UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS / FACULDADE DE ENGENHARIA
PROGRAMA DE PÓS GRADUAÇÃO EM MODELAGEM COMPUTACIONAL

Algoritmos Genéticos Adaptativos para Solucionar Problemas de Sequenciamento do Tipo Job-Shop Flexível

GUILHERME DE SOUZA FERREIRA

JUIZ DE FORA
2018

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
PROGRAMA DE PÓS GRADUAÇÃO EM MODELAGEM COMPUTACIONAL

Algoritmos Genéticos Adaptativos para Solucionar Problemas de Sequenciamento do Tipo Job-Shop Flexível

GUILHERME DE SOUZA FERREIRA

JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS / FACULDADE DE ENGENHARIA
2018

Algoritmos Genéticos Adaptativos para Solucionar Problemas de Sequenciamento do Tipo Job-Shop Flexível

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional.

Orientador: Prof. Dr. Heder Soares Bernardino

JUIZ DE FORA

INSTITUTO DE CIÊNCIAS EXATAS / FACULDADE DE ENGENHARIA

2018

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Ferreira, Guilherme de Souza.
Algoritmos genéticos adaptativos para solucionar problemas de sequenciamento do tipo job-shop flexível / Guilherme de Souza Ferreira. -- 2018.
106 f. : il.

Orientador: Heder Soares Bernardino
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, ICE/Engenharia. Programa de Pós-Graduação em Modelagem Computacional, 2018.

1. Algoritmo genético adaptativo. 2. Job-shop. 3. Sequenciamento. 4. Seleção de operadores adaptativa. 5. Seleção de parâmetros adaptativa. I. Bernardino, Heder Soares, orient. II. Título.

GUILHERME DE SOUZA FERREIRA

Algoritmos Genéticos Adaptativos para Solucionar Problemas de Sequenciamento do Tipo Job-Shop Flexível

DISSERTAÇÃO APRESENTADA AO PROGRAMA DE PÓS GRADUAÇÃO EM MODELAGEM COMPUTACIONAL DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE EM MODELAGEM COMPUTACIONAL.

Aprovada em: 22/02/2018

BANCA EXAMINADORA

Professor Dr. Heder Soares Bernardino
Orientador
Universidade Federal de Juiz de Fora

Professor Dr. Hélio José Corrêa Barbosa
Membro Interno
Universidade Federal de Juiz de Fora

Professor Dr. Eduardo Krempser da Silva
Membro Externo
Fundação Oswaldo Cruz

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, que iluminou o meu caminho e me deu forças para que eu alcançasse meus objetivos.

Aos meu pais, Darcilei e Heloisa, e também ao meu irmão, Gustavo, que com muito carinho e compreensão entenderam a importância dessa jornada, e foram a base para toda essa etapa.

À Laís, minha namorada, que além de entender o desafio ao qual eu me propunha, dedicou horas ao meu lado, deu incentivos para persistir nos meus sonhos, e compartilhou as realizações do caminho.

Pelo apoio dado por todos os meus familiares, pela Gláucia, Luiz Fernando, Igor e, em especial, à minha tia Bete, que mesmo distante sempre foi a mais entusiasmada pelos meus estudos.

Aos meus amigos do PPGMC, que dividiram momentos de aprendizado e também alegrias no laboratório, no RU ou na copa.

Aos amigos da EPR e MRS que não só entenderam a mudança, mas também me mostraram e me ajudaram a superar os obstáculos que existem em todos os caminhos.

Ao meu orientador, Professor Heder Bernardino, pelos esforços empenhados como educador, pela paciência e pela excelente orientação realizada durante o desenvolvimento do mestrado.

E aos demais colegas, professores, funcionários e técnicos do PPGMC pelos ensinamentos e suporte essenciais para minha formação.

Por fim, agradeço à UFJF, à CAPES, ao CNPQ e à FAPEMIG pela bolsa de estudos, estrutura e os auxílios financeiros, que tornaram esse trabalho possível.

"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change."

Leon C. Megginson

RESUMO

O escalonamento de tarefas é um problema de otimização combinatória no qual tenta-se sequenciar da melhor maneira os trabalhos a serem realizados em processos de produção. O intuito neste caso é atingir os objetivos de desempenho estipulados pelo tomador de decisão, tais como, minimizar o *makespan* e minimizar o atraso total. O Problema de Sequenciamento do tipo Job-Shop Flexível (FJSP) pertence a essa categoria, e caracteriza-se pela possibilidade de haver rotas tecnológicas diferentes para as tarefas e cada estágio poder ser composto por mais de uma máquina. Esse é o núcleo da tecnologia do gerenciamento de produção, pois sequenciamentos melhores podem encurtar o tempo de manufatura, reduzir os níveis de estoque, possibilitar a entrega de encomendas no tempo correto e aumentar a credibilidade dos processos e da empresa. Métodos exatos, que são computacionalmente custosos, são geralmente aplicados nos problemas de sequenciamento menores, portanto quando os problemas aumentam em tamanho, os métodos heurísticos e metaheurísticos começaram a ser aplicados. As metaheurísticas são importantes para solucionar FJSPs porque são mais rápidas do que os métodos exatos. Dentre elas, os Algoritmos Genéticos (AGs) estão entre as técnicas mais utilizadas para solucionar FJSPs e, atualmente, modelos híbridos vem sendo explorados, combinando AGs com técnicas de busca local e heurísticas para inicializar a população. No entanto, a escolha adequada dos parâmetros dos AGs é um trabalho difícil, recaindo num outro problema de otimização. Os Algoritmos Genéticos Adaptativos (AGAs) foram introduzidos para lidar com essa adversidade, uma vez que podem ajustar os parâmetros dos AGs durante o processo de busca. Portanto, o objetivo da presente dissertação é analisar diferentes técnicas adaptativas desenvolvidas para AGAs, com o intuito de reduzir o tempo de configuração dos AGs quando aplicados a FJSPs. Além disso, serão propostas alterações para as técnicas de atribuição de crédito e de seleção de operadores. Os estudos foram realizados em instâncias de diferentes tamanhos e os AGAs são comparados com AGs tradicionais. Duas diferentes análises foram realizadas baseadas em cenários no qual o tomador de decisão tem pouco tempo para configurar os algoritmos. Na Análise I, os AGAs tiveram desempenho semelhante aos AGs tradicionais, mas são interessantes por possuírem um menor número de parâmetros e, consequentemente, um menor tempo de configuração. Na Análise II, os AGAs geraram melhores resultados do que aqueles obtidos pelos AGs, o que os tornam apropriados para o caso em que há incerteza no processo produtivo e menor tempo de configuração.

Palavras-chave: algoritmo genético adaptativo; job-shop; sequenciamento; seleção de operadores adaptativa; seleção de parâmetros adaptativa;

ABSTRACT

Scheduling is a combinatorial optimization problem, in which one tries ordering the tasks to be performed in the processing units. The objective is to achieve the best values with respect to the performance indicators chosen by the decision-maker, such as, minimize the makespan and minimize the total lateness. The Flexible Job-Shop Scheduling Problem (FJSP) belongs to this category, and its characteristics are the different technological routes for the tasks and that each stage may consist of more than one machine. This is the technological core of the production management, as better schedules may reduce the manufacturing time, reduce the inventory, deliver the order in the right time, and raise the reliability of the process and the company. Exact methods, as they are computationally expensive, are usually employed for small scheduling problems, then heuristic and metaheuristic methods become interesting techniques for this type of problem. Metaheuristics are important to solve FJSPs as they are faster than the exact methods, and among then, Genetic Algorithms (GAs) are one of the most used techniques to solve FJSPs and, currently, they have been hybridized with local search and heuristics to initialize their population. However, to set up GAs is a hard-work and often generates another optimization problem. Adaptive Genetic Algorithms (AGAs) were introduced to work around this problem as they adapt the parameters of the GAs during the search process. Therefore, the objective of this dissertation is to analyze different adaptive techniques developed for AGAs with the purpose of reducing the setup time of GAs when they are applied to FJSPs. In addition, modifications will be proposed for the operator selection techniques and for credit assignment schemes. The studies were performed in instances of different sizes, and the AGAs are compared with traditional GAs. Two different analyzes were performed based on scenarios in which the decision maker does not has to much time to configure the algorithms. In Analysis I, some AGAs performed similarly to the traditional GAs, but they are more interesting as they have a smaller number of parameters, thus a shorter configuration time. In Analysis II, some AGAs generated better results than those obtained by GAs, which makes them appropriate for the case when there is uncertainty in the production process and the decision maker does not have too much time to configure the algorithm.

Key-words: adaptive genetic algorithm; job-shop; scheduling; adaptive operator selection; adaptive parameter control;

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AP	*Adaptive Pursuit*
APEX	*Adaptive Pursuit com valor EXtremo*
AE	*Algoritmo Evolutivo*
AG	*Algoritmo Genético*
AGA	*Algoritmo Genético Adaptativo*
LS	*Busca Local*
DMAB	*Dynamic Multi-Armed Bandit*
DMABEX	*Dynamic Multi-Armed Bandit com valor EXtremo*
FJSP	*Flexible Job-Shop Scheduling Problem*
NEH	*Heurística construtiva proposta por Nawaz, Enscore e Ham*
IA	*Inteligência Artificial*
IM	*Interchange pairwise neighborhhod mutation*
JSP	*Job-Shop Scheduling Problem*
LOX	*Linear-Order Crossover*
MAB	*Multi-Armed Bandit*
RDMAB	*Normalized-Roulette Dynamic Multi-Armed Bandit*
OPX	*Order-Preserving one-point Crossover*
PH	*Page-Hinkley*
PMX	*Partially Mapped Crossover*
PD	*Perfis de Desempenho*
CPPC	*Predctive Parameter Control selecionado a partir de intervalos*
PPC	*Predictive Parameter Control*
PP	*Programação da Produção*
RMAB	*Rank-based Multi-Armed Bandit*

SM	*Shift Mutation*
SLMAB	*SLiding Multi-Armed Bandit*
SLMABEX	*SLiiding Multi-Armed Bandit com valor EXtremo*
SR	*Sum of Ranks*

LISTA DE SÍMBOLOS

al	Algoritmo
A	Ambiente de máquina
Al	Conjunto de algoritmos
$\mathcal{C}$	Conjunto de estágios
$\mathcal{M}$	Conjunto de máquinas
Ta	Conjunto de problemas
$\mathcal{J}$	Conjunto de tarefas
t_{jc}	Duração da tarefa j no estágio c
c	Estágio
$\rho_{al}(\tau)$	Fração de problemas resolvidos pelo algoritmo al, com desempenho dentro de um fator τ do melhor comportamento obtido, considerando todos os algoritmos analisados.
λ	Gatilho do *Page-Hinkley*
W	Janela temporal dos métodos adaptativos
J_m	Job-Shop
FJ_c	Job-Shop Flexível
m	Máquina
$\bar{\mathcal{R}}_{i,l}$	Média das recompensas do parâmetro i até a utilização l
$C_{\max}$	*Makespan*
i^*	Melhor operador do método *Adaptive Pursuit*
$\circ$	Indica que não há restrições nas tarefas
D	Nível de decaimento do *Rank-based Multi-Armed Bandit*
nm_{al}	Número de algoritmos que pertencem ao Al
des	Número de componentes removidos na fase de destruição da busca local
K	Número de parâmetros

nm_{ta}	Número de problemas que pertencem ao Ta
u_i	Número de vezes que o parâmetro i foi utilizado
u_i^s	Número de vezes que o parâmetro i foi utilizado com sucesso
$n_{i,t}$	Número de vezes utilizadas que o parâmetro i no tempo t
Γ	Objetivo que será buscado
o_{jc}	Operação da tarefa j no estágio c
i	Parâmetro
δ	Parâmetro de tolerância do *Dynamic Multi-Armed Bandit*
C	Parâmetro que balanceia a explotação e a exploração nos modelos baseados no *Multi-Armed Bandit*
$\mathcal{P}_i$	Probabilidade do parâmetro i
$\mathcal{P}_{\max}$	Probabilidade máxima
$\mathcal{P}_{\min}$	Probabilidade mínima
ta_{pr}	Problema
$\mathcal{Q}_i$	Qualidade do parâmetro i
$\hat{q}_{i,t}$	Qualidade estimada empiracmente para o parâmetro i no tempo t
$rd_{ta,al}$	Razão entre o desempenho do algoritmo al e o melhor algoritmo quando resolve o problema ta
RI	Recompensa instantânea no tempo t
$\mathcal{R}_i(t)$	Recompensa do parâmetro i no tempo t
B	Restrições das tarefas
j	Tarefa
α	Taxa de adaptação do método *Adaptive Pursuit*
β	Taxa de aprendizado do método *Adaptive Pursuit*
t	Tempo atual
f_j	Tempo de fluxo da tarefa j

t_i	Tempo em que o operador i foi utilizado pela última vez
$\vec{\pi}$	Vetor que contém as tarefas removidas na busca local

SUMÁRIO

1 INTRODUÇÃO

Nos processos decisórios presentes no cotidiano, busca-se otimizar algum objetivo específico, tal como: minimizar custos, maximizar algum desempenho próprio ou de um objeto, e minimizar o tempo gasto para realizar determinada atividade. Essa busca pela melhor decisão também ocorre nos processos empresariais, nos quais procura-se o melhor retorno associado ao risco de fundos de investimento, a melhor sequência de produção de determinado produto, a melhor rota para entrega, a melhor maneira de alocar recursos, entre outros.

O sequenciamento ou o escalonamento de processos é uma atividade operacional na qual deve-se selecionar a ordem em que as atividades devem ser processadas em um conjunto de máquinas (Werner, 2011), com o intuito de atingir os objetivos de desempenho previamente escolhidos. Trata-se de um processo operacional realizado pelo tomador de decisões em diferentes empresas manufatureiras ou prestadoras de serviços. A seguir, pode-se verificar diferentes processos decisórios, os quais podem ser modelados como um problema de escalonamento de processos (Liu; Kozan, 2009; Pinedo, 2012):

- Numa companhia, ao determinar a ordem de fabricação de um produto ou de atendimento aos clientes;
- Num aeroporto, ao decidir qual avião decolar ou aterrissar, ou atribuir um avião a uma determinada pista de aterrissagem;
- Durante uma obra civil, ao alocar uma equipe para realizar determinado tipo de serviço;
- O escalonador de processos de um computador, ao escolher qual programa será executado em qual unidade de processamento;
- Numa ferrovia, ao montar a grade horária dos trens, define-se qual trem deve passar primeiro em uma determinada estação ferroviária.

Esses problemas, além de poderem ser modelados com diferentes complexidades, podem ainda se diferenciar devido às características do sistema. Quanto à tarefa, podem distinguir-se pela inclusão de tempo em que a tarefa estará disponibilizada para entrada no sistema, tempo em que a tarefa é prometida ao cliente e o nível de prioridade destes. Quanto aos objetivos, pode-se considerar, entre outros: *makespan* (tempo total em que o sistema processa todas as atividades), soma do atraso total ponderado, e maximização da utilização dos recursos. Finalmente, o modelo pode contemplar as seguintes particularidades: tempo de preparação, tempo gasto para realizar o ajuste das máquinas

antes de iniciar determinadas tarefas; ambiente *no-wait*, cenário em que uma tarefa deve concluir o processamento sem pausas durante as operações; e recirculação, cenário em que uma determinada tarefa pode voltar a realizar operação numa máquina que já havia processado-a.

O sequenciamento de processos possui grande importância no ambiente empresarial, sendo que, no cenário atual, as empresas recebem diversas pressões de mercado. O fato de os consumidores requererem variedade nos produtos, os ciclos de vida de produtos mais curtos e a pressão competitiva para reduzir custos, resultam na necessidade de reduzir os níveis de estoque. Contrapondo a essas características, tem-se que as companhias devem responder rapidamente às constantes mudanças na quantidade demandada, que consequentemente sugerem que as corporações empreguem um maior nível de estoque. Como resultado desse contexto empresarial, as empresas buscam estratégias de sequenciamento eficazes e mais eficientes, a fim de apoiar as recorrentes decisões que impactam nesses objetivos conflitantes. Isso culmina na busca por bons algoritmos de escalonamento de processos.

Os problemas de sequenciamento de processos são complexos e são definidos na literatura como problemas de otimização combinatória. Desta forma, possuem um número finito de soluções e, exceto em casos muito restritos, são classificados como problemas NP-difícil. Segundo Chaudhry & Khan (2016), a natureza prática desses problemas e a complexidade mencionada fez com que surgissem muitos estudos para solucioná-los. Algoritmos exatos como o método *Branch & Bound* e a programação dinâmica são computacionalmente custosos para encontrar soluções ótimas para o problema de escalonamento de processos com espaço grande de busca (Ripon; Tsang; Kwong, 2007). Por consequência, torna-se razoável a busca por boas soluções, as quais se encontram próximas à solução ótima. Portanto, introduziu-se a aplicação de heurísticas e meta-heurísticas (Jain; Meeran, 1999) nesses problemas.

Métodos heurísticos são utilizados quando não há um algoritmo eficiente para solucionar os problemas. Atualmente, existem inúmeras meta-heurísticas e muitas já foram utilizadas para solucionar problemas de sequenciamento de processos. Sua atratividade se deve ao fato de que esses métodos são uma alternativa mais ágil, as quais obtém soluções com menor esforço computacional, quando comparada aos métodos baseados em *Branch & Bound* (Çaliş; Bulkan, 2015).

O sequenciamento de processos divide-se em diversas classes de problemas, porém nessa dissertação o estudo se restringirá aos Problemas de *Job-Shop* Flexível (FJSPs). Nesse ambiente, segundo Pinedo (2012), as tarefas devem ser processadas em estágios que estão dispostos em série. Cada estágio possui máquinas em paralelo, as quais são capazes de processar quaisquer tarefas existentes. Além disso, Pinedo (2012) cita a principal

característica desse sistema, que é a possibilidade de diferenciar as rotas tecnológicas das tarefas. Um exemplo desse ambiente são as ferrovias, em que o processo de elaboração de uma grade horária de trens (horários que os trens passarão na estação ferroviária) pode ser modelado como um problema de sequenciamento, e como os trens podem ter rotas diferentes, resulta em um problema do tipo *Job-Shop*. Ademais, as estações ferroviárias podem ter mais de uma linha, o que torna a elaboração da grade de trens um problema de máquinas paralelas e, por isso, caracteriza-se como um FJSP.

Çaliş & Bulkan (2015) mostraram que minimizar o *makespan* é o objetivo mais estudado na literatura, representando 54,84% dos trabalhos apresentados. Esse objetivo é definido como o tempo total que o sistema utiliza para completar todas as tarefas (Chaudhry; Khan, 2016). Segundo Çaliş & Bulkan (2015), em sua revisão de literatura, identificaram que os Algoritmos Genéticos (AGs) são as estratégias de Inteligência Artificial (IA) mais utilizadas para solucionar problemas de sequenciamento do tipo *Job-Shop* (JSP). Essas meta-heurísticas estão presentes em 26,4% dos trabalhos investigados naquele artigo.

Similar a esses resultados, Chaudhry & Khan (2016) mostraram em sua revisão de literatura que os algoritmos mais utilizados para solucionar problemas de sequenciamento do tipo *Job-Shop* flexível (FJSPs) são os algoritmos híbridos e os Algoritmos Evolutivos (AEs), que aparecem em 35,03% e em 23,86% dos trabalhos investigados, respectivamente. Em ambas as categorias citadas, os AGs são as técnicas mais utilizadas, presentes em 34% das publicações referidas em (Chaudhry; Khan, 2016). O uso preferencial por AEs, sobretudo dos AGs, é que esses evoluem uma população de soluções, ao contrário dos outros algoritmos tradicionais de otimização os quais evoluem uma única solução, o que evita a estagnação em ótimos locais (Ripon; Tsang; Kwong, 2007).

Os AGs possuem parâmetros como: forma de representação, tamanho da população, operadores de movimento (recombinação e mutação) e suas taxas de ocorrência (Karafotias; Smit; Eiben, 2012). Além disso, eles são dependentes do correto ajuste desses parâmetros para obterem sucesso durante a busca de soluções. Portanto, configurá-los não é uma tarefa muito fácil, uma vez que ao tomar como exemplo um AG com três tamanhos populacionais diferentes, três operadores de recombinação, dois operadores de mutação, um operador de seleção, três valores possíveis para as taxas de recombinação e três alternativas para as taxas de mutação, tem-se como resultado 162 ($3^4 \times 2$) configurações disponíveis. A situação pode tornar-se ainda mais complexa, visto que a melhor configuração de um AG é dependente do problema que está sendo resolvido (Krempser; Fialho; Barbosa, 2012). Além disso, um conjunto diferente de parâmetros e valores pode ser adequado para cada etapa do processo de busca. Sendo assim, mesmo que seja avaliada e utilizada a melhor configuração do AGs para um dado problema, essa configuração pode não ser a melhor

quando aplicada a outros problemas (Aleti; Moser, 2016) e ainda pode não ser a melhor durante todo o processo evolutivo.

Geralmente, em um AG tradicional, os parâmetros são escolhidos após uma série de execuções com o intuito de determinar o conjunto de parâmetros que gera os melhores resultados. Segundo Hinterding, Michalewicz & Eiben (1997), esse método possui algumas desvantagens, como:

- Erros cometidos pelo usuário ao configurar um AG podem causar perda de desempenho;
- Essa seleção de parâmetros é uma tarefa demorada;
- Os valores ótimos dos parâmetros podem variar durante o processo de busca.

Ainda segundo Hinterding, Michalewicz & Eiben (1997), pode-se citar três categorias que realizam a seleção de parâmetros de maneira dinâmica dentro de um AG:

- Determinística: quando existe uma estratégia determinística para ajustar os parâmetros. A escolha dos parâmetros ocorre sem receber nenhuma forma de retorno da população e, geralmente, utiliza regras baseadas no número de gerações decorridas no processo do AG.
- Adaptativa: quando o AG recebe alguma forma de retorno do processo evolutivo, por exemplo, da função de aptidão, e esse retorno impacta na probabilidade dos parâmetros a serem utilizados.
- Autoadaptativa: quando cada indivíduo possui uma parte em seu cromossomo que identifica qual parâmetro será aplicado. Essa sequência não é utilizada para cálculo da aptidão, porém a ideia principal é que bons parâmetros irão gerar bons indivíduos, e esses estarão mais suscetíveis a sobreviver e serem utilizados novamente para gerar novos indivíduos. Dessa maneira, os melhores parâmetros serão propagados.

Os algoritmos que selecionam os parâmetros dinamicamente reduzem o tempo de configuração e teste de um algoritmo, uma vez que escolhem os parâmetros que estão disponíveis na sua estrutura, por exemplo, os diferentes tipos de operadores de recombinação.

Devido a essas necessidades, os Algoritmos Genéticos Adaptativos (AGAs) surgiram para tratar o problema de configurar um AG. Desde o trabalho original de Davis (1989), o qual foi o primeiro artigo publicado sobre AGAs, até a revisão de literatura elaborada por Aleti & Moser (2016), muito se desenvolveu no que concerne a tais algoritmos.

Um AG multiobjetivo e adaptativo foi proposto em (Chang; Hsieh; Wang, 2007). Aquele algoritmo adapta as taxas de recombinação e mutação de acordo com o aptidão do indivíduo gerado em relação ao seu predecessor. O problema estudado nesse artigo tinha o objetivo de reduzir o *makespan* e o tempo de atraso total. Chang, Hsieh & Wang (2007), ao compararem o algoritmo adaptativo proposto, mostraram que os resultados de seus algoritmos foram superiores aos de outros métodos adaptativos.

Xing *et al.* (2007) propuseram um AG adaptativo baseado na aptidão para solucionar JSPs e evitar que o algoritmo fique preso num ótimo local. O modelo adapta somente as taxas de ocorrência da recombinação e da mutação. Segundo Xing *et al.* (2007), os experimentos foram realizados sobre duas instâncias de problemas de sequenciamento do tipo Job-Shop e os resultados obtidos indicam que o algoritmo converge rapidamente, evitando ótimos locais.

Yang *et al.* (2008) propuseram um AGA para adaptar as probabilidades de ocorrência dos operadores de movimento. O algoritmo proposto foi aplicado a duas instâncias e tem por objetivo reduzir o *makespan*. Seus resultados mostraram que o algoritmo converge rapidamente, quando está sendo comparado com um AG.

Hongyan & Hong (2010) desenvolveram um AG multiobjetivo autoadaptativo denominado *Self-Adaptive Genetic Algorithm* (SAGA). O SAGA, pela definição mencionada anteriormente é considerado um algoritmo adaptativo, pois utiliza a aptidão média e mínima da população e a aptidão dos filhos gerados para determinar a taxa de recombinação e mutação. Esse cálculo também envolve os limites inferiores e superiores para delimitar os parâmetros. O algoritmo foi comparado com um Algoritmo Genético e mostrou-se capaz de convergir rapidamente.

O trabalho de Pan *et al.* (2011) propôs um AGA para solucionar problemas de FJSP. Esse trabalho utiliza a adaptação através de equações determinísticas para selecionar as melhores taxas de ocorrência do operador de recombinação, de mutação e também a quantidade de indivíduos que irão sobreviver através do esquema de elitismo. Além disso, aquela proposta também altera a área de ação dos operadores de movimento sobre o cromossomo. Naquele trabalho, o AGA superou os resultados de um AG tradicional.

No trabalho proposto por Wang & Tang (2011) foi desenvolvido um AGA para solucionar JSPs que adaptativamente altera as taxas dos operadores de movimento. Para isso, o algoritmo utiliza um método baseado em modulação hormonal (função monótona utilizada para representar a quantidade de hormônio excretado por uma glândula e que possui como característica a não-negatividade das variáveis) e atua de forma parecida com a técnica de busca em escalada (*Hill Climbing*). Os resultados obtidos através de três experimentos mostraram que o algoritmo proposto é mais eficiente do que os AGs.

Uma seleção adaptativa para um AG celular foi proposto em (Li *et al.*, 2014). O objetivo principal da adaptação naquele algoritmo é evitar a estagnação num ótimo local. Para esse fim, o algoritmo seleciona os indivíduos que sobreviverão para a próxima geração de acordo com a aptidão normalizada dos indivíduos vizinhos. No estudo, Li *et al.* (2014) avaliou o objetivo de minimizar o *makespan* testando seu algoritmo em duas instâncias. Os experimentos mostraram que os algoritmos são eficientes para solucionar JSP.

Em (Nalepa; Cwiek; Kawulok, 2015), um algoritmo memético adaptativo foi proposto para reduzir o *makespan*. A proposta desse artigo utiliza a adaptação para selecionar os parâmetros deste método. De acordo com os autores, a adaptação foi introduzida no método para realizar um comprometimento entre a procura por diversidade (exploração) e a busca por intensificação (explotação). Os resultados obtidos nas experimentações realizadas por Nalepa, Cwiek & Kawulok (2015), mostraram que o algoritmo proposto é eficaz para solucionar os problemas, e até mais eficiente que o AG utilizado nas comparações.

Em (Yan; Wu, 2015) foram selecionados quatro operadores de recombinação e quatro operadores de mutação da literatura. A partir daí, a adaptação tenta selecionar a melhor combinação dos operadores, permitindo somente o uso de um operador de cada tipo. O algoritmo utiliza uma janela temporal para armazenar a aptidão anterior devido a utilização de uma combinação de operadores e, assim, calcular o crédito que será atribuído a eles. Nesse esquema, as probabilidades de ocorrência dos operadores são modificadas nas gerações ímpares enquanto os melhores valores são usados nas gerações pares. Yan & Wu (2015) testaram os algoritmos em uma instância e mostraram que é possível automatizar a seleção de operadores, sendo que a introdução de uma janela temporal, tornou a busca automática por operadores mais efetiva, o que fez o algoritmo proposto superar AGs comuns.

Em (Wang; Cai; Li, 2016), foi proposto um Algoritmo Genético multipopulacional para solucionar JSPs no qual a adaptação é utilizada para ajustar a probabilidade de ocorrência dos operadores de movimento de acordo com a aptidão. O objetivo do algoritmo proposto é de reduzir o *makespan* e obteve resultados superiores a AGs e outras metaheurísticas.

Um esquema adaptativo incorporado a uma evolução diferencial foi proposto por Zhang *et al.* (2016) com o objetivo de reduzir o *makespan*. Para tal, foi desenvolvido nesse algoritmo um operador que seleciona entre dois diferentes tipos de mutação já estabelecidos, sendo que a adaptação ocorre de acordo com a diversidade do indivíduo.

Zuo, Gong & Jiao (2016) desenvolveram um algoritmo memético adaptativo que utiliza a técnica de Multi-Armed Bandit para selecionar entre dois operadores estocásticos e três diferentes métodos de busca local.

A abordagem em (Riley; Mei; Zhang, 2016) é uma programação genética para solucionar JSPs. Esse trabalho evolui regras de despacho e faz uso da adaptação para guiar o algoritmo a concentrar-se nos terminais mais essenciais. São inseridos pesos nos terminais para guiar o processo de mutação para focar nos terminais mais importantes. O ajuste dos pesos dos terminais são adaptados baseados na frequência em que aparecem nos indivíduos e aptidão deles.

Huang, Wang & Liang (2016) desenvolveram um AG para tratar FSJPs baseado na aptidão da população. A adaptação restringe as taxas de ocorrência dos operadores de movimento, e beneficiou ao acelerar a convergência do algoritmo. O algoritmo proposto por Huang, Wang & Liang (2016) foi aplicado aos JSPs em três instâncias diferentes. A proposta aumentou a convergência do algoritmo e melhorou o retorno quando comparado a AGs tradicionais.

Um algoritmo adaptativo foi desenvolvido em (Lu *et al.*, 2017) para tratar o problema de sequenciamento do tipo *flow-shop*. Diferentemente dos demais trabalhos, a adaptação nesse caso busca aumentar a diversidade entre os indivíduos. O algoritmo é aplicado a um problema multiobjetivo (minimizar o *makespan* e a quantidade consumida de energia) real. A proposta realizada apresentou uma melhoria das soluções quando comparado a algoritmos já estabelecidos na literatura, quando estão solucionando um caso real de uma empresa chinesa.

O cenário empresarial altera-se dinamicamente devido às diversas pressões recebidas, o que gera a necessidade de sequenciamentos mais eficientes e eficazes. Para alcançá-los, têm-se utilizado os AGs a fim de solucionar FJSPs, mas configurá-los demanda tempo e os erros ocasionados por má configuração expõem os decisores a possíveis perdas de desempenho. Para contornar esses problemas, os AGAs foram aplicados com o intuito de solucionar FJSPs, uma vez que procuram as melhores configurações para os AGs durante o processo de busca. Porém, os algoritmos adaptativos desenvolvidos para FJSPs geralmente utilizam a recompensa recebida para alterar a probabilidade de ocorrência dos parâmetros sem avaliar as diferentes técnicas existentes de seleção de operadores e maneiras de atribuir crédito. Na literatura, e em outras aplicações, diferentes esquemas foram desenvolvidos para selecionar e para atribuir crédito aos operadores (Thierens, 2005; da Costa *et al.*, 2008; Fialho; Schoenauer; Sebag, 2010; Fialho *et al.*, 2010; Aleti; Moser, 2011).

Essa dissertação tem como objetivo projetar e analisar AGAs quando aplicados aos FJSPs. Este trabalho difere dos anteriores, no contexto dos FJSPs, por utilizar e comparar as técnicas de seleção de operadores existentes e também algumas variações na forma de assimilar crédito para os AGAs. Além disso, são propostas modificações para as técnicas de operadores de seleção e na forma de assimilar o crédito.

O estudo diferencia-se da literatura de algoritmos adaptativos para solucionar FJSPs, uma vez que serão realizadas comparações entre métodos adaptativos existentes, o que não foi realizado nos trabalhos da literatura. Um segundo ponto que vale ser mencionado, é que os AGAs estudados terão maior liberdade para escolha dos parâmetros, uma vez que selecionarão tanto as taxas de ocorrência quanto os operadores de recombinação e mutação que serão utilizados durante o processo evolutivo do AG. Os AGAs ainda escolherão a taxa de ocorrência da busca local, tornando a configuração de um AG híbrido mais fácil de ser realizada, por possuir menos parâmetros.

Estudos preliminares mostraram que a técnica de seleção de operadores *Adaptive Pursuit* (AP) quando comparada aos AGs tradicionais foram promissoras para solucionar FJSPs (Ferreira; Bernardino, 2017a). Posteriormente, foram propostos AGAs híbridos com busca local em (Ferreira; Bernardino, 2017b) e os resultados foram comparados aos alcançados por AGs híbridos. A técnica AP com a recompensa baseada em valores extremos foi utilizada e os resultados mais uma vez superaram aqueles obtidos pelos AGs. Vale ressaltar que os AGAs mostraram ainda sua capacidade de adaptação quando (i) configurados levando-se em consideração apenas problemas pequenos e (ii) sendo aplicados a problemas maiores. Os resultados sugerem que mais investigações com outras técnicas desenvolvidas para AGAs devem ser realizadas, uma vez que os AGAs apresentam a possibilidade de reduzir o tempo de configuração desses algoritmos.

Houve também contribuição quanto à condução dos experimentos no contexto do FJSPs. Um primeiro experimento foi usado para selecionar parâmetros para as técnicas de busca usando um grupo de instâncias e, posteriormente, as melhores configurações escolhidas a partir desse primeiro caso foram utilizadas para analisar comparativamente o desempenho das técnicas num segundo conjunto de problemas. Deseja-se com isso (i) verificar se um AGA pode ter desempenho similar ao de um AG quando ambos são configurados especificamente para resolver um determinado tipo de problema, (ii) analisar a capacidade de generalização dos AGAs, (iii) estudar a aplicação dos diferentes mecanismos envolvidos no processo de adaptação de parâmetros quando resolvendo FJSPs, (iv) investigar a incorporação de adaptação também na busca local, e (v) determinar métodos de busca adequados para FJSPs sob diferentes perspectivas. O estudo realizado nessa dissertação utiliza instâncias geradas de acordo com o proposto por Taillard (1993), mas incorpora conteúdo para adaptá-las ao tipo de problema abordado aqui.

A dissertação está organizada em seis capítulos, sendo que o Capítulo 2 descreve os problemas de sequenciamento de processos e como ele se encaixa no contexto empresarial. No Capítulo 3 serão apresentados os conceitos das técnicas de busca utilizadas, como os AGs, os AGAs, as heurísticas construtivas e a busca local. Já o Capítulo 4 apresenta mais detalhes dos AGAs utilizados e as contribuições realizadas. Os experimentos computacionais são

descritos no Capítulo 5, juntamente com análises dos resultados. Finalmente, o Capítulo 6 apresenta a conclusão da dissertação.

2 O SEQUENCIAMENTO DE PROCESSOS

Nesse capítulo serão explicados os conceitos que determinam o que é o problema de sequenciamento de processos. A Seção 2.1 explica a importância do problema de sequenciamento e como ele se encaixa no contexto empresarial. Na Seção 2.2 encontram-se os principais tipos de problemas existentes e suas características, e são definidos o objetivo a ser adotado aqui, além da interpretação da solução no gráfico de Gantt.

2.1 O SEQUENCIAMENTO DE PROCESSOS NO AMBIENTE EMPRESARIAL

O sequenciamento de produção é uma das atividades operacionais que causam impacto nos objetivos de desempenho de uma empresa. É um problema prático e relevante em várias áreas como sistemas flexíveis de manufatura, planejamento da produção, logística, comunicação, entre outros. Sua importância e relevância fez com que pesquisadores de diversas áreas contribuíssem sobre diferentes perspectivas, passando desde métodos exatos a aplicações de heurísticas e meta-heurísticas.

No contexto empresarial, o sequenciamento de produção é realizado pela atividade de Programação da Produção (PP). Essa área possui como propósito garantir que os processos de produção ocorram eficaz e eficientemente e produzam os produtos e serviços requeridos pelos consumidores (Chambers; Johnston; Slack, 2002). Ainda segundo Chambers, Johnston & Slack (2002), a PP de uma empresa está encarregada de quatro atividades:

- Programação: Define o instante de tempo em que deve-se iniciar uma atividade;
- Carregamento: Dita a quantidade de trabalho que deve ser alocado num centro de trabalho;
- Sequenciamento: Estabelece a ordem em que as tarefas serão executadas; e
- Monitoramento e Controle: Verifica se as atividades estão sendo realizadas conforme o plano.

Pinedo (2012) define o sequenciamento como a alocação de um conjunto de tarefas em um conjunto de recursos durante determinado período de tempo, buscando otimizar um ou mais objetivos. Segundo esse autor, o sequenciamento de produção pode ser impactado pelas decisões de planejamento de médio e longo prazo, como pode ser visto na Figura 1. O processo tenta, então, otimizar o fluxo de produção baseado nos níveis de estoque, recursos disponíveis, e previsões de demanda. De fato, bons sequenciamentos possuem capacidade de reduzir os estoques, reduzir os custos e aumentar a confiabilidade das operações.

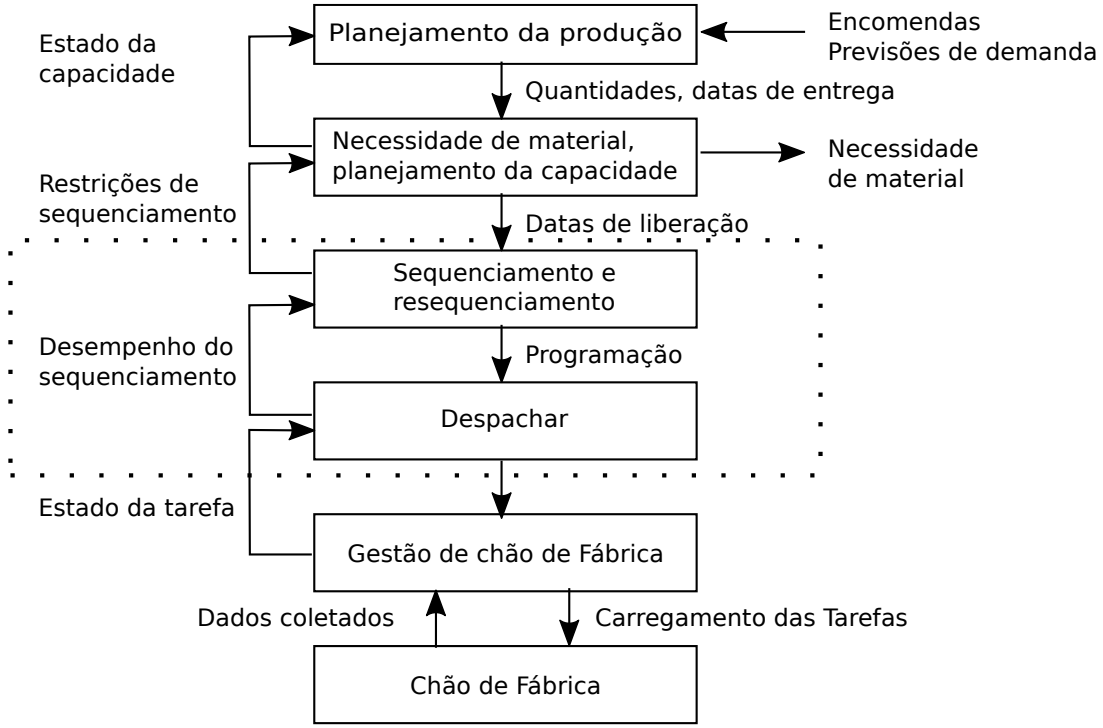

Figura 1 – Diagrama de fluxo de informações em um sistema de manufatura. (Adaptado de Pinedo (2012)).

Tanto as tarefas como os recursos podem ter diferentes formas dentro de uma organização, podendo ser aplicados tanto em empresas de bens de manufatura (uma indústria automobilística ao escolher em qual sequência os modelos de automóveis entrarão na linha de produção) quanto em empresas prestadoras de serviço (por exemplo, quando uma ferrovia deve sequenciar a passagem dos seus trens por diferentes estações ferroviárias para estabelecer uma grade horária).

2.2 O PROBLEMA DE SEQUENCIAMENTO DE PRODUÇÃO DO TIPO JOB-SHOP

O número de tarefas e de máquinas, e as restrições de sequenciamento, as quais indicam como as operações devem ser sequenciadas, definem os tipos de escalonamento de processos como sequenciamento (Hart; Ross; Corne, 2005): máquina única, máquinas paralelas, *flow-shop*, *job-shop* e *open-shop*. A notação introduzida em (Graham *et al.*, 1979) utiliza um padrão que contém 3 parâmetros para classificar esses problemas como $A|B|\Gamma$.

O parâmetro A indica o ambiente de máquina, ou seja, o fluxo em que as tarefas serão designadas às máquinas e aos estágios. Para o sequenciamento do tipo *Job-Shop* (J_m), existem m máquinas em série, na qual cada tarefa j deve ser operada. Os JSPs diferem do *flow-shop* quanto à rota tecnológica, isto é, a ordem das máquinas em que as tarefas são executadas. No ambiente *flow-shop* as tarefas são processas na mesma ordem tecnológica,

ou seja, passam pela máquina 0, depois são executadas na máquina 1 e assim por diante. Já nos JSPs essas rotas tecnológicas podem ser distintas. O cenário que será estudado aqui é o sequenciamento do tipo *Job-Shop* Flexível (FJ_c), no qual existem c estágios em série (ao invés das máquinas em série), e em cada estágio existe um número de máquinas em paralelo, sendo que essas máquinas podem realizar qualquer tarefa. Cada tarefa j deve então passar nos c estágios, e estas podem ter rotas tecnológicas diferentes.

O parâmetro B descreve seis possíveis características das tarefas, como recirculação, preempção, restrições de recursos, relações de precedência, datas de lançamento, entre outras. No modelo estudado aqui não foram contempladas nenhuma dessas restrições. Assim, este elemento será representado pela notação $\circ$.

Por fim, o parâmetro Γ representa o objetivo a ser minimizado, que pode ser minimizar o atraso total, minimizar a quantidade de tarefas atrasadas, minimizar o *makespan*, entre outros. O problema trabalhado nesta dissertação terá como objetivo minimizar o *makespan*, definido como:

$$C_{\max} = \max_{j \in \mathcal{J}} \{f_j\} \ , \tag{2.1}$$

em que f_j é o tempo no qual a tarefa j é concluída. O *makespan* é equivalente ao instante de tempo em que a última tarefa a deixar o sistema é finalizada. Para o seu cálculo, é necessário saber o tempo gasto por cada tarefa j em cada recurso. Portanto, pode-se denotar o problema resolvido aqui por $FJ_c| \circ |C_{\text{máx}}$.

Como mencionado anteriormente, no contexto do FJSP, as tarefas não têm a mesma rota tecnológica, mas possuem um tempo de processamento predefinido. Para problemas clássicos, como aqueles que serão considerados aqui, o número de operações é igual ao de estágios.

Então, no contexto FJSP deve-se processar um conjunto de tarefas $\mathcal{J}$ em um conjunto de estágios $\mathcal{C}$, que pode ser definido formalmente como (Ripon; Tsang; Kwong, 2007):

- cada estágio c possui um conjunto de máquinas $\mathcal{M}_c$ que podem processar todas as tarefas;
- cada tarefa $j \in \mathcal{J}$ deve ser processada por uma máquina $m \in \mathcal{M}_c$, a qual pertence a um estágio $c \in \mathcal{C}$;
- o processamento de cada tarefa j no estágio c é chamado de operação o_{jc};
- a operação o_{jc} requer o uso exclusivo de uma máquina m que pertence a um estágio c por um tempo t_{jc}, chamado de tempo de processamento;

– cada tarefa j possui uma rota tecnológica (sequência operacional) que pode ser distinta;

– cada operação o_{jc}, após iniciada, deve ser finalizada na mesma máquina e não deve ser interrompida até ser concluída;

– cada máquina j processa uma operação após a outra.

A Tabela 1 apresenta um exemplo dos dados de entrada utilizados para o modelo. Nesse exemplo, a tarefa 1 deve ser processada pela ordem dos estágios (1 – 2 – 3) com os respectivos tempos de duração (3 – 2 – 2). Além disso, cada instância fornece também o número de máquinas disponíveis para cada estágio, como pode ser observado na Tabela 2. Essas máquinas paralelas são consideradas idênticas, ou seja, capazes de processar as tarefas com a mesma duração, conforme descrito no modelo de Ripon, Tsang & Kwong (2007).

Tabela 1 – Exemplo de uma instância para um problema de três tarefas e três estágios.

j	c	t_{jc}	c	t_{jc}	c	t_{jc}
1	1	3	2	2	3	2
2	1	2	3	1	2	4
3	2	4	3	3	1	1

Tabela 2 – Um exemplo do número de máquinas em cada estágio para um problema que contém 10 estágios.

c	1	2	3
$\mathcal{M}_c$	2	1	1

A Figura 2 apresenta as rotas tecnológicas das tarefas apresentadas na Tabela 1. No FJSP as rotas tecnológicas das tarefas podem ser distintas. Nesse ambiente, a máquina 2 do estágio 1 também pode processar a tarefa 2, assim como a máquina 1 do estágio 1 também pode processar as tarefas 1 e 3.

Uma das formas de visualização de uma solução para o problema e também para o objetivo *makespan* é o Gráfico de Gantt, que foi desenvolvido por Henry Gantt em 1917, o qual apresenta um gráfico de barras horizontais, em que o eixo horizontal representa o tempo e o eixo vertical representa os vários estágios e/ou máquinas nos quais as tarefas devem passar (Pinedo, 2012). Geralmente cores são utilizadas para identificar as tarefas que estão dispostas nas barras.

Um gráfico de Gantt está disponível na Figura 3, ele apresenta uma possível solução para o exemplo fornecido na Tabela 1. O gráfico também permite a visualização do início

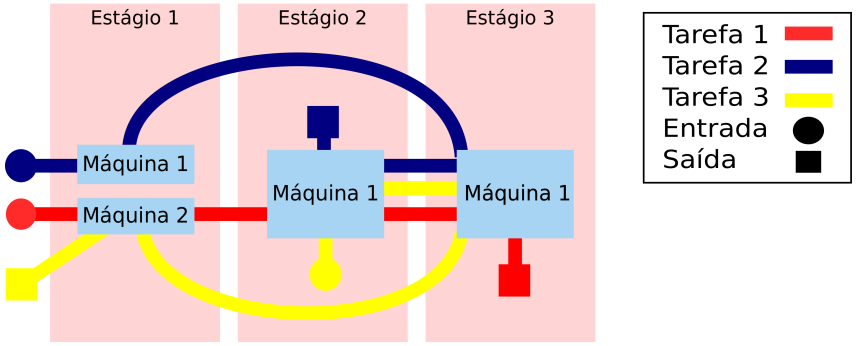

Figura 2 – No problema de Job-Shop Scheduling as rotas tecnológicas das tarefas podem variar.

e fim do processamento das tarefas em cada estágio. Na Figura 3 é possível verificar que as tarefas 1, 2 e 3 terminam, respectivamente, nos tempos 12, 8 e 8. O *makespan* é o tempo que o sistema completa todas as tarefas. Como a última a terminar é a tarefa 1, no tempo 12, o *makespan* também é 12.

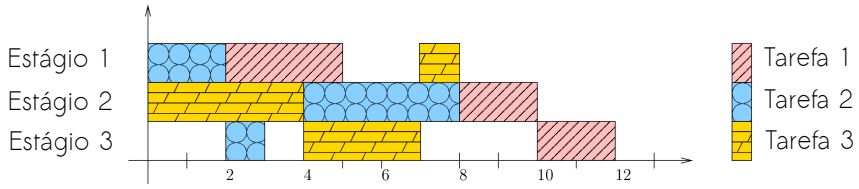

Figura 3 – Exemplo do Gráfico de Gantt para um problema de Job-Shop Flexível com três tarefas e três estágios (Adaptado de Acrogenesis)[1].

No contexto dessa dissertação, os problemas de sequenciamento que serão tratados são aqueles que precisam de várias execuções, uma vez que o fluxo de entrada e saída de tarefas do sistema é alto. Portanto, nesse cenário as técnicas de busca serão executadas diversas vezes, e retrata um cenário no qual o gestor não possuirá muito tempo para configurar as técnicas antes de utilizá-las.

[1] Adaptado de `https://acrogenesis.com/or-tools/documentation/user_manual/manual/ls/jobshop_def_data.html` em 15/01/2018

3 TÉCNICAS DE BUSCA

Neste capítulo serão apresentadas as técnicas de busca utilizadas aqui para solucionar o problema de sequenciamento do tipo job-shop flexível. A Seção 3.1 apresenta um Algoritmo Genético, sua representação e os operadores de movimento utilizados. A Seção 3.2 descreve a heurística construtiva usada para iniciar parte da população dos Algoritmos Genéticos. A Seção 3.3 detalha a busca local presente na estrutura dos algoritmos. A Seção 3.4 traz as técnicas adaptativas implantadas.

3.1 ALGORITMO GENÉTICO

Como citado nos trabalhos de Çaliş & Bulkan (2015) e Chaudhry & Khan (2016), os Algoritmos Genéticos (AGs) têm sido amplamente utilizados para solucionar problemas de sequenciamento. Os AGs foram propostos por Holland (1975) e a partir de 1980 começaram a ser aplicados a problemas de sequenciamento. Essa técnica possui características tais como: evoluir uma população de soluções (indivíduos), o que permite soluções diferentes e uma maior exploração do espaço de busca; e fácil hibridização com outras técnicas de busca, como acoplar uma heurística construtiva para iniciar parte da população e/ou utilizar uma busca local durante o processo de busca para melhorar sua convergência.

Em um AG, cada indivíduo (cromossomo) representa uma solução para o problema descrito. Entre as representações para FJSPs, essa dissertação segue a recomendação feita por Jorapur *et al.* (2014), que concluíram que a operação baseada em tarefas gera melhores resultados e ainda indica que o resultado desse tipo de codificação pode ser melhorado com a ajuda de outros métodos, como uma Busca Local (LS).

Na representação baseada em tarefas, um cromossomo representa uma permutação de tamanho igual ao número de tarefas. Então as tarefas são executadas de acordo com a ordem em que elas aparecem no cromossomo. A Figura 4 ilustra um indivíduo para essa representação que, ao ser decodificado, indica que a tarefa 9 é a primeira a ser executada, a tarefa 3 é a segunda, e assim por diante, até a tarefa 0 ser realizada.

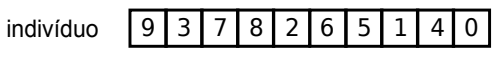

Figura 4 – Representação baseada em tarefas para um problema com 10 tarefas.

Após definir a representação, deve-se definir o modelo de seleção parental, os operadores movimento (recombinação e mutação) específicos para essa representação, e como os indivíduos serão selecionados para a próxima geração.

Uma etapa importante de um AG é a avaliação de indivíduo, nessa etapa cada indivíduo tem sua aptidão mensurada de acordo com o objetivo específico que o tomador de decisão procura. Nesse trabalho, o *makespan* para cada indivíduo será calculado na etapa de avaliação. A Figura 5 apresenta um indivíduo que possui 3 tarefas, de acordo com a representação mencionada. O *makespan* para esse indivíduo pode ser avaliado através da representação das atividades no gráfico de Gantt, como o demonstrado pela Figura 6. Para esse caso, o valor encontrado é 12 unidades, que é o tempo total gasto para o sistema processar todas as atividades.

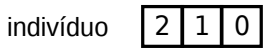

Figura 5 – Indivíduo com 3 tarefas.

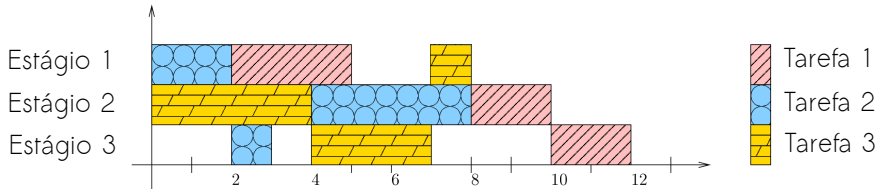

Figura 6 – Gráfico de Gantt para um indivíduo que possui três tarefas (Adaptado de Acrogenesis)[1].

Para selecionar os indivíduos que serão recombinados (seleção parental), será utilizado o torneio binário. Num torneio binário são selecionados aleatoriamente dois indivíduos. Em seguida, esses indivíduos são comparados através da aptidão e o mais apto é escolhido para ser o primeiro pai. O processo é repetido para selecionar o segundo pai e assim esses pais passam pelo processo de recombinação (Eiben; Smith *et al.*, 2003).

Nessa dissertação, os cinco operadores que seguem, dos quais três são de recombinação e dois são de mutação, foram escolhidos e todos eles estão disponíveis em (Werner, 2011):

– O *Order-Preserving one-point crossover* (OPX) seleciona aleatoriamente um ponto de corte. As tarefas são copiadas do pai 1 para o filho até esse ponto. Os genes vazios no filho são completados pelas tarefas ausentes de acordo com o posicionamento delas no pai 2, como demostrado na Figura 7.

– O *Linear-Order crossover* (LOX) usa dois pontos de corte. Conforme ilustrado na Figura 8, o segmento entre estes dois cortes é copiado do pai 1 para a mesma posição

[1] Adaptado de `https://acrogenesis.com/or-tools/documentation/user_manual/manual/ls/jobshop_def_data.html` em 15/01/2018

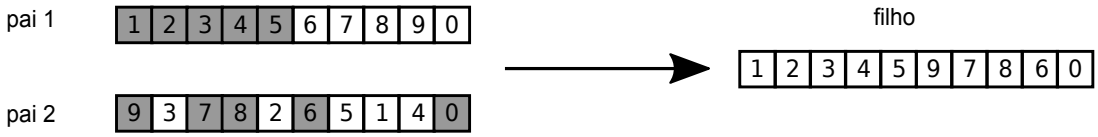

Figura 7 – Um exemplo da recombinação OPX.

no filho. Os genes vazios são completados a partir de elementos do pai 2 (somente as tarefas que estão ausentes na nova solução), mantendo a mesma ordem em que eles aparecem originalmente, isto é, mantendo suas posições relativas.

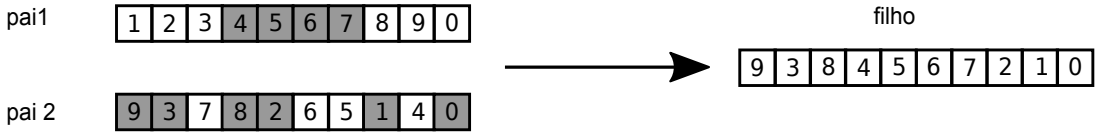

Figura 8 – Um exemplo da recombinação LOX.

– O *Partially mapped crossover* (PMX) seleciona dois pontos de corte e esse segmento é copiado do pai 1 para a mesma posição no filho-protótipo. O genes vazios são completados a partir do segundo pai, nas mesmas posições. O filho-protótipo é quase sempre infactível e, para corrigi-lo, as tarefas que aparecem duplicadas são mapeadas do pai 1 para o pai 2. Como pode ser visto na Figura 9, as tarefas 7 e 4 estão duplicadas no filho-protótipo. A duplicidade da tarefa 4 é solucionada ao mapeá-la para a tarefa 8 e substituí-la fora do segmento, como pode ser visualizado na seta de cor vermelha. A segunda repetição requer um passo a mais para solucioná-la. Destacada pela seta de cor azul, a tarefa 7 é mapeada para a tarefa 5, mas substituí-la pela tarefa 5 irá causar em uma nova duplicidade. Com um outro passo, a tarefa 5 é mapeada para a tarefa 2. Assim, a tarefa 7 (início do processo) pode ser substituída pela tarefa 2 (final do processo) e um filho factível é criado.

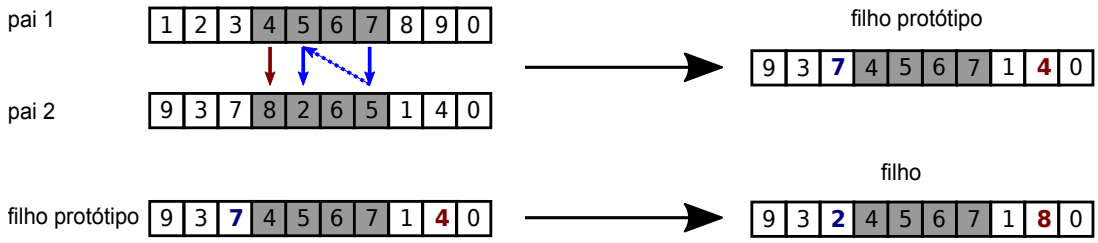

Figura 9 – Um exemplo da recombinação PMX.

– A *Shift Mutation* (SM) seleciona aleatoriamente um gene e muda para uma posição diferente e arbitrária. As outras tarefas são alteradas mantendo a ordem relativa. A Figura 10 representa a SM, na qual a tarefa 5 é alterada para a primeira posição no cromossomo.

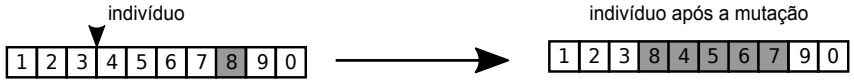

Figura 10 – Um exemplo da mutação SM.

– A *Pairwise interchange neighborhood mutation* (IM) seleciona arbitrariamente duas diferentes tarefas e suas respectivas posições são trocadas no cromossomo. A Figura 11 exemplifica esse tipo de mutação.

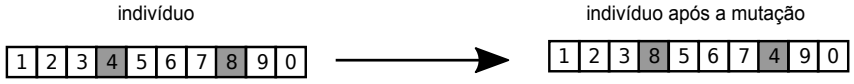

Figura 11 – Um exemplo da mutação IM.

Para a seleção dos sobreviventes será utilizado um mecanismo elitista, o qual impede que o indivíduo com melhor aptidão seja perdido. Para isso, o melhor indivíduo da geração atual é mantido, descartando-se o pior indivíduo gerado (Eiben; Smith *et al.*, 2003).

Um pseudocódigo de um AG é apresentado no Algoritmo 1.

Algoritmo 1: Algoritmo Genético

```
1 INICIALIZAR a população;
2 AVALIAR cada candidato;
3 enquanto A condição de término não é satisfeita faça
4     enquanto O número de indivíduos é menor do que o tamanho da população
        faça
5         SELECIONAR os pais;
6         RECOMBINAR o par de pais;
7         MUTAR cada novo indivíduo gerado;
8         AVALIAR os novos indivíduos;
9     SELECIONAR indivíduos para a nova geração;
```

3.2 HEURÍSTICA CONSTRUTIVA

Uma das vantagens dos AGs consiste em permitir sua combinação com outras técnicas. Para isso, nessa dissertação foi avaliada a possibilidade de iniciar parte da população (20%) através de uma heurística construtiva. Em problemas de sequenciamento, é comum utilizar heurísticas construtivas pois essas são capazes de chegar em soluções boas sem um grande esforço computacional. Segundo Ronconi (2004), uma heurística construtiva explora características específicas do problema que está sendo estudado. São

métodos simples e fáceis de implementar, visto que não possuem tantos parâmetros quando comparados às meta-heurísticas.

A heurística proposta por Nawaz, Enscore & Ham (1983) (NEH) tem sido uma das mais utilizadas quando se trata de problemas de sequenciamento de produção. Inicialmente, essa heurística foi desenvolvida para problemas de *flow-shop*, ambiente no qual as rotas tecnológicas das tarefas são iguais. Contudo, devido à representação adotada nesta dissertação, a NEH pode ser aplicada para o ambiente *job-shop* flexível.

Para iniciar a heurística é necessário somar o tempo de operação de cada tarefa. Posteriormente, as tarefas são ordenadas de maneira decrescente do tempo total de operação. Então, as duas primeiras tarefas (aquelas que possuem o maior tempo total de operação) são ordenadas para todas as sequências possíveis e é calculado o *makespan* para todas as sequências geradas. A sequência de menor *makespan* é adotada. Para isso, a ordem relativa das atividades dessa sequência é fixada e mantida até o fim do algoritmo. No próximo passo, realiza-se o mesmo procedimento com a próxima tarefa da lista (a terceira em maior tempo total de operação), ou seja, são mapeadas todas as sequência possíveis e aquela que tiver o menor *makespan* é escolhida, sendo essa ordem relativa das tarefas fixadas. O procedimento continua até que todas as tarefas sejam alocadas e o indivíduo possua todas as tarefas.

O Algoritmo 2 apresenta os passos dessa heurística e a Figura 12 ilustra o desenvolvimento do algoritmo para as tarefas apresentadas na Tabela 3. Observa-se que ao ordenar as tarefas de maneira decrescente do tempo de processamento total, as tarefas 2 e 1 são aquelas de maior tempo, e por isso são as escolhidas para iniciar o algoritmo. A ordem das tarefas 2-1 tem menor *makespan* do que a ordem das tarefas 1-2, e por isso é tomado como a melhor ordem relativa. Posteriormente adiciona-se a tarefa 0 nas diferentes posições. Como o ordenamento 2-1-0 possui o menor *makespan* essa é a solução de acordo com a heurística NEH.

Algoritmo 2: Heurística NEH

```
1 CALCULAR, para cada tarefa, a duração total de todas as operações;
2 ORDENAR as tarefas na ordem decrescente de seus tempos de processamento;
3 SELECIONAR as duas primeiras tarefas ordenadas, e então calcular o makespan
    para todas as possíveis sequências;
4 ESCOLHER a alternativa que minimiza o makespan, e adotar a ordem relativa
    dessas tarefas até o fim do algoritmo;
5 enquanto Há tarefas a serem selecionadas faça
6 |   SELECIONAR a nova tarefa e avaliar o makespan para todas as sequências
  |     possíveis;
7 |   ESCOLHER a alternativa que gera o menor makespan e sempre manter a ordem
  |     relativa dessas tarefas;
```

Tabela 3 – Exemplo de entrada para desenvolvimento da heurística NEH.

Tareja j	Tempo de Processamento Total
0	4
1	5
2	6

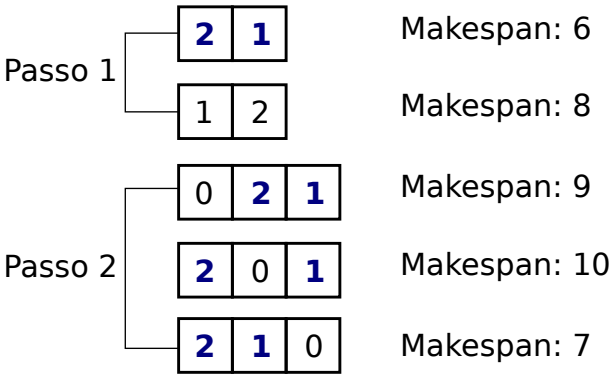

Figura 12 – Um exemplo do desenvolvimento da heurística NEH

3.3 BUSCA LOCAL

Como mencionado em (Jorapur *et al.*, 2014), uma busca local pode fazer um Algoritmo Genético convergir mais rápido. Para isso, foi implementada a heurística proposta em (Ruiz; Stützle, 2007) após os operadores de mutação.

O algoritmo proposto por Ruiz & Stützle (2007) gera uma série de soluções via iterações de uma heurística construtiva composta por duas fases. A primeira, chamada de destruição, remove um percentual de tarefas (genes) de uma solução completa (cromossomo). Na fase destrutiva dessa heurística, um número *des* de tarefas são removidas aleatoriamente de um indivíduo. Essas *des* tarefas são alocadas numa lista, na mesma ordem em que foram removidas. Após as *des* tarefas terem sido removidas do sequenciamento, a fase destrutiva se encerra, e a fase construtiva é iniciada, em que as tarefas removidas são recolocadas na solução.

Após o término da primeira fase, duas listas são formadas. A primeira lista contém as tarefas que não foram removidas, já a segunda lista contém as tarefas que foram removidas. Para completar o cromossomo novamente, as tarefas que foram removidas serão novamente adicionadas de acordo com a ordem que estão armazenadas na segunda lista. Para isso, a primeira tarefa dessa segunda lista é adicionada na primeira lista em todas as sequências possíveis, mantendo a posição relativa das demais tarefas que já pertenciam a essa primeira lista. A sequência que gera o menor *makespan* (melhor solução) é escolhida e tem a ordem relativa dessas atividades fixada. O processo de adição

Algoritmo 3: Busca Local

```
1 REMOVER aleatoriamente des tarefas do indivíduo;
2 ARMAZENAR essas tarefas removidas no π⃗;
3 enquanto Há tarefas em π⃗ faça
4 |   SELECIONAR a nova tarefa e avaliar o makespan para todas as sequências
  |     possíveis;
5 |   ESCOLHER a alternativa que gera o menor makespan e sempre manter a ordem
  |     relativa dessas tarefas;
```

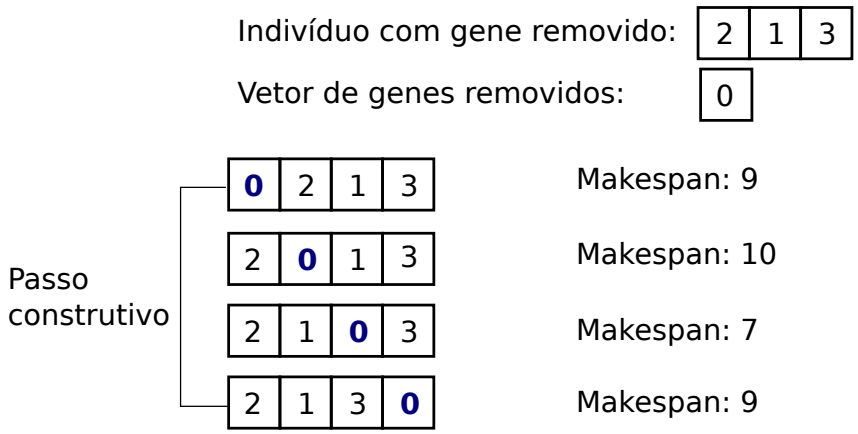

Figura 13 – Fase construtiva da busca local utilizada no algoritmo.

é repetido para as demais tarefas que foram retiradas até que o indivíduo possua todas as tarefas novamente. O Algoritmo 3 apresenta os passos dessa heurística e a Figura 13 apresenta a etapa de construção do algoritmo.

É importante ressaltar que quando a busca local é utilizada, agrega-se o número total de avaliações realizadas ao critério de parada.

3.4 ALGORITMOS GENÉTICOS ADAPTATIVOS

Os estudos de Eiben, Hinterding & Michalewicz (1999) evidenciaram o fato de que os parâmetros de um AG devem ser corretamente configurados para melhorar sua eficiência. Os mesmos autores destacam que configurar esses parâmetros é uma tarefa demorada, pois ao configurar um AG simples, como o descrito na Tabela 4, resulta-se em 162 possibilidades disponíveis. Além disso, Karafotias, Hoogendoorn & Eiben (2015) indicam que os melhores parâmetros podem alterar durante o processo de busca, e Srinivas & Patnaik (1994) complementam ao dizer que os parâmetros devem ser ajustados tanto para evitar que o algoritmo fique preso num ótimo local, quanto para aumentar a taxa de convergência. Por fim, Krempser, Fialho & Barbosa (2012) citam que a configuração de um algoritmo é dependente do problema.

Tabela 4 – Combinações de Parâmetros Possíveis para um Algoritmo Genético

Parâmetros	Valores		
Tamanho da População	50	76	100
Taxa de recombinação	0,7	0,8	0,9
Operador de recombinação	PMX	LOX	OPX
Taxa de mutação	0,3	0,4	0,5
Operador de mutação	SM	IM	–

Para contornar o problema de configurar um AG e evitar outros problemas que são ocasionados pela má configuração do AG, foram adotados nessa dissertação os esquemas adaptativos para realizar o ajuste automático dos parâmetros. Os Algoritmos Genéticos Adaptativos (AGA) são algoritmos que monitoram a qualidade das soluções geradas ao utilizar certos parâmetros dos algoritmos através de *feedbacks* da população (Eiben; Hinterding; Michalewicz, 1999; Aleti; Moser, 2016). Como exemplo, pode-se citar um componente de mutação que, ao melhorar um indivíduo, terá a sua probabilidade de ser utilizado aumentada. O pseudocódigo de um AGA pode ser visto no Algoritmo 4.

Algoritmo 4: Algoritmo Genético Adaptativo.

```
1  INICIALIZAR a população;
2  AVALIAR cada candidato;
3  enquanto A condição de término não é satisfeita faça
4      enquanto O número de indivíduos é menor do que o tamanho da população faça
5          SELECIONAR os operadores de movimento e as taxas de ocorrência que serão utilizados ;
6          SELECIONAR os pais;
7          RECOMBINAR o par de pais;
8          MUTAR cada novo indivíduo gerado;
9          AVALIAR os novos indivíduos;
10         RECOMPENSAR os operadores de movimento e as taxas de ocorrência de acordo baseado no sucesso desses componentes ;
11     SELECIONAR indivíduos que formarão a nova geração;
```

Através de uma comparação entre o Algoritmo 1 e o Algoritmo 4, pode-se perceber que dois passos são acrescentados neste último; as instruções adicionais são aquelas nas linhas 5 e 10. O primeiro passo utiliza um mecanismo de roleta para sortear aleatoriamente os componentes de recombinação, de mutação, e as taxas de ocorrência que serão utilizados. O segundo passo altera a probabilidades de utilização dos operadores de movimento e das taxas de ocorrência, através de diferentes processos que devem ser avaliados durante a construção de um AGA, denominados de Atribuição de Crédito e a Técnicas de Seleção de Operadores, que serão explicados nas próximas seções.

3.4.1 **Atribuição de Crédito**

A atribuição de crédito é necessária pois é nela que a técnica de adaptação define: o cálculo do impacto impacto esperado de determinado operador de movimento ou taxa de ocorrência, quais operadores de movimento e taxas de ocorrência devem receber crédito, e a forma como os créditos são designados aos operadores de movimento e as taxas de ocorrência. Fialho (2011) lista algumas possibilidades para cada um desses pontos.

Para mensurar o impacto dos parâmetros, questiona-se o que deve ser considerado uma resposta positiva e o que deve ser considerado uma resposta negativa. Os métodos comparam a aptidão do indivíduo gerado com: a aptidão dos pais, a aptidão da população, a mediana ou algum quartil da aptidão da população, entre outros (Fialho, 2011). Nesta dissertação, para os operadores e as taxas de recombinação, o sucesso é definido quando a aptidão do indivíduo gerado supera a dos ancestrais diretos. Já para os operadores, para as taxas de mutação e para as taxas de busca local, o sucesso é atribuído quando o desempenho do indivíduo gerado supera o desempenho médio da população.

Quando torna-se necessário definir quais operadores de movimento e taxas de ocorrência devem receber crédito, Fialho (2011) levanta a discussão se somente o último operador que modificou o indivíduo deve ser recompensado, ou se o caminho que o trouxe até esse formato também deve ser recompensado. Por exemplo, recompensar os operadores que geraram o ancestral. Contudo, nesta dissertação limita-se a recompensar os últimos operadores de movimento cada tipo, isto é, somente os operadores de recombinação que atuam sobre a última geração serão recompensados. O que exclui os operadores de recombinação que formaram os ancestrais desses indivíduos.

Quanto às diferentes formas de designar o crédito, busca-se minimizar os efeitos da aleatoriedade de um Algoritmo Genético. Assim, alguns métodos foram propostos para minimizar os problemas gerados por sorteios que podem beneficiar ou prejudicar um operador. Definindo-se a recompensa recebida por um operador i como $\mathcal{R}(i)$ e a aptidão de um indivíduo x como $\mathcal{F}(x)$, pode-se então destacar as seguintes formas de designação de crédito:

- A Recompensa Instantânea (RI) é a forma pela qual o operador é recompensado somente pela última aplicação. Normalmente são contabilizados somente os valores positivos de recompensa, isto é:

$$\mathcal{R}(i) = \max(\mathcal{F}(i(x)) - \mathcal{F}(x), 0) \ ; \tag{3.1}$$

- No valor EXtremo (EX), que foi proposto em (Fialho *et al.*, 2008), existe uma janela temporal de tamanho W em que são armazenadas as últimas utilizações do operador

i, seguindo a política Primeiro que Entra, Primeiro que Sai. A partir da RI que o operador i receberia no tempo t, então a recompensa utilizando o EX para esse operador será:

$$\mathcal{R}(i) = \max\{RI(t_i), i = 1 \cdots W\} \ ; \tag{3.2}$$

– A recompensa pode ser atribuída de acordo com a qualidade dos parâmetros inferida através dos métodos de aprendizado de máquina (Aleti; Moser, 2016). Por exemplo, em (Aleti; Moser, 2011) foi assumida a linearidade por partes do desempenho dos operadores, então a partir dos dois últimos passos, interpola-se o comportamento do operador e o extrapola para prever o comportamento do operador no próximo passo, pela seguinte equação:

$$\text{Previsão de sucesso} = a_0 + a_1 \times (t + 1) \ , \tag{3.3}$$

onde a variável t é a variável independente, a_1 é a inclinação da reta e a_0 é o valor que intercepta o eixo das ordenadas;

– A recompensa pode ser atribuída de acordo com esquemas de ranqueamento, como o proposto por Fialho, Schoenauer & Sebag (2010). Para isso, tem-se uma janela temporal de tamanho W, que armazena os impactos recentes dos operadores, os quais são ranqueados em ordem decrescente dos valores de melhoria da aptidão nela armazenados. A posição r da janela ranqueada recebe o valor $(W - r)$. Então, um fator de decaimento $D \in [0, 1]$ é aplicado sobre esses valores. O método de assimilação de crédito *Sum of Ranks* (SR) credita os operadores de acordo com a soma da classificação das suas recompensas recebidas normalizada pela soma das recompensas recebidas por todos os operadores. Portanto, a soma da atribuição é igual a 1. Sendo K o número de operadores, a recompensa do operador i no tempo t é definida como:

$$SR_{i,t} = \frac{\sum_{op_r=i} D^r(W - r)}{\sum_{r=1}^{W} D^r(W - r)} \ , \tag{3.4}$$

no qual o hiperparâmetro D define o nível de distorção da distribuição da classificação.

Todos esses esquemas de recompensa apresentados serão avaliados nesta dissertação.

3.4.2 **Técnica de Seleção de Operadores**

Uma vez definido como o crédito será computado para os operadores, é preciso escolher qual técnica de seleção será aplicada. Essas técnicas armazenam a qualidade estimada de um operador de movimentos e das taxas de ocorrência e a aplica para aumentar ou reduzir a probabilidade de utilização desse com o intuito de escolher qual operador de movimento e taxa de ocorrência são melhores.

Similar ao problema de escolher os operadores movimento e as taxas de ocorrência é o problema disponível na literatura denominado de *Multi-Armed Bandit* (MAB), no qual um conjunto de recursos deve ser empregado em alternativas concorrentes de modo a maximizar o ganho esperado.

O exemplo clássico desse tipo de problema é um apostador que está numa fileira de máquinas de caça-niqueis e deve escolher em qual máquina jogar, quantas vezes deve jogar, em qual ordem deve jogar nessas máquinas, e se deve continuar na mesma máquina ou se deve escolher outra. Para isso, cada máquina fornece uma recompensa aleatória baseada numa distribuição de probabilidades para aquela máquina específica. Portanto, o apostador deve maximizar a soma das recompensas recebidas, lindando, a cada tentativa de usar uma máquina, com o conflito entre utilizar a máquina que o remunera melhor ou obter mais informações da recompensa das outras máquinas.

No entanto, o problema de escolher o melhor operador de movimento e as melhores taxas de ocorrência pode ser ainda mais desafiador, uma vez que o estado é não estacionário, ou seja, as distribuições de probabilidade que geram as recompensas dos operadores de movimento e das taxas de ocorrência dos operadores variam com o tempo. Assim, as técnicas que foram introduzidas no contexto do MAB foram adaptadas para lidar com o estado não estacionário, e de modo a definir as probabilidade de seleção dos operadores e da sua taxa de ocorrência.

As técnicas já concebidas e empregadas nessa dissertação serão explicadas nas próximas seções.

3.4.2.1 *Adaptive Pursuit*

Thierens (2005) estudou os algoritmos de rápida convergência para *learning automata* (um tipo de aprendizado de máquina) e adaptou esses algoritmos para o estado não estacionário. Segundo Thierens (2005), se as distribuições de recompensas ficam estacionárias por algum tempo, o *Adaptive Pursuit* (AP) converge mais rápido e acurado para o valor do parâmetro (operador de movimento ou taxa de ocorrência) com as melhores distribuições de recompensas, quando comparado com *Probability Matching* (método tradicional para acumular recompensas definido em (Goldberg, 1990)), aumentando probabilidade desse melhor valor de parâmetro ser utilizado. Além disso, o método ainda mantém-se sensível às alterações nas distribuições de recompensa.

A partir da taxa de adaptação α e a recompensa do operador $\mathcal{R}_i$, a qualidade de um operador i no tempo $t+1$ do método AP é definida como:

$$\mathcal{Q}_i(t+1) = \mathcal{Q}_i(t) + \alpha[\mathcal{R}_i(t) - \mathcal{Q}_i(t)] \ . \tag{3.5}$$

O método AP utiliza limites inferior e superior para as probabilidades dos operadores. Dessa forma, possibilita que um operador que não é bom no início do processo de busca possa ser selecionado em outro estágio, caso esse se torne uma escolha boa. Definindo então o número de componentes como K, a taxa de aprendizado como β, a probabilidade mínima como $\mathcal{P}_{\min}$, a probabilidade máxima como $\mathcal{P}_{\max} = [1 - (K-1)\mathcal{P}_{\min}$, e o melhor operador como $i^* = \max_i[\mathcal{Q}_i(t+1)]$, então a probabilidade do melhor componente é calculada como:

$$\mathcal{P}_{i^*}(t+1) = \mathcal{P}_{i^*}(t) + \beta[\mathcal{P}_{\max} - \mathcal{P}_{i^*}(t)] \ , \tag{3.6}$$

e a probabilidade para os outros operadores é calculada como:

$$\mathcal{P}_i(t+1) = \mathcal{P}_i(t) + \beta[\mathcal{P}_{\min} - \mathcal{P}_i(t)] \ . \tag{3.7}$$

O pseudocódigo do AP está disponível no Algoritmo 5.

Algoritmo 5: Algoritmo *Adaptive Pursuit*

```
   entrada: P,Q,K,P_min,α,β
1  P_máx ← 1 − (K − 1)P_min;
2  para i ← 1 até K faça
3      P_i(t_0) ← 1/K; Q_i(t_0) ← 1,0;
4  enquanto Não terminar faça
5      i ← SelecionaProporcionalmenteOperador(P);
6      R_i(t) ← RecebeRecompensa(i);
7      Q_i(t+1) = Q_i(t) + α[R_i − Q_i(t)];
8      i* ← max(Q_i(t+1));
9      P_i*(t+1) = P_i*(t) + β[P_max − P_i*(t)];
10     enquanto i ← 1 até K faça
11         se i ≠ i* então
12             P_i(t+1) = P_i(t) + β[P_min − P_i(t)];
```

3.4.2.2 *Dynamic Multi-Armed Bandit*

A técnica do Dynamic Multi-Armed Bandit (DMAB) foi proposta por da Costa *et al.* (2008) para adaptar o Multi-Armed Bandit (MAB) para problemas não estacionários. Para o problema de ajustar os parâmetros de um AG, o MAB busca utilizar o melhor operador até que ele deixe de trazer boas recompensas e assim passe a explorar um novo cenário. O modelo empregado utiliza o algoritmo *Upper Confidence Bound* (UCB1), proposto por Auer, Cesa-Bianchi & Fischer (2002), o qual seleciona em cada tempo t o componente $i \in \{1, \ldots, K\}$ de acordo com a seguinte equação:

$$UCB1 = \max_{i=1\ldots K} \left(\hat{q}_{i,t} + C\sqrt{\frac{2log\sum_K n_{k,t}}{n_{i,t}}} \right) \ , \tag{3.8}$$

no qual $\hat{q}_{i,t}$ é a qualidade estimada empiricamente, e $n_{i,t}$ é o número de utilizações do parâmetro i. C é um fator de escala que realiza um *trade-off* entre explotação, que é o primeiro termo da Equação (3.8) que favorece a utilização dos bons operadores, e exploração, o segundo termo da Equação (3.8) que favorece a utilização de outros operadores.

Para aplicar essa técnica no contexto dinâmico, sendo esse o problema de ajustar os parâmetros de um AG, da Costa *et al.* (2008) utilizaram as ideias do MAB adicionadas de um teste estatístico chamado de Page-Hinkley (PH). O PH possui a capacidade de detectar alterações no ambiente (distribuições de recompensa) e, quando detecta, o teste reinicia o MAB.

$\bar{\mathcal{R}}_l$ é definido como a média da recompensa do operador até a utilização l, e_l como: $\mathcal{R}_l - \bar{\mathcal{R}} + \delta$, sendo que δ é um parâmetro de tolerância Então, o PH monitora a diferença entre M_t, que é o valor máximo de m_t para $l = 1, \cdots, t$ e uma variável aleatória m_t, que é o somatório de e_l. Quando o valor é maior do que o gatilho λ especificado pelo usuário, o PH é acionado e reinicia o MAB. A Equação (3.9) apresenta as equações do PH. O parâmetro λ controla o *trade-off* entre falso alarme e a não detecção de alterações na distribuição, e o parâmetro δ torna o teste robusto quando lida com ambientes que são alterados vagarosamente.

$$
\begin{aligned}
&\bar{\mathcal{R}}_l = \frac{1}{l}\sum_{i=1}^{l} r_i \ , \\
&m_t = \sum_{l=1}^{t}(\mathcal{R}_l - \bar{\mathcal{R}}_l + \delta) \ , \\
&M_t = \text{máx}\{m_l, l = 1 \cdots t\} \ , \\
&PH_t = M_t - m_t \ , \\
&Retornar(PH_t > \lambda) \ .
\end{aligned}
\tag{3.9}
$$

Portanto, o DMAB pode ser considerado o MAB com adição do teste PH, sendo que o último é utilizado para reiniciar o MAB quando são detectadas alterações no ambiente. O Algoritmo 6 apresenta o pseudocódigo do DMAB.

3.4.2.3 *Sliding Multi-Armed Bandit*

Fialho *et al.* (2010) apresenta duas desvantagens do DMAB. A primeira consiste no fato de que o teste PH somente pode ser disparado em casos de alterações abruptas, sendo que a recompensa de um operador geralmente cai lentamente, tornando difícil calibrar o gatilho λ. A segunda desvantagem é que, após acionado o gatilho, toda a memória do MAB é perdida e os operadores precisam iniciar um novo processo.

Algoritmo 6: Algoritmo *Dynamic Multi-Armed Bandit*

entrada: $K, C, \lambda, \delta = 0{,}15$

para $i \leftarrow 1$ *até* K **faça**
  $n_i \leftarrow 0$;
  $\hat{q}_i \leftarrow 0{,}0$;
  $m_i \leftarrow M_i \leftarrow 0{,}0$;
enquanto *Não terminar* **faça**
  se *um ou mais operadores não foram aplicados* **então**
    $op \leftarrow$ selecionado uniformemente entre os operadores não aplicados;
  senão
    $op \leftarrow \max_{i=1\cdots K} \left(\hat{q}_{i,t} + C\sqrt{\frac{2log\sum_K n_{k,t}}{n_{i,t}}} \right)$;
  Operador *op* é aplicado, impactando o processo de busca;
  $\mathcal{R}_{op} \leftarrow$ RecebeRecompensa(op);
  $n_{op} \leftarrow n_{op} + 1$;
  $\hat{q}_{op} \leftarrow \left(\frac{(n_{op} - 1) \times \hat{q}_{op} + \mathcal{R}_{op}}{n_{op}} \right)$;
  $m_{op} \leftarrow m_{op} + (\mathcal{R}_{op} - \hat{q}_o p + \delta)$;
  se $|m_{op}| > M_{op}$ **então**
    $M_{op} \leftarrow |m_{op}|$;
  senão se $M_{op} - |m_{op}| > \lambda$ **então**
    Reiniciar as variáveis do MAB e do PH $(n, \hat{q}, m, M)$;

Por isso, Fialho *et al.* (2010) propuseram um novo operador de seleção *bandit-based*, chamado de *Sliding Multi-Armed Bandit* (SLMAB). A ideia por trás desse método é seguir a dinâmica do ambiente sem ser muito sensível e algumas vezes controverso, como o mecanismo de reinício utilizado no DMAB.

Portanto, como regra de atualização, o SLMAB utiliza o número de passos decorridos desde a última aplicação em que o operador i foi utilizado (t_i). Para preservar o *trade-off* entre exploração ($\hat{q}$) e explotação (n), o SLMAB mantém um contador $n_{i,t}$ que reflete a frequência de aplicação do operador i até o passo t. Considerando uma janela tempo de tamanho W, o SLMAB utiliza as seguintes equações em seu escopo:

$$\left.\begin{aligned} \hat{q}_{i,t+1} &= \hat{q}_{i,t} \times \frac{W}{W + (t - t_i)} + \mathcal{R}_{i,t} \times \frac{1}{n_{i,t} + 1} \ , \\ n_{i,t+1} &= n_{i,t} \times \left(\frac{W}{W + (t - t_i)} + \frac{1}{n_{i,t} + 1} \right) \ . \end{aligned}\right\} \tag{3.10}$$

Desse modo, o pseudocódigo para o SLMAB pode ser visto no Algoritmo 7.

Algoritmo 7: Algoritmo *Sliding Multi-Armed Bandit*

entrada: K, C, W

para $i \leftarrow 1$ *até* K **faça**
  $n_i \leftarrow 0$;
  $\hat{q}_i \leftarrow 0{,}0$;
  $last_i \leftarrow 0$;
enquanto *Não terminar* **faça**
  se *um ou mais operadores não foram aplicados* **então**
    $op \leftarrow$ selecionado uniformemente entre os operadores não aplicados;
  senão
    $op \leftarrow \max_{i=1\cdots K} \left(\hat{q}_{i,t} + C\sqrt{\frac{2log\sum_K n_{k,t}}{n_{i,t}}} \right)$;
  Operador *op* é aplicado, impactando o processo de busca;
  $\mathcal{R}_{op} \leftarrow$ RecebeRecompensa(op);
  $\hat{q}_{op} \leftarrow \hat{q}_{op} \times \left(\frac{W}{W + (t - t_{\text{op}})} \right) + \mathcal{R}_{op} \times \left(\frac{1}{n_{op} + 1} \right)$;
  $n_{op} \leftarrow n_{op} \times \left(\frac{W}{W + (t - t_{op})} + \frac{1}{n_{op} + 1} \right)$;
  $t_{op} \leftarrow t$;
  $t \leftarrow t + 1$;

3.4.2.4 *Rank-based Multi-Armed Bandit*

As maiores críticas aos esquemas adaptativos baseados no MAB dizem respeito à sensibilidade de seus hiperparâmetros que devem ser ajustados. Além disso, essas técnicas são baseadas em estatísticas aplicadas às recompensas adquiridas após a utilização de determinados operadores. Ao passo em que esses valores são utilizados, a configuração do método adaptativo pode tornar-se dependente do problema, uma vez que os valores da aptidão podem variar, em escala, de um problema para outro (Fialho, 2011).

Esses problemas motivaram Fialho, Schoenauer & Sebag (2010) a desenvolver um método baseado no ranqueamento das soluções, denominado *Rank-based Multi-Armed Bandit* (RMAB). A ideia dessa proposta é que as melhores recompensas devem influenciar mais no crédito atribuído a cada operador. Contudo, os resultados preliminares mostraram que inserir essa forma de crédito não se ajustava bem às técnicas baseadas no MAB (Fialho; Schoenauer; Sebag, 2010). Segundo Fialho (2011), os resultados pioraram uma vez que há duas camadas de estatísticas que diluem características interessantes do método proposto: uma gerada pelo SR e a outra do próprio MAB. No entanto, as saídas do SR podem ser utilizadas no termo $\hat{q}$ da Equação 3.8. Para garantir um nível mínimo de exploração, o termo n do MAB foi modificado para contabilizar o número de vezes em que cada operador aparece na janela temporal. O pseudocódigo do RMAB pode ser visto no Algoritmo 8.

Algoritmo 8: Algoritmo *Rank-based Multi-Armed Bandit*

entrada: K, C

para $i \leftarrow 1$ *até* K **faça**
  $n_i \leftarrow 0$;
  $\hat{q}_i \leftarrow 0{,}0$;
enquanto *Não terminar* **faça**
  se *um ou mais operadores não foram aplicados* **então**
    $op \leftarrow$ selecionado uniformemente entre os operadores não aplicados;
  senão
    $op \leftarrow \max_{i=1\cdots K} \left(\hat{q}_{i,t} + C\sqrt{\frac{2log\sum_K n_{k,t}}{n_{i,t}}} \right)$;
  Operador *op* é aplicado, impactando o processo de busca;
  para $i \leftarrow$ *até* K **faça**
    $\hat{q}_i \leftarrow RecebeRecompensa(i)$;
    $n_i \leftarrow RecebeTempo(JanelaTemporal, i)$;

3.4.2.5 *Predictive Parameter Control*

Dentre as alternativas disponíveis para adaptar parâmetros, Aleti & Moser (2011) propuseram uma abordagem baseada em métodos preditivos. Por isso, o *Predictive Parameter Control* (PPC) utiliza a medida de qualidade de desempenho anteriores na forma de uma série temporal para ajustar os parâmetros durante o processo de busca.

A cada geração o método preditivo atualiza a aptidão média da população, o total de vezes que determinado parâmetro foi aplicado, e o total de vezes que esse operador foi aplicado com sucesso (o sucesso é contabilizado quando a aptidão do indivíduo gerado supera a aptidão média da população). Assim, o algoritmo calcula a taxa de sucesso de determinado operador como:

$$\text{Taxa de sucesso} = \frac{u_i^s}{u_i} \ , \tag{3.11}$$

em que u_i é a quantidade de vezes que o parâmetro i é utilizado, e u_i^s é o número de vezes que o parâmetro i é utilizado com sucesso.

A partir das taxas de sucesso das duas últimas gerações, o PPC extrapola a qualidade de determinado operador de movimento ou taxa de ocorrência para a próxima geração. Os algoritmos preditivos foram subdivididos em dois diferentes algoritmos: o primeiro é o PPC, que possui operadores de movimento e taxas de ocorrência discretos; o segundo algoritmo é o CPPC no qual a taxa de ocorrência é selecionada através de um intervalo contínuo. Sendo K o número de elementos ou de intervalos que um parâmetro pode receber, pode-se verificar o pseudocódigo dos algoritmos preditivos no Algoritmo 9.

Algoritmo 9: Algoritmo *Predictive Parameter Control*

```
para i ← 1 até K faça
    P(i) ← 1/K;
    u_i(t_0) ← 0;
    u_i^s(t_0) ← 0;
enquanto Não terminar faça
    i ← SelecionarProporcionalOperador(P);
    se Está nas duas primeiras iterações então
        se Houver sucesso na criação do indivíduo então
            u_i^s(t) ← u_i^s(t − 1) + 1;
        u_i(t) ← u_i(t − 1) + 1;
        CalculaTaxadeSucesso(u_i^s, u_i);
    senão
        se Houver sucesso na criação do indivíduo então
            u_i^s(t) ← u_i^s(t − 1) + 1;
        u_i(t) ← u_i(t − 1) + 1;
        CalculaTaxadeSucesso(u_i^s, u_i);
        InfereSucesso(TaxadeSucesso);
        EstimaSucessonoPróximoPasso(Regressão);
```

4 ALGORITMOS GENÉTICOS ADAPTATIVOS PARA SOLUCIONAR PROBLEMAS DE SEQUENCIAMENTO DO TIPO JOB-SHOP

O estudo desenvolvido nesta dissertação envolve o projeto e a aplicação de algoritmos adaptativos para solucionar FJSPs. Os AGs possuem algumas vantagens frente a outros métodos, no entanto a configuração de seus parâmetros não é uma tarefa rápida, podendo ainda comprometer seus resultados. Portanto, os AGAs foram desenvolvidos a fim de solucionar esses problemas.

Como descrito no Capítulo 3, diferentes AGAs estão disponíveis na literatura, tanto no sentido de diversificar seus tipos de atribuição de crédito (Thierens, 2005; Fialho *et al.*, 2008; Aleti; Moser, 2016) quanto os seus operadores de seleção (Thierens, 2005; da Costa *et al.*, 2008; Fialho; Schoenauer; Sebag, 2010; Fialho *et al.*, 2010; Aleti; Moser, 2011).

Esses estudos têm se mostrado eficientes para guiar os processos decisórios dos AGAs. Contudo, para os problemas de sequenciamento de produção foram utilizados métodos simples e não foram realizadas comparações entre as diferentes técnicas adaptativas. Assim sendo, o objetivo dessa dissertação de mestrado é confrontar esses esquemas adaptativos para solucionar os FJSPs. Além disso, foram desenvolvidas duas propostas para Atribuição de Crédito e uma Técnica de Seleção de Operadores.

4.1 CONTRIBUIÇÃO NA ATRIBUIÇÃO DE CRÉDITO

Como explicado na Seção 3.4.1, Aleti & Moser (2011) assumiram que a qualidade observada nas soluções candidatas geradas pelos operadores é linear por partes. Embora esse comportamento possa ser observado na Figura 14, deseja-se verificar outros cenários, nos quais o comportamento dos operadores possa ser melhor inferido com base em outras formas de interpolação.

A partir disso, pressupõe-se que o comportamento dos parâmetros possa ser inferido utilizando uma função exponencial, que é dada pela equação:

$$\text{Previsão de sucesso} = a_0 e^{a_1(t)} \ , \tag{4.1}$$

onde a_0 e a_1 são coeficientes do modelo, e o termo a_1 denota se a função é crescente (quando $a_1 > 0$) ou decrescente (quando $a_1 < 0$). Para interpolar a função serão utilizados a taxa de sucesso de um determinado operador nas duas últimas gerações. Para isso, a função exponencial será linearizada. Finalmente, pode-se tentar prever a taxa de sucesso futuro desses operadores ao extrapolar o modelo obtido.

Além disso, com o intuito de utilizar mais informações dos operadores, foi proposto um método no qual o comportamento dos parâmetros é expresso por uma função quadrática,

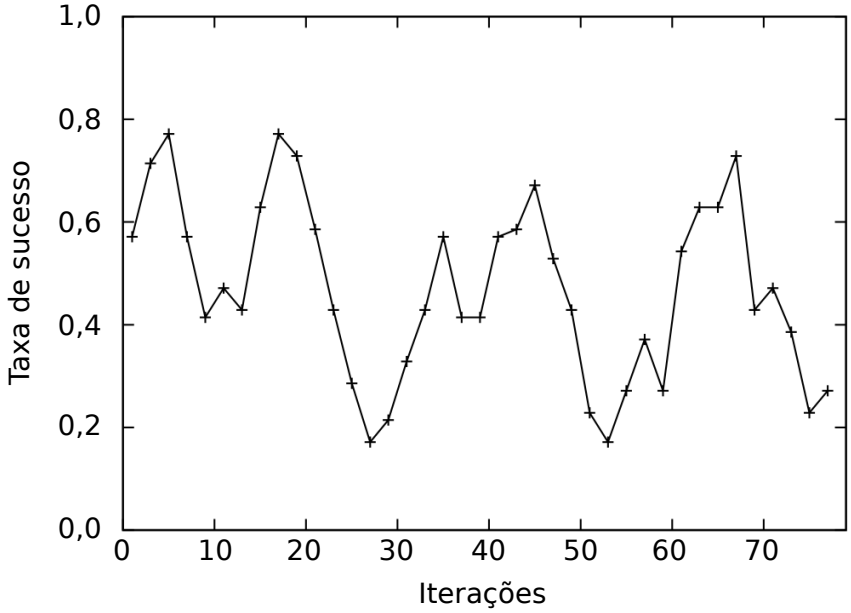

Figura 14 – Taxa de sucesso de um operador de recombinação. Adaptado de (Aleti; Moser, 2011).

dada pela equação:

$$\text{Previsão de sucesso} = a_0 + a_1 \times (t) + a_2 \times (t)^2 \ , \tag{4.2}$$

onde t é a variável, e a_0, a_1 e a_2 são os coeficientes do modelo. Serão utilizados as taxas de sucesso dos operadores nas últimas três gerações para interpolar uma função quadrática. A partir disso, irá se extrapolar o sucesso dos operadores para o próximo passo de tempo. Por ter mais dados acerca do comportamento de um operador, ao utilizar a taxa de sucesso das três últimas gerações, espera-se que a função quadrática seja capaz de inferir com maior precisão o sucesso do operador na próxima geração.

As duas interpolações que serão extrapoladas para prever o sucesso de um operador na próxima geração. Para isso, são utilizadas as taxas de sucesso obtidas nas gerações imediatamente predecessoras. Esses esquemas de atribuição de crédito serão aplicados para os dois métodos preditivos (PPC e CPPC).

4.2 CONTRIBUIÇÃO PARA AS TÉCNICAS DE SELEÇÃO DE OPERADORES

De acordo com da Costa *et al.* (2008), e também explicado na Seção 3.4.2.2, o DMAB é muito sensível à qualidade dos operadores selecionados para o sorteio. Isso ocorre porque essa técnica testa cada operador somente uma vez e, a partir daí, começa a utilizá-lo até que esse operador pare de gerar indivíduos melhores ou o PH seja reiniciado. Portanto, foram desenvolvidas algumas técnicas para evitar que um bom (ou mau) sorteio possam influenciar na qualidade estimada dos operadores, como o método de atribuição

de crédito baseada no valor extremo discutido na Seção 3.4.1 e as técnicas de seleção de operadores SLMAB e RMAB, explicados nas Seções 3.4.2.3 e 3.4.2.4, respectivamente.

Para resolver esse problema é proposto o *normalized-Roulette Dynamic Multi-Armed Bandit* (RDMAB). Nesse modelo, os operadores são utilizados por toda uma janela temporal W antes que ocorra o início do DMAB, sendo que nesse mesmo período as probabilidades de ocorrência desses operadores são ajustadas de acordo com sua recompensa normalizada, como mostra a seguinte equação:

$$\mathcal{P}_i = \frac{\sum_{t_0}^{T} \hat{q}_{i,t}}{\sum_{i=1}^{K} \sum_{t_0}^{T} \hat{q}_{i,t}} \quad . \tag{4.3}$$

A partir da segunda geração, o DMAB é inicializado e só é interrompido através do PH, que dá início novamente ao processo anterior, por mais uma janela temporal W. O pseudocódigo pode ser visualizado no Algoritmo 10.

4.3 OS ALGORITMOS GENÉTICOS ADAPTATIVOS PROPOSTOS

Com base na literatura sobre o tema, ao desenvolver os AGAs seguiu-se a orientação para a representação do indivíduo e também para os operadores genéticos que serão utilizados.

Na estrutura dos AGAs serão disponibilizados cinco operadores de movimento, sendo três de recombinação e dois de mutação. Esses operadores serão escolhidos pelo próprio AGA, durante o processo de busca, devendo somente ser escolhido um operador de cada tipo, como ocorre em um AG padrão. Os operadores disponíveis são: PMX, OPX, LOX, SM e IM, explicados anteriormente na Seção 3.1. Além disso, os AGAs selecionarão automaticamente a taxa de ocorrência desses operadores.

No que se refere a seleção parental e a seleção de sobreviventes para a próxima geração, os AGAs terão os mesmos componentes descritos na Seção 3.1, isto é, um torneio binário é utilizado na seleção parental e uma seleção elitista, na qual sobrevive o melhor indivíduo, é utilizada para a seleção de sobreviventes para próxima geração.

Ainda, os AGAs poderão ter diferentes elementos para compor sua estrutura, gerando AGAs híbridos. Para isso, os AGAs poderão ter 20% dos seus indivíduos iniciais gerados pela heurística NEH, descrita na Seção 3.2, podem utilizar uma busca local descrita na Seção 3.3 após a mutação, ou ainda ter estas duas hibridizações no mesmo AGA. Vale ressaltar que o AGA poderá alterar a taxa de ocorrência da busca local.

Algoritmo 10: Algoritmo *Normalized-Reward Dynamic Multi-Armed Bandit*

entrada: $K, C, \lambda, W, \delta = 0{,}15$

para $i \leftarrow 1$ *até* K **faça**
  $n_i \leftarrow 0$;
  $\hat{q}_i \leftarrow 0{,}0$;
  $m_i \leftarrow M_i \leftarrow 0{,}0$;
$Flag = 0$;
enquanto *Não terminar* **faça**
  se $Flag < W$ **então**
    se *um ou mais operadores não foram aplicados* **então**
      $op \leftarrow$ selecionado uniformemente entre os operadores não aplicados;
    senão
      $\mathcal{P}_i = \frac{\sum_{t_0}^{T} \hat{q}_{i,t}}{\sum_{i=1}^{K} \sum_{t_0}^{T} \hat{q}_{i,t}}$;
    $Flag \leftarrow Flag + 1$;
  senão
    $op \leftarrow \text{máx}_{i=1 \cdots K} \left(\hat{q}_{i,t} + C \sqrt{\frac{2log \sum_K n_{k,t}}{n_{i,t}}} \right)$;
    Operador *op* é aplicado, impactando o processo de busca;
    $\mathcal{R}_{op} \leftarrow$ RecebeRecompensa(op);
    $n_{op} \leftarrow n_{op} + 1$;
    $\hat{q}_{op} \leftarrow \left(\frac{(n_{op} - 1) \times \hat{q}_{op} + \mathcal{R}_{op}}{n_{op}} \right)$;
    $m_{op} \leftarrow m_{op} + (\mathcal{R}_{op} - \hat{q}_o p + \delta)$;
    se $|m_{op}| > M_{op}$ **então**
      $M_{op} \leftarrow |m_{op}|$;
    senão se $M_{op} - |m_{op}| > \lambda$ **então**
      Reiniciar as variáveis do MAB e do PH $(n, \hat{q}, m, M)$;
      $Flag = 0$;

Ao utilizar as diferentes técnicas de seleção de operadores e métodos de recompensa, dez diferentes AGAs são obtidos. Esses AGAs são definidos conforme apresentado na Tabela 5.

Tabela 5 – AGAs considerados aqui e que serão utilizados na fase de experimentos.

Sigla	Técnica de Seleção de Operadores	Método de Recompensa
AP	*Adaptive Pursuit*	Recompensa Instantânea
APEX	*Adaptive Pursuit*	Recompensa baseada em Valor Extremo
DMAB	*Dynamic Multi-Armed Bandit*	Recompensa Instantânea
DMABEX	*Dynamic Multi-Armed Bandit*	Recompensa baseada em Valor Extremo
SLMAB	*Sliding Multi-Armed Bandit*	Recompensa Instantânea
SLMABEX	*Sliding Multi-Armed Bandit*	Recompensa baseada em Valor Extremo
RMAB	*Rank-based Multi-Armed Bandit*	SR
RDMAB	*normalized-Roulette Dynamic Multi-Armed Bandit*	Recompensa Instantânea
PPC	*Predictive Parameter Control*	N/A
CPPC	*Predictive Parameter Control* com intervalos	N/A

O trabalho proposto nessa dissertação difere daquelas presentes na literatura dos FJSPs, uma vez que utilizam de diferentes técnicas de operador de seleção (AP, APEX, RMAB, SLMAB e PPC) e também diferentes formas de atribuição de crédito (métodos preditivos e SR). Além disso, destaca-se como contribuição para tratar os FJSPs, a possibilidade dos AGAs adaptarem um número maior de parâmetros, não se limitando às taxas de ocorrência dos operadores como acontece em (Huang; Wang; Liang, 2016) ou à seleção dos operadores como ocorre em (Yan; Wu, 2015; Zuo; Gong; Jiao, 2016).

5 EXPERIMENTOS COMPUTACIONAIS

Para comparação e avaliação dos métodos adaptativos, tanto os presentes na literatura quanto os aqui propostos, os algoritmos foram confrontados através de instâncias geradas pelo algoritmo descrito por Taillard (1993). Uma vez que foram estudados os FJSPs, e como a proposta de Taillard (1993) não foi desenvolvida para este tipo de ambiente, as instâncias geradas foram adaptadas, atribuindo-lhes o número de máquinas paralelas pertencentes a cada estágio, de acordo com um sorteio aleatório, no qual foram permitidos somente os valores inteiros pertencentes ao intervalo [1, 3]. Através desse método, foram geradas 25 instancias para cada tamanho de tarefas $\times$ estágios ($10 \times 10, 20 \times 10, 20 \times 20, 50 \times 10$ e 50×50), totalizando 125 instancias que serão utilizadas para comparar as diversas formas dos algoritmos adaptativos. O critério de parada é definido como o número de avaliações da função objetivo, sendo esse igual a $(800 + 10\mathcal{J})\mathcal{J}^2$, em que $\mathcal{J}$ é número de tarefas, como utilizado por Arroyo & Pereira (2011).

Os dez AGAs descritos no Capítulo 4 serão configurados de acordo com a Tabela 6, que apresenta: os parâmetros, como eles serão configurados, e os valores permitidos. Quanto a forma que serão configurados os parâmetros podem ser: "Usuário", quando é definido pelo usuário; "AGAs", quando é adotado pelo AGA; e "Constante", quando o parâmetro é mantido fixo nos experimentos realizados aqui. Além disso, os hiperparâmetros disponíveis para as técnicas adaptativas estão disponíveis na Tabela 7.

Tabela 6 – Configurações disponíveis para os parâmetros e como são controlados/selecionados.

Parâmetros	Como é configurado	Valores dos Parâmetros
Tamanho da População	Usuário	50; 76; 100
Operador de recombinação	AGAs	PMX; OPX; LOX
Taxa de recombinação	AGAs	0,7; 0,8; 0,9[1]
Operador de mutação	AGAs	SM, IM
Taxa de mutação	AGAs	0,3; 0,4; 0,5[1]
Número de indivíduos do critério elitista	Constante	1 indivíduo
Utilização da NEH	Usuário	APLICADO; N/A
Utilização da LS	Usuário	APLICADO; N/A
Taxa da LS	AGAs	0,015; 0,030; 0,045[1]
Parâmetro de destruição da LS (*des*)	Constante	10% dos genes

[1] Para o CPPC é utilizado o intervalo entre esses valores.

Um AG também foi utilizado como base de comparação para as técnicas adaptativas. Esse AG possui a codificação, os esquemas de seleção dos pais, a seleção de sobreviventes para próxima geração e o critério de parada iguais às que foram implementadas para os AGAs, e estão descritas na Seção 4.3.

Para configurar esse AG, serão utilizadas todas as variações dos parâmetros possíveis

Tabela 7 – Hiperparâmetros disponíveis nos métodos adaptativos e seus valores.

AGA	Hiperparâmetros	Valores dos hiperparâmetros
AP	α	20%, 50%; 80%
AP	β	20%, 50%; 80%
AP	$\mathcal{P}_{\min}$	Constante = 5%
APEX	α	20%, 50%; 80%
APEX	β	20%, 50%; 80%
APEX	$\mathcal{P}_{\min}$	Constante = 5%
APEX	W	Constante = Tamanho da População
DMAB	C	0,01; 0,1; 1; 10; 100
DMAB	γ	0,01; 0,1; 1; 10; 100
DMAB	δ	Constante = 0,15
DMABEX	C	0,01; 0,1; 1; 10; 100
DMABEX	γ	0,01; 0,1; 1; 10; 100
DMABEX	W	Constante = Tamanho da População
DMABEX	δ	Constante = 0,15
SLMAB	C	0,01; 0,1; 1; 10; 100
SLMAB	W	Constante = Tamanho da População
SLMABEX	C	0,01; 0,1; 1; 10; 100
SLMABEX	W	Constante = Tamanho da População
RMAB	C	0,01; 0,1; 1; 10; 100
RMAB	D	0,25; 0,50; 0,75; 1,00
PPC	método preditivo	Linear; Quadrático; Exponencial
CPPC	método preditivo	Linear; Quadrático; Exponencial
RDMAB	C	0,01; 0,1; 1; 10; 100
RDMAB	γ	0,01; 0,1; 1; 10; 100
RDMAB	W	Constante = Tamanho da População
RDMAB	δ	Constante = 0,15

descritos na Tabela 6. Este algoritmo base, denotado aqui como GA, será composto por um operador de recombinação, um operador de mutação e poderá ter um operador de busca local (pela seleção de parâmetros, observou-se que o desempenho do GA melhora com a busca local).

Somado a esse, foi desenvolvido um segundo algoritmo, denominado MS, que realiza sorteios aleatórios em que a probabilidade está distribuída uniformemente para escolher os operadores de movimento e as taxas de ocorrência que serão utilizados. Os valores dos parâmetros que serão escolhidos aleatoriamente serão os mesmos que o AG e os AGAs poderão utilizar. Itens como tamanho da população e os esquemas de seleção serão os definidos pelo usuário, como representado também na Tabela 6.

Os algoritmos serão comparados de diferentes maneiras utilizando o perfil de desempenho, medidas estatísticas e testes estatísticos para mostrar quais são os melhores resultados obtidos pelos algoritmos e em quais situações esses resultados foram obtidos.

5.1 PERFIS DE DESEMPENHO

Para comparar os algoritmos desenvolvidos foram utilizado os Perfis de Desempenho (PDs). Os PDs são ferramentas gráficas desenvolvidas por Dolan & Moré (2002) para facilitar a visualização e interpretação dos resultados de experimentos com grandes quantidade de dados. Os PDs são gerados considerando um conjunto de problemas Ta de instâncias de prolemas ta_x, com $x = 1, 2, \cdots, nm_{ta}$, e um conjunto Al de algoritmos al_y com $y = 1, 2, \cdots, nm_{al}$. Considerando também a métrica de desempenho que será utilizada para comparação $C_{ta,al} > 0$ (nesse caso a métrica utilizada será a média do *makespan* entre as execuções independentes), a razão de desempenho é definida como:

$$rd_{ta,al} = \frac{C_{ta,al}}{\min\{C_{ta,al} : al \in Al\}} \ . \tag{5.1}$$

Então, os PDs de um algoritmo al são definidos como:

$$\rho_{al}(\tau) = \frac{1}{nm_{ta}} |\{ta \in Ta : rd_{ta,al} \leq \tau\}| \ , \tag{5.2}$$

no qual $\rho_{al}(\tau)$ representa a fração de problemas resolvidos pelo algoritmo al, com desempenho dentro de um fator τ do melhor comportamento obtido, considerando todos os algoritmos analisados. Posteriormente, será calculada a área normalizada sob a curva dos PDs.

Os perfis de desempenho serão traçados de acordo com o *makespan* médio obtido para cada algoritmo e serão utilizados para comparar os diferentes métodos de busca.

5.2 ANÁLISE I

Nesta primeira análise, os algoritmos são configurados através de instâncias de determinado tamanho e, posteriormente, as configurações que resultam em melhores resultados são usadas para resolver outras instâncias desse mesmo tamanho. Com esse tipo de experimento, busca-se avaliar quais algoritmos são melhores para solucionar um problema específico.

A Análise I se encaixa nos sistemas de produção em que o gestor recebe muitas atividades e todas elas são despachadas rapidamente, ou seja, o fluxo de entrada e saída de tarefas no sistema produtivo é alto. Além disso, nesse sistema, a quantidade de tarefas é uniforme e o número de estágios pelos quais essas tarefas passam é constante. Por fim, o tomador de decisão não sabe as características das tarefas que chegarão no futuro.

Assim, a Análise I foi dividida em duas partes: na primeira, nomeada de Fase de Configuração, os algoritmos serão configurados utilizando 25 (das 125) instâncias do problema. Os algoritmos serão comparados através da área normalizada sob a curva do

perfil de desempenho, que avalia uma combinação entre eficiência e confiabilidade dos métodos. As comparações nesta primeira fase serão realizadas entre técnicas similares (por exemplo, o AP será comparado com suas diferentes possibilidades de configuração) e somente a melhor configuração de cada técnica passará para a segunda fase. A segunda fase da Análise I, chamada de Fase de Comparação, avalia comparativamente os diferentes tipos de algoritmos (por exemplo, a melhor configuração obtida pelo AP será comparada a melhor configuração obtida pelo DMAB) através dos resultados obtidos nas 100 instâncias restantes.

Em cada uma das fases, os algoritmos serão testados em diferentes categorias sendo as cinco primeiras nomeadas de: 10×10, em que foram considerados os problemas de 10 tarefas e 10 estágios; 20×10, em que foram considerados os problemas de 20 tarefas e 10 estágios; e assim por diante. Além dessas, foi inserida uma sexta categoria que contém todos os problemas.

Portanto, os melhores ajustes para os problemas de 10 tarefas e 10 máquinas serão utilizados para resolver as novas instâncias de 10 tarefas e 10 máquinas, e assim por diante. Como resultado, espera-se verificar quais são os algoritmos com o melhor desempenho no que se refere à qualidade das soluções obtidas quando aplicados aos FJSPs.

5.2.1 **Fase de Configuração**

Na fase de configuração foram selecionadas 20% das instâncias de cada tamanho e, a partir daí, os métodos adaptativos, o GA e o MS, descritos nas seções anteriores, terão seus melhores parâmetros selecionados. Os 12 diferentes algoritmos, que serão configurados com os parâmetros disponíveis na Tabela 6. Quando forem AGAs, estes serão ajustados também com os hiperparâmetros que estão disponíveis na Tabela 7.

Através de duas execuções independentes para cada instância (total de 50 execuções independentes), os algoritmos serão comparados entre suas configurações disponíveis. Destaca-se que os 12 diferentes algoritmos serão comparados apenas entre suas diferentes configurações, dentro de seis categorias. Cinco dessas categorias são definidas de acordo com o tamanho do problema, e cada uma dispõe de cinco instâncias. A última engloba todos os problemas, logo tem 25 instâncias. Portanto, 72 algoritmos serão selecionados nessa fase.

Para comparar estes algoritmos foram utilizados os PDs, descritos previamente, sobre todas as configurações disponíveis para um algoritmo. Por exemplo, todas as combinações de parâmetros dos GAs serão testados através dos perfis de desempenho, e assim, somente a melhor delas, para cada categoria, será utilizada na fase de comparação. Desse modo, a área normalizada dos PDs será o indicador utilizado em cada comparação.

Como resultado dessa fase, as melhores combinações dos parâmetros para cada algoritmo, as quais serão utilizadas na próxima fase para comparar as diferentes técnicas, estão disponíveis na Tabela 23 do APÊNDICE A.

5.2.2 **Fase de Comparação**

Os algoritmos adaptativos, GA e MS são comparados aqui usando as configurações encontradas durante a fase anterior . Para isso, a Fase de Comparação contém as 100 instâncias que não foram utilizadas na Fase de Configuração, sendo que serão realizadas cinco execuções independentes para cada instância. Dessa forma, nessa fase as 12 diferentes configurações dos algoritmos, para cada categoria, serão aplicadas para solucionar as novas instâncias, que não foram utilizadas na Fase de Configuração.

As melhores configurações encontradas para cada categoria, se restringirão a solucionar as novas instância de mesma categoria, ou seja, as melhores configurações encontradas para solucionar problemas de 10 tarefas e 10 estágios serão aplicadas somente as novas instâncias desse mesmo tamanho, mais detalhes sobre essas categorias estarão descritos nas próximas seções.

5.2.2.1 *Resolução dos problemas com 10 tarefas e 10 estágios*

Nessa categoria 20 instâncias de problemas envolvendo 10 tarefas e 10 estágios foram utilizadas. Como na fase anterior, os PDs foram realizados considerando o desempenho médio do *makespan* para as cinco execuções independentes. A Figura 15 representa os PDs para esses algoritmos e é possível observar que o DMABEX é o primeiro a conseguir resolver os problemas, com melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$. Além disso, o DMABEX é o que tem melhor desempenho, quando considera-se a área normalizada dos PDs, o que pode ser visualizado na Figura 16. O DMABEX e o GA são os métodos que apresentam o melhor resultado obtido na maior quantidade de instâncias (em 90% delas).

Além disso, medidas estatísticas como a média, mediana, valor máximo, valor mínimo e desvio-padrão foram realizados para comparar as 12 versões dos algoritmos. Os resultados foram normalizados em relação ao maior *makespan* médio para cada instância, e estão disponíveis na Tabela 8, na qual os melhores resultados estão em negrito. Os *boxplots* dos resultados dos algoritmos estão representados na Figura 17. Esses diagramas representam a variação de dados observados de uma variável numérica por meio de quartis.

Como o objetivo dos algoritmos é minimizar o *makespan*, os menores valores alcançados são os melhores. Ao analisar a média, o DMABEX obteve os melhores

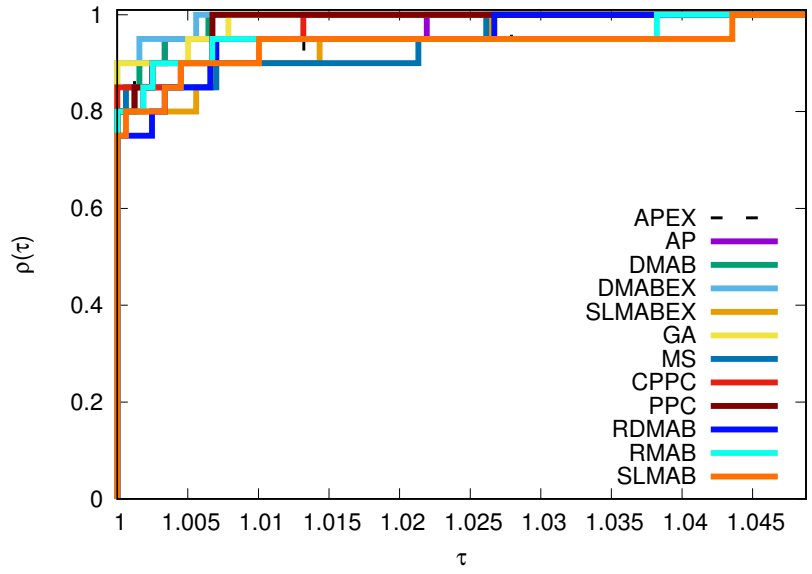

Figura 15 – PDs dos algoritmos ao resolverem os problemas de 10 tarefas e 10 estágios.

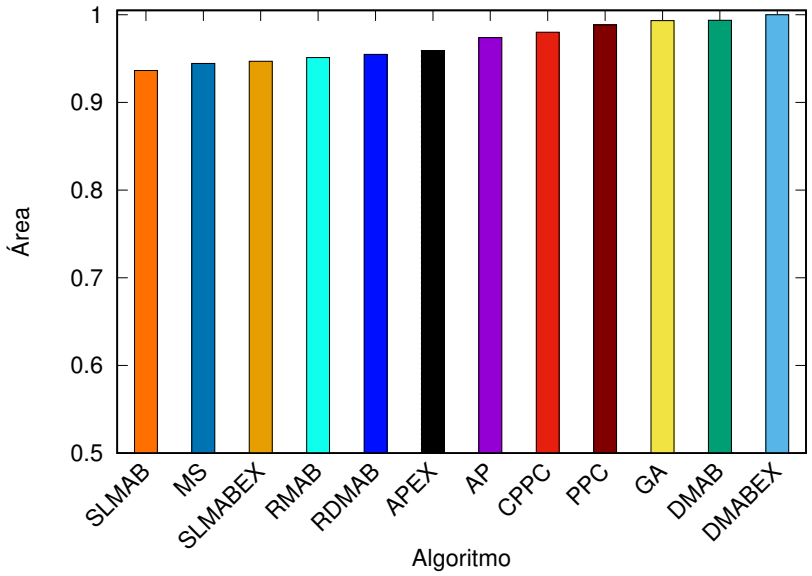

Figura 16 – Área normalizada dos PDs dos algoritmos ao resolverem os problemas de 10 tarefas e 10 estágios.

Tabela 8 – Resultados estatísticos para os algoritmos ao resolverem os problemas de 10 tarefas e 10 estágios.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9972	0,9966	0,9957	**0,9955**	0,9977	0,9957	0,9978	0,9960	0,9963	0,9974	0,9975	0,9981
Desvio-padrão	0,0066	0,0074	0,0094	0,0106	0,0063	0,0088	**0,0042**	0,0093	0,0087	0,0057	0,0057	**0,0042**
Mínimo	0,9709	0,9728	0,9644	**0,9583**	0,9720	0,9658	0,9833	0,9647	0,9626	0,9792	0,9753	0,9826
Mediana	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

resultados. Além disso, esse algoritmo apresenta um melhor desempenho em relação aos demais, quando comparado pelo melhor resultado alcançado (mínimo). Contudo, o pior

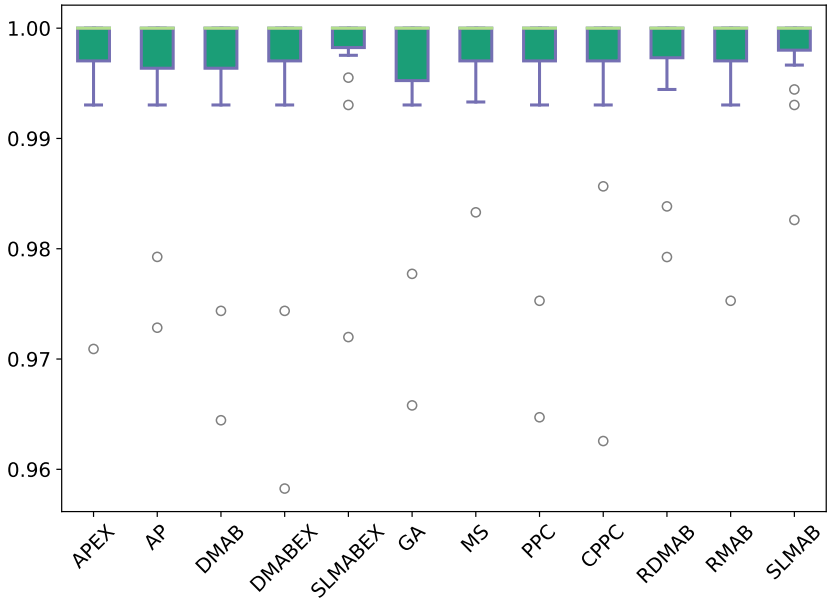

Figura 17 – *Boxplot* dos resultados dos algoritmos ao resolverem os problemas de 10 tarefas e 10 estágios.

resultado alcançado (máximo), bem como o resultado da mediana foram semelhantes para todos os algoritmos.

Além disso, foi realizado o teste de Kruskal & Wallis (1952) para avaliar se há diferença estatisticamente significante entre esses algoritmos. Os testes de Kruskal-Wallis foram realizados utilizando a biblioteca SciPy, implementado e executado em Python 2.7.12. Trata-se de um teste não paramétrico, o qual pode ser utilizado para comparar três ou mais populações, sendo que sua hipótese nula é de que todas as populações possuem funções de distribuições iguais, ou seja, não se pode dizer que os resultados dos algoritmos são diferentes. Portanto, caso seja significante (p-valor $< 0{,}05$), o teste rejeita a hipótese nula, o que diz que pelo menos uma das amostras é diferente das demais.

O p-valor obtido foi igual a 0,9996, e através de uma confiança de 95%, tem-se que o p-valor $> 0{,}05$, o que não rejeita a hipótese nula. Portanto, para esses problemas pode-se dizer que os resultados obtidos para esses algoritmos ao resolverem os problemas de 10 tarefas e 10 estágios são estatisticamente semelhantes.

5.2.2.2 *Resolução dos problemas com 20 tarefas e 10 estágios*

Nessa categoria, os diferentes algoritmos foram comparados para a solução dos problemas de 20 tarefas e 10 estágios, também com o intuito de minimizar o *makespan*. A Figura 18 apresenta os PDs para os algoritmos quando tentam resolver os problemas dessa categoria. Pode-se observar que o algoritmo PPC é o que consegue resolver todos os problemas com melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$.

Porém, ao calcular a área normalizada dos PDs, o DMABEX é o melhor algoritmo. O método AP é o que apresenta o melhor resultado para o maior número de instâncias (em 70% dos casos).

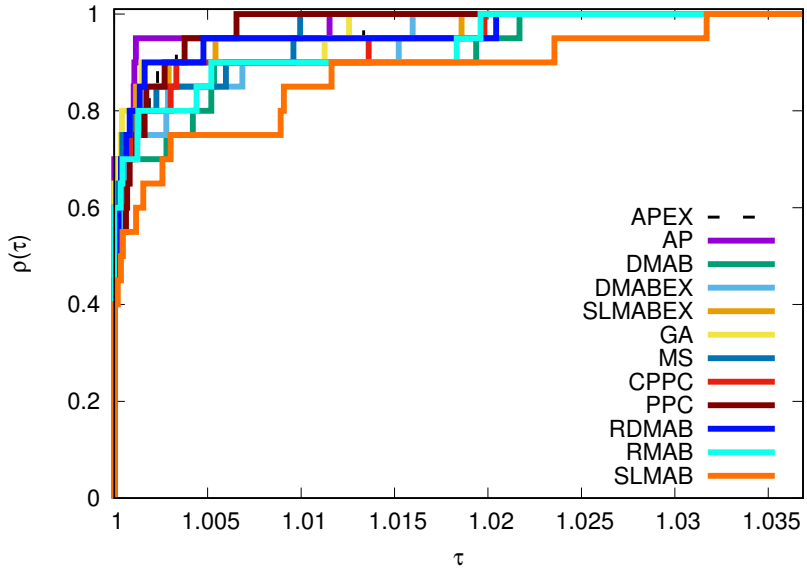

Figura 18 – PDs dos algoritmos ao resolverem os problemas de 20 tarefas e 10 estágios.

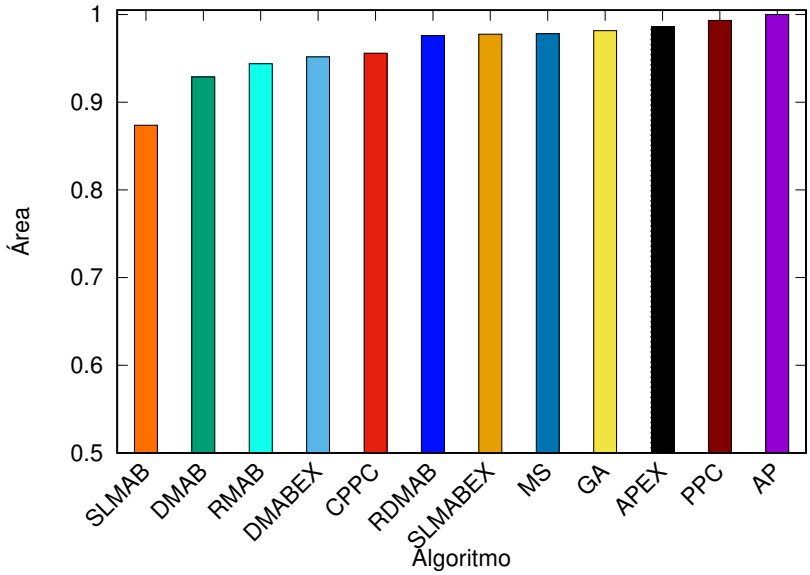

Figura 19 – Área normalizada dos PDs dos algoritmos ao resolverem os problemas de 20 tarefas e 10 estágios.

Ao calcular as medidas estatísticas, que podem ser visualizados na Tabela 9, e os *boxplots*, que são apresentados na Figura 20, pode-se verificar que o PPC obteve o melhor resultado médio, enquanto o algoritmo AP conseguiu o melhor resultado e a melhor

mediana. Quando é verificado o pior resultado, todos os algoritmos possuem resultados semelhantes.

Tabela 9 – Resultados estatísticos para os algoritmos ao resolverem os problemas de 20 tarefas e 10 estágios.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9963	0,9959	0,9980	0,9973	0,9965	0,9964	0,9965	**0,9961**	0,9972	0,9966	0,9976	0,9997
Desvio-padrão	0,0068	0,0073	0,0030	0,0041	0,0060	0,0054	0,0058	0,0075	0,0040	0,0058	0,0035	**0,0007**
Mínimo	0,9711	**0,9693**	0,9903	0,9840	0,9745	0,9802	0,9786	0,9699	0,9885	0,9770	0,9883	0,9977
Mediana	0,9994	**0,9992**	0,9996	0,9993	0,9995	0,9995	0,9994	0,9997	0,9996	0,9997	0,9994	1,0000
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

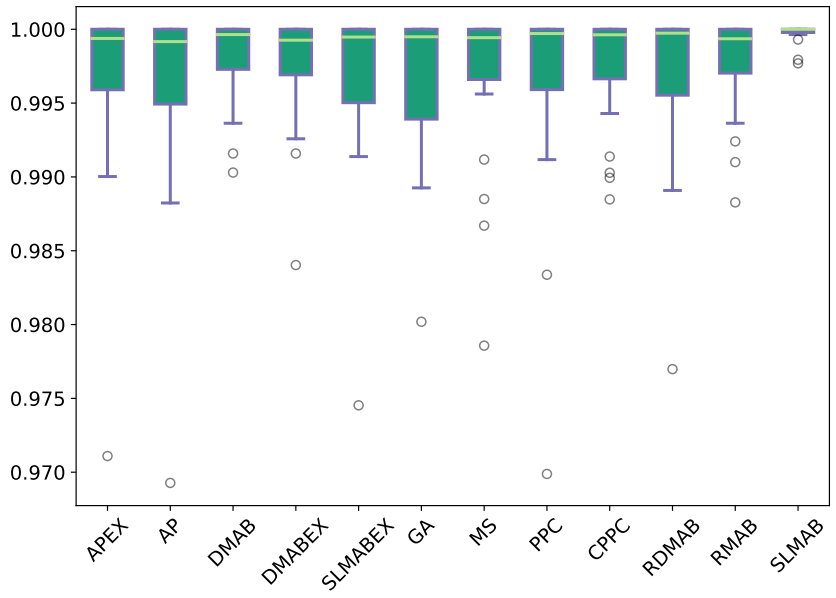

Figura 20 – *Boxplot* dos resultados dos algoritmos ao resolverem os problemas de 20 tarefas e 10 estágios.

O p-valor obtido para esses algoritmos, ao realizar o teste de Kruskal-Wallis foi de 0,2978, e através de uma confiança de 95% temos que o p-valor $> 0{,}05$, o que não rejeita a hipótese nula. Portanto, para esses problemas pode-se dizer que o resultado obtidos pelos algoritmos possuem desempenhos estatisticamente semelhantes.

5.2.2.3 *Resolução dos problemas com 20 tarefas e 20 estágios*

Quando estão sendo solucionados os problemas com 20 tarefas e 20 estágios, o GA conseguiu resolver os problemas com melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), e obteve a maior área normalizada, conforme pode ser visualizado na Figura 21 e Figura 22, respectivamente. O GA é a técnica que apresenta o melhor desempenho na maioria das instâncias, em 30% delas.

As medidas estatísticas mostram também que o GA possui um desempenho superior quando comparado através dos resultados mínimo, média, máximo e a melhor mediana.

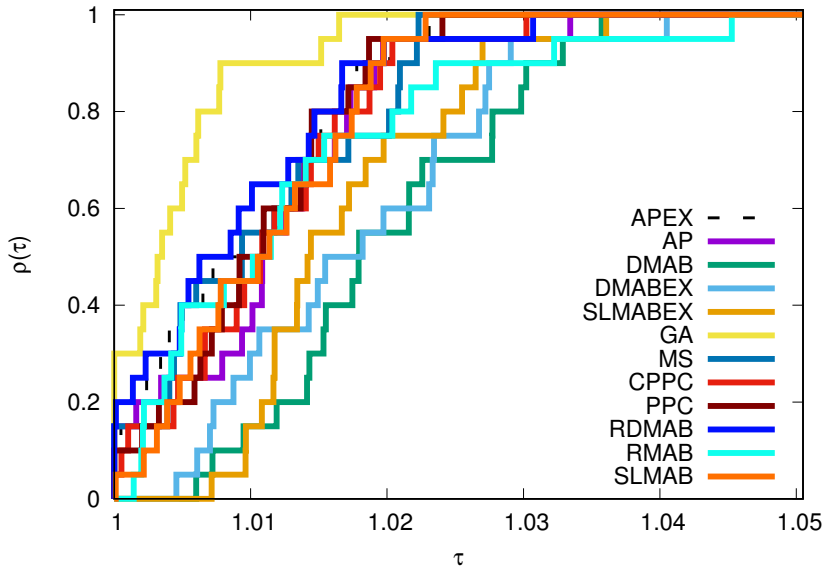

Figura 21 – PDs dos algoritmos ao resolverem os problemas de 20 tarefas e 20 estágios.

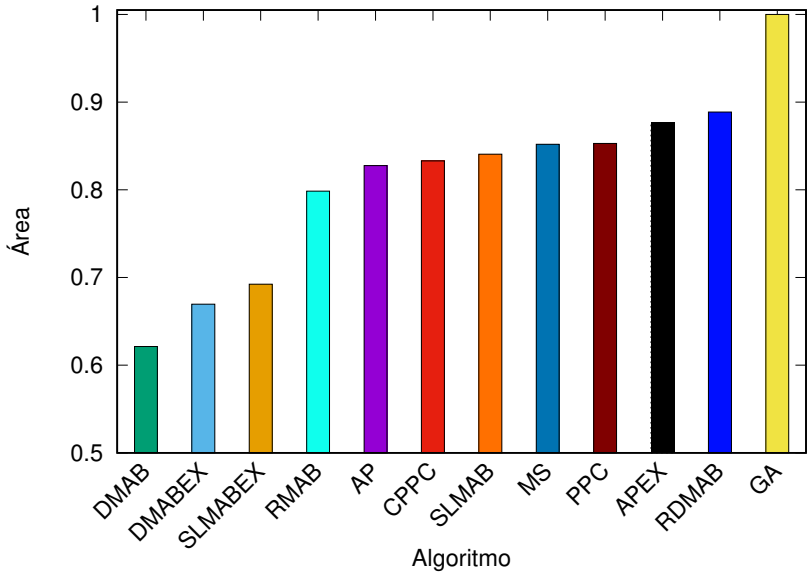

Figura 22 – Área normalizada dos PDs dos algoritmos ao resolverem os problemas de 20 tarefas e 20 estágios.

Esses resultados podem ser visualizados na Tabela 10 e a variação dos dados pode ser conferidas nos *boxplots* disponíveis na Figura 23.

Tabela 10 – Resultados estatísticos para os algoritmos ao resolverem os problemas de 20 tarefas e 20 estágios.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9844	0,9864	0,9946	0,9926	0,9918	**0,9795**	0,9854	0,9854	0,9861	0,9839	0,9875	0,9858
Desvio-padrão	0,0067	0,0082	0,0070	**0,0048**	0,0072	0,0088	0,0095	0,0089	0,0065	0,0086	0,0084	0,0082
Mínimo	0,9702	0,9687	0,9704	0,9830	0,9747	**0,9567**	0,9655	0,9611	0,9762	0,9677	0,9728	0,9733
Mediana	0,9860	0,9870	0,9957	0,9924	0,9920	**0,9812**	0,9861	0,9869	0,9866	0,9866	0,9885	0,9858
Máximo	0,9978	1,0000	1,0000	1,0000	1,0000	**0,9936**	1,0000	0,9975	1,0000	1,0000	1,0000	1,0000

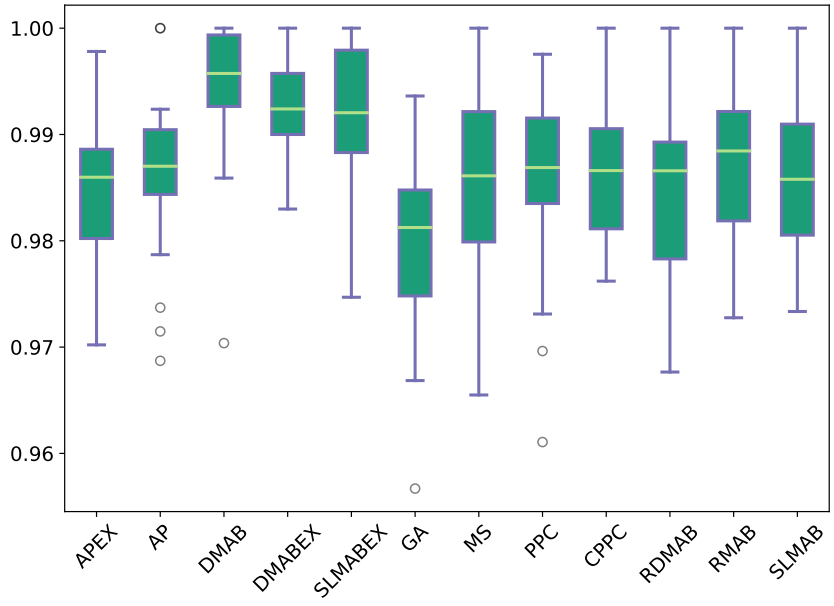

Figura 23 – *Boxplot* dos resultados dos algoritmos ao resolverem os problemas de 20 tarefas e 20 estágios.

O teste de Kruskal-Wallis retornou um p-valor igual a 0,0000, e através de uma confiança de 95% pode-se rejeitar a hipótese de que os resultados obtidos pelos algoritmos são iguais. Quando a hipótese é rejeitada, deve-se realizar um teste para verificar em relação a quais algoritmos essa diferença estatística é observada.

Para isso, o teste de Conover & Iman (1979) será utilizado para estabelecer as relações de dominância estocástica. Os testes de Conover-Iman foram realizados através da biblioteca conover.test disponível para R. Nesse teste, a hipótese nula para cada comparação entre pares é definida como: a probabilidade de selecionar aleatoriamente um valor do primeiro grupo amostral, deve-ser maior do que selecionar aleatoriamente um valor do segundo grupo, em 1/2. Esse método apresenta algumas opções disponíveis para ajustar o p-valor das múltiplas comparações. Portanto, utilizou-se o ajuste proposto por Benjamini & Hochberg (1995), que controla a taxa de descobertas falsas e é apropriado para testes independentes. Os p-valores são ordenados em ordem decrescente e ajustados, assim depois que a hipótese nula em relação ao primeiro é rejeitada, todos os demais na sequência também o são.

O teste de Conover-Iman compara os algoritmos por meio da mediana dos valores obtidos. Por isso, selecionamos o algoritmo cujo os resultados obtidos tiveram a melhor (menor) mediana (para esse caso foi o GA) e realizamos o teste sempre comparando as demais técnicas a esse. Os resultados do teste de Conover-Iman estão no APÊNDICE B na Tabela 24, e com 95% de confiança não permite afirmar estatisticamente que o desempenho do GA é superior ao dos algoritmos APEX, MS, CPPC, RDMAB e SLMAB. No entanto,

o GA foi estatisticamente superior ao AP, DMAB, DMABEX, SLMABEX, PPC e RMAB.

5.2.2.4 *Resolução dos problemas com 50 tarefas e 10 estágios*

Nessa categoria, os algoritmos são comparados ao solucionarem os problemas de 50 tarefas e 10 estágios. Ao analisar os PDs, verifica-se que o MS foi o melhor algoritmo para resolver esse tipo de problema, com melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$. Porém, esse resultado foi bem próximo do RDMAB, que é o algoritmo que possui a maior área normalizada. O RDMAB é o algoritmo que apresenta o melhor desempenho na maioria das instâncias (em 25% delas).

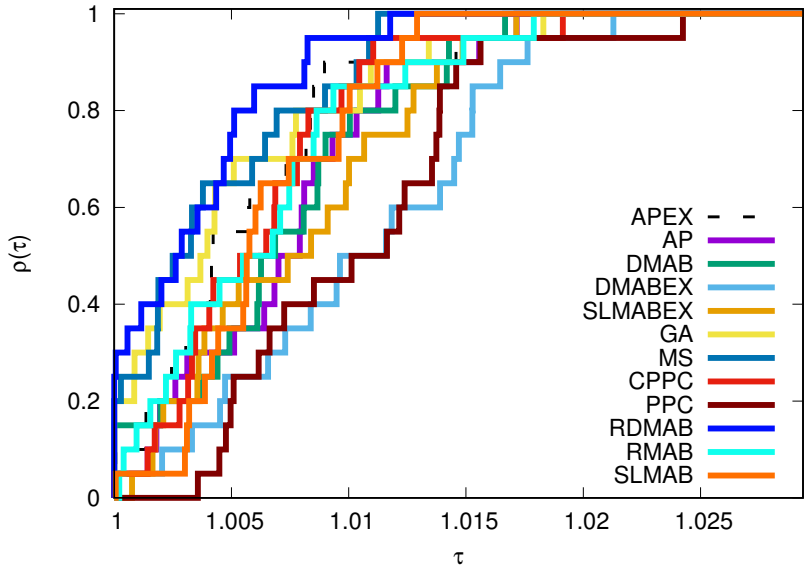

Figura 24 – PDs dos algoritmos ao resolverem os problemas de 50 tarefas e 10 estágios.

As medidas estatísticas na Tabela 11 e as variações dos dados apresentados na Figura 26 mostram que o RDMAB é o algoritmo com melhor desempenho médio, a melhor mediana e o melhor valor máximo. Já quando são comparados pelo valor mínimo, o DMAB e o RDMAB alcançaram resultados semelhantes.

Tabela 11 – Resultados estatísticos para os algoritmos ao resolverem os problemas de 50 tarefas e 10 estágios.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,991	0,993	0,993	0,996	0,993	0,991	0,990	0,992	0,996	**0,989**	0,992	0,992
Desvio-padrão	0,006	0,004	0,006	0,005	0,005	0,005	0,005	0,004	**0,003**	0,005	0,005	0,004
Mínimo	0,978	0,984	**0,976**	0,981	0,978	0,980	0,979	0,983	0,987	**0,976**	0,983	0,979
Mediana	0,993	0,993	0,994	0,997	0,995	0,991	0,991	0,993	0,997	**0,990**	0,991	0,992
Máximo	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	**0,996**	1,000	1,000

O teste de Kruskal-Wallis retornou o p-valor de 0,0000 e, com a confiança de 95%, pode-se rejeitar a hipótese de que os resultados obtidos pelos algoritmos são iguais. Então,

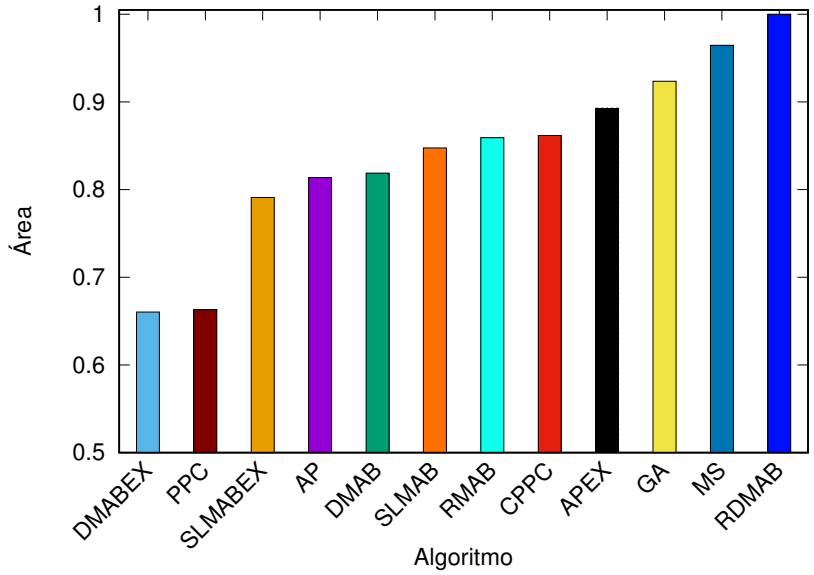

Figura 25 – Área normalizada dos PDs dos algoritmos ao resolverem os problemas de 50 tarefas e 10 estágios.

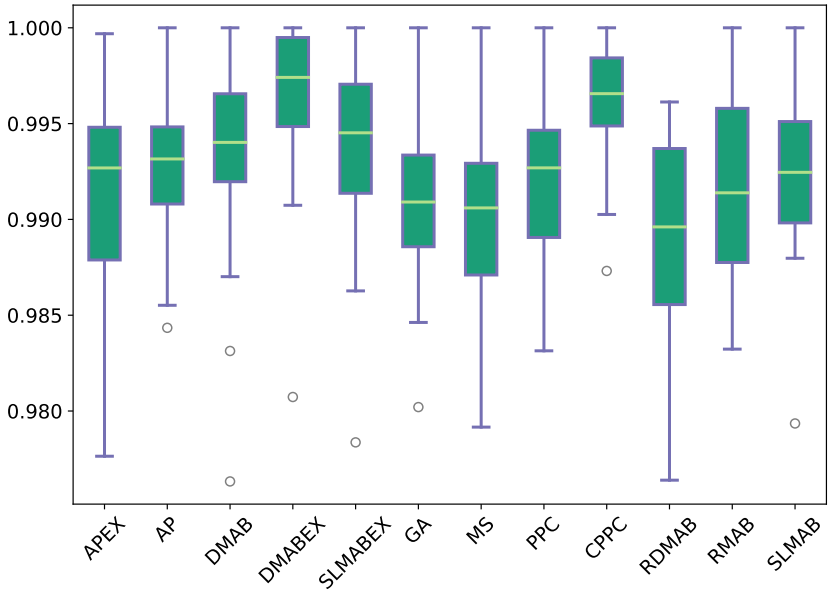

Figura 26 – *Boxplot* dos resultados dos algoritmos ao resolverem os problemas de 50 tarefas e 10 estágios.

foi realizado o teste de Conover-Iman, no qual os resultados alcançados, com 95% de confiança, indicam que RDMAB detém desempenho estatisticamente superior ao DMAB, DMABEX, SLMABEX, PPC e o CPPC, o que pode ser conferido no APÊNDICE B na Tabela 25. No entanto, observa-se nessa que o RDMAB é estatisticamente similar ao APEX, AP, GA, MS, RMAB e o SLMAB.

5.2.2.5 *Resolução dos problemas com 50 tarefas e 50 estágios*

Ao analisar os algoritmos quando foram resolvidos os problemas de 50 tarefas e 50 estágios, na Figura 27, verifica-se que o GA consegue solucionar os problemas com melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$. E na Figura 28, esse mesmo algoritmo conseguiu a melhor área normalizada. O GA é o algoritmo que apresenta o melhor desempenho na maioria das instâncias (em 45% delas).

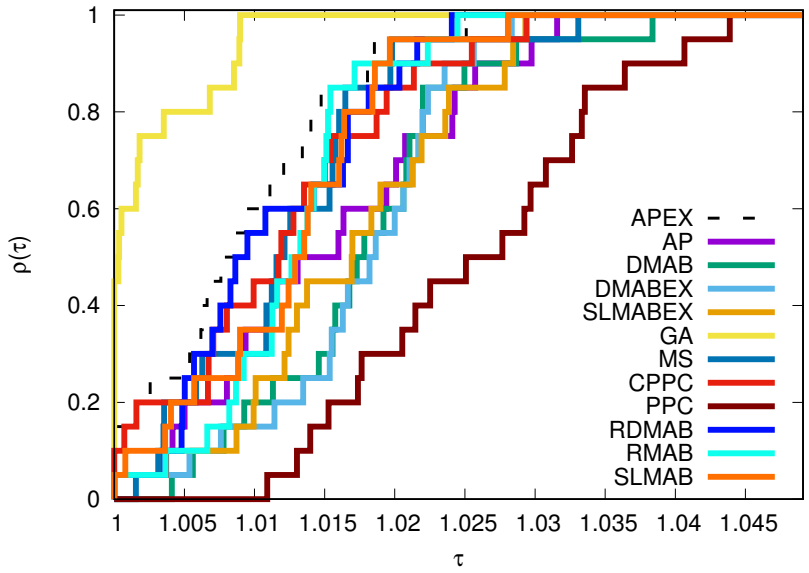

Figura 27 – PDs dos algoritmos ao resolverem os problemas de 50 tarefas e 50 estágios.

A Tabela 12 contém medidas estatísticas que mostram que os resultados obtidos pelo GA são superiores aos resultados obtidos pelos AGAs, obtendo os melhores resultados quando comparados através dos melhores resultados em relação à média, mediana e também máximo e mínimo. A variação dos valores observados podem ser visualizados nos *boxplots* da Figura 29.

O teste de Kruskal-Wallis aplicado a esses problemas retornou um valor de 0.0000, que com uma confiança de 95%, pode-se rejeitar a hipótese nula, ou seja, os algoritmos possuem desempenho estatisticamente diferentes. Ao realizar o teste de Conover-Iman, com uma confiança de 95%, o GA não tem desempenho superior ao APEX, porém é superior

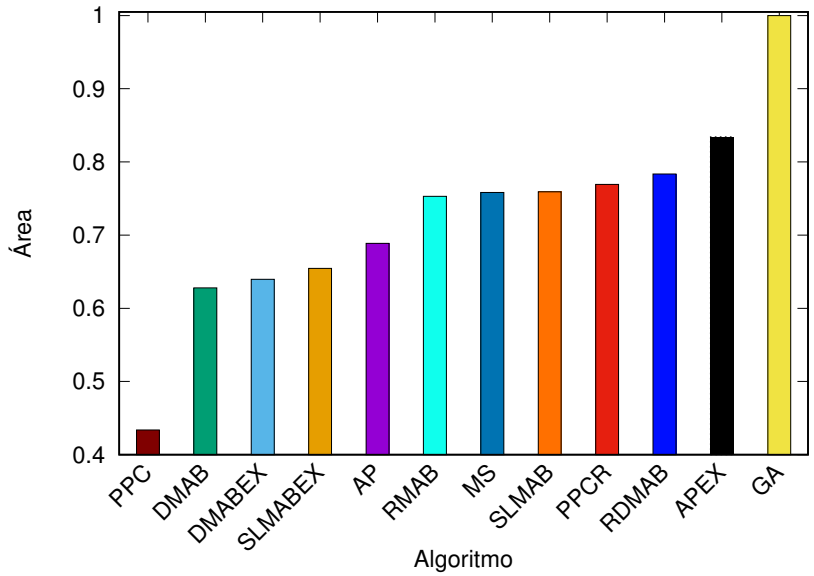

Figura 28 – Área normalizada dos PDs dos algoritmos ao resolverem os problemas de 50 tarefas e 50 estágios.

Tabela 12 – Resultados estatísticos para os algoritmos ao resolverem os problemas de 50 tarefas e 50 estágios.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9806	0,9864	0,9889	0,9885	0,9879	**0,9739**	0,9836	0,9832	0,9968	0,9826	0,9839	0,9836
Desvio-padrão	0,0061	0,0074	0,0078	0,0074	0,0096	0,0072	0,0071	0,0078	**0,0055**	0,0087	0,0079	0,0080
Mínimo	0,9661	0,9716	0,9724	0,9715	0,9675	**0,9580**	0,9714	0,9677	0,9848	0,9722	0,9649	0,9692
Mediana	0,9820	0,9862	0,9893	0,9885	0,9921	**0,9733**	0,9829	0,9838	1,0000	0,9798	0,9862	0,9842
Máximo	0,9955	1,0000	1,0000	1,0000	1,0000	**0,9863**	1,0000	0,9969	1,0000	1,0000	0,9945	1,0000

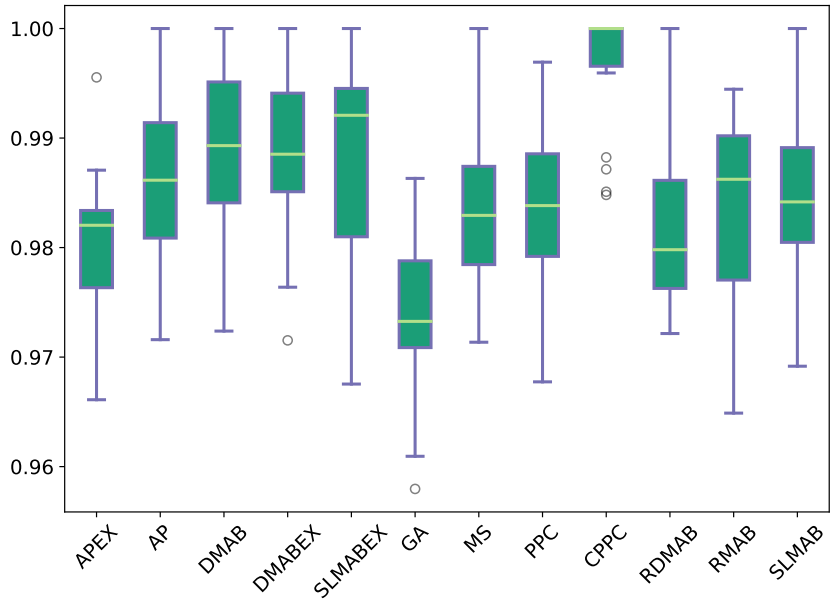

Figura 29 – *Boxplot* dos resultados dos algoritmos ao resolverem os problemas de 50 tarefas e 50 estágios.

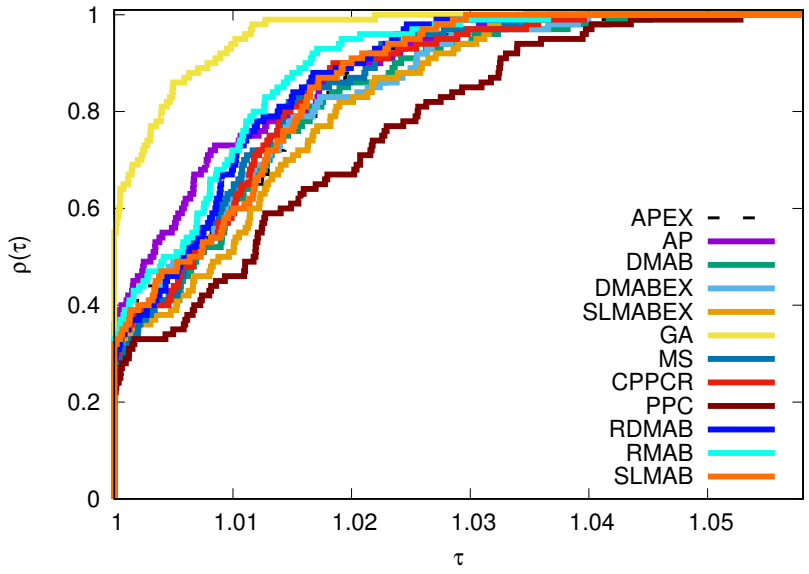

Figura 30 – PDs dos algoritmos ao resolverem os todos os problemas.

ao AP, DMAB, DMABEX, SLMABEX, RMAB, MS, PPC, CPPC, RDMAB, RMAB e SLMAB. Os resultados desse teste estão disponíveis no APÊNDICE B na Tabela 26.

5.2.2.6 *Resolução de todos os problemas*

Nessa categoria, todos os problemas solucionados nas seções anteriores foram agrupados. Então, aqueles algoritmos capazes de resolver as 25 instâncias da fase de configuração com maior área normalizada irão ser comparados nessa categoria.

Portanto, as 100 instâncias solucionadas nas seções anteriores estão agrupadas agora para uma única análise. As Figuras 30 e 31 mostram, respectivamente, que o GA é o melhor algoritmo, tanto para resolver com melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), quanto para maior área normalizada. O GA é o algoritmo que apresenta o melhor desempenho na maioria das instâncias (em 55% delas).

As medidas estatísticas mostram que o AP obteve o melhor resultado (menor *makespan*). No entanto, o GA apresenta a melhor média e a melhor mediana dos *makespans*. Essas observações podem ser encontradas na Tabela 13 e na Figura 32.

Tabela 13 – Resultados estatísticos para os algoritmos ao resolverem todas as instâncias.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	100	100	100	100	100	100	100	100	100	100	100	100
Média	0,9923	0,9906	0,9930	0,9926	0,9938	**0,9859**	0,9919	0,9921	0,9970	0,9911	0,9902	0,9918
Desvio-padrão	0,0089	0,0098	0,0076	0,0076	0,0076	0,0121	0,0086	0,0087	**0,0046**	0,0083	0,0098	0,0083
Mínimo	0,9613	**0,9499**	0,9672	0,9678	0,9685	0,9568	0,9655	0,9583	0,9812	0,9644	0,9561	0,9690
Mediana	0,9954	0,9930	0,9944	0,9938	0,9966	**0,9864**	0,9939	0,9939	1,0000	0,9922	0,9922	0,9937
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

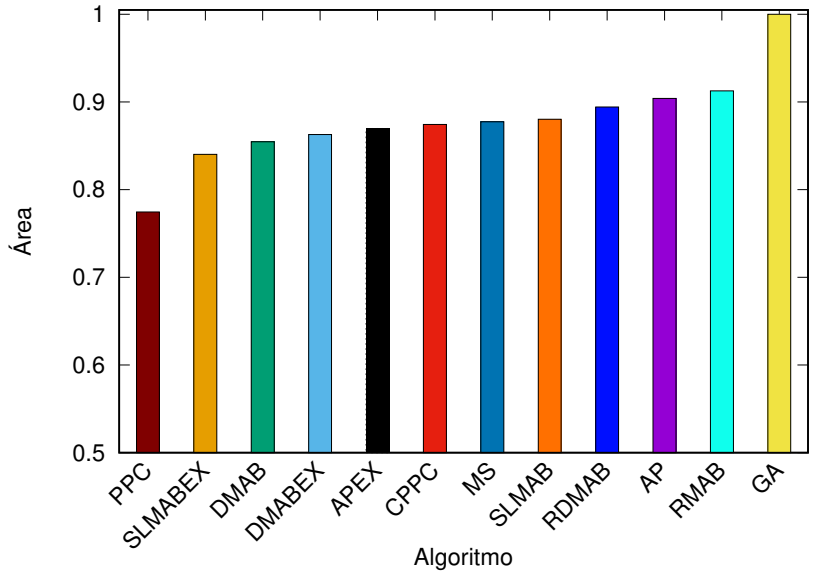

Figura 31 – Área normalizada dos PDs dos algoritmos ao resolverem todos os problemas.

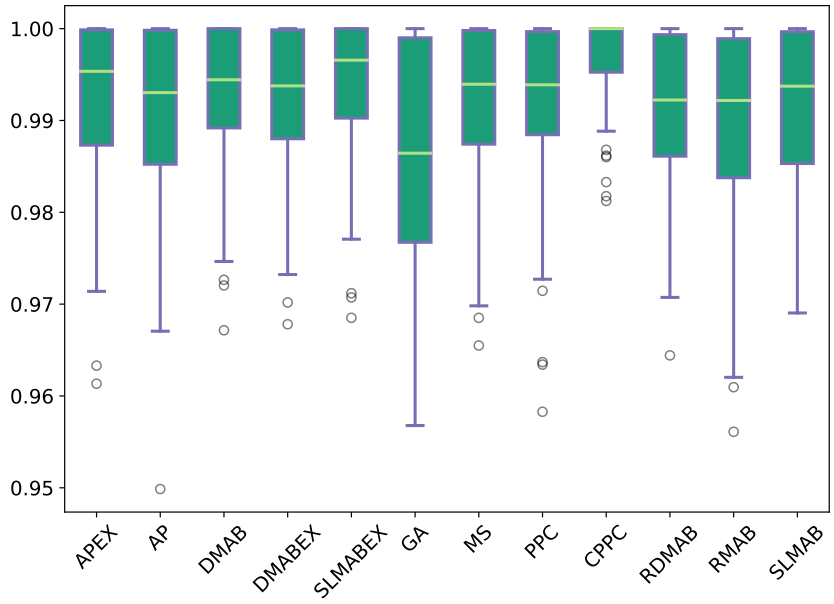

Figura 32 – *Boxplot* dos resultados dos algoritmos ao resolverem todos os problemas.

Ao realizar o teste de Kruskal-Wallis, obteve-se o valor de 0,0000 e com uma confiança de 95%, pode-se rejeitar a hipótese nula, ou seja, os valores obtidos dos desempenho dos algoritmos são estatisticamente diferentes. Então, é realizado o teste de Conover-Iman, com uma confiança de 95%. Os resultados desse teste encontram-se no APÊNDICE B, na Tabela 27. Para esses problemas, o GA não tem desempenho superior ao RMAB, no entando possui desempenho superior ao APEX, AP, DMAB, DMABEX, SLMABEX, MS, PPC, CPPC, RDMAB, RMAB e SLMAB.

5.2.3 **Conclusões da Análise I**

Ao comparar as meta-heurísticas para solucionar FJSPs, num sistema de produção que se assemelha àquele no qual o tomador de decisões lida com um alto fluxo de entrada e saída de tarefas, foi possível verificar que os resultados obtidos por esses métodos são estatisticamente semelhantes quando estão sendo solucionados os problemas de 10 tarefas e 10 estágios, e os de 20 tarefas e 10 estágios. Para esses casos, indica-se os algoritmos que são mais rápidos de serem configurados, como o MS e os AGAs quando comparados ao GA convencional, uma vez que têm menos parâmetros. A Tabela 14 apresenta, em síntese, os algoritmos que obtiveram os melhores resultados em relação à mediana ou que conseguiram resultados estatisticamente similares à esses para cada categoria.

Tabela 14 – Algoritmos que obtiveram resultados estatisticamente superiores na Análise I (por categoria).

	10×10	20×10	20×20	50×10	50×50	TODOS
AP	X	X		X		
APEX	X	X	X	X	X	
DMAB	X	X				
DMABEX	X	X				
SLMAB	X	X	X	X		
SLMABEX	X	X				
PPC	X	X				
CPPC	X	X	X			
RDMAB	X	X	X	X		
RMAB	X	X		X		X
GA	X	X	X	X	X	X
MS	X	X	X	X		

Entretanto, quando os problemas aumentam de tamanho, os resultados obtidos por alguns algoritmos começaram a se diferenciar estatisticamente. Para os problemas de 20 tarefas e 20 estágios, o GA apresentou a melhor média e também o melhor resultado através dos PDs. No entanto, esse algoritmo não obteve desempenho estatisticamente superior às opções adaptativas APEX, CPPC, RDMAB e SLMAB. Para os problemas de 50 tarefas e 10 estágios, o RDMAB foi estatisticamente superior aos algoritmos DMAB, DMABEX, SLMABEX, PPC e CPPC. No entanto, não é possível afirmar que esse algoritmo é estatisticamente superior ao GA. O GA foi o algoritmo de melhor desempenho na análise que considera os problemas de 50 tarefas e 50 estágios, e também na análise que recebe todos os problemas. Porém, não é possível diferenciá-lo estatisticamente dos algoritmos APEX e do RMAB, nas respectivas categorias.

Conclui-se para Análise I, na qual os algoritmos são aplicados aos problemas da mesma categoria em que são treinados, que o MS é o mais interessante quando não há

tempo suficiente para testar todas as configurações dos AGs e ou AGAs, principalmente quando trata-se de problemas de até 50 tarefas e 10 estágios. Quando se está solucionando problemas grandes, com 50 tarefas e 50 estágios, o resultados obtidos pelo GA foram estatisticamente superiores aos resultados obtidos pelo MS.

Quando comparados aos AGAs, a técnica APEX se mostrou estatisticamente semelhante ao GA para tratar os problemas envolvendo 50 tarefas e 50 estágios ou menores. Portanto, pode-se dizer que o APEX é uma técnica adaptativa interessante quando pode-se definir a categoria de problemas que deverão ser solucionados. Isso, pois essa técnica tem menos parâmetros que um AG tradicional, sendo mais rápido de configurá-lo, e obtém resultados estatisticamente semelhantes ao AG.

Já quando estão sendo avaliados todos os problemas conjuntamente, os resultados obtidos pelo GA não foram estatisticamente superiores aos resultados obtidos pelo método RMAB. Portanto indica-se os AGs quando há tempo suficiente para configurar os parâmetros, enquanto o RMAB é indicado quando não há tempo suficiente para configurar todos os parâmetros, uma vez que o GA possui três parâmetros a mais que o RMAB. Isso torna o RMAB uma escolha adequada, por ser mais rápido ao configurar.

5.3 ANÁLISE II

Na Análise II, os algoritmos serão configurados através de instâncias de tamanho pequeno (10 tarefas e 10 estágios) e posteriormente serão aplicadas a problemas maiores. Nesse tipo de experimento será testada a capacidade de generalização dos algoritmos.

A Análise II se encaixa naqueles ambientes onde não há tempo suficiente para configurar o algoritmo de busca de forma adequada, sendo ainda mais restritivo que o cenário apresentado na Análise I. Portanto, o utilizador irá configurar o algoritmo através das instâncias com menos tarefas e menos máquinas, as quais o algoritmo é executado mais rápido, e irá aplicar a melhor configuração em um conjunto de instâncias com maior número de tarefas e máquinas. Esse cenário pode ser visto também na situação em que o tomador de decisões não conhece o número de tarefas que irá sequenciar, sendo que podem haver tarefas que dependem de diferentes quantidades de estágios. Somado a isso, há uma alta taxa de entrada e saída de tarefas e o tomador de decisões deve configurar repetitivamente o algoritmo.

A Análise II é dividida em duas partes: na primeira, os algoritmos serão configurados através de parte das instâncias de maneira similar ao que ocorre na Fase de Configuração da Análise I, ou seja, para cada técnica serão realizadas variações nas suas configurações, através dos parâmetros previamente selecionados, e somente os melhores configurações para cada uma das técnicas serão utilizadas na segunda fase. É importante relembrar

que as técnicas serão comparadas inicialmente apenas entre suas variações de parâmetros. Por exemplo, as configurações do DMAB serão comparadas somente com outras variações desse método. A Análise II diferencia-se da Análise I uma vez que apenas instâncias de 10 tarefas e 10 estágios são utilizadas na fase de configuração da Análise II.

Na segunda fase (Fase de Comparação), somente as 12 melhores configurações obtidas na Fase de Configuração, uma para cada técnica, serão comparadas entre si. Esses algoritmos irão solucionar as 100 instâncias de diferentes tamanhos. e também diferentes daquelas utilizadas na fase de configuração.

Espera-se com a Análise II verificar a capacidade de generalização dos algoritmos para solucionar FJSPs com tamanhos diferentes daqueles em que foram previamente configurados. Portanto, espera-se identificar quais algoritmos são mais robustos quando não há tempo suficiente para realizar as configurações e em quais situações esses devem ser utilizados.

5.3.1 **Fase de Configuração**

Na fase de configuração foram selecionadas 20% das instâncias de cada tamanho e, a partir daí, os métodos adaptativos, o GA e o MS, descritos nas seções anteriores, terão seus melhores parâmetros selecionados. O GA e o MS serão configurados através da Tabela 6. Já os AGAs serão configurados através dos parâmetros disponíveis na Tabela 6 e os hiperparâmetros disponíveis na Tabela 7.

Através de duas execuções independentes para cada instância os algoritmos serão comparados entre suas configurações disponíveis. Destaca-se que os 12 diferentes algoritmos serão comparados apenas entre suas diferentes configurações, para os problemas de 10 tarefas e 10 estágios.

Para comparar estes algoritmos foram utilizados os PDs, descritos previamente, sobre todas as configurações disponíveis para um algoritmo, por exemplo, todos as combinações de parâmetros dos GAs serão testados através dos perfis de desempenho, e assim, somente a melhor delas, para cada categoria, será utilizada na fase de comparação. Desse modo, a área normalizada dos PDs será o indicador utilizado em cada comparação. A Tabela 23 do APÊNDICE A apresenta as melhores configurações de parâmetros para cada algoritmo e que foram selecionadas para a Fase de Comparação.

5.3.2 **Fase de Comparação**

Os algoritmos adaptativos, GA e MS são comparados aqui usando as configurações encontradas durante a fase anterior. Para isso, a Fase de Comparação contém as 100

instâncias que não foram utilizadas na Fase de Configuração, sendo que serão realizadas cinco execuções independentes para cada instância.

Dessa forma, nessa fase as 12 versões diferentes de algoritmos serão confrontadas nas categorias nomeadas: 20×10, em que foram considerados os problemas de 20 tarefas e 10 estágios; 20×20, em que foram considerados os problemas de 20 tarefas e 20 estágios; 50×10, em que foram considerados os problemas de 50 tarefas e 10 estágios; 50×50, em que foram considerados os problemas de 50 tarefas e 50 estágios; 20 tarefas, em que foram considerados todos os problemas de 20 tarefas; 50 tarefas, em que foram considerados todos os problemas de 50 tarefas; Além dessas, foi inserida uma sétima categoria que contém todos os problemas.

A forma de experimentação empregada nesse trabalho, especialmente a presente na Análise II, não é comum em trabalhos da literatura envolvendo FJSPs e também pode ser considerado uma contribuição deste trabalho. O esquema de configurar o algoritmo de busca e avaliá-lo sobre os mesmos problemas, normalmente presente literatura (Zuo; Gong; Jiao, 2016; Kaplanoğlu, 2016), impossibilita mostrar a capacidade de generalização dos métodos adaptativos. Além disso, acredita-se que esse processo é mais similar ao que será empregado na prática, dado que normalmente não se tem tempo disponível para resolver o problema várias vezes. Esse tipo de experimento também é importante quando o tomador de decisão desconhece a característica das tarefas que vão entrar no meio, podendo ter diferentes características. Portanto, essa proposta poderá ser usada não apenas quando há métodos adaptativos envolvidos na resolução de FJSPs, mas também em outros problemas de programação de produção.

5.3.2.1 *Resolução dos problemas com 20 tarefas e 10 estágios*

Para essa análise serão usadas todas as instâncias de 20 tarefas e 10 estágios. Pode-se visualizar na Figura 33 que o algoritmo CPPC conseguiu resolver as instâncias desse tipo com menor perda de desempenho, ou seja, consegue resolver todas as instâncias possuindo o melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$. O resultado se manteve quando é avaliada a maior área normalizada, conforme a Figura 34. O CPPC é o algoritmo que apresenta o melhor desempenho na maioria das instâncias (em 65% delas).

Na Tabela 15 estão representados as medidas estatísticas quando são resolvidos os problemas de 20 tarefas e 10 estágios. O algoritmo CPPC possui melhor resultado quando comparado às médias e aos melhores resultados obtidos dentre os algoritmos. Já ao observar a mediana, apresentou resultados semelhantes, quando comparado ao método APEX e SLMABEX. A Figura 35 apresenta os *boxplots* para os algoritmos.

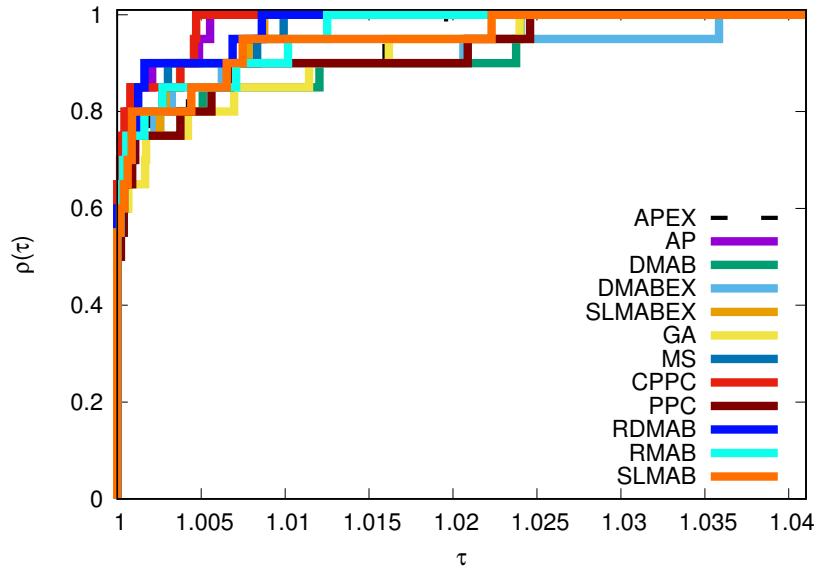

Figura 33 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 20 tarefas e 10 estágios.

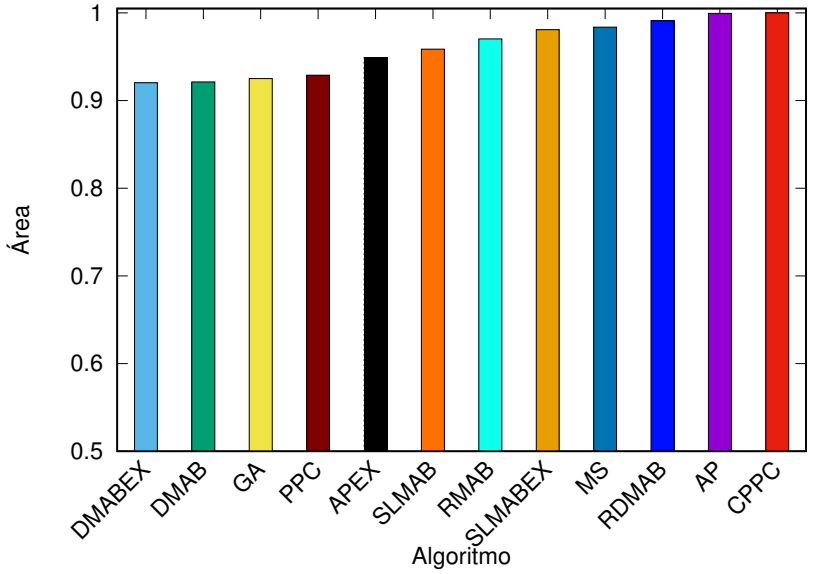

Figura 34 – Área normalizada dos PDs dos algoritmos ajustados através problemas pequenos, porém solucionando instâncias de 20 tarefas e 10 estágios.

Tabela 15 – Resultados estatísticos para as instâncias de 20 máquinas e 10 estágios, quando estão sendo resolvidas pelos algoritmos ajustados através de problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9979	0,9962	0,9988	0,9989	0,9968	0,9987	0,9967	0,9986	**0,9961**	0,9964	0,9971	0,9975
Desvio-padrão	0,0039	0,0088	0,0027	**0,0018**	0,0066	0,0030	0,0066	0,0027	0,0085	0,0071	0,0058	0,0047
Mínimo	0,9843	0,9674	0,9883	0,9943	0,9738	0,9885	0,9734	0,9891	**0,9654**	0,9720	0,9775	0,9829
Mediana	**0,9994**	0,9998	0,9998	0,9998	**0,9994**	0,9999	0,9995	0,9997	**0,9994**	1,0000	0,9998	0,9995
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

Ao realizar o teste de Kruskal-Wallis, foi encontrado um p-valor de 0,9925, e com

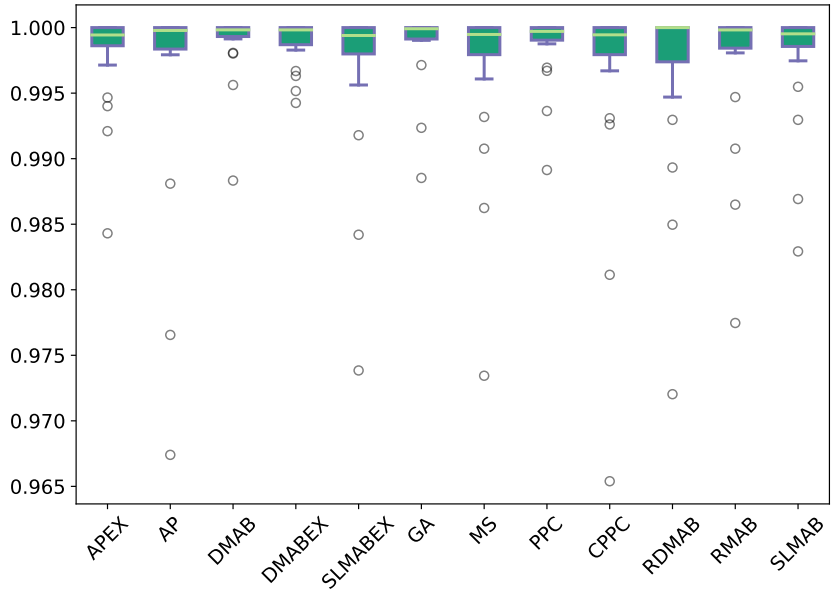

Figura 35 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos, ao resolverem as instâncias de 20 tarefas e 10 estágios.

uma confiança de 95%, não rejeita a hipótese de nulidade, ou seja, não se pode afirmar que os resultados obtidos pelos algoritmos são estatisticamente diferentes.

5.3.2.2 *Resolução dos problemas com 20 tarefas e 20 estágios*

Nessa seção as melhores configurações dos algoritmos para os problemas de 10 tarefas e 10 estágios são aplicadas para solucionar os problemas com 20 tarefas e 20 estágios. De acordo com a Figura 36 e a Figura 34, o algoritmo RDMAB conseguiu resolver os problemas com melhor melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), e também apresentou a melhor área normalizada. Para esses problemas, o RDMAB é o algoritmo que obtém os melhores resultados na maioria das instâncias (em 20% delas).

A Tabela 16 apresenta as medidas estatísticas, que mostram que o algoritmo RDMAB apresenta melhor média, o RMAB apresenta a melhor mediana, enquanto o MS obteve o melhor resultado dentre os valores mínimos e máximos. A variação dos valores observados pode ser vista na Figura 38.

Tabela 16 – Resultados estatísticos para as instâncias de 20 máquinas e 20 estágios, quando estão sendo resolvidas pelos ajustes realizados para problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9803	0,9827	0,9938	0,9933	0,9804	0,9905	0,9783	0,9949	0,9788	**0,9771**	0,9778	0,9794
Desvio-padrão	0,0088	0,0079	0,0059	0,0059	0,0074	0,0068	0,0078	**0,0055**	0,0067	0,0072	0,0066	0,0076
Mínimo	0,9675	0,9629	0,9813	0,9820	0,9680	0,9730	**0,9571**	0,9836	0,9672	0,9612	0,9649	0,9586
Mediana	0,9789	0,9840	0,9950	0,9940	0,9801	0,9910	0,9796	0,9962	0,9788	0,9787	**0,9772**	0,9799
Máximo	0,9966	0,9932	1,0000	1,0000	0,9918	1,0000	**0,9891**	1,0000	0,9959	0,9913	0,9902	0,9909

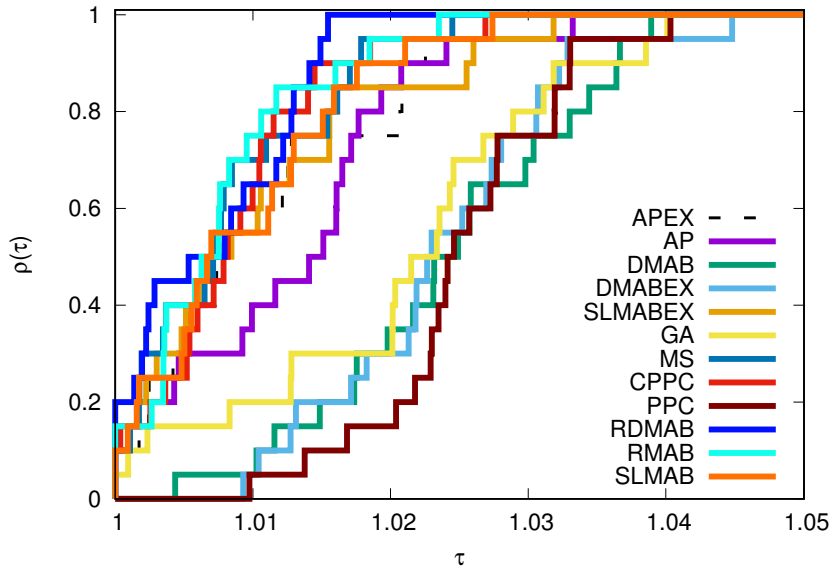

Figura 36 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 20 tarefas e 20 estágios.

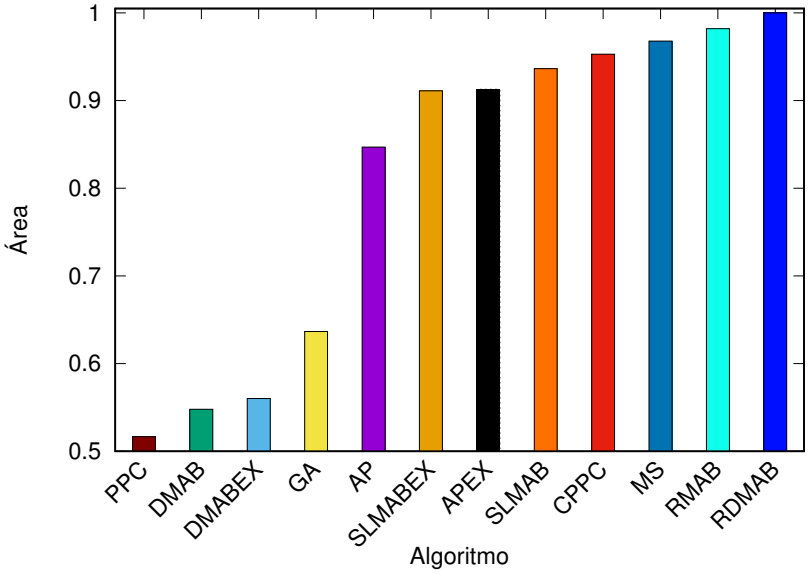

Figura 37 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 20 tarefas e 20 estágios.

Através de 95% de confiança, ao realizar o teste não paramétrico de Kruskal-Wallis, obteve-se o valor de 0,0000, portanto pode-se rejeitar a hipótese nula e identifica-se que há diferença estatística entre os valores obtidos pelos algoritmos. Dada a rejeição do teste de Kruskal-Wallis, foi realizado o teste de Conover-Iman, com 95% de confiança. Pode-se concluir através dos resultados obtidos, presentes no APÊNDICE B na Tabela 28, que o RMAB não é superior aos algoritmos APEX, SLMABEX, MS, CPPC, RDMAB e SLMAB. Porém, o RMAB é superior ao AP, DMAB, DMABEX, GA e PPC.

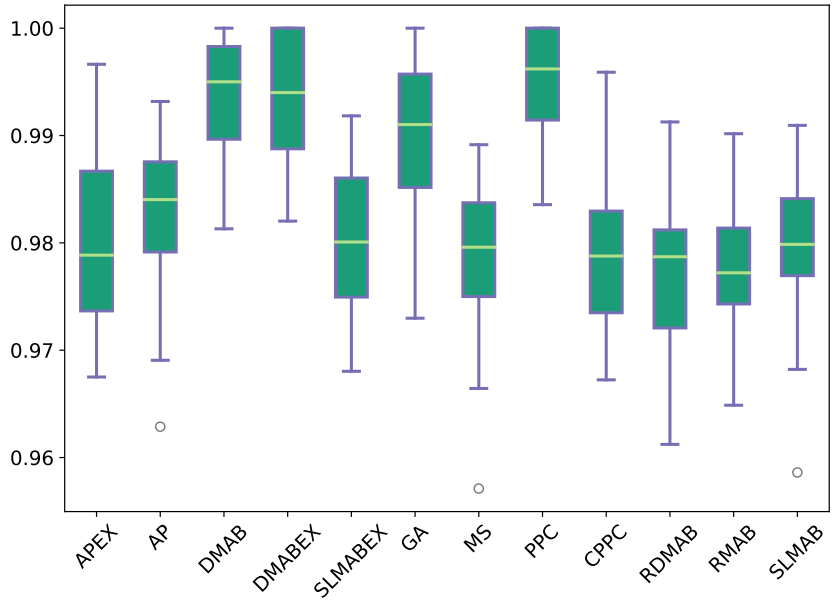

Figura 38 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem as instâncias de 20 tarefas e 20 estágios.

5.3.2.3 *Resolução dos problemas com 50 tarefas e 10 estágios*

A Figura 39 e a Figura 40 mostram os PDs quando são solucionados os problemas de 50 tarefas e 10 estágios, utilizando as configurações de problemas pequenos. Observa-se

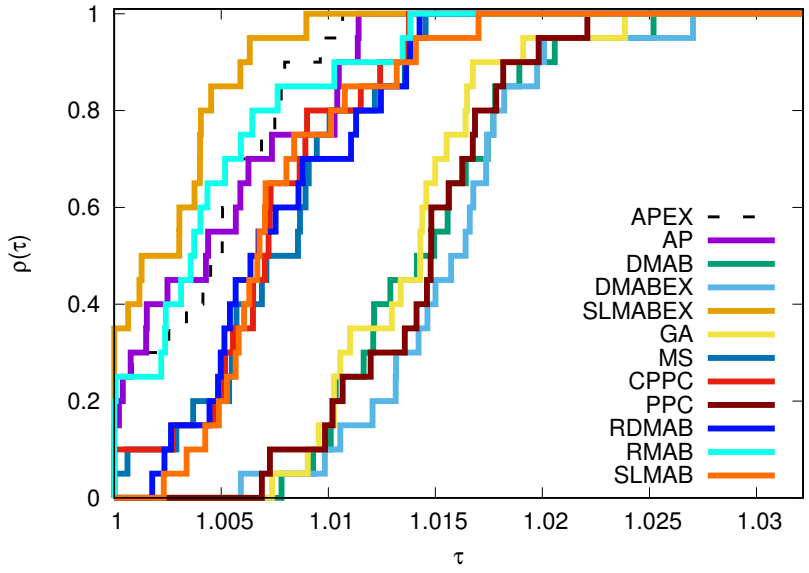

Figura 39 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas e 10 estágios.

que o SLMABEX apresenta os melhores resultados dos algoritmos para esse contexto, pois resolveu todos os problemas com melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), e com a maior área normalizada. Para esses problemas, o SLMABEX

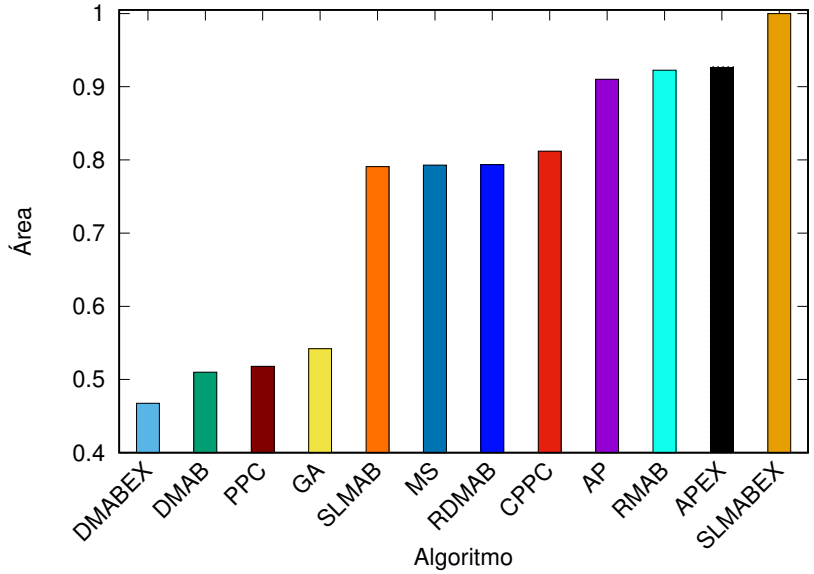

Figura 40 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas e 10 estágios.

obtém os melhores resultados na maioria das instâncias (em 35% delas).

As medidas estatísticas, disponíveis na Tabela 17 revela que o SLMABEX apresenta os melhores resultados para a média, para a mediana, para o valor máximo e para o valor mínimo. As variações dos resultados observados dos algoritmos podem ser visualizadas nos *boxplots* disponíveis na Figura 41.

Tabela 17 – Resultados estatísticos para as instâncias de 50 máquinas e 10 estágios, quando estão sendo resolvidas pelos ajustes realizados para problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9875	0,9879	0,9976	0,9986	**0,9858**	0,9968	0,9908	0,9974	0,9903	0,9907	0,9876	0,9908
Desvio-padrão	0,0050	0,0047	0,0025	**0,0022**	0,0058	**0,0022**	0,0037	0,0026	0,0041	0,0036	0,0043	0,0034
Mínimo	0,9781	0,9806	0,9924	0,9918	**0,9737**	0,9915	0,9855	0,9900	0,9807	0,9849	0,9794	0,9848
Mediana	0,9876	0,9887	0,9976	0,9996	**0,9844**	0,9972	0,9894	0,9972	0,9901	0,9910	0,9872	0,9904
Máximo	0,9965	0,9957	1,0000	1,0000	**0,9944**	1,0000	0,9972	1,0000	0,9963	0,9962	0,9961	0,9973

O teste estatístico de Kruskal-Wallis retornou o p-valor de 0,0000, logo conclui-se, para uma confiança de 95%, que os resultados obtidos do desempenho dos algoritmos são estatisticamente diferentes, uma vez que rejeitamos a hipótese nula. Os resultados do teste de Conover-Iman encontram-se no APÊNDICE B na Tabela 29. Então, com 95% de confiança, não se pode dizer que o SLMABEX possui desempenho superior ao AP, ao APEX e ao RMAB, mas é superior ao DMAB, DMABEX, GA, MS, PPC, CPPC, RDMAB, RMAB e SLMAB.

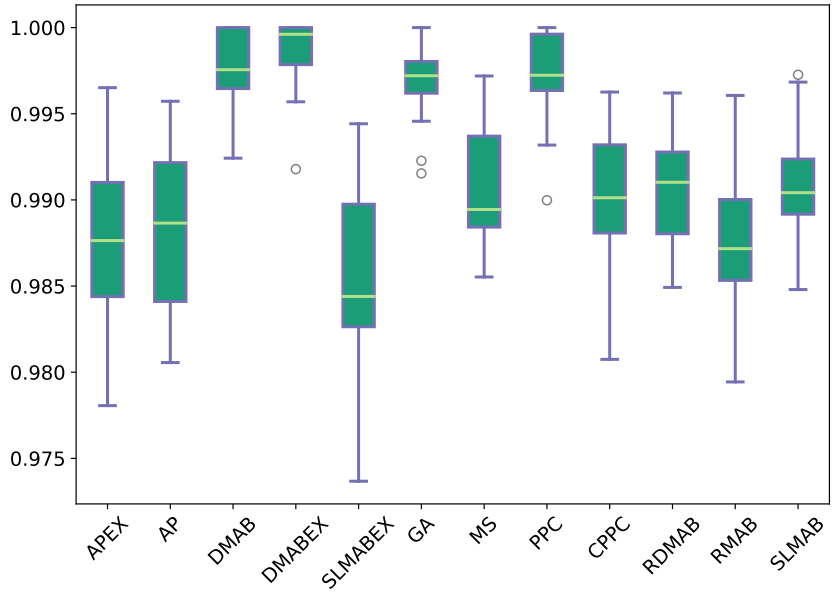

Figura 41 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem as instâncias de 50 tarefas e 10 estágios.

5.3.2.4 *Resolução dos problemas com 50 tarefas e 50 estágios*

Ao comparar os algoritmos utilizados para solucionar os problemas com 50 tarefas e 50 estágios, verifica-se que o RMAB é o algoritmo com melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), o que pode ser visto na Figura 27. Na Figura 28, verifica-se que esse algoritmo também apresenta a melhor área normalizada. O APEX é o algoritmo que apresenta os melhores resultados para a maiorias das instâncias (em 30% delas).

As medidas estatísticas disponíveis na Tabela 18 e as variações dos resultados observados apresentados na Figura 44 mostram que o RMAB apresentou a melhor média e mediana entre os resultados, enquanto o APEX apresentou o melhor valor máximo e mínimo.

Tabela 18 – Resultados estatísticos para as instâncias de 50 máquinas e 50 estágios, quando estão sendo resolvidas pelos ajustes realizados para os problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	20	20	20	20	20	20	20	20	20	20	20	20
Média	0,9754	0,9763	0,9969	0,9980	0,9772	0,9937	0,9835	0,9954	0,9803	0,9827	**0,9743**	0,9827
Desvio-padrão	0,0054	0,0063	**0,0029**	**0,0029**	0,0074	0,0051	0,0060	0,0036	0,0049	0,0050	0,0053	0,0053
Mínimo	**0,9612**	0,9658	0,9910	0,9888	0,9642	0,9835	0,9728	0,9885	0,9730	0,9724	0,9648	0,9764
Mediana	0,9771	0,9778	0,9971	0,9993	0,9774	0,9932	0,9833	0,9957	0,9805	0,9830	**0,9741**	0,9819
Máximo	**0,9818**	0,9851	1,0000	1,0000	0,9908	1,0000	0,9964	1,0000	0,9873	0,9921	0,9839	0,9976

O teste de Kruskal-Wallis retornou o p-valor de 0,0000 e pode-se afirmar que os algoritmos são diferentes estatisticamente, com 95% de confiança. Por isso, o teste de Conover-Iman foi realizado e retornou que os resultados obtidos do RMAB são superiores aos resultados do DMAB, DMABEX, GA, MS, PPC, CPPC, RDMAB, e SLMAB, mas

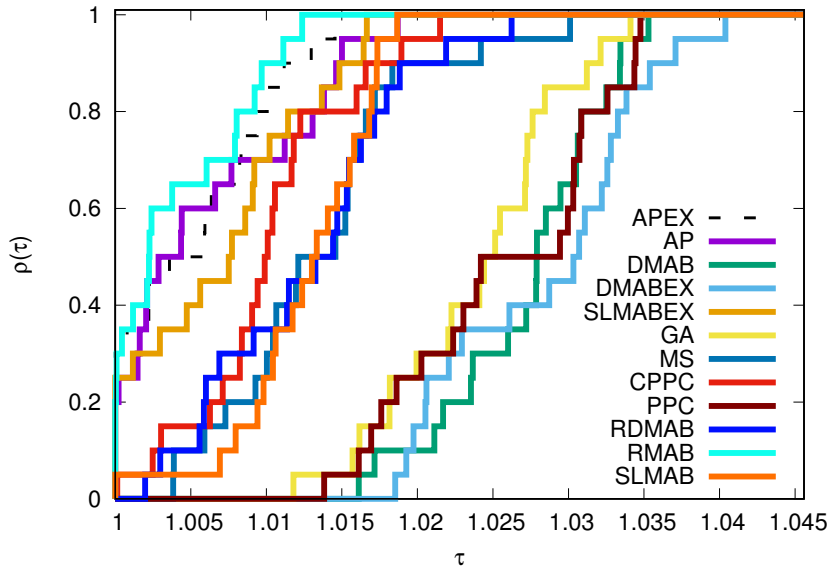

Figura 42 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas e 50 estágios.

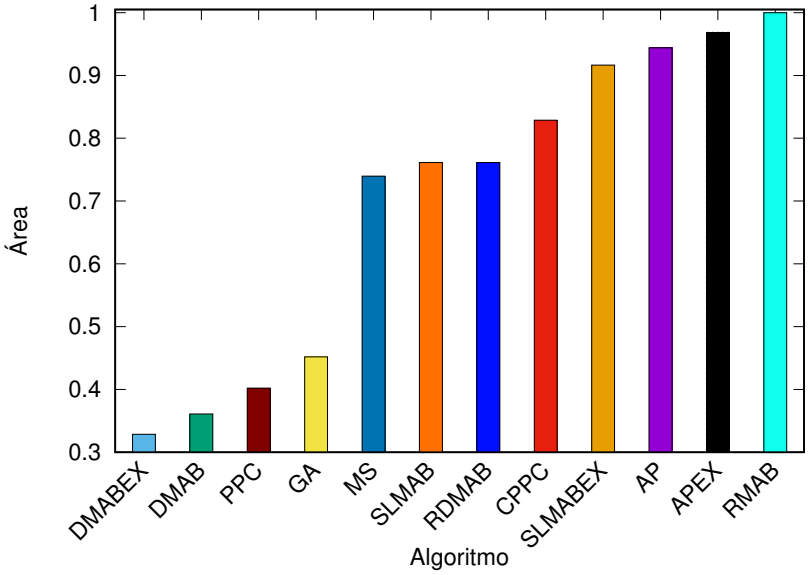

Figura 43 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas e 50 estágios.

não é estatisticamente superior aos resultados obtidos do AP, APEX e SLMABEX. O resultados do teste de Conover-Iman encontram-se no APÊNDICE B na Tabela 26

5.3.2.5 *Resolução dos problemas com 20 tarefas*

Nessa categoria, os problemas com 20 tarefas serão analisados com os diferentes algoritmos ajustados pelas instâncias pequenas. Portanto, 40 instâncias foram executadas com as melhores configurações de cada técnica disponível. A Figura 45 e a Figura 46

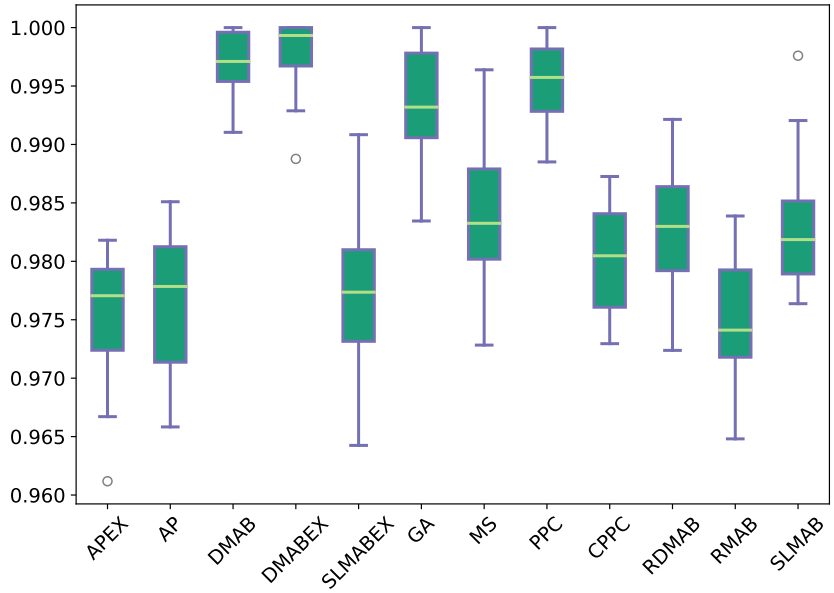

Figura 44 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem as instâncias de 50 tarefas e 50 estágios.

mostram que o RDMAB apresentou as melhores configurações para solucionar os problemas dessa categoria, tanto ao analisar o melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$), quanto por possuir a melhor área normalizada. O RDMAB é o algoritmo que apresenta os melhores resultados para a maioria das instâncias (em 40% delas).

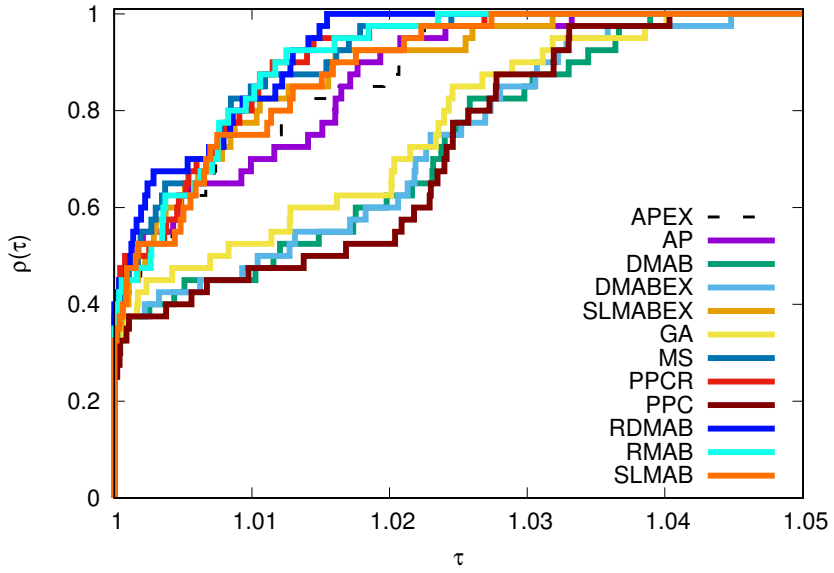

Figura 45 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 20 tarefas.

As medidas estatísticas presentes na Tabela 19 e as variações dos resultados obtidos na Figura 47 mostram que o RDMAB é o algoritmo que apresenta o melhor resultado

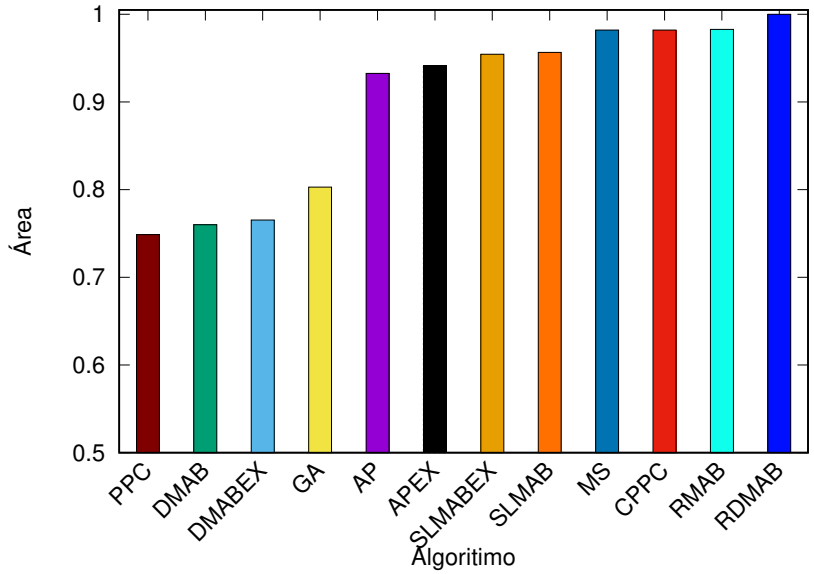

Figura 46 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 20 tarefas.

médio e também a melhor mediana. O MS é o algoritmo que apresentou o melhor valor mínimo.

Tabela 19 – Resultados estatísticos para as instâncias de 20 máquinas, quando estão sendo resolvidas pelos ajustes para os problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	40	40	40	40	40	40	40	40	40	40	40	40
Média	0,9891	0,9894	0,9963	0,9961	0,9886	0,9946	0,9875	0,9967	0,9875	**0,9868**	0,9875	0,9885
Desvio-padrão	0,0112	0,0107	0,0052	0,0051	0,0108	0,0066	0,0117	**0,0047**	0,0116	0,0121	0,0116	0,0111
Mínimo	0,9675	0,9629	0,9813	0,9820	0,9680	0,9730	**0,9571**	0,9836	0,9654	0,9612	0,9649	0,9586
Mediana	0,9927	0,9912	0,9991	0,9990	0,9907	0,9981	0,9870	0,9993	0,9865	**0,9848**	0,9867	0,9880
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

O teste de Kruskal-Wallis retornou o p-valor de 0,0000, portanto há diferença estatística entre os algoritmos testados. Consequentemente, o teste de Conover-Iman foi realizado e seus resultados encontram-se no APÊNDICE B na Tabela 31. Com uma confiança de 95%, pode-se afirmar que o RDMAB tem desempenho superior ao DMAB, ao DMABEX, ao GA e ao PPC, porém não é superior estatisticamente AP, APEX, SLMABEX, MS, CPPC, RDMAB, RMAB e SLMAB.

5.3.2.6 *Resolução dos problemas com 50 tarefas*

As 40 instâncias que envolvem os problemas de 50 tarefas foram usadas nessa categoria. Pode-se observar pela Figura 48 e pela Figura 49 que o APEX e o RMAB apresentaram melhores resultados ao avaliar o melhor desempenho relativo no pior caso; menor τ tal que $\rho(\tau) = 1$. No entanto, o algoritmo RMAB apresentou a melhor área normalizada. O SLMABEX é o algoritmo que apresenta o melhor resultado para a maioria das instâncias (em 30% delas).

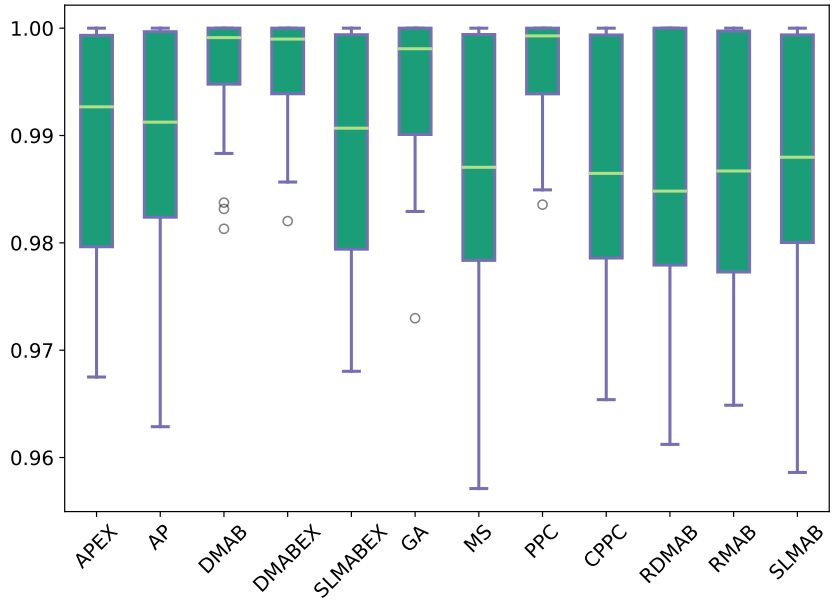

Figura 47 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem as instâncias de 20 tarefas.

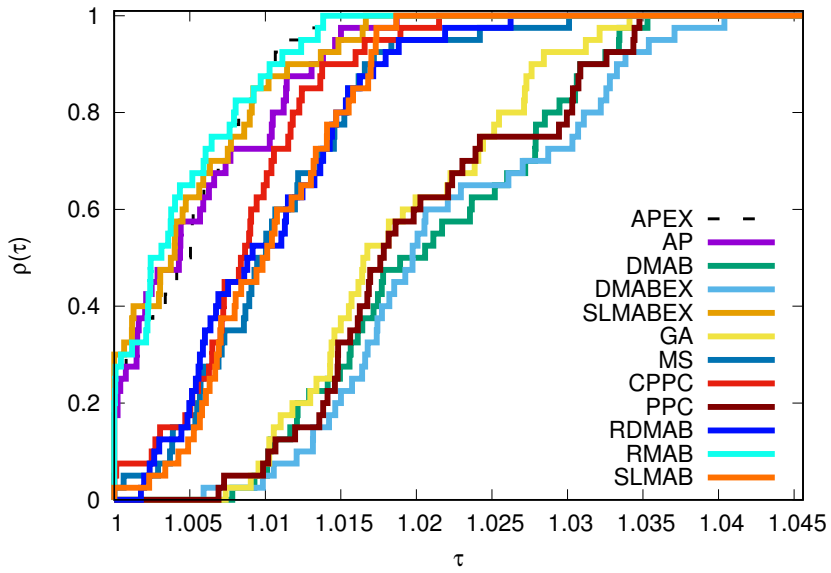

Figura 48 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas.

As medidas estatísticas presentes na Tabela 20 e na Figura 50 mostram que os resultados obtidos pelo algoritmo RMAB apresenta a melhor média e a melhor mediana. O APEX apresentou o melhor mínimo e o SLMABEX apresentou o melhor máximo.

O teste não paramétrico de Kruskal-Wallis retornou, para esses problemas, um p-valor de 0,0000, então pode-se rejeitar a hipótese nula, com 95% de confiança, portanto os resultados obtidos pelos algoritmos são estatisticamente diferentes. Portanto, foi realizado o teste de Conover-Iman para comparar os algoritmos. Os resultados encontram-se no

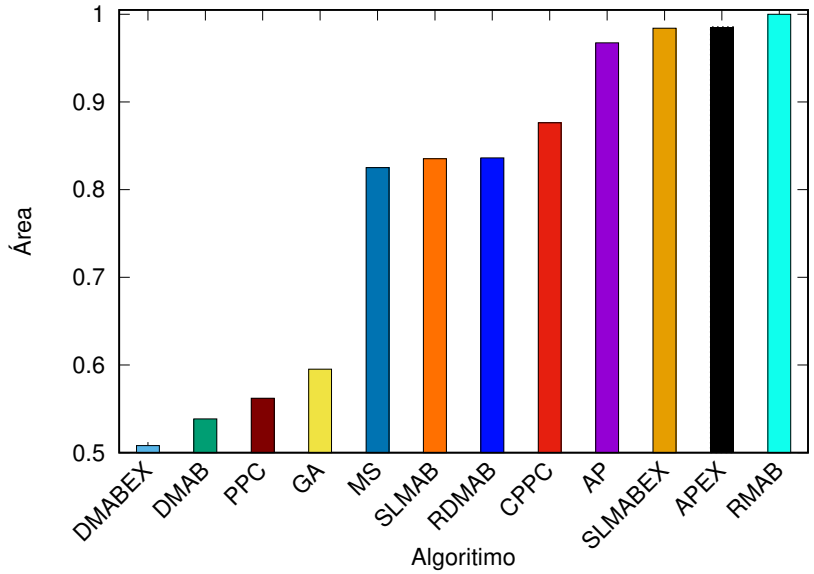

Figura 49 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando instâncias de 50 tarefas.

Tabela 20 – Resultados estatísticos para as instâncias de 50 máquinas, quando estão sendo resolvidas pelos ajustes para os problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	40	40	40	40	40	40	40	40	40	40	40	40
Média	0,9815	0,9821	0,9972	0,9983	0,9815	0,9952	0,9871	0,9964	0,9853	0,9867	**0,9810**	0,9868
Desvio-padrão	0,0080	0,0080	0,0027	**0,0026**	0,0079	0,0042	0,0062	0,0032	0,0067	0,0059	0,0083	0,0060
Mínimo	**0,9612**	0,9658	0,9910	0,9888	0,9642	0,9835	0,9728	0,9885	0,9730	0,9724	0,9648	0,9764
Mediana	0,9810	0,9827	0,9976	0,9996	0,9826	0,9966	0,9880	0,9969	0,9866	0,9871	**0,9806**	0,9866
Máximo	0,9965	0,9957	1,0000	1,0000	**0,9944**	1,0000	0,9972	1,0000	0,9963	0,9962	0,9961	0,9976

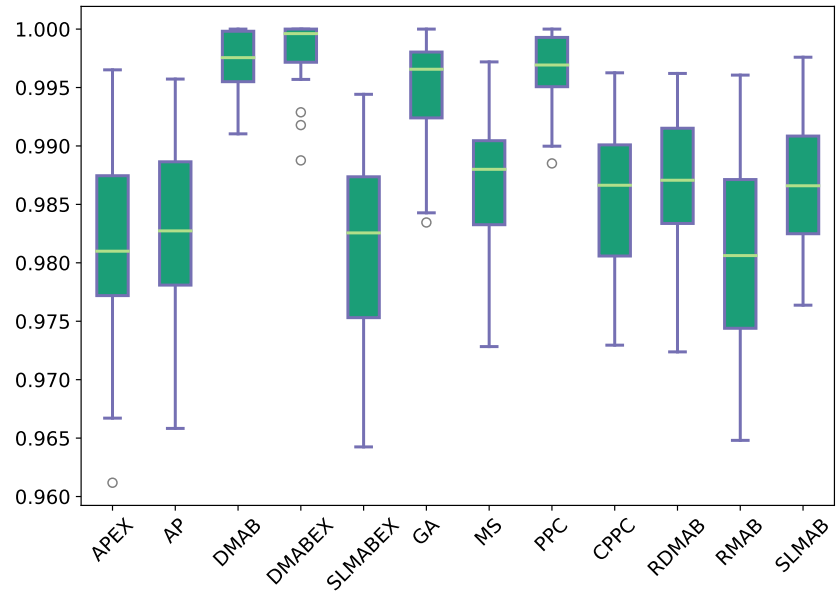

Figura 50 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem as instâncias de 50 tarefas.

APÊNDICE B na Tabela 32, os quais mostram que o RMAB possui desempenho superior ao: DMAB, DMABEX, GA, MS, PPC, CPPC, RDMAB e SLMAB, apesar disso não é superior aos algoritmos AP, APEX e SLMABEX.

5.3.2.7 *Resolução de todos os problemas*

Nessa categoria todas as 100 instâncias disponíveis na fase de comparação de testes serão solucionadas através das configurações de 10 tarefas e 10 estágios. O melhor algoritmo para resolver as instâncias com o melhor desempenho relativo no pior caso (menor τ tal que $\rho(\tau) = 1$) é o RDMAB, porém o RMAB possui a melhor área normalizada. O SLMABEX é o algoritmo que apresenta os melhores resultados na maioria das instâncias (em 42% delas).

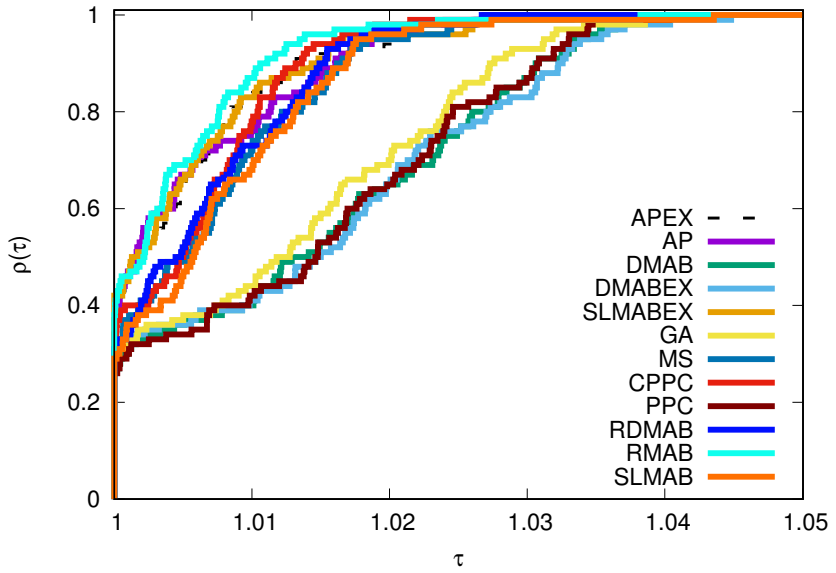

Figura 51 – PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando todas as instâncias.

As medidas estatísticas apresentados na Tabela 21 e os *boxplots* presentes na Figura 53 mostram que o RMAB possui o melhor valor médio e também a melhor mediana. No entanto, o MS é o algoritmo que mais conseguiu se deslocar dos outros algoritmos, obtendo o melhor valor mínimo.

Tabela 21 – Resultados estatísticos para todas as instâncias, quando estão sendo solucionadas pelos ajuste para os problemas pequenos.

	APEX	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
Quantidade	100	100	100	100	100	100	100	100	100	100	100	100
Média	0,9877	0,9879	0,9966	0,9968	0,9876	0,9951	0,9894	0,9965	0,9884	0,9889	**0,9869**	0,9897
Desvio-padrão	0,0108	0,0105	0,0055	0,0060	0,0107	0,0063	0,0095	**0,0054**	0,0101	0,0098	0,0111	0,0092
Mínimo	0,9612	0,9629	0,9644	0,9583	0,9642	0,9658	**0,9571**	0,9647	0,9626	0,9612	0,9648	0,9586
Mediana	0,9891	0,9887	0,9987	0,9996	0,9884	0,9972	0,9897	0,9989	0,9893	0,9894	**0,9868**	0,9904
Máximo	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

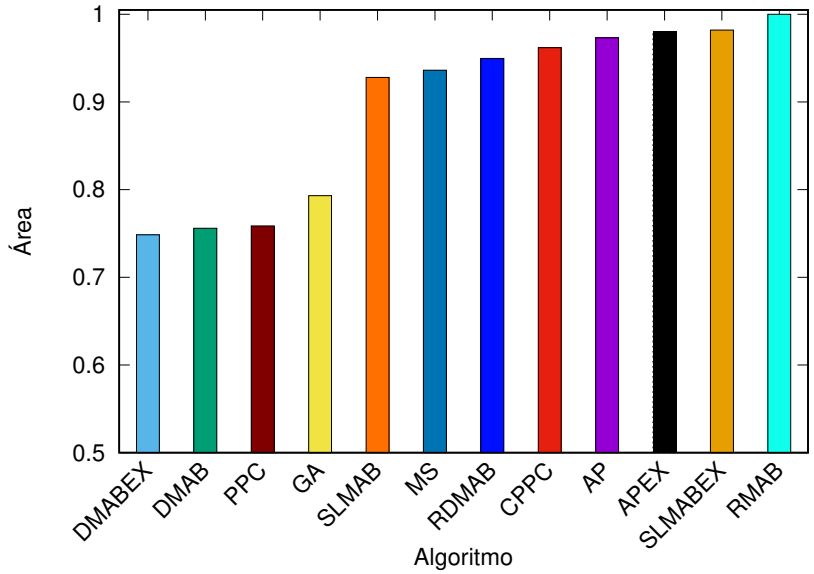

Figura 52 – Área normalizada dos PDs dos algoritmos ajustados através de problemas pequenos, porém solucionando todas as instâncias.

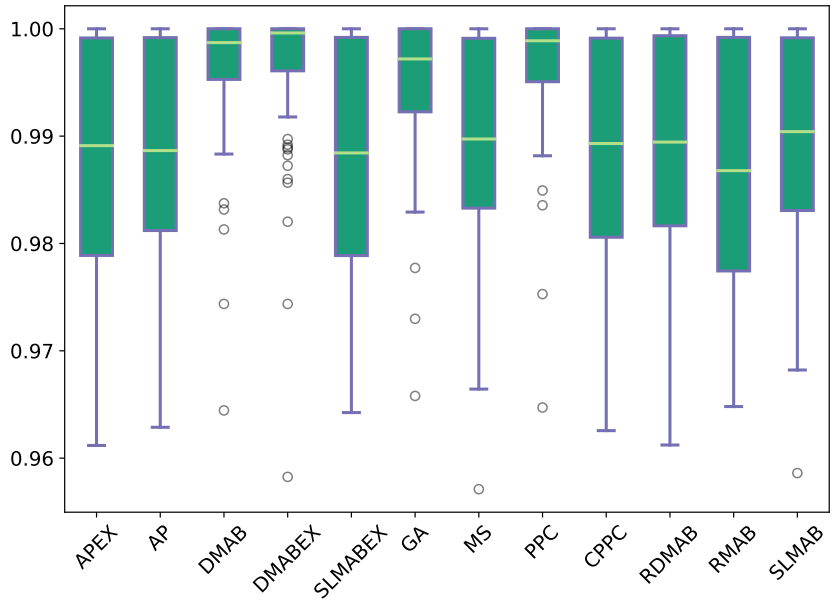

Figura 53 – *Boxplot* dos resultados dos algoritmos ajustados a problemas pequenos ao resolverem todas as instâncias.

O teste de Kruskal-Wallis retornou o p-valor de 0,0000, e com uma confiança de 95%, mostra que os resultados obtidos pelos algoritmos são estatisticamente diferentes. Consequentemente, o teste de Conover-Iman foi realizado, o qual tem seus resultados disponíveis no APÊNDICE B na Tabela 33. O RMAB apresentou desempenho superior aos algoritmos DMAB, DMABEX, GA, e PPC, segundo os testes realizados realizados, cuja confiança é de 95%. No entanto, o RMAB não se mostrou estatisticamente superior ao AP, APEX, SLMABEX, MS, CPPC, RDMAB e SLMAB.

5.3.3 **Conclusões da Análise II**

Ao comparar as meta-heurísticas para solucionar FJSPs, num sistema de produção mais restrito do que a Análise I, quanto ao tempo de configuração dos algoritmos, com alta taxa de entrada e saída de tarefas do ambiente, observa-se que os algoritmos que podem adaptar seus componentes sobressaíram-se na maioria dos casos, como é demonstrado pela Tabela 22.

Tabela 22 – Algoritmos que obtiveram resultados estatisticamente superiores na Análise II (por categoria).

	20×10	20×20	50×10	50×50	20 tarefas	50 tarefas	TODOS
AP	X		X	X	X	X	X
APEX	X	X	X	X	X	X	X
DMAB	X						
DMABEX	X						
SLMAB	X	X			X		X
SLMABEX	X	X	X	X	X	X	X
PPC	X						
CPPC	X	X			X		X
RDMAB	X	X			X		X
RMAB	X	X	X	X	X	X	X
GA	X						
MS	X	X			X		X

Ao aplicar os algoritmos nas instâncias de 20 tarefas e 10 estágios, o CPPC apresentou os melhores resultados, no entanto não é possível diferenciá-lo estatisticamente dos demais algoritmos, de acordo com o teste de Kruskal-Wallis realizado. Nos testes realizados para 20 tarefas e 20 estágios, o algoritmo de melhor desempenho foi o RDMAB, que obteve resultados estatisticamente melhores do que aqueles encontrados pelo GA e pelos outros AGAs. No entanto, o método não obteve resultado superior ao MS. O RDMAB destacou-se também ao solucionar as instâncias de 20 tarefas, independentemente do número de máquinas. Seu desempenho foi superior ao do GA e dos esquemas adaptativos DMAB, DMABEX e PPC. No entanto, não houve novamente diferença entre os resultados obtidos pelo RDMAB e pelo MS. Para esses problemas, o MS é o algoritmo mais rápido de configurar, por possuir somente o parâmetro de tamanho da população.

Para as instâncias de 50 tarefas e 10 estágios, o melhor algoritmo foi o SLMABEX, que apresentou desempenho superior ao GA, ao MS e também a alguns AGAs, não sendo superior ao AP, APEX e RMAB. Para os problemas de 50 tarefas e 50 estágios, o RMAB apresentou o melhor resultado médio entre os algoritmos e, estatisticamente, o teste de Conover-Iman indicou que o desempenho dele é superior ao DMAB, ao DMABEX, ao MS, ao PPC, ao CPPC, ao RDMAB, ao SLMBAB e também superior ao GA. Quando

todas as instâncias de 50 tarefas estão agrupadas, o desempenho do RMAB é superior ao dos algoritmos DMAB, DMABEX, GA, MS,PPC, CPPC, RDMAB e ao SLMAB. Nos problemas com 50 tarefas independente do número de estágios testados aqui, os algoritmos adaptativos, no qual podemos destacar o APEX, RMAB e SLMABEX, obtiveram resultados estatisticamente superiores aos dos demais algoritmos. Finalmente, quando é realizada uma análise sobre todas as instâncias, o RMAB também é o algoritmo que apresenta melhor desempenho médio, sendo estatisticamente superior ao DMAB, ao DMABEX, ao GA e ao PPC.

Conclui-se pela Análise II que quando o cenário de tempo é muito restrito e o tomador de decisão deve configurar os algoritmos muitas vezes, o MS é a melhor escolha quando estão sendo resolvidos problemas pequenos (10 tarefas e 10 estágios) e médios (20 tarefas independente do número de estágios). Isso ocorre porque esse algoritmo possui o menor tempo de configuração, uma vez que deve-se escolher somente os parâmetros de tamanho da população, se vai utilizar ou não busca local, e se vai usar ou não a heurística NEH na inicialização. Porém, se houver uma necessidade de ajuste mais fino e houver tempo para configurar alguns poucos parâmetros, os métodos adaptativos podem conseguir resultados melhores. Já quando estão sendo resolvidos problemas maiores (50 tarefas independente do número de estágios), os métodos adaptativos mostraram melhor capacidade de generalização e, por isso, são os métodos mais aconselhados para a aplicação. Já quando são analisados todos os problemas conjuntamente, o método MS é o mais indicado quando há restrição de tempo para configuração dos algoritmos e o método RMAB é o melhor algoritmo quando deseja-se os melhores resultados.

Ainda, entre os métodos adaptativos, podem ser destacados os algoritmos APEX, RMAB e SLMABEX porque esses algoritmos estão entre aqueles que obtiveram os melhores resultados estatisticamente em todas as categorias presentes na Análise II.

6 CONCLUSÃO

Foram estudados nesta dissertação os Algoritmos Genéticos Adaptativos (AGAs) para solucionar Problemas de Sequenciamento do tipo Job-Shop Flexível (FJSPs), bem como foram propostas modificações para as técnicas adaptativas existentes. Na construção do modelo, foram utilizadas outras heurísticas para compor o AGA, como a heurística NEH, e a busca local proposta por Ruiz & Stützle (2007). A representação utilizada é a JBR, e por isso alguns operadores de recombinação e mutação específicos foram usados.

Para avaliar a qualidade dos diferentes AGAs, e também compará-los com um algoritmo genético padrão e com um segundo algoritmo que escolhe aleatoriamente os operadores de movimento, foram realizados experimentos computacionais com instâncias criadas de acordo com um gerador de instâncias da literatura. Estes experimentos foram executados com o intuito de minimizar o *makespan* e duas análises foram realizadas.

Na Análise I as diferentes técnicas foram configuradas em diferentes categorias. A melhor configuração de cada técnica e cada categoria foi então selecionada para uma segunda fase, na qual as diferentes técnicas foram comparadas.

O melhor algoritmo para os problemas de até 20 tarefas e 20 estágios foi o MS (método que seleciona os algoritmos de acordo com uma distribuição de probabilidades uniforme), uma vez que as demais técnicas não superaram seus resultados, e sendo que o MS é o algoritmo que é mais rápido de configurar, porque tem menos parâmetros. Para os problemas de 50 tarefas independente do número de estágios o APEX (*Adaptive Pursuit* com valor extremo) é o algoritmo recomendado, uma vez que ele possui menos parâmetros para ser configurado. Vale ressaltar, que se for necessário implementar uma única técnica adaptativa para solucionar problemas específicos, que se encaixam nas categorias de 10 tarefas e 10 estágios até 50 tarefas e 50 estágios, o APEX é o algoritmo recomendado, porque é a técnica que possui menos parâmetros e que não é superada por nenhum dos outros algoritmos. Já quando diferentes tipos de problemas estão sendo solucionados, o RMAB (*Rank-based Multi-Armed Bandit*) é o melhor algoritmo, uma vez que tem menos parâmetros que o GA (Algoritmo Genético tradicional desenvolvido para comparar com os algoritmos) e seus resultados foram estatisticamente superiores aos das demais técnicas.

Na Análise II, as técnicas são configuradas a partir de instâncias pequenas, para o problema de 10 tarefas e 10 estágios, e as melhores configurações de cada algoritmo foram aplicadas para solucionar problemas maiores.

Quando são analisados os problemas de 20 tarefas independente do número de estágios, o MS é o algoritmo recomendado. Isso porque os resultados desse algoritmo são melhores ou similares aos dos melhores algoritmos e ainda possui menos parâmetros.

Quando a análise comparativa é feita levando-se em consideração os problemas com 50 tarefas, independente do número de estágios, ressalta-se o uso de três algoritmos adaptativos: o APEX, o RMAB e o Sliding Multi-Armed Bandit com valor Extremo (SLMABEX). Já quando estão sendo analisadas as soluções obtidas a partir de todas as instâncias, o MS volta a ser uma boa técnica a ser adotada, uma vez que seus resultados são tão bons quanto as soluções obtidas pelos demais algoritmos, sendo que ainda possui o menor tempo de configuração.

Portanto, identifica-se que para problemas pequenos (10 tarefas e 10 estágios) e médio (20 tarefas independente do número de estágios) a utilização de um algoritmo que faz a escolha aleatória dos componentes é recomendado quando não há tempo disponível para configurar corretamente um algoritmo. No entanto, ressalta-se a importância do método APEX, que na Análise I só não é interessante quando há diferentes classes de problemas (categoria TODOS), e na Análise II está entre os melhores algoritmos em todas as categorias.

Portanto, o MS e os métodos adaptativos se mostraram melhores escolhas quando comparado ao GA, quando deseja-se reduzir o tempo de configuração, ou ainda quando almeja-se aplicá-los a diferentes classes de problemas a partir de uma única configuração. Para os problemas maiores recomenda-se o APEX e o para problemas de classes diferentes o RMAB.

Contudo, constata-se também que os AGs são melhores escolhas quando há a possibilidade de realizar um ajuste das configurações, principalmente tratando-se de instâncias maiores, a partir de 20 tarefas e 20 estágios.

Destaca-se que o objetivo principal dessa dissertação é avaliar os AGAs aplicados aos FJSPs a fim de reduzir os problemas de configuração desses algoritmos e, desse modo, identificar alguns algoritmos que se sobressaem nos resultados obtidos. Pode-se destacar assim o RMAB, o APEX e o SLMABEX.

Duas linhas diferentes são observadas para realizar trabalhos futuros. A primeira, refere-se ao estudo dos problemas de FJSPs, no qual pode-se aplicar as técnicas desenvolvidas aqui em modelos mais complexos, tais como: inclusão de tempo de preparação (*setup*), ambientes de máquinas não relacionadas, ou ambientes sem tempo de espera entre os estágios. Já na perspectiva do algoritmo, pode-se avaliar outras técnicas de busca local para serem inseridas e selecionadas pelas técnicas adaptativas. Além disso, é possível realizar um trabalho para verificar qual dos algoritmos converge mais rápido para uma solução pré-estabelecida. Por fim, ressalta-se a importância de desenvolver um Algoritmo Genético Adaptativo Multiobjetivo para solucionar FJSPs, uma vez que geralmente os problemas reais envolvem a resolução de dois ou mais objetivos ao mesmo tempo, como

minimizar o *makespan*, minimizar o atraso total ponderado e maximizar a utilização das máquinas.

REFERÊNCIAS

ALETI, Aldeida; MOSER, Irene. Predictive parameter control. In: ACM. **Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation**. [S.l.], 2011. p. 561–568.

ALETI, Aldeida; MOSER, Irene. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. **ACM Computing Surveys (CSUR)**, ACM, v. 49, n. 3, p. 56, 2016.

ARROYO, José Elias Claudio; PEREIRA, Ana Amélia de Souza. A grasp heuristic for the multi-objective permutation flowshop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 55, n. 5, p. 741–753, 2011.

AUER, Peter; CESA-BIANCHI, Nicolo; FISCHER, Paul. Finite-time analysis of the multiarmed bandit problem. **Machine learning**, v. 47, n. 2-3, p. 235–256, 2002.

BENJAMINI, Yoav; HOCHBERG, Yosef. Controlling the false discovery rate: a practical and powerful approach to multiple testing. **Journal of the royal statistical society. Series B (Methodological)**, JSTOR, p. 289–300, 1995.

ÇALIŞ, Banu; BULKAN, Serol. A research survey: review of AI solution strategies of job shop scheduling problem. **Journal of Intelligent Manufacturing**, v. 26, n. 5, p. 961–973, out. 2015.

CHAMBERS, Stuart; JOHNSTON, Robert; SLACK, Nigel. Administração da produção. **São Paulo: Atlas**, 2002.

CHANG, Pei-Chann; HSIEH, Jih-Chang; WANG, Chih-Yuan. Adaptive multi-objective genetic algorithms for scheduling of drilling operation in printed circuit board industry. **Applied Soft Computing**, v. 7, n. 3, p. 800–806, jun. 2007.

CHAUDHRY, Imran Ali; KHAN, Abid Ali. A research survey: review of flexible job shop scheduling techniques. **International Transactions in Operational Research**, v. 23, n. 3, p. 551–591, maio 2016.

CONOVER, WJ; IMAN, Ronald L. On multiple-comparisons procedures. **Los Alamos Sci. Lab. Tech. Rep. LA-7677-MS**, 1979.

da Costa, Luis; FIALHO, Alvaro; SCHOENAUER, Marc; SEBAG, Michèle. Adaptive operator selection with dynamic multi-armed bandits. In: **Proceedings of the 10th annual Conference on Genetic and Evolutionary Computation**. [S.l.]: ACM, 2008. p. 913–920.

DAVIS, Lawrence. Adapting operator probabilities in genetic algorithms. In: **International Conference on Genetic Algorithms**. [S.l.: s.n.], 1989. p. 61–69.

DOLAN, Elizabeth D; MORÉ, Jorge J. Benchmarking optimization software with performance profiles. **Mathematical programming**, Springer, v. 91, n. 2, p. 201–213, 2002.

EIBEN, Ágoston E.; HINTERDING, Robert; MICHALEWICZ, Zbigniew. Parameter control in evolutionary algorithms. **IEEE Transactions on evolutionary computation**, v. 3, p. 124–141, 1999.

EIBEN, Agoston E; SMITH, James E *et al.* **Introduction to evolutionary computing**. [S.l.]: Springer, 2003.

FERREIRA, Guilherme Souza; BERNARDINO, Heder Soares. An adaptive genetic algorithm for solving job-shop scheduling problems. In: **XXXVIII CILAMCE – Ibero-Latin American Congress on Computational Methods in Engineering**. Florianópolis, SC: ABMEC, 2017. p. 1–18.

FERREIRA, Guilherme Souza; BERNARDINO, Heder Soares. An adaptive pursuit genetic algorithm for solving job-shop scheduling problems. In: **Congresso Brasileiro de Inteligência Computacional**. Niterói, RJ: ABRICOM, 2017. p. 1–12.

FIALHO, Álvaro; COSTA, Luis Da; SCHOENAUER, Marc; SEBAG, Michèle. Analyzing bandit-based adaptive operator selection mechanisms. **Annals of Mathematics and Artificial Intelligence**, v. 60, n. 1, p. 25–64, 2010.

FIALHO, Álvaro; SCHOENAUER, Marc; SEBAG, Michèle. Toward comparison-based adaptive operator selection. In: **Proceedings of the 12th annual conference on Genetic and evolutionary computation**. [S.l.]: ACM, 2010. p. 767–774.

FIALHO, Álvaro. **Adaptive Operator Selection for Optimization**. Tese (Doutorado) — Université Paris Sud XI, 2011.

FIALHO, Álvaro; COSTA, Luis Da; SCHOENAUER, Marc; SEBAG, Michèle. Extreme value based adaptive operator selection. In: **International Conference on Parallel Problem Solving from Nature (PPSN)**. [S.l.]: Springer, 2008. p. 175–184.

GOLDBERG, David E. Probability matching, the magnitude of reinforcement, and classifier system bidding. **Machine Learning**, Springer, v. 5, n. 4, p. 407–425, 1990.

GRAHAM, Ronald L; LAWLER, Eugene L; LENSTRA, Jan Karel; KAN, AHG Rinnooy. Optimization and approximation in deterministic sequencing and scheduling: a survey. **Annals of discrete mathematics**, Elsevier, v. 5, p. 287–326, 1979.

HART, Emma; ROSS, Peter; CORNE, David. Evolutionary scheduling: A review. **Genetic Programming and Evolvable Machines**, Springer, v. 6, n. 2, p. 191–220, 2005.

HINTERDING, Robert; MICHALEWICZ, Zbigniew; EIBEN, Agoston E. Adaptation in evolutionary computation: A survey. In: **International Conference on Evolutionary Computation**. [S.l.]: IEEE, 1997. p. 65–69.

HOLLAND, John Henry. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI, 1975.

HONGYAN, Xie; HONG, Huo. Research on job-shop scheduling problem based on Self-Adaptation Genetic Algorithm. In: **International Conference on Logistics Systems and Intelligent Management**. [S.l.]: IEEE, 2010. v. 2, p. 940–943.

HUANG, Ming; WANG, Lu-ming; LIANG, Xu. An improved adaptive genetic algorithm in flexible job shop scheduling. In: **International Conference on Cloud Computing and Internet of Things (CCIOT)**. [S.l.]: IEEE, 2016. p. 177–184.

JAIN, Anant Singh; MEERAN, Sheik. Deterministic job-shop scheduling: Past, present and future. **European journal of operational research**, Elsevier, v. 113, n. 2, p. 390–434, 1999.

JORAPUR, Vedavyasrao; PURANIK, V. S.; DESHPANDE, A. S.; SHARMA, M. R. Comparative Study of Different Representations in Genetic Algorithms for Job Shop Scheduling Problem. **Journal of Software Engineering and Applications**, v. 07, n. 07, p. 571–580, 2014.

KAPLANOĞLU, Vahit. An object-oriented approach for multi-objective flexible job-shop scheduling problem. **Expert Systems with Applications**, v. 45, p. 71–84, mar. 2016.

KARAFOTIAS, Giorgos; HOOGENDOORN, Mark; EIBEN, Ágoston E. Parameter control in evolutionary algorithms: Trends and challenges. **Transactions on Evolutionary Computation**, IEEE, v. 19, n. 2, p. 167–187, 2015.

KARAFOTIAS, Giorgos; SMIT, Selmar K; EIBEN, AE. A generic approach to parameter control. In: SPRINGER. **European Conference on the Applications of Evolutionary Computation**. [S.l.], 2012. p. 366–375.

KREMPSER, Eduardo; FIALHO, Álvaro; BARBOSA, Helio. Adaptive operator selection at the hyper-level. **Parallel Problem Solving from Nature (PPSN)**, p. 378–387, 2012.

KRUSKAL, William H; WALLIS, W Allen. Use of ranks in one-criterion variance analysis. **Journal of the American statistical Association**, Taylor & Francis, v. 47, n. 260, p. 583–621, 1952.

LI, Zi Mu; ZHANG, Yi; ZHENG, Xiao Dong; WAN, Xing Yu. Solving the Problem of General Job Shop Problem by Using Improved Cellular Genetic Algorithm. **Advanced Materials Research**, v. 945-949, p. 3130–3135, jun. 2014.

LIU, Shi Qiang; KOZAN, Erhan. Scheduling trains as a blocking parallel-machine job shop scheduling problem. **Computers & Operations Research**, Elsevier, v. 36, n. 10, p. 2840–2852, 2009.

LU, Chao; GAO, Liang; LI, Xinyu; PAN, Quanke; WANG, Qi. Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. **Journal of Cleaner Production**, v. 144, p. 228–238, fev. 2017.

NALEPA, Jakub; CWIEK, Marcin; KAWULOK, Michal. Adaptive memetic algorithm for the job shop scheduling problem. In: **International Joint Conference on Neural Networks (IJCNN)**. [S.l.]: IEEE, 2015. p. 1–8.

NAWAZ, Muhammad; ENSCORE, E. Emory; HAM, Inyong. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, v. 11, n. 1, p. 91–95, 1983.

PAN, Ying; ZHANG, W. X.; GAO, T. Y.; MA, Qin-Yi; XUE, D. J. An adaptive Genetic Algorithm for the Flexible Job-shop Scheduling Problem. In: **International Conference on Computer Science and Automation Engineering (CSAE)**. [S.l.]: IEEE, 2011. v. 4, p. 405–409.

PINEDO, Michael. **Scheduling**. [S.l.]: Springer, 2012.

RILEY, Michael; MEI, Yi; ZHANG, Mengjie. Improving job shop dispatching rules via terminal weighting and adaptive mutation in genetic programming. In: IEEE. **Congress on Evolutionary Computation (CEC)**. [S.l.], 2016. p. 3362–3369.

RIPON, Kazi Shah Nawaz; TSANG, Chi-Ho; KWONG, Sam. An evolutionary approach for solving the multi-objective job-shop scheduling problem. In: **Evolutionary Scheduling**. [S.l.]: Springer, 2007. p. 165–195.

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

RUIZ, Rubén; STÜTZLE, Thomas. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, v. 177, n. 3, p. 2033–2049, mar. 2007.

SRINIVAS, Mandavilli; PATNAIK, Lalit M. Adaptive probabilities of crossover and mutation in genetic algorithms. **Transactions on Systems, Man, and Cybernetics**, IEEE, v. 24, n. 4, p. 656–667, 1994.

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

THIERENS, Dirk. An adaptive pursuit strategy for allocating operator probabilities. In: **Annual conference on Genetic and Evolutionary Computation**. [S.l.]: ACM, 2005. p. 1539–1546.

WANG, Lei; CAI, Jing-Cao; LI, Ming. An adaptive multi-population genetic algorithm for job-shop scheduling problem. **Advances in Manufacturing**, v. 4, n. 2, p. 142–149, jun. 2016.

WANG, Lei; TANG, Dun-bing. An improved adaptive genetic algorithm based on hormone modulation mechanism for job-shop scheduling problem. **Expert Systems with Applications**, v. 38, n. 6, p. 7243–7250, jun. 2011.

WERNER, Frank. Genetic algorithms for shop scheduling problems: a survey. **Preprint**, v. 11, p. 31, 2011.

XING, Yingjie; CHEN, Zhentong; SUN, Jing; HU, Long. An improved adaptive genetic algorithm for job-shop scheduling problem. In: **International Conference on Natural Computation (ICNC)**. [S.l.]: IEEE, 2007. v. 4, p. 287–291.

YAN, Jin; WU, Xiuli. A genetic based hyper-heuristic algorithm for the job shop scheduling problem. In: IEEE. **International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)**. [S.l.], 2015. v. 1, p. 161–164.

YANG, Gui; LU, Yujun; LI, Ren-wang; HAN, Jin. Adaptive genetic algorithms for the job-shop scheduling problems. In: **World Congress on Intelligent Control and Automation (WCICA)**. [S.l.]: IEEE, 2008. p. 4501–4505.

ZHANG, Haijun; YAN, Qiong; ZHANG, Guohui; JIANG, Zhiqiang. A chaotic differential evolution algorithm for flexible job shop scheduling. In: SPRINGER. **Asian Simulation Conference**. [S.l.], 2016. p. 79–88.

ZUO, Yi; GONG, Maoguo; JIAO, Licheng. Adaptive multimeme algorithm for flexible job shop scheduling problem. **Natural Computing**, set. 2016.

Tabela 23 – Algoritmos selecionados na primeira fase.

Algoritmo	Parâmetro	Categoria					
		10×10	20×10	20×20	50×10	50×50	TODOS
GA	Tamanho da População	76	100	100	100	100	100
GA	Operador de recombinação	PMX	LOX	LOX	LOX	LOX	LOX
GA	Taxa de recombinação	0,9	0,8	0,9	0,8	0,9	0,7
GA	Operador de mutação	SM	SM	SM	IM	SM	SM
GA	Taxa de mutação	0,5	0,5	0,4	0,3	0,4	0,5
GA	Hibridização	NEH + LS	NEH + LS	LS	NEH + LS	Não	Não
GA	Taxa da LS	0,03	0,015	0,03	0,03	N/A	N/A
AP	Tamanho da População	50	100	76	76	100	100
AP	α	80%	80%	80%	20%	50%	20%
AP	β	80%	80%	50%	50%	50%	80%
AP	Hibridização	LS	NEH + LS	NEH + LS	Não	Não	Não
APEX	Tamanho da População	50	100	100	76	100	100
APEX	α	20%	50%	80%	80%	20%	50%
APEX	β	80%	80%	20%	80%	20%	80%
APEX	Hibridização	NEH + LS	NEH + LS	NEH + LS	NEH	NEH	LS
DMAB	Tamanho da População	76	100	76	76	50	76
DMAB	C	0,1	0,01	1	10	0,1	1
DMAB	γ	100	1	1	10	1	1
DMAB	Hibridização	LS	NEH	NEH	NEH	NEH	NEH
DMABEX	Tamanho da População	76	50	76	100	76	76

Tabela 23 continuada da página anterior.

Algoritmo	Parâmetro	Categoria					
		10×10	20×10	20×20	50×10	50×50	TODOS
DMABEX	C	0,1	0,01	10	100	1	10
DMABEX	γ	1	0,1	0,01	0,01	10	0,01
DMABEX	Hibridização	LS	LS	NEH	NEH	Não	NEH
MS	Tamanho da População	100	76	76	100	100	76
MS	Hibridização	NEH + LS	NEH + LS	LS	Não	Não	LS
SLMAB	Tamanho da População	100	76	76	50	100	100
SLMAB	C	1	0,1	1	100	100	0,01
SLMAB	Hibridização	LS	NEH	NEH + LS	NEH + LS	NEH	NEH + LS
SLMABEX	Tamanho da População	100	100	76	76	50	100
SLMABEX	C	100	100	0,01	100	0,01	0,1
SLMABEX	Hibridização	NEH	NEH + LS	Não	NEH	NEH	LS
RMAB	Tamanho da População	76	100	100	50	50	76
RMAB	C	100	0,1	1	1	10	100
RMAB	D	0,50	100	0,25	0,75	100	0,75
RMAB	Hibridização	LS	NEH + LS	LS	NEH + LS	NEH + LS	NEH + LS
PPC	Tamanho da População	76	100	100	50	50	50
PPC	Regressão	Exponencial	Linear	Linear	Quadrática	Quadrática	Quadrática
PPC	Hibridização	NEH + LS	LS	NEH + LS	Não	Não	Não
CPPC	Tamanho da População	76	100	100	100	76	100
CPPC	Regressão	Linear	Quadrática	Linear	Linear	Quadrática	Linear
CPPC	Hibridização	NEH + LS	LS	NEH + LS	Não	NEH	NEH + LS

Tabela 23 continuada da página anterior.

Algoritmo	Parâmetro	Categoria					
		10×10	20×10	20×20	50×10	50×50	TODOS
RDMAB	Tamanho da População	100	50	100	100	100	76
RDMAB	C	0,1	0,01	100	0,1	100	100
RDMAB	γ	0,01	1	1	0,1	1	0,01
RDMAB	Hibridização	LS	NEH + LS	NEH + LS	NEH	NEH	LS

APÊNDICE B – Testes de Conover-Iman

Quando o teste de Kruskal-Wallis detectou diferenças nos resultados obtidos pelos algoritmos, foi aplicado o teste de Conover-Iman para identificar aqueles algoritmos com performance diferentes. Os testes de Conover-Iman foram realizados através da biblioteca conover.test disponível para R. Nesse teste, a hipótese nula para cada comparação entre pares é definida como: a probabilidade de selecionar aleatoriamente um valor do primeiro grupo amostral, deve-ser maior do que selecionar aleatoriamente um valor do segundo grupo, em 1/2. Esse método apresenta algumas opções disponíveis para ajustar o p-valor das múltiplas comparações. Portanto, utilizou-se o ajuste proposto por Benjamini & Hochberg (1995), que controla a taxa de descobertas falsas e é apropriado para testes independentes. Os p-valores são ordenados em ordem decrescente e ajustados, assim depois que a hipótese nula em relação ao primeiro é rejeitada, todos os demais na sequência também o são.

As tabelas a seguir apresentam o retorno da execução do teste de Conover-Iman e deve-se rejeitar ou aceitar a hipótese de acordo com a confiança estabelecida. Para essa dissertação, foi estabelecida uma confiança de 95,0%. Portanto, ao ler uma tabela deve-se comparar um algoritmo na coluna com aquele representado na linha, e se o p-valor encontrado for maior que 0,05 diz-se que não há superioridade estatística no algoritmo, por exemplo: Ao comparar os resultados obtidos pelo GA, que obteve a melhor mediana quando estão sendo resolvidos os problemas de 20 tarefas e 20 estágios, com os resultados obtidos pelo APEX, verifica-se na Tabela 24 que o teste de Conover-Iman retornou um valor de 0,2689, e como esse valor é maior que 0,05 (95,0% de confiança), não há superioridade estatística entre os algoritmos. No entanto, quando compara os resultados obtidos pelo GA e e DMAB, o resultado do teste de Conover-Iman na Tabela 24 é de 0,000. Como esse valor é menor do que 0,05, afirma-se com 95,0% de confiança que há uma superioridade estatística entre os algoritmos. Para saber qual algoritmo é superior estatisticamente basta avaliar a melhor mediana, que para esse caso é o GA.

Tabela 24 – Testes de Conover-Iman para os algoritmos ao resolverem os problemas de 20 tarefas e 20 estágios.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,4466	0,0000	0,0008	0,0040	0,2689	0,6396	0,5622	0,6489	0,9568	0,2758	0,6884
AP	–	0,0015	0,0187	0,0544	0,0280	0,8654	0,9271	0,8703	0,5311	0,8630	0,8021
DMAB	–	–	0,5748	0,2782	0,0000	0,0007	0,0009	0,0007	0,0001	0,0042	0,0006
DMABEX	–	–	–	0,7555	0,0000	0,0080	0,0113	0,0082	0,0013	0,0364	0,0064
SLMABEX	–	–	–	–	0,0000	0,0285	0,0359	0,0279	0,0058	0,1042	0,0231
GA	–	–	–	–	–	0,0539	0,0413	0,0543	0,2160	0,0135	0,0652
MS	–	–	–	–	–	–	0,9605	0,9817	0,6929	0,6834	0,9434
PPC	–	–	–	–	–	–	–	0,9490	0,6418	0,7492	0,9151
CPPC	–	–	–	–	–	–	–	–	0,6873	0,6893	0,9394
RDMAB	–	–	–	–	–	–	–	–	–	0,3381	0,7278
RMAB	–	–	–	–	–	–	–	–	–	–	0,6627

Tabela 25 – Testes de Conover-Iman para os algoritmos ao resolverem os problemas de 50 tarefas e 10 estágios.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,5975	0,3973	0,0053	0,2531	0,6008	0,3719	0,8987	0,0086	0,2847	0,8405	0,7854
AP	–	0,8153	0,0275	0,6110	0,2468	0,1237	0,6809	0,0409	0,0809	0,7701	0,8119
DMAB	–	–	0,0611	0,8094	0,1452	0,0628	0,4906	0,0842	0,0394	0,5596	0,6153
DMABEX	–	–	–	0,1281	0,0004	0,0001	0,0077	0,8857	0,0001	0,0104	0,0133
SLMABEX	–	–	–	–	0,0703	0,0266	0,3075	0,1708	0,0161	0,3704	0,4170
GA	–	–	–	–	–	0,7687	0,5088	0,0007	0,6329	0,4431	0,3852
MS	–	–	–	–	–	–	0,2932	0,0002	0,8875	0,2468	0,2148
PPC	–	–	–	–	–	–	–	0,0114	0,2265	0,9036	0,8667
CPPC	–	–	–	–	–	–	–	–	0,0001	0,0142	0,0186
RDMAB	–	–	–	–	–	–	–	–	–	0,1884	0,1521
RMAB	–	–	–	–	–	–	–	–	–	–	0,9096

Tabela 26 – Testes de Conover-Iman para os algoritmos ao resolverem os problemas de 50 tarefas e 50 estágios.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,0229	0,0013	0,0016	0,0048	0,0587	0,2501	0,2543	0,0000	0,5025	0,1430	0,2246
AP	–	0,3877	0,4472	0,6673	0,0000	0,3003	0,2956	0,0003	0,1247	0,4608	0,3310
DMAB	–	–	0,9586	0,7221	0,0000	0,0493	0,0483	0,0069	0,0126	0,0901	0,0571
DMABEX	–	–	–	0,7969	0,0000	0,0569	0,0573	0,0051	0,0167	0,1117	0,0657
SLMABEX	–	–	–	–	0,0000	0,1175	0,1150	0,0016	0,0436	0,2064	0,1300
GA	–	–	–	–	–	0,0017	0,0016	0,0000	0,0075	0,0006	0,0014
MS	–	–	–	–	–	–	0,9828	0,0000	0,6867	0,8092	0,9585
PPC	–	–	–	–	–	–	–	0,0000	0,6919	0,8048	0,9558
CPPC	–	–	–	–	–	–	–	–	0,0000	0,0000	0,0000
RDMAB	–	–	–	–	–	–	–	–	–	0,4875	0,6545
RMAB	–	–	–	–	–	–	–	–	–	–	0,8534

Tabela 27 – Testes de Conover-Iman para os algoritmos ao resolverem todos os problemas.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,4091	0,8332	0,9276	0,4673	0,0006	0,7501	0,7977	0,0001	0,3192	0,2053	0,6288
AP	–	0,2874	0,4638	0,0660	0,0253	0,6539	0,6063	0,0000	0,8514	0,7317	0,7891
DMAB	–	–	0,8115	0,6105	0,0002	0,6240	0,6581	0,0002	0,2109	0,1146	0,4837
DMABEX	–	–	–	0,4123	0,0009	0,7873	0,8202	0,0001	0,3664	0,2397	0,6671
SLMABEX	–	–	–	–	0,0000	0,2444	0,2906	0,0036	0,0398	0,0170	0,1702
GA	–	–	–	–	–	0,0034	0,0025	0,0000	0,0432	0,0922	0,0069
MS	–	–	–	–	–	–	0,9168	0,0000	0,5506	0,4052	0,8501
PPC	–	–	–	–	–	–	–	0,0000	0,4937	0,3574	0,8184
CPPC	–	–	–	–	–	–	–	–	0,0000	0,0000	0,0000
RDMAB	–	–	–	–	–	–	–	–	–	0,8222	0,6560
RMAB	–	–	–	–	–	–	–	–	–	–	0,4979

Tabela 28 – Testes de Conover-Iman para os algoritmos ao solucionarem as instâncias de 20 tarefas e 20 estágios, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,2596	0,0000	0,0000	0,9850	0,0000	0,6084	0,0000	0,5653	0,2882	0,3575	0,9029
AP	–	0,0000	0,0000	0,2571	0,0011	0,0609	0,0000	0,0508	0,0123	0,0200	0,1823
DMAB	–	–	0,9538	0,0000	0,3148	0,0000	0,7140	0,0000	0,0000	0,0000	0,0000
DMABEX	–	–	–	0,0000	0,3609	0,0000	0,6747	0,0000	0,0000	0,0000	0,0000
SLMABEX	–	–	–	–	0,0000	0,6115	0,0000	0,5684	0,2910	0,3608	0,9040
GA	–	–	–	–	–	0,0000	0,1432	0,0000	0,0000	0,0000	0,0000
MS	–	–	–	–	–	–	0,0000	0,9467	0,6637	0,7346	0,7035
PPC	–	–	–	–	–	–	–	0,0000	0,0000	0,0000	0,0000
CPPC	–	–	–	–	–	–	–	–	0,6955	0,7910	0,6717
RDMAB	–	–	–	–	–	–	–	–	–	0,8987	0,3595
RMAB	–	–	–	–	–	–	–	–	–	–	0,4652

Tabela 29 – Testes de Conover-Iman para os algoritmos ao solucionarem as instâncias de 50 tarefas e 10 estágios, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,8839	0,0000	0,0000	0,4249	0,0000	0,0257	0,0000	0,0531	0,0288	0,9197	0,0197
AP	–	0,0000	0,0000	0,3068	0,0000	0,0426	0,0000	0,0888	0,0474	0,8267	0,0325
DMAB	–	–	0,3940	0,0000	0,5417	0,0000	0,9029	0,0000	0,0000	0,0000	0,0000
DMABEX	–	–	–	0,0000	0,1049	0,0000	0,3101	0,0000	0,0000	0,0000	0,0000
SLMABEX	–	–	–	–	0,0000	0,0016	0,0000	0,0048	0,0019	0,4944	0,0011
GA	–	–	–	–	–	0,0000	0,6628	0,0000	0,0000	0,0000	0,0000
MS	–	–	–	–	–	–	0,0000	0,8217	0,9585	0,0195	0,9296
PPC	–	–	–	–	–	–	–	0,0000	0,0000	0,0000	0,0000
CPPC	–	–	–	–	–	–	–	–	0,8515	0,0424	0,7451
RDMAB	–	–	–	–	–	–	–	–	–	0,0220	0,9159
RMAB	–	–	–	–	–	–	–	–	–	–	0,0145

Tabela 30 – Testes de Conover-Iman para os algoritmos ao solucionarem as instâncias de 50 tarefas e 50 estágios, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,4951	0,0000	0,0000	0,2780	0,0000	0,0000	0,0000	0,0040	0,0000	0,5687	0,0000
AP	–	0,0000	0,0000	0,7045	0,0000	0,0001	0,0000	0,0343	0,0002	0,1985	0,0004
DMAB	–	–	0,3590	0,0000	0,2002	0,0000	0,5429	0,0000	0,0000	0,0000	0,0000
DMABEX	–	–	–	0,0000	0,0253	0,0000	0,1200	0,0000	0,0000	0,0000	0,0000
SLMABEX	–	–	–	–	0,0000	0,0003	0,0000	0,0869	0,0010	0,0946	0,0016
GA	–	–	–	–	–	0,0000	0,5302	0,0000	0,0000	0,0000	0,0000
MS	–	–	–	–	–	–	0,0000	0,0714	0,7394	0,0000	0,6571
PPC	–	–	–	–	–	–	–	0,0000	0,0000	0,0000	0,0000
CPPC	–	–	–	–	–	–	–	–	0,1390	0,0005	0,1785
RDMAB	–	–	–	–	–	–	–	–	–	0,0000	0,8901
RMAB	–	–	–	–	–	–	–	–	–	–	0,0000

Tabela 31 – Testes de Conover-Iman para os algoritmos ao solucionarem as instâncias de 20 tarefas, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	1,0000	0,0128	0,0128	0,9740	0,0560	0,9170	0,0050	0,9064	0,9128	0,9708	1,0000
AP	–	0,0171	0,0170	0,9919	0,0813	0,8743	0,0072	0,8661	0,8713	0,9049	0,9603
DMAB	–	–	0,9881	0,0063	0,9032	0,0037	0,9866	0,0053	0,0041	0,0044	0,0078
DMABEX	–	–	–	0,0064	0,9105	0,0039	0,9811	0,0059	0,0043	0,0045	0,0078
SLMABEX	–	–	–	–	0,0271	0,9608	0,0052	0,9860	0,9738	0,9915	0,9868
GA	–	–	–	–	–	0,0151	0,5921	0,0140	0,0146	0,0176	0,0355
MS	–	–	–	–	–	–	0,0033	1,0000	0,9958	0,9877	0,9653
PPC	–	–	–	–	–	–	–	0,0081	0,0045	0,0035	0,0049
CPPC	–	–	–	–	–	–	–	–	1,0000	0,9931	0,9962
RDMAB	–	–	–	–	–	–	–	–	–	0,9995	0,9606
RMAB	–	–	–	–	–	–	–	–	–	–	0,9598

Tabela 32 – Testes de Conover-Iman para os algoritmos ao solucionarem as instâncias de 50 tarefas, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,6898	0,0000	0,0000	0,9674	0,0000	0,0004	0,0000	0,0231	0,0014	0,8978	0,0011
AP	–	0,0000	0,0000	0,7249	0,0000	0,0019	0,0000	0,0756	0,0062	0,5769	0,0053
DMAB	–	–	0,2406	0,0000	0,0903	0,0000	0,5113	0,0000	0,0000	0,0000	0,0000
DMABEX	–	–	–	0,0000	0,0029	0,0000	0,0521	0,0000	0,0000	0,0000	0,0000
SLMABEX	–	–	–	–	0,0000	0,0005	0,0000	0,0266	0,0016	0,8634	0,0013
GA	–	–	–	–	–	0,0000	0,3610	0,0000	0,0000	0,0000	0,0000
MS	–	–	–	–	–	–	0,0000	0,2256	0,7605	0,0002	0,7941
PPC	–	–	–	–	–	–	–	0,0000	0,0000	0,0000	0,0000
CPPC	–	–	–	–	–	–	–	–	0,3930	0,0150	0,3667
RDMAB	–	–	–	–	–	–	–	–	–	0,0008	0,9542
RMAB	–	–	–	–	–	–	–	–	–	–	0,0007

Tabela 33 – Testes de Conover-Iman para os algoritmos ao solucionarem todas as instâncias, através das configurações de problemas pequenos.

	AP	DMAB	DMABEX	SLMABEX	GA	MS	PPC	CPPC	RDMAB	RMAB	SLMAB
APEX	0,9260	0,0000	0,0000	0,9464	0,0000	0,6071	0,0000	0,8866	0,6852	0,8384	0,4936
AP	–	0,0000	0,0000	0,8837	0,0000	0,6917	0,0000	0,9497	0,7567	0,7533	0,5862
DMAB	–	–	0,5722	0,0000	0,3530	0,0000	0,9942	0,0000	0,0000	0,0000	0,0000
DMABEX	–	–	–	0,0000	0,0567	0,0000	0,5795	0,0000	0,0000	0,0000	0,0000
SLMABEX	–	–	–	–	0,0000	0,5907	0,0000	0,8425	0,6261	0,8828	0,4419
GA	–	–	–	–	–	0,0001	0,3477	0,0000	0,0001	0,0000	0,0002
MS	–	–	–	–	–	–	0,0000	0,7500	0,9311	0,4197	0,8787
PPC	–	–	–	–	–	–	–	0,0000	0,0000	0,0000	0,0000
CPPC	–	–	–	–	–	–	–	–	0,8296	0,6953	0,6149
RDMAB	–	–	–	–	–	–	–	–	–	0,4869	0,8469
RMAB	–	–	–	–	–	–	–	–	–	–	0,2997