passo do algoritmo;

Algoritmo 1 Algoritmo Arrefecimento Simulado com Multi-partida

Entrada: $\mathrm{I}_{iter}, T_0, T_F, \alpha etam P$

Saída: Melhor solução encontrada

início

Passo 1: Gerar as soluções iniciais $\sigma_k aleatoriamente$;
Passo 2: Seja T=T_0; $N = 0$;σ_{melhor}= a melhor σ_k entre as $P_{tamanho}$ soluções;
$TFT_{best} = obj(\sigma_{melhor})$;
Passo 3: $N = N + 1$;
Passo 4: Para k = 1 até $\mathrm{P}_{tamanho}$
Passo 4.1 Gerar a solução $\sigma_{k'}$ baseada em σ_k;
Passo 4.2
if Δ_k = *obj*($\sigma_k^{'}$)-*obj*(σ_k) ≤ 0 **then**
  Faça $\sigma_k = \sigma_k^{'}$;
else
  Gerar r U*0,1);
  if $r < Exp(\Delta_k/T)$ **then**
    Faça $\sigma_k = \sigma_k^{'}$
  else
    Descarta σ_k;
  end
end
Passo 4.3
if *obj*(σ_k) $< TFT_{best}$ **then**
  $\mathrm{sigma}_{best} = \sigma_k; TFT_{best} = obj(\sigma_k)$;
  Faça $\sigma_j = \sigma_{best} (j = 1, ..., P_{tamanho})$
end
Passo 5:
if N=I_{iter} ***then***
  T=$T \times \alpha$; $N = 0$;
  Realize a busca local para cada σ_k;
  if obj (σ_k) $< TFT_{best}$**then**
    $\sigma_{best} = \sigma_k; TFT_{best} = obj(\sigma_k)$;
    Faça $\sigma_j = \sigma_{best}$ (j=1,..., $\mathrm{P}_{tamanho}$)
  end
else
  Vai para passo 3
end
Passo 6:
if $T < T_f$ ***then*** $TerminaoprocedimentoMSA$
else *Vai para o passo 3*

fim

Afim de explorar a vizinhança de uma solução, neste algoritmo é usado as operações de troca e inserção com probabilidade de 50% para cada.

2.2.1.1 Exploração na vizinhança da solução

Para fazer a exploração da vizinhança, o algoritmo MSA utiliza as operações de inserção e troca com com probabilidade aleatória de escolha. A operação de inserção consiste em escolher aleatoriamente uma posição da solução reinseri-la a frente de outra posição escolhida. A operação de troca consiste na escolha aleatória de duas posições da solução e troca-la de lugar. O procedimento de exploração da vizinhança é feito tanto no melhoramento do SA quanto para a busca local, onde esse procedimento é repetido n vezes onde n é o tamanho do vetor de solução.

2.2.1.2 Parâmetros de desempenho

O conjunto de parâmetros de desempenho do MSA é listado a seguir:

- $I_{iteração}$: esse parâmetro é responsável por definir um número iterações realizada pelo algoritmo em uma temperatura particular.
- T_0: esse parâmetro define a temperatura inicial;
- T_F: esse parâmetro define a temperatura final do algoritmo;
- α: esse parâmetro define o coeficiente de controle de arrefecimento;
- $P_{tamanho}$: esse parâmetro define o tamanho da população de soluções mantidas pelo algoritmo.

2.2.2 Algoritmo *Discrete Artificial Bee Colony with Composite mutation strategies (CDABC)*

O algoritmo de colônia de abelhas artificiais é um algoritmo evolutivo introduzido por(KARABOGA, 2005). Este algoritmo simula o comportamento de forrageamento da colônia de abelhas. Cada ciclo da pesquisa consiste em três etapas: mover as abelhas empregadas e observadoras para as fontes de alimento, calcular suas quantidades de néctar, determinar as abelhas exploradoras e, em seguida, movê-las aleatoriamente para a possível fonte de alimento. Uma fonte de alimento representa uma possível solução para o problema a ser otimizado.

O algoritmo colonia de abelhas artificial discreta com estratégias de mutação composta *(Discrete Artificial Bee Colony with Composite mutation strategies, CDABC*) é apresentado por (LI; YIN, 2012) para compensar os defeitos do esquema de mutação única que facilmente fica preso a ótimos locais. As estrategias de mutação composta atuam sobre uma permutação de tarefa, considerada uma fonte de alimento, aplicando operações discretas para gerar uma nova fonte de alimento vizinha para diferentes abelhas. Ao final de

cada iteração, a busca local rápida é usada para melhorar o melhor indivíduo. O Algoritmo 2 exemplifica o funcionamento da CDABC.

Algoritmo 2 Algoritmo Colonia de abelha discreta

Entrada: $\mathrm{I}_{iter}, T_0, T_F, \alpha etamP$

início

Passo 1: Gere as soluções
Passo2: Avalie o *fitness* para cada individuo e determine o melhor. Passo 3: enquanto critério de para não satisfeito, faça:
Passo 3.1: fase das abelhas operarias - produzir uma nova fonte de alimento para a i-ésima abelha operaria associada a solução Π_i, usando as estrategias de mutação e escolhendo a melhor. Então, quanto à seleção, se a nova fonte de alimento é melhor que a atual, então aceita-se a nova fonte de alimento. Caso contrario, a a fonte atual é mantida.
Passo 3.2: Atualize as probabilidades para cada individuo;
Passo 3.3: fase das abelhas observadoras - selecione uma fonte de alimento para a abelha observadora de acordo com a probabilidade associada à aquela fonte de alimento. Produzir uma nova solução para a observadora usando a estrategia de mutação composta e avalie-a. Se a nova fonte de alimento é melhor que a atual, então aceita-se a nova fonte de alimento. Caso contrario, a a fonte atual é mantida.
Passo 3.4: fase das abelhas exploradoras - se uma solução não melhora por um pre-determinado numero de iterações chamado "limite", então está fonte de alimento é abandonada pela abelha operaria e a abelha se torna uma exploradora, a exploradora gera uma fonte de alimento realizando varias operações de "inserção"na melhor fonte de alimento na população.
Passo 3.5: avalie o fitness para cada individuo e determine o melhor individuo
Passo 3.6:faça uma busca local no melhor individuo
Passo 3.7: vá para o Passo 3
Passo 4: retorne a melhor solução encontrada

fim

2.2.2.1 Estrategia de mutação

Para resolver o PFSP, (LI; YIN, 2012) propôs um composto de estratégias de mutação que é empregado no algoritmo DABC. isto é feito para evitar que as soluções fiquem presas em diferentes ótimos locais. As operações de mutação usadas no DABC são:

- SWAP mutation: escolhe aleatoriamente duas posições diferentes de uma permutação de tarefas e as trocam.
- INSERT mutation: seleciona aleatoriamente um job e o realoca para trás de outro job selecionado.
- INVERSE mutation: inverte a subsequência entre as duas diferentes posições aleatórias de uma permutação de trabalho.

- Adjacent exchange: escolha aleatoriamente duas posições adjacentes de uma permutação de trabalho e troque-as.

Cada método para a geração de fontes alimentares vizinhas pode ter diferentes desempenhos durante o processo de evolução. No processo de busca de vizinhança, cada estrategia de mutação é usada para gerar uma nova fonte de alimento. Então, a melhor é escolhida ser for melhor que a fonte atual. A Tabela 3 apresenta exemplos de cada uma das operações de mutação.

Mutação	Origem	Resultado
Insertion	3 1 4 5 6 2 8 7	3 1 8 4 5 6 2 7
Inverse	3 1 4 5 6 2 8 7	3 1 4 8 2 6 5 7
Swap	3 1 4 5 6 2 8 7	3 1 4 8 6 2 5 7
Adjacent exchange	3 1 4 5 6 2 8 7	3 1 4 5 6 8 2 7

Tabela 3 – Exemplos das operações de mutação. A posição sublinhada é onde ocorre a operação.

2.2.2.2 Parâmetros de desempenho

O conjunto de parâmetros de desempenho do CDABC é listado a seguir:

- *sn*: esse parâmetro é responsável por definir o número de abelhas operarias;
- *Observador_tamanho*: esse parâmetro define o número de abelhas observadoras;
- *limite*: esse parâmetro define o número de não melhoramento que uma fonte de alimento pode ter antes de ser descartada e substituída;
- *mut_rate*: esse parâmetro define a porcentagem de alteração feita a melhor fonte encontrada para gerar uma fonte nova para a abelha exploradora;
- $fator\_bl$: esse parâmetro define número de execuções da busca local na melhor solução encontrada na iteração e na melhor global, o número de busca é calculado como $numeroDeBuscas = fator\_bl * tam\_fonte$.

2.2.3 Algoritmo *Biased Random-Key Evolutionary Clustering Search (BRKeCS)*

BRKeCS é um algoritmo genético híbrido usado para solucionar instâncias de Flow Shop Permutacional. O algoritmo, nomeado como *Biased Random-Key Evolutionary Clustering Search (BRKeCS)*(FONSECA; OLIVEIRA, 2017), é baseado no algoritmo genético de chaves aleatórias viciadas(acrônimo em inglês *BRKGA*) e na busca guiada por agrupamentos (acrônimo em inglês *CS*).

Um algoritmo genético de chaves aleatórias viciadas (biased random-key genetic algorithm – BRKGA) é uma metaheurística evolutiva para problemas de otimização discreta e global baseada no algoritmo de chaves aleatórias de (BEAN, 1994). Cada solução é representada por um vetor de n chaves aleatórias, onde uma chave aleatória é um número real, gerado aleatoriamente, no intervalo contínuo [0, 1). Um decodificador mapeia um vetor de chaves aleatórias numa solução do problema de otimização e calcula o custo desta solução. O algoritmo começa com uma população de p vetores de chaves aleatórias. A cada iteração, os vetores são particionados em um pequeno conjunto com os melhores elementos (o conjunto elite) e o restante (o conjunto não-elite). Todos os elementos elite são copiados sem alteração para a população da próxima iteração. Um número pequeno de vetores de chaves aleatórias (os mutantes) é adicionado à população da próxima iteração. O restante da população da próxima iteração é composta de soluções geradas pela combinação uniforme parametrizada de (SPEARS; JONG, 1995) de pares de soluções, onde uma solução é elite e a outra não (RESENDE,).

Clustering Search (CS) é uma forma genérica de combinar meta-heurísticas de otimização com algoritmos de agrupamento visando detectar regiões promissoras do espaço de busca para que essas regiões seja iterativamente exploradas por heurísticas específicas do problema (OLIVEIRA, 2004). CS divide dinamicamente o espaço de busca em *clusters* a partir das soluções candidatas amostradas pela meta-heurística de suporte.

Algoritmo 3 Algoritmo BRKeCS

Entrada: Conjunto de parâmetros
Saída: Melhor solução encontrada
início

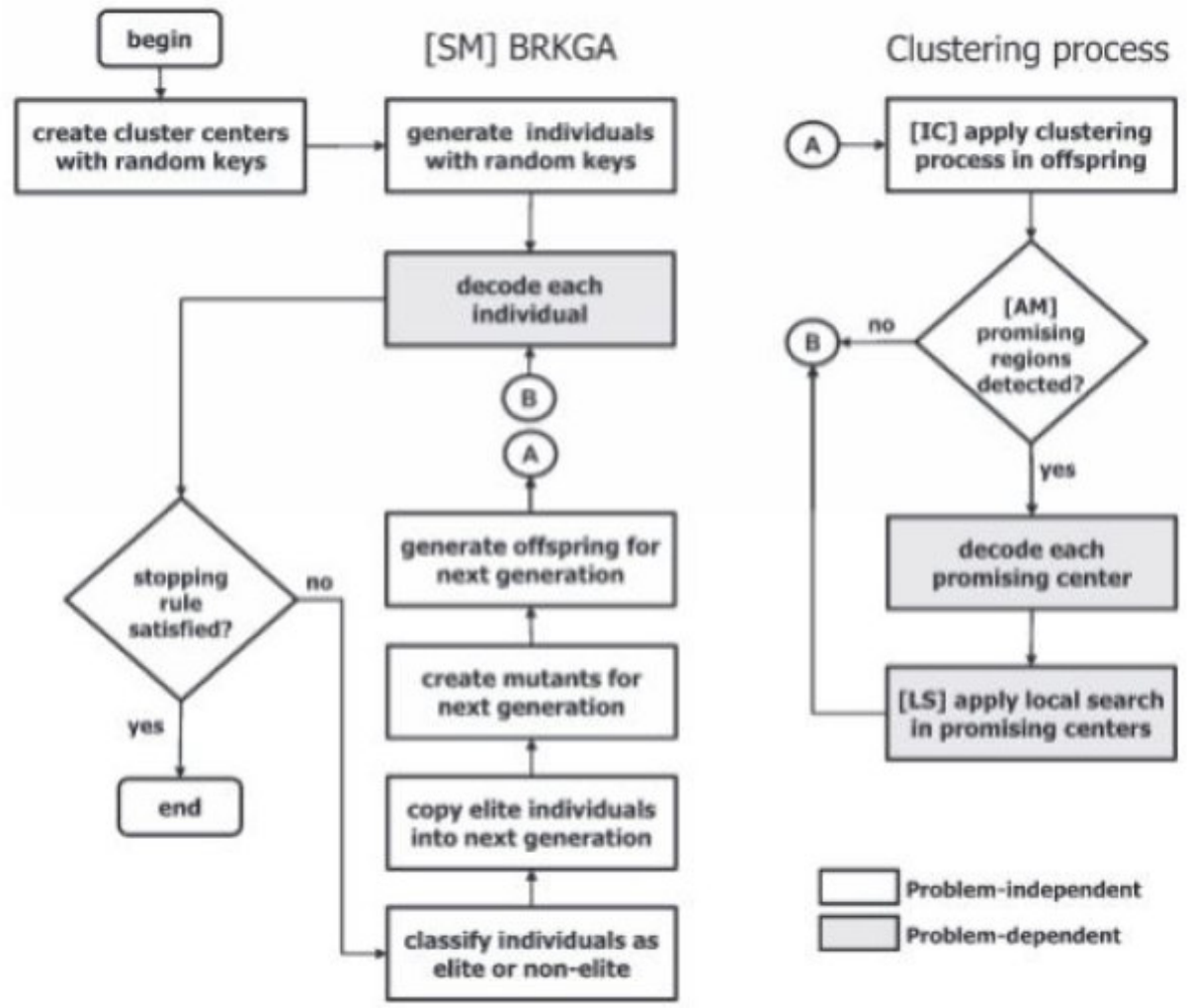
Passo 1: inicialize o valor da melhor solução encontrada
Passo 2: enquanto critério de parada não satisfeito, faça:
Passo 2.1: Gere uma população P com vetores de n chaves aleatórias e avalie o custo de cada solução em P;
Passo 2.2: Particione P em dois grupos, um com as soluções mais bem avaliadas (População elite - P_e) e outro com o restante(População não elite - P_{ne};
Passo 2.3 Inicialize a próxima geração com a elite
Passo 2.4: Gere um conjunto de mutantes P_m e adicione à população da próxima geração;
Passo 2.5: Para completar a população da próxima geração, gere $(|P| - |P_e| - |P_m|)$ indivíduos usando *BLX*, escolhendo aleatoriamente um pai da P_n e outro da P_{ne}.
Passo 2.6: Adicione, individualmente a População gerada(com exceção da elite) ao *cluster*;
Passo 2.7 Atualize a População e a melhor solução encontrada;
Passo 3: retorne a melhor solução encontrada

fim

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 4.

Figura 4 – Modelo conceitual do BRKeCS. Apenas o decodificador e a busca local são dependente do problema.



Fonte – (FONSECA; OLIVEIRA, 2017).

2.2.3.1 Cruzamento

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

2.2.3.2 Assimilação

A operação de assimilação ocorre sempre que um *cluster* é ativado por um indivíduo gerado pela meta-heurística, causando uma pertubação na localização do centro,

em geral, proporcional à distância para o indivíduo assimilado. 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.

2.2.3.3 Parâmetros de desempenho

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

1. Parâmetros da busca local 2-opt:

- altura: esse parâmetros é responsável por definir um número de trocas realizadas por busca local 2-OPT dentro de um mesmo segmento de largura;
- largura: responsável por definir o segmento de tarefas dentro uma solução de PFSP a qual seriam realizadas as trocas;
- r_{max}: defini 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;
- porcentagem de indivíduos em um cluster com relação ao tamanho da população para que o mesmo seja considerado promissor.

3 Sintonização de meta-heurística por método de corrida

Neste Capítulo é descrito o conceito de configuração de algoritmos. Também é apresentado o pacote *Irace*, com descrição sobre método de corrida e funcionamento geral do pacote.

3.1 Configuração de algoritmos

Muitos algoritmos de otimização são configuráveis, ou seja, possuem parâmetros que devem ser ajustados, e suas performance em um problema especifico geralmente depende de uma determinada configuração desses parâmetros. Este é o caso de muitos tipos de algoritmos que vão desde métodos exatos, como *branch-and-bound* e as técnicas implementadas em *solvers* de programação inteira moderna, até métodos heurísticos, como busca local ou meta-heurísticas(LÓPEZ-IBÁÑEZ et al., 2016b). Por isso, é crucial a utilização de métodos de sintonização de parâmetros para selecionar o conjunto ótimo de parâmetros.

A importância da sintonia é reconhecida pela comunidade cientifica (BARR et al., 1995): é evidente para quem tem alguma experiencia 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.

Para um dado algoritmo, considerando um conjunto de instância de problema e uma medida de custo, o objetivo da sintonização automática é encontrar a melhor configuração de parâmetros que minimize a medida de custo para um conjunto de experimentos de aplicação do algoritmo ao conjunto de instâncias.

(BIRATTARI; KACPRZYK, 2009) apresenta uma definição geral para o problema de sintonizar uma meta-heurística, considerando os seguintes objetos:

- Θ é o conjunto de configurações candidatas;
- I é o conjunto de instancias;
- P_I é a medida de probabilidade sobre o conjunto de instâncias I, sendo $P_I(i)$ a probabilidade que a instância i seja selecionada para ser resolvida;
- $t : I \rightarrow IR$ é a função que relaciona o tempo computacional para cada instância;

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

Com base nesses conceitos, o problema de configuração de meta-heurísticas pode ser descrito pela 7-tupla $\{\langle\Theta, I, P_I, P_C, t, C, T\rangle\}$. A solução deste problema é a configuração $\bar{\theta}$ tal que:

$$\bar{\theta} = argmin_{\theta} C(\theta) \tag{3.1}$$

Espera-se encontrar o custo $C(\theta)$ expresso pela equação(BIRATTARI; KACPRZYK, 2009):

$$C(\theta) = E_{I,C[c]} = \int c dP_C(c|\theta, i) dP_I(i) \tag{3.2}$$

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.

Em relação a estratégias para otimização de parâmetros de meta-heurísticas, na grande maioria dos casos, meta-heurísticas são sintonizadas manualmente em um procedimento de tentativa e erro, guiado por algumas regras básicas. Essa abordagem é tipicamente adotada na maioria dos trabalhos de pesquisa que tratam de meta-heurísticas. Poucos pesquisadores declaram isso explicitamente e mostram estar cientes da limitação da abordagem - ver, por exemplo, (BREEDAM, 1995) e (GENDREAU; HERTZ; LAPORTE, 1994) - enquanto a grande maioria nem sequer dá uma palavra sobre o assunto(BIRATTARI; KACPRZYK, 2009).

Em relação a estrategia para sintonização automática de meta-heurísticas, a abordagem de força bruta é uma maneira simples de prover um ajuste para uma meta-heurística. Na estratégia de força bruta o poder computacional é atribuído uniformemente

para todas as configurações candidatas previamente definidas. Deste modo, a estratégia executa o mesmo número de experimentos para cada configuração candidata. O que gera um problema: configurações candidatas de qualidade inferior são testadas tanto quanto aquelas consideradas boas. Para evitar este problema, o método de Corrida tradicional emprega uma estatística robusta para avaliar as configurações candidatas durante o processo de sintonização e descartar aquelas consideradas estatisticamente inferiores, assim que coletar evidências suficientes contra elas. A eliminação de candidatos inferiores acelera o procedimento e permite avaliar as configurações em mais instâncias, assim, obter estimativas mais confiáveis do seu comportamento(BIRATTARI; KACPRZYK, 2009).

3.2 Irace

O pacote *Irace (Iterated racing)* implementa um procedimento de corrida iterada, que é uma extensão do procedimento iterado *F-race(I/F-Race)* proposto por(BALAPRAKASH; BIRATTARI; STÜTZLE, 2007). O principal objetivo do *irace* é configurar automaticamente os algoritmos de otimização encontrando as configurações mais apropriadas, considerando um conjunto de instâncias de um problema de otimização (LÓPEZ-IBÁNEZ et al., 2011).

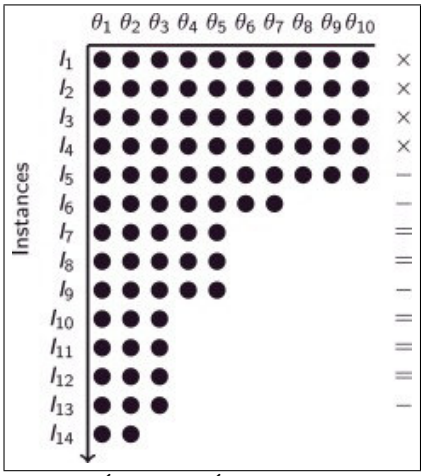
Corrida iterada consiste de três passos: (1) Amostragem de novas configurações de acordo com uma distribuição particular inicial; (2) aplicar e selecionar as melhores configurações das recém-amostradas; e (3) atualizar a distribuição de amostragem para influenciar as próximas corridas. Estes três passos são repetidos até que um critério de parada seja alcançado (LÓPEZ-IBÁNEZ et al., 2011).

No pacote *Irace*, cada parâmetro configurável tem associado uma distribuição de amostragem independente, além das restrições e condições entre os parâmetros. A distribuição amostral é uma distribuição normal truncada para parâmetros numéricos ou uma distribuição discreta para parâmetros categóricos.Parâmetros ordinais são tratados como numéricos (inteiros). A atualização das distribuições consiste em modificar a média e o desvio padrão no caso da distribuição normal, ou os valores de probabilidade discreta das distribuições discretas. A atualização polariza as distribuições para aumentar a probabilidade de amostrar, em futuras iterações, os valores dos parâmetros nas melhores configurações encontradas até o momento.(LÓPEZ-IBÁÑEZ et al., 2016b).

Após novas configurações serem amostradas, as melhores configurações são selecionadas por meio de corridas. Uma corrida começa com um conjunto finito de configurações candidatas. Em cada etapa da corrida, as configurações candidatas são avaliadas em uma única instância. Depois de várias etapas, as configurações candidatas que apresentam desempenho estatisticamente pior do que pelo menos uma outra são descartadas, e a corrida continua com as configurações sobreviventes restantes. Esse procedimento continua

até atingir um número mínimo de configurações sobreviventes, um número máximo de instâncias que foram usadas ou um orçamento computacional predefinido. Esse orçamento computacional pode ser um tempo de computação geral ou um número de experimentos, em que um experimento é a aplicação de uma configuração a uma instância(LÓPEZ-IBÁÑEZ et al., 2016b). Na Figura 5 é mostrado um exemplo de corrida com 10 configurações.

Figura 5 – Corrida para configuração automática de algoritmo (LÓPEZ-IBÁÑEZ et al., 2016b). Cada linha é a avaliação de uma configuração em uma instância. 'x' significa que nenhum teste estatistico é realizado, '-', significa que o teste descartou pelo menos uma configuração, '=' significa que o teste não descartou nenhuma configuração. Neste exemplo $T^{furst} = 5$ e $T^{each} = 1$.



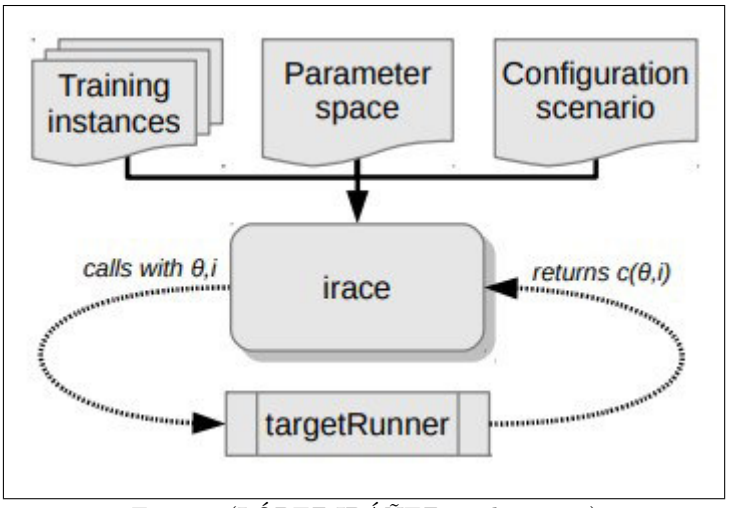
Fonte – (LÓPEZ-IBÁÑEZ et al., 2016b).

O pacote irace fornece uma ferramenta de configuração automática para ajustar os algoritmos de otimização, ou seja, encontrar automaticamente boas configurações para os valores dos parâmetros de um algoritmo (de destino) salvando o esforço que normalmente requer ajuste manual. a Figura 6 dá um esquema geral de como *irace* funciona. *Irace* recebe como entrada uma definição de espaço de parâmetro correspondente aos parâmetros do algoritmo de destino que será ajustado, um conjunto de instâncias para as quais os parâmetros devem ser ajustados e um conjunto de opções para *irace* que definem o cenário de configuração (LÓPEZ-IBÁÑEZ et al., 2016a).

O espaço de parâmetros define os parâmetros a serem configurados, seus tipos, domínios e restrições. Na descrição dos parâmetros são definidos os tipos dos parâmetros(inteiro, real, ordinal ou categórico); rótulos para passar os valores destes parâmetros por linha de comando; o domínio dos parâmetros, que pode ser o limite superior e inferior para os tipos inteiros e reais, ou o conjunto de valores para parâmetros categóricos ou ordinais; Opção condicional que determina se o parâmetro está ativado ou desativado, tornando o parâmetro condicional. parâmetros condicionais são apenas quando outros tem certos valores.

O conjunto de instâncias de treinamento que na prática é uma amostra finita e

Figura 6 – Esquema do fluxo de informação do *irace*



Fonte – (LÓPEZ-IBÁÑEZ et al., 2016a).

representativa do conjunto de todas as instancias. o conjunto de instancias pode ser dado explicitamente, como um diretório, para o *irace* ou, como alternativa, as instâncias podem ser lidas de um arquivo de instância.

O cenário de configuração irá fornecer toda a informação necessária para otimizar os parâmetros do algoritmo. Os parâmetros mais importantes do cenário de configuração são: *maxExperiments*, que define o número máximo de experimentos realizado pelo *irace*. onde cada experimento é a execução de uma configuração em uma instancia; *targetRunner*, serve como medidor de custo, sendo responsável por fazer a chamada ao algoritmo de acordo com a configuração de parâmetros encontrada e retorna como saída o desempenho do algoritmo; *firstTest*, define o número de instancia a serem vista antes do primeiro teste estatístico, a primeira iteração é muito importante para definir a distribuição dos parâmetros, por padrão é 5; *eachTest*, define o número de instâncias vista cada iteração; *testType*, define o tipo de teste estatístico podendo ser o teste t, que usa o desempenho médio das configurações para analisar as diferenças entre as configurações, ou teste de *Friendman*, que usa a classificação das configurações para analisar as diferenças entre seu desempenho.

4 Meta-aprendizagem

Neste capítulo, os principais conceitos inerentes à meta-aprendizagem são apresentados.

4.1 Conceito

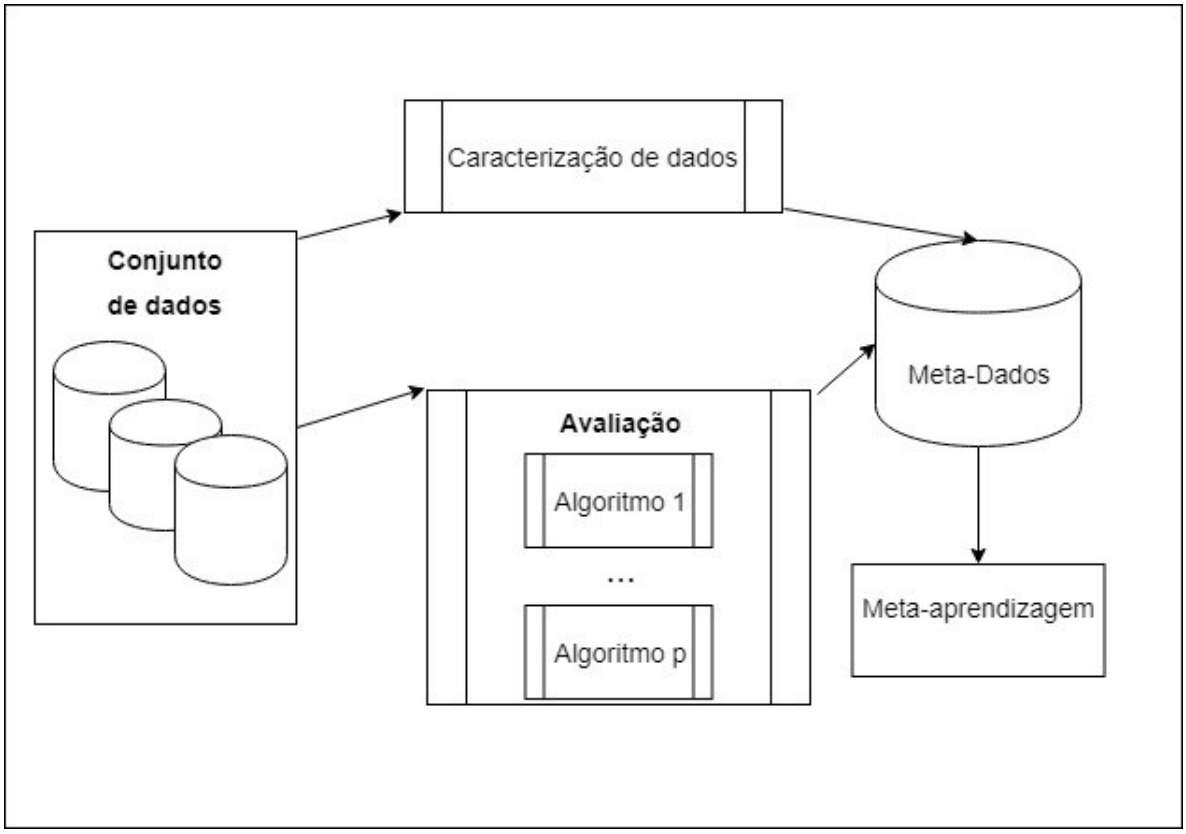
Projetar um algoritmo para resolver um novo problema é uma tarefa difícil. As dificuldades surgem principalmente da variedade significativa de parâmetros que precisam ser ajustados e da própria escolha do algoritmo que pode não ser o melhor para resolver determinado problema. Isso motiva o crescimento do interesse em pesquisas de técnicas para automatizar o projeto de algoritmos em otimização, aprendizagem de máquinas e outras áreas da informática. Como é o caso da meta-aprendizagem, que com base nos dados de treinamentos que compreende características de entrada e a classe de saída que mostra qual algoritmo é o melhor em um conjunto de algoritmos, a tarefa é aprender um mapeamento para produzir uma previsão de qual algoritmo será melhor para instâncias de testes não vistas (SMITH-MILES, 2009).

Meta-aprendizagem é uma subárea de aprendizagem de Máquinas que investiga a seleção do melhor algoritmo para uma determinada tarefa (BRAZDIL et al., 2008). A Meta-aprendizagem estuda como os sistemas de aprendizagem podem aumentar em eficiência através da experiência, almejando ainda entender como o aprendizagem em si pode se tornar flexível de acordo com o domínio ou a tarefa em estudo (VILALTA; DRISSI, 2002). Além disso, alguns dos classificadores induzem modelos que podem ser explicados e podem ser usados como ferramenta adicional para melhor entender a estrutura do problema (MIRANDA et al., 2018).

Um processo de meta-aprendizagem aborda o problema de seleção de algoritmos de forma semelhante a um processo de aprendizagem tradicional, através da indução de um meta-modelo, que pode ser visto como o meta-conhecimento capaz de explicar por que certos algoritmos funcionam melhor do que outros em conjuntos de dados com características específicas. O meta-modelo também pode ser usado para prever o melhor algoritmo para um novo conjunto de dados / problema (SERBAN et al., 2013). A Figura 7 apresenta o processo de meta-aprendizagem.

Um dos principais desafios em meta-aprendizagem é definir quais são as meta-características que efetivamente descrevem quão fortemente um conjunto de dados corresponde ao viés de cada algoritmo (BRAZDIL et al., 2008). Outro desafio é definir qual o métrica de desempenho melhor retrata a performance de um conjunto de algoritmo para

Figura 7 – Processo de Meta-Aprendizagem.



Fonte – Adaptado de (BRAZDIL et al., 2008).

uma instância especifica.

Meta-aprendizagem tem sido utilizado para a seleção de algoritmo em diversos domínios como por exemplo em problemas de selecionar o melhor classificador para conjuntos de dados desbalanceados (MORAIS; MIRANDA; SILVA, 2016), classificação de dados de expressão genética (SOUZA; CARVALHO; SOARES, 2008) e para para a recomendação de algoritmos de segmentação de imagem (CAMPOS; BARBON; MANTOVANI, 2016). A tarefa de seleção de algoritmos pode ser vista como um problema de aprendizagem, lançando-a no âmbito da predição. Para tal, usa-se uma meta-base, onde cada meta-exemplo corresponde a um conjunto de dados. Para cada meta-exemplo, as características preditivas (meta-características) são extraídas do conjunto de dados correspondente e os objetivos são os desempenhos (meta-rotulo) de um conjunto de algoritmos quando são aplicados no conjunto de dados (BRAZDIL et al., 2008).

Algumas das aplicações de meta-aprendizagem para recomendação de algoritmos são abordadas em (DÔRES et al., 2016) que utiliza uma estrutura de meta-aprendizagem para recomendação de algoritmos na previsão de falha de software e (KANDA et al., 2016) que utiliza meta-aprendizagem para recomendar meta-heurísticas para o problema do caixeiro viajante.

Nesse sentido, meta-aprendizagem tem se mostrado promissora como uma abordagem capaz de produzir modelos de associação entre instâncias de problemas de otimização e *solvers*. Sendo mais específico, um sistema de recomendação, baseado em meta-aprendizagem, deve ser capaz de selecionar a melhor meta-heurística para resolver uma instância de problema em particular. Para conseguir isso, são extraídas características de um conjunto de instâncias de problemas, para as quais são conhecidos os desempenhos de várias meta-heurísticas, e um algoritmo de aprendizagem de máquina pode ser usado para aprender associações entre as características e as meta-heurísticas com melhores desempenhos. Dessa forma, um modelo pode ser induzido a prever qual meta-heurística deve ser usada para resolver um problema, dadas as suas características, mesmo que ele seja totalmente desconhecido a priori (PAVELSKI; DELGADO; KESSACI, 2018).

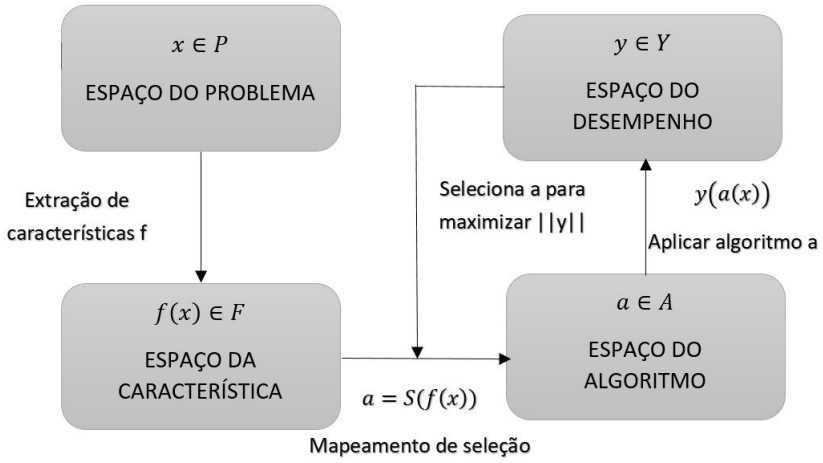
4.2 Histórico

Em 1976 John Rice, (RICE, 1976), apresentou um modelo formal abstrato que pode ser usado para explorar a questão: Com tantos algoritmos disponíveis, qual provavelmente é o melhor para o meu problema e instância específica? A estrutura do modelo abstrato proposto por Rice, Figura 8, contém 4 componentes essenciais: o espaço de problema P representa o conjunto de instâncias de uma classe de problema; o espaço de característica F contém as características mensuráveis das instâncias geradas por um processo computacional de extração de características aplicado a P; o espaço do algoritmo A é o conjunto de todos os algoritmos considerados para abordar o problema; o espaço de desempenho Y representa o mapeamento de cada algoritmo para um conjunto de métricas de desempenho. Essas quatro componentes permitem analisar qual algoritmo do conjunto de algoritmo pode apresenta o melhor resultado para uma instância visto que o teorema de *NFL (No Free Lunch*) para otimização (WOLPERT; MACREADY et al., 1997) afirma que, se o algoritmo A superar o algoritmo B em algumas funções de custo, então deve existir exatamente muitas outras funções em que B supera a A. os teoremas da NFL sugerem que é necessário afastar-se de uma abordagem de caixa preta para seleção de algoritmos.

Deve-se entender mais sobre as características do problema para selecionar o algoritmo mais adequado que idealmente levará em consideração propriedades estruturais conhecidas do problema ou instância. A escolha das características dependem tanto do domínio do problema quanto do conjunto de algoritmo escolhido. As características devem ser escolhidas para que as complexidades variáveis das instâncias do problema sejam expostas, quaisquer propriedades estruturais conhecidas dos problemas sejam capturadas e quaisquer vantagens e limitações conhecidas dos diferentes algoritmos estejam relacionadas as características (SMITH-MILES, 2009).

O projeto do algoritmo automatizado apareceu, na área de aprendizagem de

Figura 8 – Estrutura do Problema de Seleção de Algoritmo proposto por (RICE, 1976).



Fonte – (RICE, 1976).

máquina, como uma extensão natural dos primeiros trabalhos com foco na seleção de algoritmos automatizados, segundo (PAPPA et al., 2014). A área de meta-aprendizagem tomou o desafio e começou a tomar forma no final dos anos oitenta, mas foi formalmente introduzida em 1992 com o projeto MLT(*Machine Learning Toolbox*) (KODRATOFF et al., 1992), destinado a criar um sistema especializado, chamado Consultant-2, para ajudar na seleção ou recomendação do melhor algoritmo para um determinado problema. Este primeiro projeto foi seguido por outros dois, nomeadamente Statlog (MICHIE; SPIEGELHALTER; TAYLOR, 1994) e METAL (BRAZDIL; SOARES; COSTA, 2003). A principal diferença, apresentada nos três projetos, entre meta-aprendizagem e a abordagem tradicional de aprendizagem de máquina, foi seu nível de adaptação. Ao aprender no nível base, concentra-se na acumulação de experiência em uma tarefa de aprendizagem específica; aprendendo no nível meta, a experiência é acumulada pelo desempenho de múltiplas aplicações de um sistema de aprendizagem. Posteriormente, a meta-aprendizagem desenvolveu outros ramos de pesquisa, como a combinação de modelos e, mais recentemente, a geração automatizada de algoritmos (PAPPA; FREITAS, 2009).

Atualmente, os domínios de aplicação do problema de seleção de algoritmo são variados, desde problemas para determinar qual melhor classificador para uma base de dados, a problemas de satisfação de restrição, otimização, regressão, predição de falha de software, dentre outros. A própria escolha do melhor algoritmo para selecionar a função de mapeamento S consiste de outro problema de seleção de algoritmo. Para a comunidade de aprendizagem de máquinas, porém, esse era um problema de aprendizagem padrão. Com base nos dados de treinamento (meta-dados), que compreendem padrões de entrada $f(x)$, e a classe de saída, que mostra qual algoritmo é o melhor, a tarefa é determinar o mapeamento S para produzir uma previsão de do melhor algoritmo para instâncias de

teste, ainda não conhecidas (SMITH-MILES, 2009).

4.3 *Meta-características*

O objetivo do meta-aprendizagem é relacionar o desempenho dos algoritmos de aprendizagem com as características dos dados, ou seja, as meta-características. Assim sendo, é necessário ter métodos para extrair as características boas o suficiente para representar o problema abordado. Os principais conjuntos de meta-características são: medidas simples (MS), medidas estatísticas (ME), medidas de teoria da informação (MTI), medidas de complexidade de dados (MCD), medidas discriminantes (MD), medidas de modelo (MM), medidas de rede complexa (MRC) e *Landmarkers* (LM).Nesta seção será apresentada as propriedades de cada conjunto afim de clarificar onde podem ser usados.

Em (MICHIE; SPIEGELHALTER; TAYLOR, 1994), são apresentadas medidas que descrevem conjuntos de dados. Essas medidas são de três tipos: (i) medidas simples, como o número de exemplos; (ii) baseadas em estatística, como a assimetria dos atributos; e (iii) informação teórica, como o ganho de informação dos atributos. As características extraídas dos conjuntos de dados também são variadas e tem relação direta com o tipo de dados do conjunto de dados. Medidas simples conseguem extrair atributos numéricos diretamente do conjunto de dados. Medidas estatísticas extraem dados numérico complexo e medidas de teoria da informação extraem valores de atributos categóricos.

As medidas de complexidade de dados representam a complexidade de um problema de classificação considerando aspectos como a sobreposição nos valores dos recursos, a separabilidade das classes e propriedades geométricas ou topológicas (BASU; HO, 2006). Exemplos de medidas de complexidade de dados são discriminante de Fisher, Dimensão fractal do conjunto de dados, Dispersão do conjunto de dados.

Medidas de análise discriminante são aplicáveis a atributos numéricos. Como exemplo dessas medidas tem-se o *Fract*, que descreve a importância relativa do maior autovalor como indicação para a importância da 1ª função discriminante; correlação canônica, que é um indicador para o grau de correlação entre a função discriminante mais significante e a distribuição da classe; Número de funções discriminantes (LINDNER; STUDER, 1999).

Na caracterização de dados baseada em modelo, um modelo é induzido a partir dos dados e as meta-características baseiam-se nas propriedades desse modelo. Caso o modelo puder ser relacionado aos algoritmos candidatos, abordagens assim tendem a fornecer recursos úteis (BHATT; THAKKAR; GANATRA, 2012).

Uma rede complexa é um grafo rede com características topológicas não triviais, isto é, recursos que não ocorrem em redes simples, como redes ou grafos aleatórios,

mas geralmente ocorrem em grafos reais. As redes complexas são uma área de pesquisa ativa devido ao grande número de aplicações do mundo real, como a regulação de genes, redes informáticas e sociais (GANGULY; DEUTSCH; MUKHERJEE, 2009). Exemplos de características baseadas em Redes complexas são: grau médio, média do menor caminho ponderado e não ponderado, média do diâmetro do grafo(GANGULY; DEUTSCH; MUKHERJEE, 2009).

Landmarkers são estimativas rápidas do desempenho do algoritmo em um determinado conjunto de dados. As estimativas podem ser obtidas executando versões simplificadas dos algoritmos. Uma maneira alternativa de obter estimativas de desempenho rápido é executar os algoritmos cujo desempenho pretende-se estimar em uma amostra dos dados, obtendo os chamados marcadores terrestres de sub-amostragem (BHATT; THAKKAR; GANATRA, 2012). Um exemplo de abordagem baseada em *landmarkers* são as características baseadas em propriedades de meta-heurísticas encontradas em (KANDA et al., 2016)

As características extraídas dos conjuntos de dados também são variadas e têm relação direta com o tipo de dados do conjunto de dados. Medidas simples conseguem extrair atributos numéricos diretos, medidas estatísticas permitem extrair dados numéricos complexos e medidas de teoria da informação, por sua vez, são úteis para atributos categóricos.

Existem ainda medidas baseadas em representação de grafo como medidas de vértice e aresta e medidas de rede complexa. Grafos são muito populares para representar instâncias de problemas de otimização ou matrizes de dados.

4.4 Medidas de desempenho

A medida de desempenho define os critérios a serem usados para avaliar um algoritmo para um problema específico. Algoritmos diferentes podem ser apropriados para diferentes problemas de acordo com esta medida (PAPPA et al., 2014). O uso e interpretação de tais métricas não se restringem apenas a avaliar a dificuldade de resolver determinada instância de problema por um algoritmo mas também a outros fatores que podem ser priorizados como tempo de execução, eficácia, simplicidade.

Alguns exemplos de medidas de desempenho utilizadas na literatura são: equilíbrio que é a distância euclidiana entre a previsão gerada e o equilíbrio ideal entre a taxa de verdadeiro positivo e a taxa de falsos positivos (MENZIES; GREENWALD; FRANK, 2007), é utilizada para medição em previsão de falhas de software; relação ajustada de índices que combina informações sobre a precisão e o tempo de execução total dos algoritmos de aprendizagem (BRAZDIL; SOARES; COSTA, 2003), usada para ranqueamento de algoritmos de classificação em um conjunto de dados; métrica de desempenho ponderada,

que é uma métrica usada para melhorar a medição em conjuntos de dados desbalanceados (MORAIS; MIRANDA; SILVA, 2016); função de custo usada em problemas de otimização para definir o custo de uma solução.

As medidas de desempenho usadas para cada caso de meta-aprendizagem devem considerar fatores como conjunto de algoritmos utilizados, conjunto de dados, a natureza do sistema de decisão (recomendação simples ou ranqueada), tempo de execução de algoritmo e eficacia.

4.5 Meta-aprendizagem em diversas áreas

Uma abordagem de aprendizagem para o problema de seleção de algoritmo pode ser aplicada a qualquer domínio, desde que haja dados suficientes disponíveis para aprender um mapeamento sobre os metadados (SMITH-MILES, 2009). Na literatura, os domínios de aplicação do meta-aprendizagem aplicado a seleção de algoritmo é amplo: predição de falha de software((DÔRES et al., 2016)), classificação de conjunto de dados ((MORAIS; MIRANDA; SILVA, 2016), (MORAIS; PRATI, 2013), (JR et al., 2012)), otimização ((KANDA et al., 2016)), dentre outros. Nesta seção é apresentada abordagem de aprendizagem para seleção de algoritmos em vários domínios interdisciplinares.

Em classificação, tem sido proposto métodos para selecionar técnicas de amostragem para conjuntos de dados desbalanceado (MORAIS; MIRANDA; SILVA, 2016), i.e., dominada por uma classe majoritária. Em maior detalhe, o método recomenda uma solução para um novo problema com base em soluções armazenadas para problemas anteriores (meta-data). A proposta foi avaliada em 29 bases de classificação, as meta-características usadas para representar o conjunto de dados são do tipo simples e estatística e o desempenho baseia-se na média ponderada.

Em (MORAIS; PRATI, 2013), investiga-se a adoção de medidas utilizadas para avaliar propriedades de redes complexas na caracterização da complexidade de conjuntos de dados em aplicações de aprendizagem de máquina. Essas medidas são obtidas a partir de uma representação baseada em grafo de um conjunto de dados. Para avaliar as medidas, são utilizadas bases de classificação da UCI (DUA; GRAFF, 2017) em uma estrutura de meta-aprendizagem, em uma comparação pareada (*pairwise comparison*).

(AMADINI et al., 2015) apresenta em seu trabalho o impacto das técnicas de seleção de características no desempenho do seletor de algoritmo SUNNY,tomando como referência os benchmarks da biblioteca AS (ASlib)(BISCHL et al., 2016). SUNNY (AMADINI; GABBRIELLI; MAURO, 2014) é um seletor de algoritmos originalmente adaptado para Problemas de Satisfação de Restrição e mais tarde, adaptado para lidar com Problemas de Otimização de Restrição. O experimento foi feito usando os seguintes avaliadores de atributos: *Information Gain*, *Gain Ratio*, *Symmetrical Uncertainty*, *Relief* e *OneR*. Os

resultados indicam que um punhado de recursos é suficiente para alcançar um desempenho similar, senão melhor, da abordagem original SUNNY que usa todos os recursos disponíveis.

Em (MISIR; SEBAG, 2017), é apresentada uma abordagem de filtragem colaborativa para seleção de algoritmos (*Algorithm Selection - AS*). A filtragem colaborativa (*Colaborative Filtering* - CF), popularizada pelo desafio da Netflix, visa recomendar os itens que um usuário provavelmente irá gostar, com base nos itens anteriores que ela gostou e nos itens que gostaram outros usuários. Seleção de algoritmos requer experimentos extensivos e computacionalmente caros para aprender um modelo de desempenho, com todos os algoritmos lançados em todas as instâncias do problema, enquanto que filtragem colaborativa pode explorar uma matriz esparsa, com alguns algoritmos lançados em cada instância de problema . Além disso, AS aprende um modelo de desempenho como uma função da representação inicial da instância, enquanto que CF cria fatores latentes para descrever algoritmos e instâncias e usa as métricas latentes associadas para recomendar algoritmos para uma instância específica do problema. Uma contribuição principal do sistema de recomendador de algoritmo proposto é para lidar com o problema de início a frio - emitir recomendações para uma nova instância de problema - através da modelagem não linear dos fatores latentes com base na representação inicial da instância.

Em (ROSSI et al., 2014), apresenta-se um método baseado em meta-aprendizagem para a seleção de periódica algoritmos em ambientes em mudança de tempo, denominado *MetaStream*. Ele funciona ao mapear as características extraídas do passado e os dados recebidos para o desempenho de modelos de regressão, a fim de escolher entre algoritmos de aprendizagem único ou sua combinação. Os resultados experimentais para dois problemas de regressão real mostraram que o MetaStream é capaz de melhorar o desempenho geral do sistema de aprendizagem em comparação com um método de linha de base e uma abordagem baseada em conjunto.

Em (KANDA et al., 2016), uma abordagem de meta-aprendizagem baseada em algoritmos de classificação de rótulos é aplicada na solução de instâncias do problema do caixeiro viajante (*Traveling Sallesman Problem -TSP*) . Nesse trabalho, quatro conjuntos diferentes de meta-características com base em diferentes medidas das propriedades das instâncias *TSP* são investigadas: a) medidas de borda e vértice; b) medidas de rede complexas; c) propriedades de meta-heurísticas e; d) *landmarkers*. Os modelos são investigados em quatro cenários *TSP* diferentes, apresentando variações de simetria e força de conexão. Os resultados experimentais indicam que os modelos de meta-aprendizagem podem prever com precisão rankings de meta-heurísticas para diferentes cenários *TSP*. As boas soluções podem ser obtidas a partir da predição de *rankings*, independentemente do algoritmo de aprendizagem usado no meta-nível. Os resultados experimentais também mostram que a definição do conjunto de meta-características tem um impacto importante na qualidade das soluções obtidas (KANDA et al., 2016)

4.6 Aprendizagem de máquina

Em meta-aprendizagem, algoritmos candidatos a uma dada instância são representados de duas formas: melhor algoritmo ou *ranking* de algoritmo. A escolha do classificador depende dessa decisão. Nesta seção, são apresentados quatro classificadores que podem ser empregados no meta-nível, destacando uma breve descrição, principais características e eventuais vantagens.

4.6.1 Random Forest

Aprendizagem por Árvore de Decisão (*Decision Tree - DT*) gera um modelo preditivo a partir do conjunto de caminhos decisórios desde a característica armazenada no nó raiz, passando por nós intermediários, até alcançar o rótulo armazenado em nós folhas (MICHIE; SPIEGELHALTER; TAYLOR, 1994).

Florestas aleatórias (*Random Forest - RF*) formam uma combinação de preditores de árvores de decisão, de modo que cada uma delas depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição para todas as árvores na floresta (BREIMAN, 2001). RF têm sido empregadas em meta-aprendizagem para seleção de algoritmos (DÔRES et al., 2016) e é conhecida por seu bom desempenho, bem como interpretabilidade (ROSSI et al., 2014).

Algumas vantagens do uso de Arvore de decisão são: gerar meta-modelos mais inteligíveis, pois mostram as regras usadas pelo meta-modelo para produzir o resultado em sua saída (KANDA et al., 2016); ter grande capacidade de pesquisa por relacionamentos de dados, mesmo quando não são aparentes (PARMEZAN; LEE; WU, 2017).

4.6.2 *Multilayer perceptron - MLP*

MLP constitui-se em uma espécie de rede neural artificial pertencente ao paradigma conexionista, que foi inspirado no estudo das conexões neurais do sistema nervoso central e baseia-se na combinação de unidades simples altamente conectadas. Neste modelo, pode haver uma ou mais camadas de neurônios entre as camadas de entrada de dados e a saída de resultados. Essas camadas intermediárias são unidades que não interagem diretamente com o meio ambiente e funcionam como combinadoras de características. Se houver conexões apropriadas entre as unidades de entrada e um conjunto considerável de unidades intermediárias, sempre pode ser encontrada a representação que produzirá o mapeamento correto entre entrada de dados e saída de resultados (classificação). Um dos algoritmos mais utilizados para treinar esse tipo de rede é a *backpropagation*(HAYKIN et al., 2009).

Algoritmos baseados em *MLP* têm sido muito usados em seleção de algoritmos

para problemas de otimização, sendo capaz também de operar em cenários multi-rótulo. Para usar a *MLP* em técnicas de ranqueamento de rótulos, *m MLP* conseguem formular um *ranking* para cada um dos rótulos, a partir das saídas ordenadas (KANDA et al., 2016).

4.6.3 Máquina de vetores de suporte - SVM

Os SVMs assumem os dados linearmente separáveis no espaço de característica e constrói um hiperplano linear que separa as classes positivas e negativas pela maior margem. A margem é definida como a distância do hiperplano da amostra de classe positiva e negativa mais próxima presente nos dados históricos. Obviamente, o limite de decisão real pode não ser linearmente separável, para o qual o SVM usa o truque do kernel. Os dados são projetados para um espaço de característica de maior dimensão pelo uso de kernels. O truque é que os dados separáveis não linearmente no espaço de característica podem ser linearmente separáveis em um espaço dimensional maior por uma boa escolha de kernel e pela aplicação de múltiplos kernels. Os outliers e os dados ruidosos podem ainda ter um efeito severo e, portanto, um termo é adicionado para tornar o hiperplano de decisão resistente a alguns dados por uma pequena margem(KALA, 2016).

(GIANNAKOPOULOS; PIKRAKIS, 2014) fornece uma explicação dos parâmetros cruciais do SVM:

- Tipo de kernel: De acordo com a metodologia SVM, uma função kernel é usada para mapear os vetores de recursos para o "espaço kernel". Os núcleos típicos são lineares, polinomiais (de ordem 3 ou superior), e núcleo de função de base radial (RBF), etc.
- Propriedades do Kernel: dependendo do kernel selecionado, certos parâmetros do kernel precisam ser definidos para o treinamento e avaliação do classificador SVM. Por exemplo, no caso de um núcleo polinomial, a ordem do polinômio deve ser especificada pelo usuário.
- Parâmetro de restrição, C. Está relacionado com a função de custo do procedimento de treinamento SVM. À medida que o valor de C aumenta, o custo de amostras erradas também aumenta.

4.6.4 *Naives Bayes - NB*

Consiste em um algoritmo de aprendizagem a partir do paradigma estatístico que assume os dados seguindo uma distribuição normal e podem ser aproximados, por um modelo probabilístico, do conceito induzido. NB pode ser entendido como uma rede *bayesiana* especializada, tal como está em dois pressupostos importantes: os recursos são igualmente importantes e também estatisticamente independentes, de modo que, quando a classe é conhecida, conhecer o valor de um recurso não adiciona nenhuma informação

sobre o valor de outro. Em termos práticos, o algoritmo classifica um novo exemplo de acordo com a classe mais provável, dado o conjunto de dados(WITTEN et al., 2016).

Algoritmos baseados em NB têm sido usados satisfatoriamente para seleção de modelo em um cenário de fluxo de dados (ROSSI et al., 2014), bem como em problemas de meta-classificação binária usando meta-características baseadas em grafos (MORAIS; PRATI, 2013). Observe-se ainda que atributos correlacionados tendem a degradar o desempenho do classificador NB (PARMEZAN; LEE; WU, 2017).

4.6.5 K vizinhos mais próximos - *KNN*

KNN é um algoritmo de aprendizagem baseado em instâncias que busca, de acordo com alguma medida de similaridade, os k exemplos já rotulados mais próximos de um exemplo não rotulado. A classificação é, portanto, decidida por meio dos rótulos dos exemplos mais próximos (FIX; JR, 1951).

O uso de *KNN* para fornecer uma recomendação de algoritmos inclui as seguintes etapas. Inicialmente, calcular meta-características para o conjunto de dados em questão. Em segundo lugar, identificar os vizinhos mais próximos entre os conjuntos de dados existentes. Em terceiro lugar, recuperar os rankings de algoritmos associados aos vizinhos mais próximos e usar essa informação para fazer o ranking global de recomendação (SOARES; BRAZDIL; KUBA, 2004).

O classificador *KNN* permite a previsão de uma lista de classificação (DÔRES et al., 2016) com desempenho dependente de muitos fatores. Uma condição suficiente para seu bom desempenho é que as instâncias de problema no espaço inicial guardem a mesma relação de distância no espaço latente(MISIR; SEBAG, 2017), (JR et al., 2012).

5 Meta-aprendizagem aplicado para seleção de meta-heurísticas sintonizadas por método de corrida

Neste capítulo, estende-se o *framework* para seleção de algoritmos (RICE, 1976), agregando-se novos elementos para torná-lo mais genérico e aplicável a vários problemas de otimização combinatória. Os novos elementos estão relacionados com uma nova representação de instâncias de problema (ou tarefas) baseada em grafos, bem como a sintonia de algoritmos de otimização usando métodos de corrida para posterior análise de desempenho e formação das metaclasses.

5.1 Framework

O quadro geral da abordagem de meta-aprendizagem para recomendar algoritmos para instâncias de problemas de otimização ocorre em quatro fases: geração dos metadados, sintonização de algoritmos, agrupamento dos algoritmos sintonizados e construção do meta-modelo.

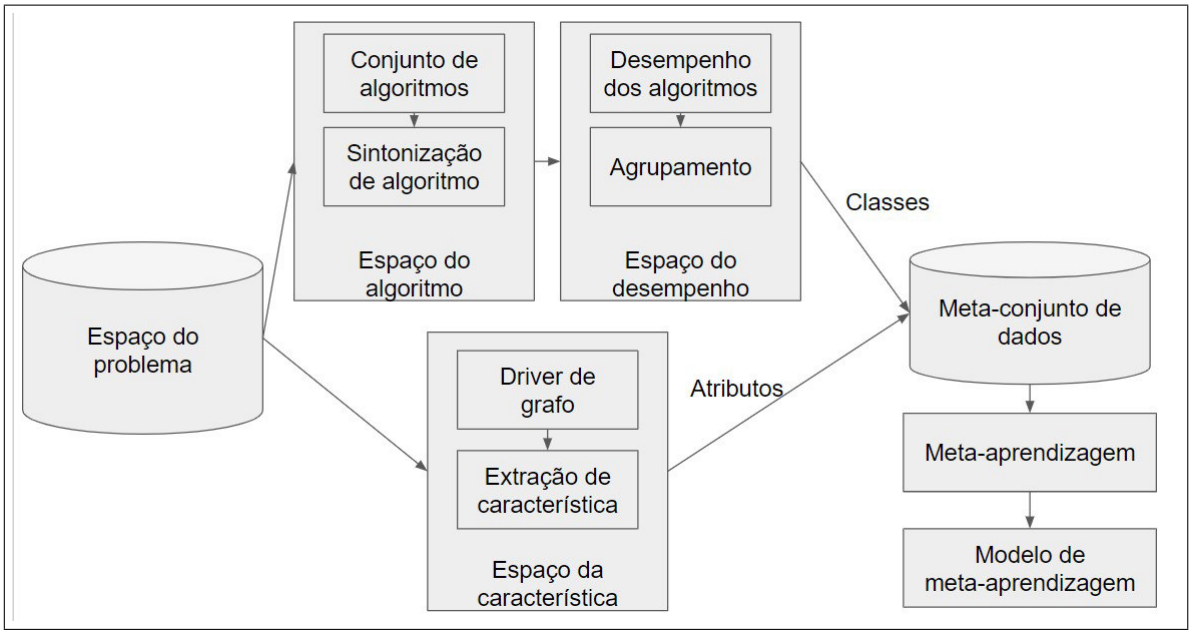
Durante a geração de metadados, as instâncias de problemas são usadas para extrair as meta-características. As meta-características são geradas a partir das propriedades extraídas das representações em grafo de cada instância, o que produz, ao final, um conjunto de *instâncias-meta-atributos*. Na sintonização, o grupo escolhido de algoritmos de otimização têm seus parâmetros ajustados separadamente pelo pacote de sintonia por corrida. Neste trabalho, propõe-se o emprego do *Irace* (*Iterated Racing for Automatic Algorithm Configuration*), devido sua popularidade e capacidade de encontrar boas configurações rapidamente.

O *Irace* produz pelo menos uma configuração de algoritmo por instância o que pode levar a um bom número de configurações semelhantes e as vezes pouco representativas. Por esse motivo, o *framework* prevê o uso de agrupadores para encontrar configurações mais representativas, ou seja, configurações que operam em domínios diferentes, e assim reduzir futuramente o número de metaclasses. As metaclasses são associadas ao melhor algoritmo otimizador para uma dada instância, considerando a melhor sintonia encontrada pelo *Irace*. Pode-se dizer que uma dada metaclasse é formada pelo par $\{algoritmo, parâmetro\}$ que obteve os melhores resultados para uma dada instância do problema de otimização, considerando um certo número de execuções do algoritmo.

Durante a construção do metamodelo, primeiramente, o conjunto de instâncias de

classificação (meta-instâncias), formado por meta-características e meta-rótulos é gerado usando as características e as classes obtidas nas etapas anteriores. Então, aplica-se o treinamento de um dado classificador sobre o conjunto de dados para se obter um meta-modelo de classificação. Espera-se que o meta-modelo seja robusto o suficiente para classificar posteriormente meta-instâncias deixadas de fora do treinamento.

Figura 9 – *Framework* proposto.



Fonte – Acervo do autor.

5.2 Espaço das meta-caracteristicas

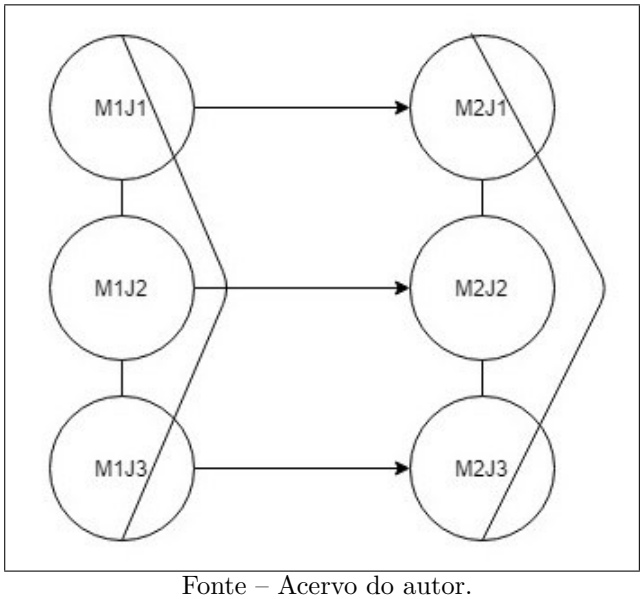
5.2.1 Representação baseada em grafo

Para este trabalho, são utilizadas métricas para extração de meta-características a partir de grafos que representam cada instância de problema de otimização. Espera-se que tais métricas possam capturar informações estruturais do problema. A representação baseada em grafo pode codificar relações locais de vizinhança, bem como as características globais associadas a espacialidade e densidade dos elementos estruturais da instância de problema (MORAIS; PRATI, 2013). Porém, alguns atributos extraídos de grafos podem ser custosos de calcular, como atributos relacionados a distancia entre dois vértices, que tornam o calculo desses atributos custoso a medida que aumenta o número de nós do grafo. Outro problema é que alguns atributos são específicos para determinado tipo de grafo, como atributos relacionados a triângulos em grafos (por exemplo: transitividade e coeficiente de agrupamento) que só podem ser calculados se os vértices dos grafos formarem ciclos.

As representações em grafos para abordagens de meta-aprendizagem em problemas de otimização já foram utilizadas anteriormente nos trabalhos de (MIRANDA et al., 2018), que utilizou para o problema *MaxSat*, e (KANDA et al., 2016), que empregou no problema do caixeiro viajante.

Para o problema *flow shop* Permutacional (*Permutation Flowshop Problem - PFSP*) foram consideradas duas representações baseadas em grafos. Na primeira representação, o chamado grafo de fluxo de trabalho é caracterizado por ser dirigido, em que cada nó do grafo representa uma tupla máquina-tarefa e contém, pelo menos, três arestas: a) aresta para o nó que representa a próxima tarefa na máquina; b) aresta para o nó que representa a tarefa anterior na máquina; e c) aresta para a próxima máquina que executa a tarefa atual, se houver próxima. Deste modo, um fluxo com 3 tarefas e 2 máquina pode ser representado como na Figura 10. Os pesos em cada aresta que sai de um nó correspondem ao tempo de execução da tarefa na máquina.

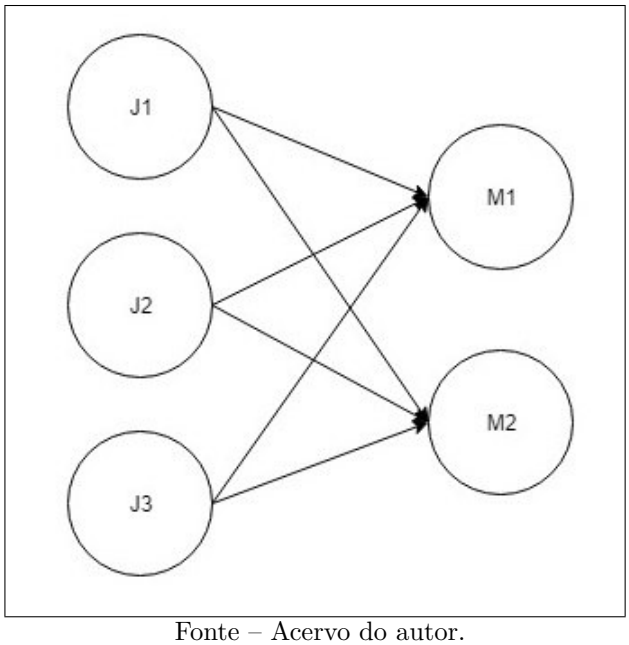
Figura 10 – Grafo de fluxo de trabalho.



Fonte – Acervo do autor.

Na segunda representação, o grafo bipartido, associando-se tarefa-máquina, é direcionado de forma que os nós são divididos em dois grupos: a) grupo formado por tarefas; e b) grupo formado por máquinas. As arestas do grafo ocorrem entre dois nós, relacionando uma tarefa a uma máquina. Cada nó-tarefa tem ligação com todos os nós-maquinas e os pesos das aresta representam o tempo de execução da tarefa em uma dada máquina. Deste modo, um fluxo com 3 tarefas e 2 máquina pode ser representado como na Figura 11.

Figura 11 – Grafo de tarefas-máquinas.



Fonte – Acervo do autor.

5.2.2 Caracterização das instâncias

Para a caracterização do problema é necessário extrair medidas estruturais a partir das instâncias do problema. Como o problema abordado é representado em grafo é importante salientar que nenhuma destas medidas está diretamente ligada a estrutura específica de uma instância *PFSP*. De fato, tal associação aparece indiretamente por meio de características do grafo que representa a instância. Observa-se, assim, a capacidade de generalização da representação usando grafo, todavia prevendo-se que a capacidade preditiva de cada métrica tende a variar entre problemas diferentes (nem tanto entre instâncias diferentes do mesmo problema).

Segue-se a descrição das medidas utilizadas:

- número de Nós (1)
- número de Arestas(2)
- densidade: a razão entre o número de arestas e de nós. dado pela formula $d = \frac{m}{n(n-1)}$
- Centralidade de grau: Centralidade de grau é definida como o número de ligações incidentes de um vértice. O grau pode ser interpretado como a probabilidade que o vértice tem de receber alguma informação da rede. A centralidade de graus para um nó v é a fração de nós aos quais ele está conectado.
- conectividade: média da vizinhança(conectividade): grau médio de vizinhos mais próximos

- grau-Número de arestas conectadas a um nó.
- peso arestas: soma dos pesos das arestas que chegam a um nó;

A centralidade de grau, conectividade, grau e peso arestas são calculados para cada nó e retiradas as seguintes medidas estatísticas como características globais do grafo: média, desvio padrão, coeficiente de variação, mínimo, máximo e entropia (NUDELMAN et al., 2004). Tendo ao final um total de $4x6 = 24$ atributos além dos outros 3, o que totaliza 27 atributos por grafo. Além dessas, outras características chegaram a ser utilizadas para o problema *PFSP*, como coeficiente de agrupamento e excentricidade do nó, porém foram descartadas devido ao tempo levado para calculá-las.

5.3 Sintonização de algoritmos

A sintonização dos algoritmos ocorre no espaço de algoritmo usando-se o pacote *Irace*. É necessário inicialmente identificar os parâmetros de desempenho do algoritmo mais efetivos para serem ajustados, bem como suas respectivas faixas de valores e restrições dos mesmos. Também são definidas as instancias que farão parte da sintonização, além de critérios de parada do sintonizador e demais parâmetros do *Irace*.

Ao final, como resultado, é obtido um conjunto de configurações para cada algoritmo. Essas tuplas de algoritmos-parâmetros são então passadas para o espaço de desempenho.

5.4 Espaço de desempenho

Nessa etapa, as tuplas algoritmos-parâmetros são validadas em um conjunto de instâncias e então agrupadas usando o *algoritmo de propagação de afinidade* (FREY; DUECK, 2007) a fim de obter-se centros de agrupamento que vão compor futuramente as metaclasses usadas na etapa de meta-aprendizagem. Esse algoritmo é utilizado por conseguir definir grupos a partir da matriz de similaridade e escolher exemplares como centro de grupo.

A validação das tuplas algoritmos-parâmetros é feita executando-se cada tupla n vezes para cada instância do conjunto, obtendo-se uma média de desempenho para cada *instancia-algoritmo-configuração*. Em seguida, as tuplas algoritmos-parâmetros são passadas como entrada para o algoritmo de agrupamento de propagação de afinidade, para o qual os desempenhos das instancias, em um algoritmo-configuração, são os atributos.

O método de agrupamento por propagação de afinidade (FREY; DUECK, 2007) recebe, como entrada, medidas de similaridade entre pares dos pontos de dados. Então, mensagens de valor real são trocadas entre pontos de dados até que um conjunto de

exemplares de alta qualidade e *clusters* correspondentes emerge gradualmente. Existem dois tipos de mensagens trocadas entre pontos de dados, e cada um leva em conta um tipo diferente de competição. As mensagens podem ser combinadas em qualquer estágio para decidir quais pontos são exemplares e, para todos os outros pontos, a que exemplar pertence.

Mensagens de responsabilidade são enviadas de pontos de dados para exemplares-candidatos e indica com que intensidade de cada ponto de dados favorece o exemplar-candidato em relação a outros exemplares de candidatos. Mensagens de disponibilidades são enviadas de exemplares-candidatos para pontos de dados e indicam em quem medida cada exemplar candidato está disponível como um centro de *cluster* para o ponto de dados. As principais características desse método são a determinação do números de *clusters* pela matriz de similaridade e a identificação de exemplares representativos dos *clusters*.

5.5 Meta-aprendizagem

Nesta etapa, o conjunto de dados é construído usando-se um conjunto de instâncias e as meta-classes obtidas na etapa anterior. A partir desse conjunto, produz-se o meta-modelo de aprendizagem. Cada uma destas tuplas de $\{algoritmos, parâmetros\}$ é considerada uma classe para o problema de aprendizagem. Desta forma, a geração de exemplos se dá por meio da execução de algoritmos diversos em uma ampla gama de instâncias.

A caracterização das instâncias do problema se dá por meio da extração das características apresentadas anteriormente. Para definição das classes, o conjunto de instancias é validado no conjunto de configurações-classes e então é determinada a classe que melhor resolve uma dada instância, baseado no melhor desempenho médio das classes na instancia.

Com o conjunto de dados já definido, a etapa de meta-aprendizagem inicia-se e esse conjunto é colocado com entrada para o algoritmo de aprendizagem de maquina supervisionado para determinação do meta-modelo capaz de prever o algoritmo de otimização que melhor solucione instâncias não incluídas no treinamento.

6 Resultados Computacionais

Neste capítulo, descreve-se o ambiente em que os experimentos foram realizados para validação do arcabouço proposto. Considera-se para esse fim, a configuração do experimento, o conjunto de instâncias do problema de otimização, bem como as ferramentas computacionais utilizadas. Ao final, os resultados obtidos são discutidos.

6.1 Configuração do Experimento

Para validar a proposta de meta-aprendizagem foi considerado o *dataset*, proposto por (VALLADA; RUIZ; FRAMINAN, 2015). Esse *benchmark* para o PFSP tem como objetivo a minimização do *makespan*(tempo total da programação) e possui 480 instâncias PFSP, separadas em *Small* (240) e *Large*(240). Esse conjunto de *benchmark* foi criado com a intenção de fornecer instâncias de *benchmark* mais difíceis do que as do conjunto de *benchmarks* da Taillard (TAILLARD, 1993).

As instâncias do grupo *Small* foram descartadas devido ao desempenhos dos algoritmos nessas instâncias serem muito semelhantes, o que impossibilitaria a seleção de um algoritmo uma vez que qualquer um escolhido apresentaria um resultado muito próximo ao outro. Deste modo, apenas o grupo de instância *Large* foi utilizado na experimentação. As 240 instancias são combinações de parâmetros n $\in \{100, 200, 300, 400, 500, 600, 700, 800\}$ e m $\in \{20, 40, 60\}$, disponibilizando-se 10 instâncias para cada combinação.

Em relação aos algoritmos, os parâmetros de desempenho considerados são apresentados na Tabela 4. A escolha dos algoritmos foi determinada pela forma em que cada algoritmo trabalha o espaço de solução. Enquanto o algoritmo Multi-start Simulated Anneling adota de escalada multi-partida no recozimento simulado, usando operações simples como inserção e troca para explorar a vizinhança de um solução; O Colônia de Abelha Discreta com Mutação Composta (CDABC), utiliza 6 operações para explorar a vizinhança de uma solução, o que torna o processo mais pesado e a exploração mais esparsa; Já o Biased Random-Key Evolutionary Clustering Search(BRKeCS) emprega uma busca 2-opt local para uma exploração mais intensa do espaço de solução.

As faixas de valores são usadas pelo pacote *Irace* para ajustar os parâmetros dos algoritmos de otimização. Quanto a execução dos algoritmos de otimização, optou-se por definir como critério de parada em $400.000,00$ chamadas a função objetivo, para cada algoritmo de otimização, tanto na sintonização quanto na construção do conjunto de dados da aprendizagem . Isto foi feito para ajustar o tempo total do experimento.

Para a configuração do cenário do *irace* foi utilizado como parâmetros

MH	Parâmetros	Tipo	Domínio
MSA	$I_{iterações}$	inteiro	(1 a 5)
	T_0(temperatura inicial)	real	(50 a 1500)
	T_F(temperatura final)	real	(0,01 a 2)
	α (coeficiente de controle de arrefecimento)	real	(0,05 a 1)
	$P_{tamanho}$	inteiro	(1 a 10)
DABC	sn (fonte de alimentos)	inteiro	(1 a 30)
	$observadoras$	inteiro	(1 a 30)
	$limite$	inteiro	(1 a 30)
	taxa de mutação	real	(0,05 a 0,5)
	fator busca local	inteiro	(1 a 30)
BRKeCS	$Altura$	inteiro	(1 a 5)
	$Largura$	inteiro	(1 a 5)
	p_e (população elite)	real	(0,05 a 0,5)
	p_m (população mutante)	real	(0,05 a 0,5)
	r_{max} (repetição da busca local no cluster)	inteiro	(1 a 10)
	λ % da população no cluster promissor	real	(0 a 1)
	p	inteiro	(1 a 1000)
	$numCl$ (número de clusters)	inteiro	(1 a 20)

Tabela 4 – Meta-heurísticas e o total de instâncias em que foram melhores.

$maxExperiments = 3000$, que define o orçamento computacional da execução do *irace*, isto é, o número de experimentos realizados antes do fim da execução do *irace*. Esse valor foi escolhido porque o aumento do número de experimentos acima de 3000 não apresentou ganho expressivo no desempenho. Os outros parâmetros foram mantidos padrão.

6.2 Sintonização de meta-heurística

Na etapa de sintonização de meta-heurística, o *irace* é empregado individualmente a cada algoritmo de otimização tendo como saída um conjunto das configurações melhores ranqueadas pelo método de corrida. Na tabela 5, mostram-se as configurações obtidas nessa etapa. Na etapa seguinte, o conjunto de configuração é passado como entrada para o algoritmo de Agrupamento por Afinidade (AA) que tem como saída os centros de cada grupo, que é representado na Tabela 5 em negrito e serão utilizadas como metaclasses na etapa de meta-aprendizagem.

6.3 Avaliação da proposta de meta-aprendizagem

Para gerar conjunto de dados da aprendizagem, foram executadas todas as meta-heurísticas(meta-classes) em todas as instâncias disponíveis do conjunto, repetidas 30 vezes em cada instância-classes para obter a média de desempenho. Considerando-se a solução obtida por cada meta-heurística em cada instância PFSP, foi atribuída uma metaclasse

Meta-heurísticas	grupo
MSA $I = 4$, $TF = 1302,36$, $T0 = 0,2$, $\alpha = 0,13$, $p = 2$	1
MSA $I = 3$, $TF = 1233,63$, $T0 = 0,13$, $\alpha = 0,11$, $p = 1$	1
MSA $I = 3$, $TF = 1281,05$, $T0 = 0,09$, $\alpha = 0,14$, $p = 2$	**1**
MSA $I = 4$, $TF = 1304,39$, $T0 = 0,10$, $\alpha = 0,13$, $p = 2$	1
MSA $I = 2$, $TF = 1276,26$, $T0 = 0,15$, $\alpha = 0,069$, $p = 3$	1
CDABC $SN = 26$, $OB = 10$, $lim = 2$, $mut = 0,29$, $bl = 25$	2
CDABC $SN = 29$, $OB = 13$, $lim = 2$, $mut = 0,30$, $bl = 25$	2
CDABC $SN = 28$, $OB = 10$, $lim = 14$, $mut = 0,27$, $bl = 23$	**2**
CDABC $SN = 22$, $OB = 14$, $lim = 15$, $mut = 0,22$, $bl = 22$	2
CDABC $SN = 26$, $OB = 12$, $lim = 16$, $mut = 0,25$, $bl = 22$	2
BRKeCS $A = 2$, $L = 1$, $p_e = 0,40$, $p_m = 0,24$, $r_{max} = 3$, $\lambda = 0,28$, $p = 57$, $numcl = 1$	3
BRKeCS $A = 2, L = 2, p_e = 0,41, p_m = 0,27, r_{max} = 2, \lambda = 0,36, p = 141, numcl = 1$	**3**
BRKeCS $A = 2$,, $L = 2$, $p_e = 0,41$, $p_m = 0,27$, $r_{max} = 1$, $\lambda = 0,24$, $p = 165$, $numcl = 1$	3
BRKeCS $A = 3$, $L = 1$, $p_e = 0,39$, $p_m = 0,27$, $r_{max} = 2$, $\lambda = 0,43$, $p = 237$, $numcl = 2$	3
BRKeCS $A = 2$, $L = 2$, $p_e = 0,41$, $p_m = 0,23$, $r_{max} = 2$, $\lambda = 0,42$, $p = 98$, $numcl = 1$	3

Tabela 5 – Configurações sintonizadas para o problema flow shop. A coluna grupo indica a qual grupo a configuração pertence e as configurações em negritos são os respectivos centros de grupo.

a cada meta-instância (problema de classificação). A metaclasse de cada meta-instância foi definida como sendo a meta-heurística otimizada que obteve o melhor resultado. Na Tabela 6 é apresentada a distribuição de instâncias por classes.

Meta-heurísticas	**Total**
MSA $I = 3$, $TF = 1281,05$, $T0 = 0,09$, $\alpha = 0,14$, $p = 2$	86
CDABC $SN = 28$, $OB = 10$, $lim = 14$, $mut = 0,27$, $bl = 23$	122
BRKeCS $A = 2$, $L = 2$, $p_e = 0,41$, $p_m = 0,27$, $r_{max} = 2$, $\lambda = 0,36$, $p = 141$, $numcl = 1$	32

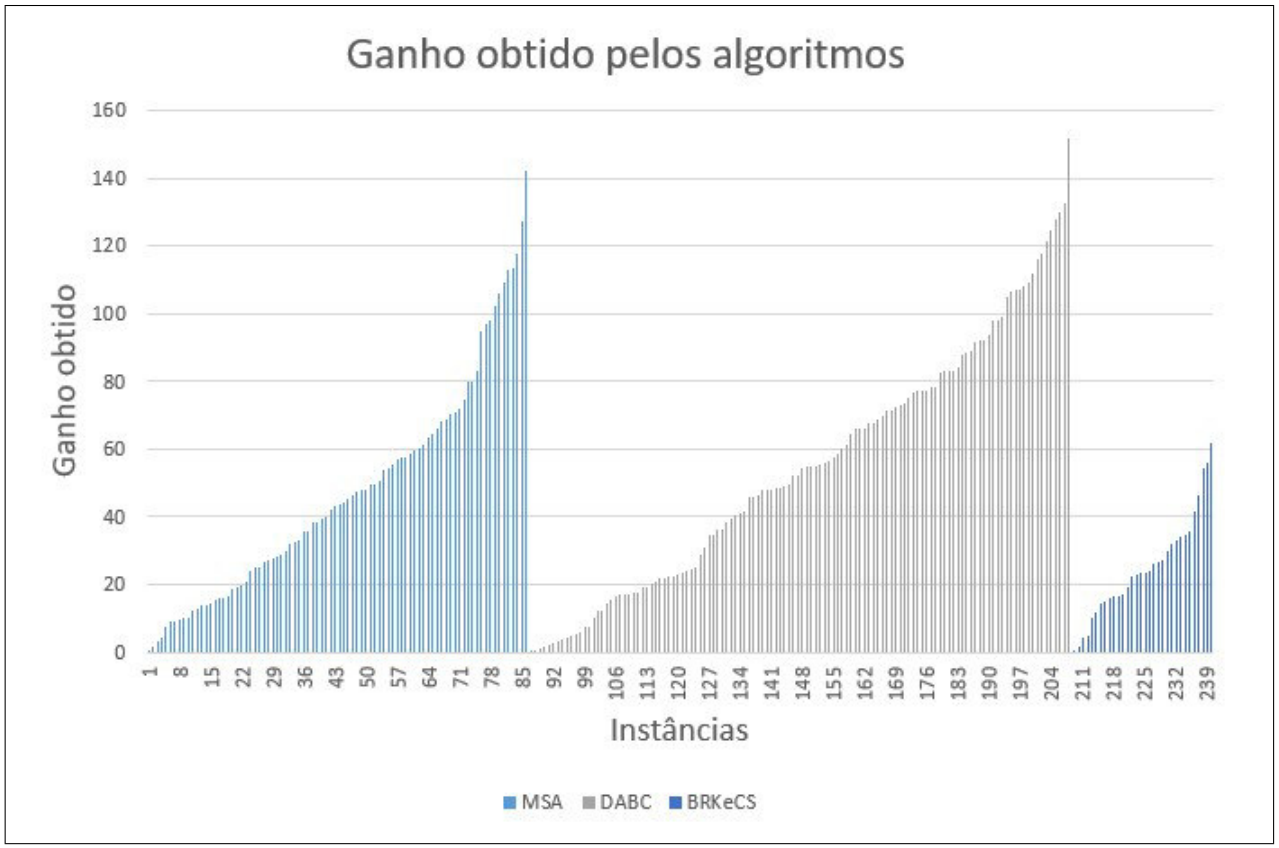
Tabela 6 – Meta-heurísticas e o total de instâncias em que foram melhores.

6.3.1 Importância da escolha

Para que o problema de seleção de algoritmos aplicado ao *flow shop* seja considerado um problema de classificação é preciso que seja possível a construção de um modelo de decisão para esse objetivo. Deste modo, é necessário que a escolha de uma MH para uma instância seja relevante, isto é, se o algoritmo A superar o algoritmo B em algumas funções de custo, então deve existir muitas outras funções em que B supera a A.

Para verificar a importância da escolha, os desempenhos dos algoritmos na instâncias são comparados a fim de apurar as diferenças entre eles (Figura 12). Isso é feito calculando o ganho obtido por cada instância ao escolher a melhor meta-heurística. O ganho obtido é calculado pela diferença de desempenho entre a segunda melhor escolha (segunda melhor meta-heurística para a instância) e a melhor escolha. Os ganhos são agrupados por cada meta-heurística e ordenados de forma crescente.

Figura 12 – Desempenho das meta-heurísticas nas instâncias.



Fonte – Acervo do autor.

No gráfico, observa-se que pode haver um ganho na escolha das meta-heurísticas, visto que a maior parte das instâncias obtiveram um ganho expressivo em relação a segunda escolha. O próximo passo é fazer o modelo de meta-aprendizagem afim de confirmar se os atributos escolhidos ajudam na classificação do problema.

6.3.2 Seleção de meta-heurísticas

Para obtenção do metamodelo de aprendizagem, foram considerados três classificadores: *Multi-layer Perceptron - MLP*, *Random Forest - RF* e *Support Vector Machine - SVM*. Eles foram executados com os parâmetros padrão dos pacotes R. A Tabela 7 mostra os parâmetros considerados para os classificadores e os pacotes R que os implementam.

Algoritmo	**Parâmetro(s)**	**Pacote**
SVM	$\sigma = 1/\#Atributos$, $C = 1$	e1071
RF	$ntree = 500$, $mtryFactor = 1$	randomForest
MLP	$maxit = 100$, $learnFunc =$ "*Std_Backpropagation*"	RSNNS

Tabela 7 – Parâmetros dos classificadores e pacotes R utilizados

O classificador *Zero Rule*, que seleciona apenas a moda, ou a meta-heurística

majoritária, foi introduzido na Tabela 8 para base de comparação. Isto é feito para determinar se a meta-aprendizagem apresenta algum ganho em relação a usar um algoritmo sintonizado para o problema.

Classificador	AUC	Acurácia	Medida F	Precisão	Sensibilidade
SVM	0.959	0.90	0.90	0.907	0.900
RF	0.955	0.896	0.896	0.897	0.896
MLP	0.962	0.892	0.892	0.893	0.892
Classe Majoritária	0.500	0.508	0.343	0.258	0.508

Tabela 8 – Desempenho dos classificadores para seleção de meta-heurísticas

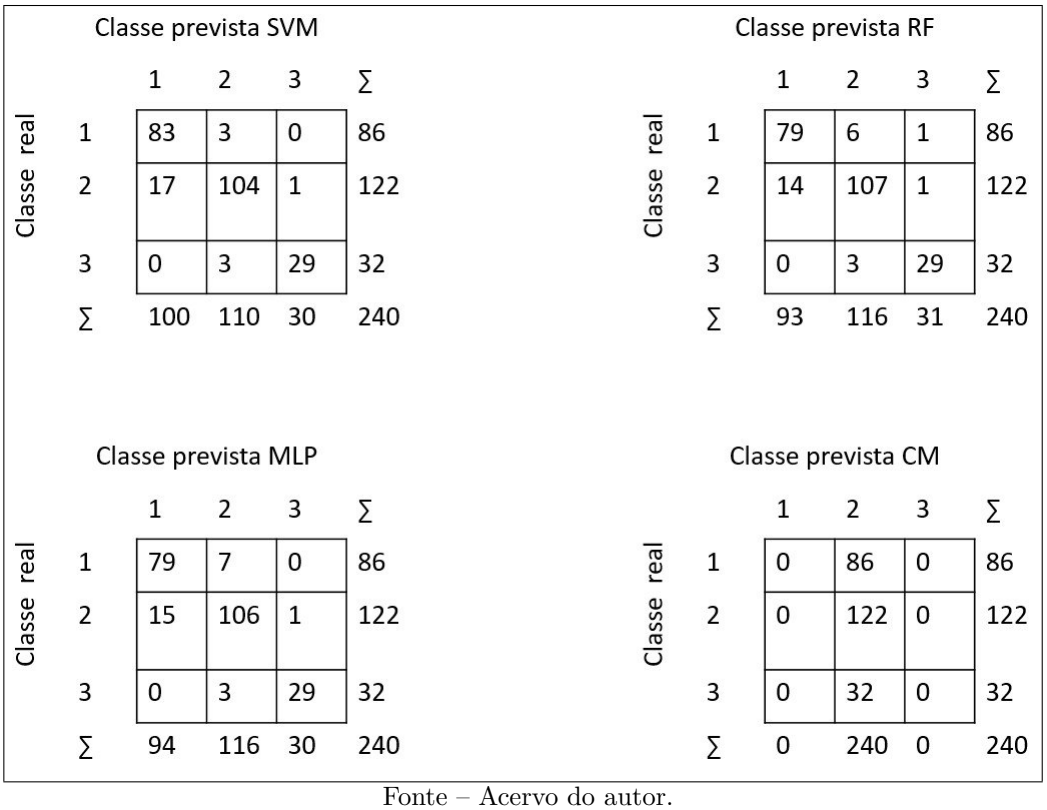
A classificação foi realizada usando validação cruzada estratificada (*Stratified Cross-Validation*)(KRSTAJIC et al., 2014). Nessa validação a variável de saída é primeiro estratificada e o conjunto de dados é dividido aleatoriamente em k-partições, certificando-se de que cada partição contém aproximadamente a mesma proporção de diferentes estratos. Para a classificação foi escolhido 10 partições, as partições são as mesmos para todos os classificadores.

Para avaliação foram usadas medidas para classificação multi classe baseada numa generalização das medidas de classificação binaria (SOKOLOVA; LAPALME, 2009). Essa métrica considera a média sobre as medidas binarias para cada classe. Observa-se que o classificador *SVM* obteve a maior acurácia, e que a medida F acompanhou o valor da Precisão e Revocação, garantindo que a acurácia obtida é confiável. Também nota-se que a métrica *AUC*, que mede a qualidade das previsões do modelo, está próxima de 1, o que indica que o modelo tem bom desempenho preditivo.

Além disso, pode ser verificado que todos os classificadores obtiveram resultados bastante superiores na seleção de meta-heurísticas quando comparados à utilização de apenas uma meta-heurística mais ajustada ao problema. O que indica que o problema é classificável para meta-heurísticas. Também indica que a utilização de meta-aprendizagem para resolução de problemas considerando um conjunto de meta-heurísticas configuradas de forma diferente pode ser mais eficiente do que a sintonização de uma única meta-heurística. Na Figura 13 é apresentado a matriz de confusão para cada classificador. Nela nota-se que a maior fonte de erro é a classe 2, que representa a MH CDABC, o que pode ser explicado por essa MH ter a menor média entre as classes de MH e então está sempre próxima dos melhores resultados.

Após a classificação realizou-se um teste de Friedman (DEMŠAR, 2006) para verificar diferenças significantes de desempenho. Foi considerado, para o teste, a "Classe Verdadeira"(as classes atribuídas as instâncias), a "Classe Majoritária"e a predição do *SVM*. O teste de Friedman foi usado com 95% de confiança sob a hipótese nula de que o desempenho é o mesmo nas três escolhas. Com a hipótese rejeitada, é feito um teste *post-hoc* usando o método de Dunn(DUNN, 1973) com 90% de confiança fazendo uma

Figura 13 – Matriz de confusão dos classificadores



Fonte – Acervo do autor.

comparação pareada para determinar o que levou a rejeição da hipótese. O resultado do teste mostrou que o *SVM* e a "Classe Verdadeira"são superiores ao classificador Majoritário e que não há diferença estatística entre o *SVM* e a "Classe Verdadeira".

6.3.3 Ganho da Meta-aprendizagem

Como foi observado, há uma indicação de que o PFSP é classificável também para meta-heurísticas. Todos os classificadores obtiveram resultados superiores na seleção de meta-heurística quando comparados à utilização de apenas uma meta-heurística mais ajustada ao problema. Porem isso não quantifica o desempenho obtido no nível base. Então, para determinar o ganho obtido com o uso de meta-aprendizagem foi realizada a comparação de desempenho entre o *ZL* e a predição do SVM para cada instância. Os resultados são apresentados na Tabela 14. O gráfico possui os eixos de ganho por instância e instâncias do problema. As instâncias estão ordenadas em função do ganho e o ganho é calculado como a diferença entre a escolha do algoritmo majoritário (*Zero Rule*) e a predição do *SVM*. Obteve-se em média ganho de $18,20$. Sendo que O *SVM* foi melhor em 112 instâncias, empatou em 110 e perdeu em 18. Isso mostra quantitativamente o ganho obtido pela meta-aprendizagem.

Outro gráfico de interesse é o gráfico de perda em relação a "Classe Verdadeira",

Figura 14 – Ganho obtido pela classificação svm em relação ao Zero Rule. As instâncias estão ordenadas em função do ganho obtido

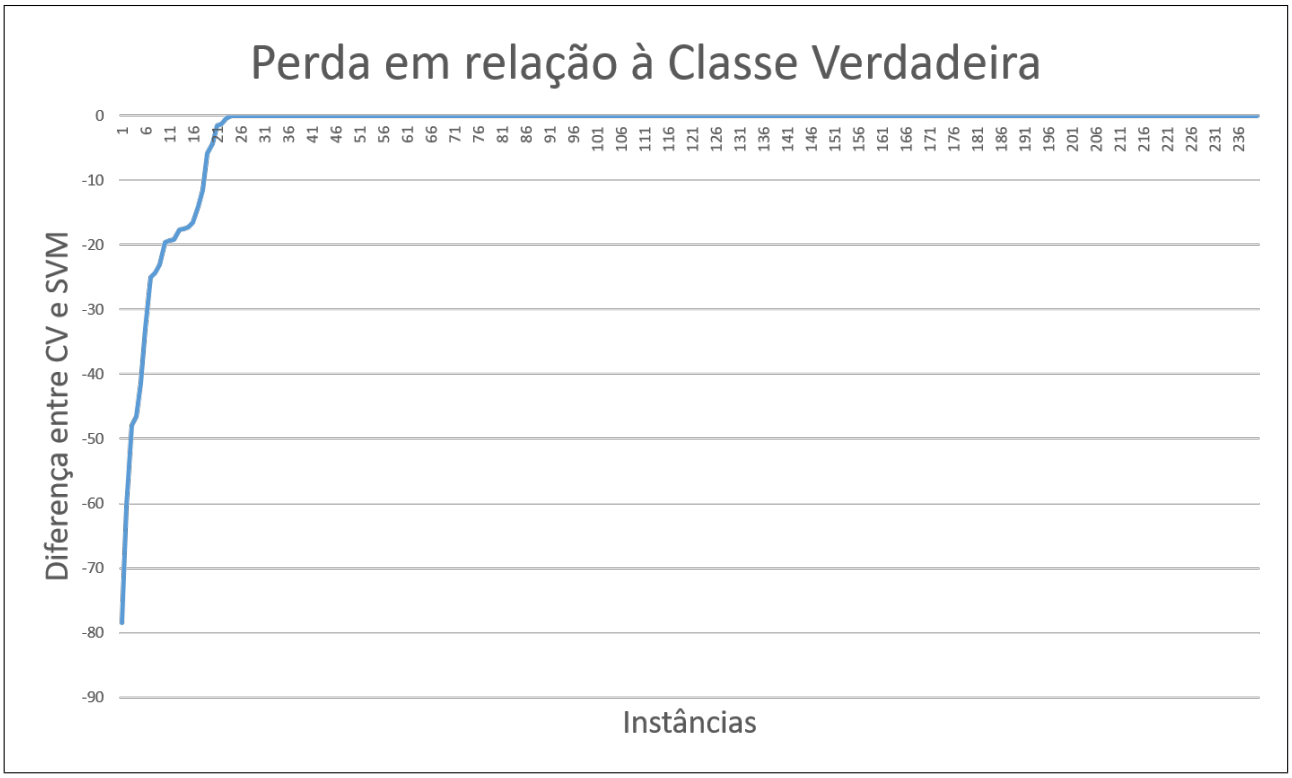
Fonte – Acervo do autor.

apresentado na Tabela 15. Nesse gráfico observa-se que a predição teve perda em 24 instancias que não foram classificadas na classe correta, o que mostra que ainda a espaço para melhorar a predição, seja ajustando os parâmetros dos classificadores seja usando seleção de atributos.

Analisando as instâncias que foram classificadas erradas, os erros podem ser explicados devido à natureza estocástica dos algoritmos, assim uma instância de um grupo que é majoritariamente de uma classe pode ser classificada em outra classe devido a aleatoriedade dos algoritmos. Como por exemplo o grupo 100x40 que é majoritariamente classificado na classe 3 mas teve 1 instancia, de 10, classificada na classe 2.

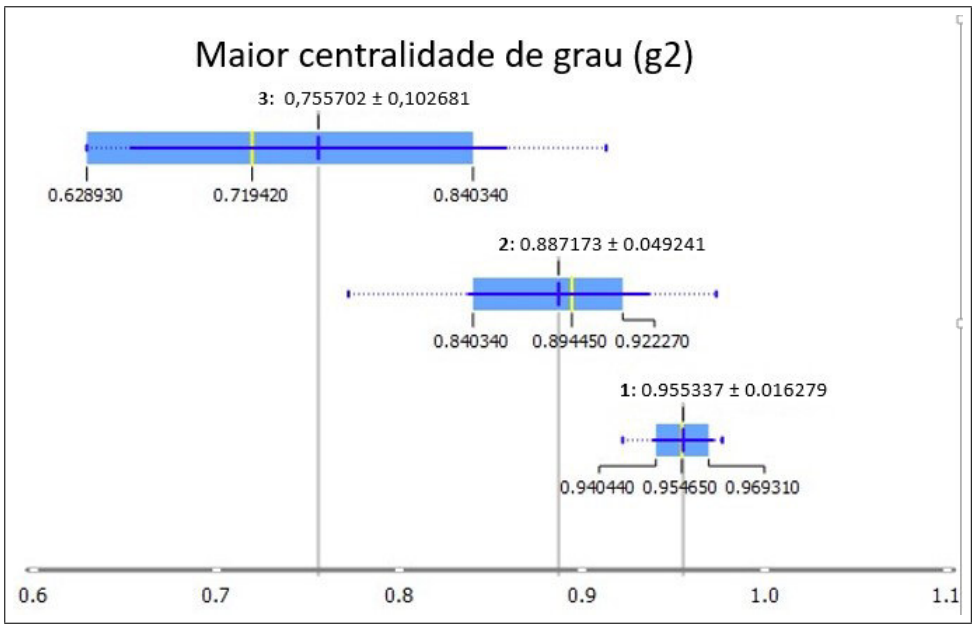
Outros grupos, especificamente 600x40 e 700x60, também são grandes focos de erros de classificação. Nesses casos, nenhum classe teve um domínio majoritário, as classes 1 e 2 obtiveram 5 instâncias em cada grupo. Analisando os valores da meta-característica "Maior Grau de centralidade"verifica-se que essas instâncias têm valores nas extremidades do box-plot apresentado na Figura 16, o grupo 600x40 tem 0,93897 e o grupo 700x60 tem 0,92227, o que coloca essas instancias no meio das classes 1 e 2 e explica a classificação em uma classe ou outra. Deste modo, para melhorar o resultado preditivo nesses casos talvez seja necessário definir uma nova representação em grafo que não considere a estrutura de grupos e seja mais voltada a características internas das instâncias.

Figura 15 – Perda em relação a Classe Verdadeira.



Fonte – Acervo do autor.

Figura 16 – Meta-características "Maior centralidade de grau do grafo 2"



Fonte – Acervo do autor.

7 Conclusão

Meta-heurísticas são estratégias de busca de alto nível que orientam a busca para regiões mais promissoras do espaço da solução e tentam escapar das soluções ótimas locais. Porém, devido a sua característica generalista, uma meta-heurística precisa ter seus parâmetros de desempenho corretamente ajustados para obtenção de resultados competitivos frente à heterogeneidade das instâncias dos problemas de otimização a que se propõe resolver. Além disso, mesmo bem sintonizados, esses algoritmos de otimização não garantem supremacia absoluta em todas as instâncias de um dado problema. Por isso, a seleção de algoritmo baseada em meta-aprendizagem tem sido investigada como estratégia de mais alto nível capaz de decidir a melhor meta-heurística para cada conjunto de instâncias, tomando por base características observáveis e muitas vezes apenas intrínsecas que forma a estrutura de cada instância.

Neste trabalho, foi proposto um *framework* de meta-aprendizagem para algoritmos de otimização sintonizados por método de corrida, buscando-se decidir o algoritmo mais apropriado para cada instância de problema. O *framework* prevê o uso de meta-características extraídas a partir de representação do problema baseada em grafos. Além disso, as melhores configurações de parâmetros encontradas pelos métodos de corrida são tratadas para garantir metaclasses mais representativas e robustas.

A abordagem foi validada experimentalmente a partir de instâncias de grande porte consideradas desafiadoras do *Permutation Flowshop Problem - PFSP*, um problema de otimização combinatória para o qual meta-heurísticas têm obtido os melhores resultados encontrados na literatura para esse tipo de instância.

Os resultados obtidos corroboram com a hipótese que meta-aprendizagem fornece elementos suficientes para o desenvolvimento de super *solvers* capazes de identificar características estruturais e classificar instâncias de acordo com essas características, podendo então selecionar melhor algoritmo de otimização para o problema. Outrossim, o uso de grafos para representar instâncias colabora com a generalidade da abordagem.

Destacam-se os seguintes resultados obtidos durante os experimentos. O *PFSP* é classificável e que as meta-características utilizadas conseguem descrever bem o problema e têm razoável desempenho *preditível*. Também tem-se verificado que o uso de meta-aprendizagem traz um ganho de desempenho se comparado a utilizar apenas uma meta-heurística sintonizada.

Destaca-se ainda, como contribuição da proposta, a extração de meta-características de instâncias que compõem um importante *benchmarking* do *PFSP*, a partir de grafos comumente encontrados na literatura, bem como a definição de metadados

para a partir do agrupamento das configurações mais representativas encontradas no processo de sintonização de meta-heurísticas.

Sugere-se, em nível de trabalhos futuros, investir mais esforços na extensão do arcabouço para outros problemas permutacionais, tais como variações do problema do caixeiro viajante, sequenciamento de padrões, alocação de navios a berços, dentre outros. Pretende-se assim compor um razoável *dataset* disponível para aplicações de meta-aprendizagem. O *framework* também requer mais investigação sobre a capacidade preditiva dos atributos mais populares extraídos a partir de grafos e a relação destes com os elementos estruturantes de cada tipo de problema. A partir de tais pesquisas, vai ser possível avançar em termos de meta-aprendizagem, além da seleção de algoritmos, para a geração de algoritmos baseados nas características das instâncias.

Referências

AMADINI, R.; BISELLI, F.; GABBRIELLI, M.; LIU, T.; MAURO, J. Feature selection for sunny: A study on the algorithm selection library. In: IEEE. *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*. [S.l.], 2015. p. 25–32. Citado na página 45.

AMADINI, R.; GABBRIELLI, M.; MAURO, J. Sunny: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, Cambridge University Press, v. 14, n. 4-5, p. 509–524, 2014. Citado na página 45.

BALAPRAKASH, P.; BIRATTARI, M.; STÜTZLE, T. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In: SPRINGER. *International workshop on hybrid metaheuristics*. [S.l.], 2007. p. 108–122. Citado na página 36.

BARR, R. S.; GOLDEN, B. L.; KELLY, J. P.; RESENDE, M. G.; STEWART, W. R. Designing and reporting on computational experiments with heuristic methods. *Journal of heuristics*, Springer, v. 1, n. 1, p. 9–32, 1995. Citado 2 vezes nas páginas 16 e 34.

BASU, M.; HO, T. K. *Data complexity in pattern recognition*. [S.l.]: Springer Science & Business Media, 2006. Citado na página 43.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, INFORMS, v. 6, n. 2, p. 154–160, 1994. Citado na página 31.

BHATT, N.; THAKKAR, A.; GANATRA, A. A survey and current research challenges in meta learning approaches based on dataset characteristics. *International Journal of soft computing and Engineering*, v. 2, n. 10, p. 234–247, 2012. Citado 2 vezes nas páginas 43 e 44.

BIRATTARI, M.; KACPRZYK, J. *Tuning metaheuristics: a machine learning perspective*. [S.l.]: Springer, 2009. v. 197. Citado 4 vezes nas páginas 17, 34, 35 e 36.

BISCHL, B.; KERSCHKE, P.; KOTTHOFF, L.; LINDAUER, M.; MALITSKY, Y.; FRÉCHETTE, A.; HOOS, H.; HUTTER, F.; LEYTON-BROWN, K.; TIERNEY, K. et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, Elsevier, v. 237, p. 41–58, 2016. Citado na página 45.

BRAZDIL, P.; CARRIER, C. G.; SOARES, C.; VILALTA, R. *Metalearning: Applications to data mining*. [S.l.]: Springer Science & Business Media, 2008. Citado 4 vezes nas páginas 17, 18, 39 e 40.

BRAZDIL, P. B.; SOARES, C.; COSTA, J. P. D. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, Springer, v. 50, n. 3, p. 251–277, 2003. Citado 2 vezes nas páginas 42 e 44.

BREEDAM, A. V. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research*, Elsevier, v. 86, n. 3, p. 480–490, 1995. Citado na página 35.

BREIMAN, L. Random forests. *Machine learning*, Springer, v. 45, n. 1, p. 5–32, 2001. Citado na página 47.

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 na página 25.

CAMPOS, G. F.; BARBON, S.; MANTOVANI, R. G. A meta-learning approach for recommendation of image segmentation algorithms. In: IEEE. *Graphics, Patterns and Images (SIBGRAPI), 2016 29th SIBGRAPI Conference on*. [S.l.], 2016. p. 370–377. Citado na página 40.

CHANG, P.-C.; CHEN, M.-H.; TIWARI, M. K.; IQUEBAL, A. S. A block-based evolutionary algorithm for flow-shop scheduling problem. *Applied Soft Computing*, Elsevier, v. 13, n. 12, p. 4536–4547, 2013. Citado na página 25.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, v. 7, n. Jan, p. 1–30, 2006. Citado na página 60.

DÔRES, S. N. das; ALVES, L.; RUIZ, D. D.; BARROS, R. C. A meta-learning framework for algorithm recommendation in software fault prediction. In: ACM. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.], 2016. p. 1486–1491. Citado 5 vezes nas páginas 18, 40, 45, 47 e 49.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 45.

DUNN, J. C. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. Taylor & Francis, 1973. Citado na página 60.

ERSEVEN, G.; AKGÜN, G.; KARAKAŞ, A.; YARIKCAN, G.; YÜCEL, Ö.; ÖNER, A. An application of permutation flowshop scheduling problem in quality control processes. In: SPRINGER. *The International Symposium for Production Research*. [S.l.], 2018. p. 849–860. Citado na página 20.

FERNANDEZ-VIAGAS, V.; RUIZ, R.; FRAMINAN, J. M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, Elsevier, v. 257, n. 3, p. 707–721, 2017. Citado na página 21.

FIX, E.; JR, J. L. H. *Discriminatory analysis-nonparametric discrimination: consistency properties*. [S.l.], 1951. Citado na página 49.

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 3 vezes nas páginas 26, 30 e 32.

FREY, B. J.; DUECK, D. Clustering by passing messages between data points. *science*, American Association for the Advancement of Science, v. 315, n. 5814, p. 972–976, 2007. Citado na página 54.

GANGULY, N.; DEUTSCH, A.; MUKHERJEE, A. *Dynamics on and of complex networks: applications to biology, computer science, and the social sciences*. [S.l.]: Springer, 2009. Citado na página 44.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, INFORMS, v. 1, n. 2, p. 117–129, 1976. Citado na página 25.

GENDREAU, M.; HERTZ, A.; LAPORTE, G. A tabu search heuristic for the vehicle routing problem. *Management science*, INFORMS, v. 40, n. 10, p. 1276–1290, 1994. Citado na página 35.

GENDREAU, M.; POTVIN, J.-Y. et al. *Handbook of metaheuristics*. [S.l.]: Springer, 2010. v. 2. Citado na página 16.

GIANNAKOPOULOS, T.; PIKRAKIS, A. *Introduction to audio analysis: a MATLAB® approach*. [S.l.]: Academic Press, 2014. Citado na página 48.

GUPTA, J. N. A functional heuristic algorithm for the flowshop scheduling problem. *Journal of the Operational Research Society*, Springer, v. 22, n. 1, p. 39–47, 1971. Citado na página 25.

HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S. *Neural networks and learning machines*. [S.l.]: Pearson Upper Saddle River, NJ, USA:, 2009. v. 3. Citado na página 47.

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 2 vezes nas páginas 16 e 20.

IVANOV, D.; DOLGUI, A.; SOKOLOV, B.; WERNER, F.; IVANOVA, M. A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0. *International Journal of Production Research*, Taylor & Francis, v. 54, n. 2, p. 386–402, 2016. Citado na página 20.

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 20.

JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, Wiley Online Library, v. 1, n. 1, p. 61–68, 1954. Citado 2 vezes nas páginas 20 e 24.

JR, E. S.; KILPATRICK, A.; NGUYEN, H.; GU, Q.; GROOMS, A.; POULIN, C. Flexible algorithm selection framework for large scale metalearning. In: IEEE COMPUTER SOCIETY. *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*. [S.l.], 2012. p. 496–503. Citado 2 vezes nas páginas 45 e 49.

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 20.

KALA, R. *On-road intelligent vehicles: Motion planning for intelligent transportation systems*. [S.l.]: Butterworth-Heinemann, 2016. Citado na página 48.

KANDA, J.; CARVALHO, A. de; HRUSCHKA, E.; SOARES, C.; BRAZDIL, P. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing*, Elsevier, v. 205, p. 393–406, 2016. Citado 8 vezes nas páginas 18, 40, 44, 45, 46, 47, 48 e 52.

KARABOGA, D. *An idea based on honey bee swarm for numerical optimization*. [S.l.], 2005. Citado na página 28.

KODRATOFF, Y.; SLEEMAN, D.; USZYNSKI, M.; CAUSSE, K.; CRAW, S. Building a machine learning toolbox. *Enhancing the Knowledge Engineering Process*, North-Holland, Elsevier Science Publishers, p. 81–108, 1992. Citado na página 42.

KRSTAJIC, D.; BUTUROVIC, L. J.; LEAHY, D. E.; THOMAS, S. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, Nature Publishing Group, v. 6, n. 1, p. 10, 2014. Citado na página 60.

LI, X.-t.; YIN, M.-h. A discrete artificial bee colony algorithm with composite mutation strategies for permutation flow shop scheduling problem. *Scientia Iranica*, Elsevier, v. 19, n. 6, p. 1921–1935, 2012. Citado 3 vezes nas páginas 26, 28 e 29.

LIN, S.-W.; HUANG, C.-Y.; LU, C.-C.; YING, K.-C. Minimizing total flow time in permutation flowshop environment. *International Journal of Innovative Computing, Information and Control*, v. 8, n. 10A, p. 6599–6612, 2012. Citado na página 26.

LINDNER, G.; STUDER, R. Ast: Support for algorithm selection with a cbr approach. In: SPRINGER. *European Conference on Principles of Data Mining and Knowledge Discovery*. [S.l.], 1999. p. 418–423. Citado na página 43.

LIU, Y.; YIN, M.; GU, W. An effective differential evolution algorithm for permutation flow shop scheduling problem. *Applied Mathematics and Computation*, Elsevier, v. 248, p. 143–159, 2014. Citado na página 25.

LIU, Y.-F.; LIU, S.-Y. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*, Elsevier, v. 13, n. 3, p. 1459–1463, 2013. Citado na página 25.

LÓPEZ-IBÁÑEZ, M.; CÁCERES, L. P.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. The irace package: User guide. *IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2016-004*, 2016. Citado 2 vezes nas páginas 37 e 38.

LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; BIRATTARI, M.; STÜTZLE, T. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, Elsevier, v. 3, p. 43–58, 2016. Citado 4 vezes nas páginas 9, 34, 36 e 37.

LÓPEZ-IBÁNEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. *The irace package, iterated race for automatic algorithm configuration*. [S.l.], 2011. Citado na página 36.

MARQUES-SILVA, J. P.; SAKALLAH, K. A. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, IEEE, v. 48, n. 5, p. 506–521, 1999. Citado na página 25.

MENZIES, T.; GREENWALD, J.; FRANK, A. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, IEEE, v. 33, n. 1, p. 2–13, 2007. Citado na página 44.

MICHIE, D.; SPIEGELHALTER, D. J.; TAYLOR, C. C. Machine learning, neural and statistical classification. Citeseer, 1994. Citado 4 vezes nas páginas 17, 42, 43 e 47.

MIRANDA, E. S.; FABRIS, F.; NASCIMENTO, C. G.; FREITAS, A. A.; OLIVEIRA, A. C. Meta-learning for recommending metaheuristics for the maxsat problem. In: IEEE. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. [S.l.], 2018. p. 169–174. Citado 3 vezes nas páginas 17, 39 e 52.

MISIR, M.; SEBAG, M. Alors: An algorithm recommender system. *Artificial Intelligence*, Elsevier, v. 244, p. 291–314, 2017. Citado 2 vezes nas páginas 46 e 49.

MORAIS, G.; PRATI, R. C. Complex network measures for data set characterization. In: IEEE. *Intelligent Systems (BRACIS), 2013 Brazilian Conference on*. [S.l.], 2013. p. 12–18. Citado 3 vezes nas páginas 45, 49 e 51.

MORAIS, R. F. de; MIRANDA, P. B.; SILVA, R. M. A meta-learning method to select under-sampling algorithms for imbalanced data sets. In: IEEE. *Intelligent Systems (BRACIS), 2016 5th Brazilian Conference on*. [S.l.], 2016. p. 385–390. Citado 2 vezes nas páginas 40 e 45.

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 na página 25.

NUDELMAN, E.; LEYTON-BROWN, K.; HOOS, H. H.; DEVKAR, A.; SHOHAM, Y. Understanding random sat: Beyond the clauses-to-variables ratio. In: SPRINGER. *International Conference on Principles and Practice of Constraint Programming*. [S.l.], 2004. p. 438–452. Citado na página 54.

OLIVEIRA, A. C. Algoritmos evolutivos híbridos com detecção de regiões promissoras em espaços de busca contínuos e discretos. *Algoritmos evolutivos híbridos com detecção de regiões promissoras em espaços de busca contínuos e discretos*, 2004. Citado na página 31.

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*, Taylor & Francis, v. 16, n. 1, p. 101–107, 1965. Citado na página 25.

PAPPA, G. L.; FREITAS, A. *Automating the design of data mining algorithms: an evolutionary computation approach*. [S.l.]: Springer Science & Business Media, 2009. Citado na página 42.

PAPPA, G. L.; OCHOA, G.; HYDE, M. R.; FREITAS, A. A.; WOODWARD, J.; SWAN, J. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, Springer, v. 15, n. 1, p. 3–35, 2014. Citado 2 vezes nas páginas 42 e 44.

PARMEZAN, A. R. S.; LEE, H. D.; WU, F. C. Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework. *Expert Systems with Applications*, Elsevier, v. 75, p. 1–24, 2017. Citado 2 vezes nas páginas 47 e 49.

PAVELSKI, L.; DELGADO, M.; KESSACI, M.-E. Meta-learning for optimization: A case study on the flowshop problem using decision trees. In: IEEE. *2018 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2018. p. 1–8. Citado na página 41.

PINEDO, M. *Scheduling*. [S.l.]: Springer, 2012. Citado 4 vezes nas páginas 18, 20, 21 e 22.

RESENDE, M. G. Introdução aos algoritmos genéticos de chaves aleatórias viciadas. Citado na página 31.

RICE, J. R. The algorithm selection problem. *Advances in computers*, Elsevier, v. 15, p. 65–118, 1976. Citado 5 vezes nas páginas 9, 17, 41, 42 e 50.

ROSSI, A. L. D.; CARVALHO, A. C. P. de Leon Ferreira de; SOARES, C.; SOUZA, B. F. D. et al. Metastream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, Elsevier, v. 127, p. 52–64, 2014. Citado 3 vezes nas páginas 46, 47 e 49.

ROTHLAUF, F. *Design of modern heuristics: principles and application*. [S.l.]: Springer Science & Business Media, 2011. Citado na página 16.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, Elsevier, v. 165, n. 2, p. 479–494, 2005. Citado na página 20.

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.

SERBAN, F.; VANSCHOREN, J.; KIETZ, J.-U.; BERNSTEIN, A. A survey of intelligent assistants for data analysis. *ACM Computing Surveys (CSUR)*, ACM, v. 45, n. 3, p. 31, 2013. Citado 2 vezes nas páginas 17 e 39.

SMITH-MILES, K. A. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, ACM, v. 41, n. 1, p. 6, 2009. Citado 5 vezes nas páginas 17, 39, 41, 43 e 45.

SOARES, C.; BRAZDIL, P. B.; KUBA, P. A meta-learning method to select the kernel width in support vector regression. *Machine learning*, Springer, v. 54, n. 3, p. 195–209, 2004. Citado na página 49.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, Elsevier, v. 45, n. 4, p. 427–437, 2009. Citado na página 60.

SOUZA, B. F. de; CARVALHO, A. de; SOARES, C. Metalearning for gene expression data classification. In: IEEE. *Hybrid Intelligent Systems, 2008. HIS'08. Eighth International Conference on*. [S.l.], 2008. p. 441–446. Citado na página 40.

SPEARS, W. M.; JONG, K. D. D. *On the virtues of parameterized uniform crossover*. [S.l.], 1995. Citado na página 31.

TAILLARD, E. Benchmarks for basic scheduling problems. *european journal of operational research*, Elsevier, v. 64, n. 2, p. 278–285, 1993. Citado na página 56.

TALBI, E.-G. *Metaheuristics: from design to implementation*. [S.l.]: John Wiley & Sons, 2009. v. 74. Citado 2 vezes nas páginas 16 e 25.

VALLADA, E.; RUIZ, R.; FRAMINAN, J. M. New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, Elsevier, v. 240, n. 3, p. 666–677, 2015. Citado na página 56.

VANSCHOREN, J. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018. Citado na página 18.

VILALTA, R.; DRISSI, Y. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, Springer, v. 18, n. 2, p. 77–95, 2002. Citado 2 vezes nas páginas 17 e 39.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. *Data Mining: Practical machine learning tools and techniques*. [S.l.]: Morgan Kaufmann, 2016. Citado na página 49.

WOLPERT, D. H.; MACREADY, W. G. et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, v. 1, n. 1, p. 67–82, 1997. Citado 2 vezes nas páginas 17 e 41.

XIE, Z.; ZHANG, C.; SHAO, X.; LIN, W.; ZHU, H. An effective hybrid teaching–learning-based optimization algorithm for permutation flow shop scheduling problem. *Advances in Engineering Software*, Elsevier, v. 77, p. 35–47, 2014. Citado na página 26.