UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE OTIMIZAÇÃO UTILIZANDO ALGORITMO GENÉTICO COM ESTRATÉGIAS DE BUSCA LOCAL E MELHORAMENTO GENÉTICO PARA MINIMIZAÇÃO DO MAKESPAN NO PROBLEMA DE PROGRAMAÇÃO DA PRODUÇÃO JOB SHOP

MONIQUE SIMPLICIO VIANA

ORIENTADOR: **PROF. DR. ORIDES MORANDIN JUNIOR**

São Carlos – SP

Novembro/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM DE OTIMIZAÇÃO UTILIZANDO ALGORITMO GENÉTICO COM ESTRATÉGIAS DE BUSCA LOCAL E MELHORAMENTO GENÉTICO PARA MINIMIZAÇÃO DO MAKESPAN NO PROBLEMA DE PROGRAMAÇÃO DA PRODUÇÃO JOB SHOP

MONIQUE SIMPLICIO VIANA

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Inteligência Artificial.

Orientador: **Prof. Dr. Orides Morandin Junior**

São Carlos – SP

Novembro/2021

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Defesa de Tese de Doutorado da candidata Monique Simplicio Viana, realizada em 12/11/2021.

Comissão Julgadora:

Prof. Dr. Orides Morandin Junior (UFSCar)

Prof. Dr. Roberto Santos Inoue (UFSCar)

Prof. Dr. Roberto Fernandes Tavares Neto (UFSCar)

Prof. Dr. Zhao Liang (FFCLRP/USP)

Prof. Dr. Mário Luiz Tronco (USP)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.
O Relatório de Defesa assinado pelos membros da Comissão Julgadora encontra-se arquivado junto ao Programa de Pós-Graduação em Ciência da Computação.

Dedico este trabalho a meus pais.

AGRADECIMENTO

Gostaria de agradecer a todos que, de alguma forma, contribuíram para que este trabalho pudesse ser realizado. Em especial agradeço:

Ao meu orientador, professor Orides, pelo auxílio e discussões, dando o suporte necessário para a realização deste trabalho. Agradeço pelas críticas, pela paciência e pela prestatividade na condução deste trabalho. Um exemplo que levarei para a vida sobre ser um professor e pesquisador de excelência.

A toda minha família, principalmente aos meus pais, pelo imenso amor que recebo todos os dias, pelo apoio e incentivo em meus estudos e a minha vó, por ser um exemplo de mulher maravilhosa, forte e com muita sabedoria.

Ao meu amado namorado Rodrigo, pelo seu jeito único e incrível de ser, por todo seu amor, carinho e por sempre apoiar e incentivar minhas escolhas profissionais.

Ao meu orientador da graduação, professor Djalma Domingos, que me auxiliou no meu primeiro contato com a pesquisa acadêmica sempre com muita dedicação e paciência.

Ao meu professor de graduação, Sérgio Borges, pelos valiosos conhecimentos, com destaque aos conselhos e dicas sobre a área acadêmica.

Ao meu amigo Cleverson, por ser um amigo muito especial, uma amizade que levarei para a vida toda.

Aos meus amigos e todos aqueles que diretamente ou indiretamente contribuíram para realização do mestrado.

À CAPES, pelo suporte financeiro.

As coisas que um dia eu imaginei como minhas maiores conquistas foram apenas o primeiro passo rumo a um futuro que apenas comecei a visualizar.

Jace Beleren

RESUMO

Muitos trabalhos da atualidade estão utilizando metaheurísticas para tratar a classe de problemas conhecida na literatura como *Job Shop Scheduling Problem* (JSSP) devido a sua complexidade, uma vez que a mesma consiste de problemas combinatórios e é pertencente ao conjunto dos problemas computacionais NP-*Hard*. Neste tipo de problema, um dos objetivos mais abordados na literatura é o de minimizar o *makespan*, o qual consiste no tempo máximo de produção de uma série de produtos (*jobs*). Tratando-se de uma situação de alocação de recursos, para solucionar instâncias de JSSP, é amplamente utilizado metaheurísticas, tais como o Algoritmo Genético (GA). Apesar dos GAs apresentarem bons resultados na literatura, é muito comum que os mesmos apresentem certas deficiências como: estagnação em soluções que são mínimos locais; dificuldade em explorar o espaço de busca de maneira satisfatória; convergência prematura; entre outras. Para contornar estas situações, é proposto neste trabalho o uso de estratégias de busca local e de melhoramento genético no GA. Sendo a primeira estratégia definida na forma da generalização e do aprimoramento de técnicas de busca local existentes na literatura. Em detalhes, o conceito de operador de busca local massiva foi generalizado; o uso de uma estratégia de busca local no operador de mutação tradicional foi aprimorado; e um novo operador de multi-*crossover* foi desenvolvido. A segunda estratégia é definida na forma de um operador especializado em direcionar a população a boas regiões no espaço de busca. Este operador possibilita manipular o material genético dos indivíduos, adicionando características que são frequentes em indivíduos bem avaliados, com a proposta de direcionar alguns indivíduos da população que se encontram perdidos no espaço de busca para uma solução mais favorável sem prejudicar a diversidade da população. Neste trabalho, é proposto a junção destas estratégias com a finalidade de definir um *framework* de técnicas do tipo GA que possui o objetivo de minimizar o *makespan* em instâncias de JSSP. O ferramental desenvolvido foi avaliado em *benchmarks* bem estabelecidos da literatura especializada e se mostra competitivo e versátil em comparação aos métodos que representam o estado da arte.

Palavras-chave: Algoritmo Genético, Busca Local, Operador de Melhoramento Genético, Programação da Produção, *Job Shop Scheduling Problem*, *Makespan*.

ABSTRACT

Many works nowadays use metaheuristics to deal with the class of problems known in the literature as Job Shop Scheduling Problem (JSSP) due to its complexity since it consists of combinatorial problems and belongs to the set of NP-Hard computational problems. In this type of problem, one of the most discussed objectives in the literature is to minimize the makespan, which consists of the maximum production time of a series of jobs. As this is a resource allocation situation, to solve JSSP instances, metaheuristics such as the Genetic Algorithm (GA) are widely used. Although GAs present good results in the literature, it is very common that they present certain deficiencies, such as: stagnation in solutions that are local minimums; difficulty in exploring the search space satisfactorily; premature convergence; among others. To overcome these situations, the use of local search and genetic improvement strategies in GA is proposed in this work. Being the first strategy defined in the form of generalization and improvement of local search techniques existing in the literature. In detail, the concept of massive local search operator was generalized; the use of a local search strategy in the traditional mutation operator has been improved; and a new multi-crossover operator was developed. The second strategy is defined in the form of an operator specialized in directing the population to good regions in the search space. This operator makes it possible to manipulate the genetic material of individuals, adding characteristics that are frequent in well-regarded individuals, with the proposal of directing some individuals in the population who are lost in the search space for a more favorable solution without harming the diversity of the population. In this work, the joining of these strategies is proposed in order to define a framework of GA techniques that have the objective of minimizing the makespan in JSSP instances. The developed material was evaluated in well-established benchmarks in the specialized literature and is competitive and versatile compared to the methods that represent the state of the art.

Keywords: Genetic Algorithm, Local Search, Genetic Improvement Operator, Production Scheduling, Job Shop Scheduling Problem, Makespan.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ABC	*Artificial Bee Colony*
ACO	*Ant Colony Optimization*
AGA	*Adaptive Genetic Algorithm*
AIS	*Artificial Immune Systems*
AMGA	*Adaptive Multipopulation Genetic Algorithm*
BA	*Bat Algorithm*
BFO	*Bacterial Foraging Algorithm*
CS	*Cuckoo Search*
CSA	*Clonal Selection Algorithm*
CSO	*Chicken Swarm Optimization*
DABC	*Discrete Artificial Bee Colony*
DCS	*Discrete Cuckoo Search*
DCSA	*Discrete Cuckoo Search and Simulated Annealing*
DE	*Differential Evolution*
DGA	*Discrete Genetic Algorithm*
DWPA	*Discrete Wolf Pack Algorithm*
F&F	*Filter and Fan*
GA	*Genetic Algorithm*
GBA	*Golden Ball Algorithm*
GIFA	*Genetic Improvement based on Frequency Analysis*
GLS	*Guided Local Search*
GRASP	*Greedy Randomized Adaptive Search Procedure*
GS	*Greedy Search*
GWO	*Grey Wolf Optimization*
HGA	*Hybrid Genetic Algorithm*
HGWO	*Hybrid Grey Wolf Optimization*
HHO	*Harris Hawks Optimizer*
HIGA	*Hybrid Immune Genetic Algorithm*
IHHO	*Improved Harris Hawks Optimizer*
IMGA	*Island Model Genetic Algorithm*
IPBGA	*Promising Initial Population Based Genetic Algorithm*

JSSP	*Job Shop Scheduling Problem*
LSGA	*Local Search Genetic Algorithm*
LSO	*Lion Swarm Optimization*
MA	*Memetic Algorithm*
MAnt	*Multiple Colony Ant Algorithm*
MeCSO	*Memetic Chicken Swarm Optimization*
MGA	*Modified Genetic Algorithm*
MKS	*Makespan*
MLP	*Multilayer Perception*
mXLSGA	*Genetic Algorithm with Local Search Strategies and Multi-Crossover*
NAGA	*Niche Adaptive Genetic Algorithm*
NIMGA	*New Island Model Genetic Algorithm*
OX2	*Order-based Crossover*
PAGA	*Parallel Agent-based Genetic Algorithm*
PBA	*Parallel Bat Algorithm*
PMX	*Partially Mapped Crossover*
PSO	*Particle Swarm Optimization*
SA	*Simulated Annealing*
SSO	*Social Spider Optimization*
SSS	*Single Seekers Society*
TA	*Threshold Accepting*
TLBO	*Teaching-Learning Based Optimization*
WPA	*Wolf Pack Algorithm*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

O procedimento de produção em empresas de manufatura é um dos principais pontos para o sucesso total dos negócios. Inúmeras empresas envolvidas na produção de bens tangíveis enfrentam problemas relacionados à programação da produção. Deste modo, a investigação por abordagens que atinjam soluções eficientes e eficazes atraiu o interesse de diversos profissionais e pesquisadores de ambas as áreas de gerenciamento de produção e de otimização combinatória. O interesse dos pesquisadores nessa área de pesquisa também foi ampliado devido à similaridade do problema combinatório de programação da produção com outros problemas, tais como problemas de roteamento, agendamento de horários, agendamento de pacotes em redes, entre outros (XHAFA; ABRAHAM, 2008).

A pesquisa acadêmica e o desenvolvimento de metodologias de solução têm se concentrado em um número limitado de problemas clássicos de programação da produção. No caso, alguns tipos de variações de problemas de programação da produção mais pesquisados são: *Single Machine Scheduling Problem, Parallel Machine Scheduling Problem, Flow Shop Scheduling Problem* (FSSP), *Job Shop Scheduling Problem* (JSSP) e *Open Shop Scheduling Problem* (OSSP) (HART; ROSS; CORNE, 2005). Neste trabalho será tratado o problema de programação da produção JSSP, por esta categoria ser uma das mais abordadas na literatura e que ainda necessita de avanços consideráveis em relação aos métodos e às instâncias disponíveis, uma vez que os métodos existentes não são capazes de tratar todos os casos com máxima eficiência.

Como um dos problemas pertencentes à classe de otimização combinatória, o JSSP é um problema bem difundido na literatura e apresenta-se como um desafio computacional. Não é uma tarefa trivial desenvolver uma abordagem para encontrar uma boa solução para este problema, isto é, uma configuração que tenha um bom resultado na medida de desempenho, dentro de um prazo razoável, mesmo considerando casos pequenos e moderados (WANG; CAI; LI, 2016).

No problema de programação da produção existem várias medidas possíveis de desempenho (*makespan*, *tardiness*, *lateness*, *maximum workload*, *total workload*, *due date*, etc.) (XHAFA; ABRAHAM, 2008). O objetivo principal deste trabalho é abordar técnicas focadas na minimização do *makespan*, o qual consiste no tempo total necessário para terminar a produção de um conjunto de *jobs*. Esta medida foi selecionada para conduzir a avaliação de eficiência do método por ser uma das medidas mais abordadas na literatura no caso de instâncias de JSSP.

As primeiras pesquisas relacionadas à programação da produção ficaram centradas em problemas simplificados, sendo que a maioria desses problemas tratavam a otimização de um único objetivo em ambientes de uma única máquina e apresentava muitas suposições simplificadas. Dados esses problemas de programação da produção simplificados, as primeiras pesquisas foram focadas em métodos exatos, isto é, métodos que garantiam a solução ótima de um dado problema. Entretanto, com a falta de recursos computacionais e a necessidade de resolver problemas de programação da produção em larga escala, foi concluído que os métodos exatos eram inviáveis. Com a nova necessidade de abordar problemas de programação da produção de larga escala foram desenvolvidos e aplicados os algoritmos que fazem uso de metaheurística (XHAFA; ABRAHAM, 2008).

Diversas abordagens da metaheurística conhecida como Algoritmo Genético (GA) foram executadas com sucesso em muitos problemas de programação da produção, em particular o JSSP. Entretanto, para problemas com uma maior complexidade, o GA precisa se conectar a métodos específicos do problema para tornar a abordagem realmente efetiva. A hibridização é uma maneira eficaz de melhorar o desempenho e a eficácia dos algoritmos genéticos. As técnicas de busca local são as formas mais comuns de hibridização e são utilizadas para melhorar o desempenho desses algoritmos (ASADZADEH, 2015).

Neste trabalho, é proposta uma nova abordagem de GA com técnicas melhoradas de busca local dedicada a encontrar boas soluções para instâncias de JSSP. Especificamente, a

proposta deste trabalho consiste na forma de um *framework* composto por estratégias de busca local agregadas a diferentes operadores de metaheurísticas do tipo GA, compreendendo a generalização de grande parte desta categoria de técnicas. Além disso, também é proposto a adição de novas estratégias de busca local como, por exemplo, um novo operador de melhoramento genético baseado em frequências de boas características presentes na população de forma a conduzir o direcionamento de indivíduos mal adaptados enquanto mantêm-se a diversidade populacional.

1.2 Motivação e Justificativa

Os problemas de programação da produção JSSP são especificados na literatura como pertencentes à classe *NP-Hard.* Sendo que para fazer a otimização de problemas pertencentes a tal classe, é aconselhado o emprego de algoritmos estocásticos, heurísticos e metaheurísticos (ÇALIS; BULKAN, 2015; LIN; GEN, 2018).

Nos últimos anos, foram desenvolvidas pesquisas para solucionar o problema de programação da produção JSSP por meio de diversas abordagens que fazem uso de metaheurísticas, tais como: ***Genetic Algorithm*** (AQUINALDO; CUCUK; YUNIARISTANTO, 2021; ABDULLAH; ALAGHA, 2021; SAIDAT *et al.*, 2021; LIU *et al.*, 2021; KALSHETTY; ADAMUTHE; KUMAR, 2020; LU; HUANG; CAO, 2020; CHEN; ZHANG; GAO, 2019; AHMADIZAR; FARAHANI, 2012; ASADZADEH, 2015; ASADZADEH; ZAMANIFAR, 2010; ESSAFI; MATI; DAUZÈRE-PÉRÈS, 2008; GONÇALVES; MENDES; RESENDE, 2005; JORAPUR *et al.*, 2016; KALANTARI; SANIEEABADEH, 2013; KUMAR *et al.*, 2016; KURDI, 2015, 2019; LIU; TSAI; CHOU, 2006; OMAR; BAHARUM; HASAN, 2006; OMBUKI; VENTRESCA, 2004; PIROOZFARD; WONG; HASSAN, 2016; TSAI *et al.*, 2008; WANG; CAI; LI, 2016; WANG; ZHENG, 2002; QING-DAO-ER-JI; WANG, 2012; WATANABE; IDA; GEN, 2005; YUSOF *et al.*, 2011; ZHANG; RAO; LI, 2008; ZHOU; FENG; HAN, 2001), ***Ant Colony Optimisation*** (ACO) (ESWARAMURTHY; TAMILARASI, 2009; HUANG; LIAO, 2008; KATO; MORANDIN; FONSECA, 2009; SEO; KIM, 2010; UDOMSAKDIGOOL; KACHITVICHYANUKUL, 2008), ***Particle Swarm Optimization*** (PSO) (WANG, 2021; YU *et al.*, 2020; MENG; ZHANG; FAN, 2016; GE *et al.*, 2008; GU; TANG; ZHENG, 2012; MORANDIN *et al.*, 2013; LIAN; JIAO; GU, 2006; LIN *et al.*, 2010; SHA; HSU, 2006), ***Simulated Annealing*** (SA) (AKRAM;

KAMAL; ZEB, 2016; VAN LAARHOVEN; AARTS; LENSTRA, 1992; ZHANG; WU, 2010), ***Tabu Search*** (TS) (ESWARAMURTHY; TAMILARASI, 2009; NOWICKI; SMUTNICKI, 2005; PONNAMBALAM; ARAVINDAN; RAJESH, 2000; PONSICH; COELLO, 2013), ***Grey Wolf Optimization*** (GWO) (JIANG, 2018; JIANG; ZHANG, 2018), ***Bat Algorithm*** (BA) (DAO *et al.*, 2018), ***Chicken Swarm Optimization*** (CSO) (SEMLALI; RIFFI; CHEBIHI, 2019), ***Golden Ball Algorithm*** (GBA) (SAYOTI; RIFFI; LABANI, 2016), ***Artificial Bee Colony*** (ABC) (BANHARNSAKUN; SIRINAOVAKUL; ACHALAKUL, 2012; CHONG *et al.*, 2006), ***Single Seekers Society*** (SSS) (HAMZADAYI; BAYKASOĞLU; AKPINAR, 2020), ***Cuckoo Search*** (CS) (ALKHATEEB; ABED-ALGUNI; AL-ROUSAN, 2021; OUAARAB; AHIOD; YANG, 2015), **Social Spider Optimization** (SSO) (ZHOU; ZHOU; ZHAO, 2021), ***Harris Hawks Optimizer*** (HHO) (LIU, 2021), **Lion Swarm Optimization** (LSO) (HUANG *et al.*, 2021), entre outros.

Os algoritmos que utilizam metaheurísticas híbridas estão se destacando dentre os vários métodos utilizados em JSSP, uma vez que estes podem combinar os méritos de diferentes algoritmos metaheurísticos. Os algoritmos híbridos geralmente apresentam um desempenho notável, incorporando a busca local para obter uma combinação adequada de esforços de busca de diversificação e intensificação. Entre a grande diversidade de estudos, é possível constatar a superioridade das metaheurísticas de comportamento de busca global com inclusão de técnicas especializadas em realizar busca local. Sendo este tipo de procedimento abordado em alguns trabalhos, recentes ou não, tais como as abordagens de Ombuki e Ventresca (2004), Watanabe, Ida e Gen (2005), Asadzadeh (2015), Sayoti, Riffi e Labani (2016), Díaz *et al.* (2017), Jiang (2018), Wang, Bing *et al.* (2018), Dao *et al.* (2018), Jiang e Zhang (2018), Semlali, Riffi e Chebihi (2019), Kurdi (2019), Yu *et al.* (2020), Lu, Huang e Cao (2020), Huang *et al.* (2021) e Alkhateeb, Abed-Alguni e Al-Rousan (2021). Desta forma, o uso de hibridizações entre métodos de busca global e busca local já é frequente nos trabalhos da literatura, sendo comum em diversos algoritmos como GA, GBA, *Memetic Algorithm* (MA), GWO, BA, CSO, CS e LSO.

Várias técnicas têm sido utilizadas nas pesquisas para tratar o problema de programação da produção com o objetivo de minimizar o *makespan*. A maioria dos trabalhos atuais trata o problema com algum método baseado em metaheurística, tais com GA, ACO, PSO, SA, GWO, BA, CSO, GBA, ABC, SSS, CS, SSO, HHO e LSO. Dentre a variedade de metaheurísticas que abordam o JSSP, o GA obteve um grande destaque por oferecer um bom desempenho devido à

sua capacidade de pesquisa global. Diversas abordagens de GA canônicos (básicos) e híbridos foram aplicadas no JSSP obtendo bons resultados.

De acordo com os autores Guo, Wang, Han (2010), um GA possui três vantagens para problemas de otimização. A primeira vantagem é que o GA não possui muitos requisitos matemáticos sobre os problemas, sendo essa característica devida à sua natureza evolutiva. Além disso, o GA executa a busca de soluções sem considerar mecanismos específicos internos do problema. A segunda vantagem é que os operadores do GA efetuam uma busca de caráter global de forma muito eficaz. A terceira vantagem é que um GA proporciona uma grande flexibilidade para se criar uma hibridização com outras heurísticas, possibilitando, assim, uma possível implementação mais eficiente em um problema específico.

Os GAs foram abordados com êxito em numerosos problemas de otimização combinatória. Entretanto, para problemas mais complexos, um GA precisa ser assimilado a características específicas do problema para tornar a abordagem realmente efetiva. A hibridização pode ser uma maneira profundamente eficaz de melhorar o desempenho dos GAs. A forma mais comum de hibridação é a junção de GAs a técnicas de buscas locais e a incorporação de conhecimento específico do domínio no processo de busca. Outra forma comum de hibridação consiste em incorporar um operador de busca local ao GA, aplicando o operador a determinados membros da população após cada geração. Essa hibridização é frequentemente realizada para produzir resultados mais adequados do que as abordagens individuais podem alcançar por conta própria (SASTRY; GOLDBERG; KENDALL, 2005).

Dentre o período de 2005 a 2016, o GA foi o algoritmo evolucionário mais utilizado em problemas de programação da produção (LIN; GEN, 2018). É perceptível que o GA apresentou um bom desempenho para problemas de otimização JSSP, devido às suas características de inspiração genética e sua capacidade de pesquisa global. No entanto, os GAs existentes para JSSP geralmente apresentam uma velocidade de convergência lenta, isto é, necessitam de um maior número de iterações para convergir de forma satisfatória, e são facilmente estagnáveis em soluções que são ótimos locais. Para aumentar a velocidade de convergência, reduzindo o número de iterações necessárias, e aprimorar a capacidade da pesquisa, muitos trabalhos agregaram técnicas de busca local nos GAs (ASADZADEH, 2015; OMBUKI; VENTRESCA, 2004; VAESSENS; AARTS; LENSTRA, 1996; WANG *et al.*, 2018; WATANABE; IDA; GEN, 2005).

Dentre diversas propostas de técnicas anexadas ao GA que levaram a bons resultados pode-se citar, o trabalho de Viana, Morandin Junior e Contreras (2020d), no qual os autores utilizam um operador intitulado *Transgênico* para solucionar instâncias de JSSP. No caso, o operador transgênico foi proposto originalmente por Amaral e Hruschka (2014) e é inspirado pela engenharia genética, na qual é possível manipular o material genético dos indivíduos, adicionando características das quais se acredita serem as mais importantes. Assim, a abordagem do operador Transgênico pode ser vista como uma estratégia de direcionamento da busca e de elitismo focada em genes específicos. Especificamente, no trabalho de Viana, Morandin Junior e Contreras (2020d), foi proposta uma modificação e adaptação do operador transgênico de Amaral e Hruschka (2014) utilizado em um GA para tratar o problema de programação reativa da produção. Uma das etapas necessárias para o sucesso deste operador é a determinação de quais genes são mais importantes para o problema, uma vez que esses genes são transmitidos do melhor indivíduo para outros indivíduos pertencentes à população visando melhorar os indivíduos da população e assim, possivelmente, encontrar um melhor resultado. Essa etapa de verificação de quais genes serão utilizados como mais significativos, possui um elevado custo computacional e é uma etapa feita anteriormente à execução principal do algoritmo, na qual é executada uma simulação gene a gene para se avaliar os efeitos que cada gene provoca na população e, ao final da execução, é analisado quais genes obtiveram um melhor impacto sobre as soluções. Apenas depois de ser feita a etapa de verificação dos genes significativos, é possível executar o algoritmo final. O trabalho em questão obteve bons resultados ao ser comparado com outras abordagens que utilizaram GA (MORANDIN, O. *et al.*, 2008), GA adaptativo (MORANDIN, O *et al.*, 2008) e ACO (KATO; MORANDIN; FONSECA, 2010). Além disso, demonstrou-se que o operador transgênico pode direcionar a população a bons resultados. Entretanto, devido à etapa de determinação dos genes feita anteriormente à execução principal do algoritmo, o tempo total para a obtenção da solução é estendido.

Dadas as contribuições alcançadas ao se utilizar GAs juntamente a técnicas de busca local e com técnicas de manipulação de material genético, tais como o operador transgênico, este trabalho compreende-se em propor um GA com técnicas melhoradas de busca local e um operador de melhoramento genético mais elaborado no sentido de não apresentar um alto custo computacional por não necessitar de uma etapa de pré-processamento. Além disso, para contornar o problema de convergência prematura devido ao fato de baixa variabilidade genética causada pelo operador de transgenia, como ocorre no trabalho de Viana, Morandin Junior e

Contreras (2020d), este operador deve ser composto por uma estratégia de manutenção de diversidade genética, tais como as baseadas em voos de Lévy (ALKHATEEB; ABED-ALGUNI; AL-ROUSAN, 2021; CHEGINI; BAGHERI; NAJAFI, 2018; OUAARAB; AHIOD; YANG, 2015). Assim, o novo operador de melhoramento genético deve ser capaz de direcionar alguns indivíduos da população por meio de manipulação genética para uma solução mais favorável ao problema sem possuir a desvantagem de causar a convergência genética da população.

1.3 Hipóteses do trabalho

A primeira hipótese deste trabalho é a de que um novo framework de metaheurísticas do tipo GA que generaliza diferentes técnicas de busca local pode minimizar o *makespan* na programação da produção JSSP e alcançar resultados competitivos ao ser comparado com trabalhos presentes na literatura.

A segunda hipótese deste trabalho é a de que o método proposto seja capaz de alcançar uma melhor convergência do que outras modificações de GA aplicadas a JSSP presentes na literatura, obtendo melhores resultados com respeito a *benchmarks* do tema.

1.4 Objetivos

1.4.1 Gerais

O objetivo do trabalho é fazer a minimização do *makespan* em cenários de programação da produção do tipo JSSP por meio de uma nova abordagem, um *framework* de técnicas do tipo GA com busca local aprimorada e um novo operador de melhoramento genético.

1.4.2 Específicos

- Definir o método de busca global GA e o método de busca local como métodos a serem utilizados em colaboração no processo de busca de menor *makespan.*

- Desenvolver um novo operador de melhoramento genético.
- Detalhar um novo *framework* de técnicas do tipo GA com operadores especializados em busca local e melhoramento genético especializado na solução de instâncias de JSSP.
- Definir todos os parâmetros a serem utilizados na implementação do método proposto.
- Selecionar os cenários do problema de programação de produção JSSP a serem usados na validação.

1.5 Delimitações do trabalho

O principal delimitador de trabalhos como este consiste no fato de que a validação não é feita em um cenário real de manufatura, ou seja, os dados utilizados para a validação do projeto não são produzidos por um sistema real de manufatura e sim por um cenário virtual de JSSP baseado em características que supostamente definem um sistema de manufatura.

O problema de programação da produção abordado segue algumas premissas, a saber:

- Cada trabalho consiste em um número finito de operações;
- O tempo de processamento para cada operação em uma determinada máquina está definido;
- Existe uma sequência pré-definida de operações que têm de ser mantida para completar cada trabalho;
- Os prazos de entrega dos *jobs* são indefinidos;
- Não são considerados os custos de *setup* ou custo de atraso;
- Não são considerados valores de tempo de transporte;
- Uma máquina pode processar apenas uma tarefa de cada vez;
- Cada trabalho visita cada máquina apenas uma vez;
- Nenhuma máquina pode lidar com mais de um tipo de tarefa;
- O sistema não pode ser interrompido até que cada operação de cada trabalho esteja no estado terminado;

- Nenhuma máquina pode interromper um trabalho e começar outro trabalho antes de terminar o anterior (HASAN; SARKER; ESSAM, 2010).

1.6 Método de pesquisa e desenvolvimento

Neste trabalho, foi utilizado como orientação principal para o método de pesquisa e desenvolvimento a engenharia de algoritmos (SANDERS, 2009), a qual é composta por etapas que direcionam a pesquisa e o desenvolvimento de algoritmos que sejam eficientes. Essa abordagem é executada de modo cíclico contendo as seguintes etapas:

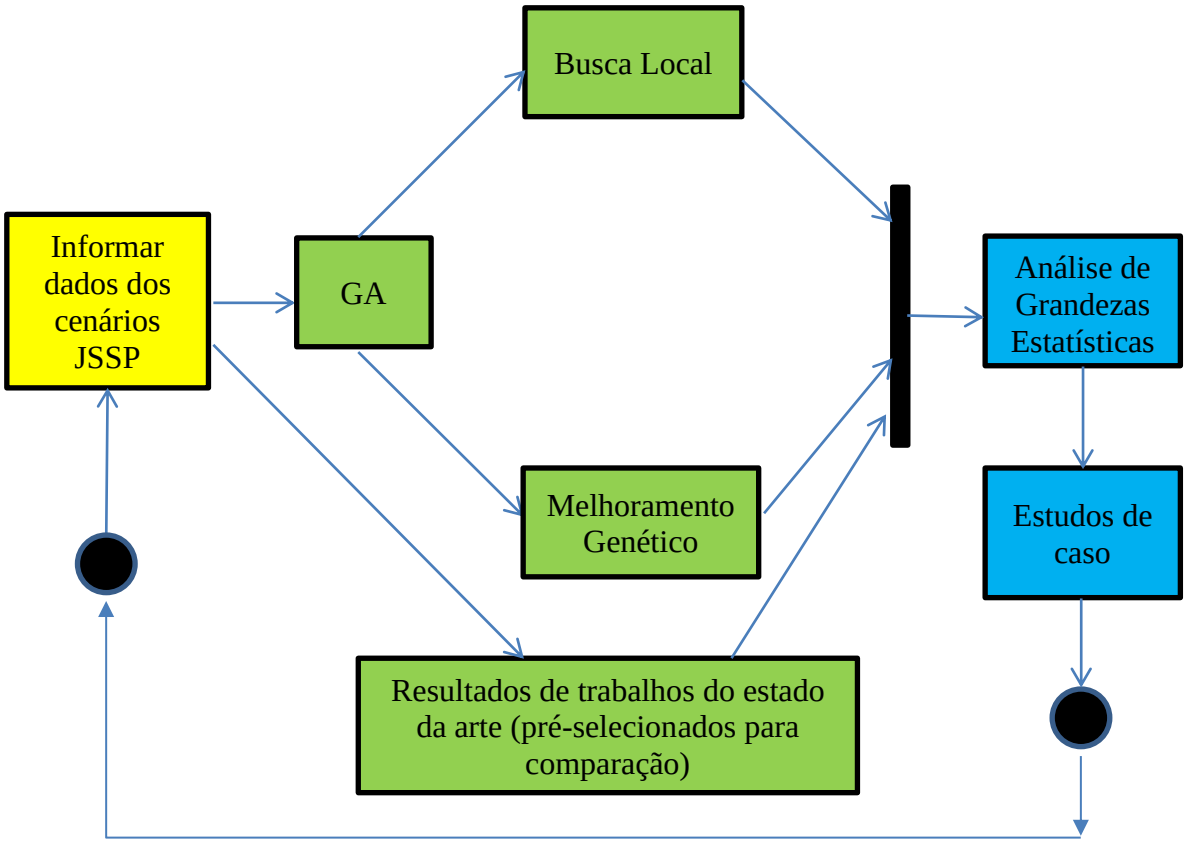
- **Design**: Essa etapa consiste na análise e na compreensão do problema abordado, e o delineamento de um algoritmo para alcançar uma solução. Devem ser consideradas a simplicidade do algoritmo e a possibilidade de reuso.
- **Análise teórica**: Essa etapa consiste em analisar teoricamente a eficiência do algoritmo e a qualidade das soluções que serão obtidas.
- **Implementação**: Essa etapa consiste em implementar o algoritmo utilizando as estruturas de dados e rotinas auxiliares adequadas.
- **Análise Experimental**: Esta etapa consiste em ensaiar o algoritmo com variadas instâncias do problema, verificar seu desempenho e comparar os resultados obtidos com outros algoritmos presentes na literatura especializada.

O trabalho proposto foi avaliado em cenários JSSP considerados tradicionais na literatura e os resultados obtidos são comparados com trabalhos relevantes que definem o estado da arte da literatura. Sendo que a metodologia de validação utilizada neste trabalho consiste na abordagem quantitativa, a qual é caracterizada pela formação de uma hipótese, definições de variáveis, quantificação na coleta de dados e de informação e uso de tratamentos estatísticos. O método quantitativo se apoia em dados estatísticos, comprovações ou validações e testes.

A validação do material desenvolvido é feita como mostra a Figura 1.1. A seção em amarelo representa os dados de cada instância JSSP que serão utilizadas para testes. Nas seções em verde, está a execução da metaheurística do tipo GA com inclusão das estratégias de Busca Local propostas e com a inclusão do Operador de Melhoramento Genético proposto. São

avaliadas diferentes configurações do material proposto, incluindo a análise do funcionamento de estratégias isoladas, isto é, as técnicas propostas são avaliadas de modo individual e também de modo conjunto, por exemplo, GA com busca local e GA com o operador de melhoramento genético. A seção em azul é referente à obtenção e à análise dos dados obtidos através das técnicas em verde. Para tal, são obtidas métricas numéricas que descrevem o desempenho do método proposto. Finalmente, a validação do material se dá com a consideração de vários estudos de caso, sendo cada um dedicado a analisar uma propriedade do método proposto e comprovar que os objetivos estabelecidos foram alcançados.

Figura 1.1. Diagrama de Atividade da Validação.



Fonte: Elaborada pela autora.

1.7 Publicações

1.7.1 Publicações diretamente relacionadas à Tese

Nesta seção, serão listados os artigos que resultaram durante o desenvolvimento deste projeto de doutorado e que são diretamente relacionados à Tese. Isto é, são apresentados os trabalhos que fazem uso de versões preliminares do material aqui proposto, que consiste no *framework* de técnicas do tipo GA, dedicado a solucionar instâncias de JSSP.

- Artigos publicados:
  - (1) "*A modified genetic algorithm with local search strategies and multi-crossover operator for job shop scheduling problem*" (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020a).
  - (2) "*Transgenic Genetic Algorithm to Minimize the Makespan in the Job Shop Scheduling Problem*" (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020d).
  - (3) "*An improved local search genetic algorithm with multi-crossover for job shop scheduling problem*" (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020c).
  - (4) "*A new genetic improvement operator based on frequency analysis for Genetic Algorithms applied to Job Shop Scheduling Problem*" (VIANA; CONTRERAS; MORANDIN JUNIOR, 2021).

- Artigo em processo de revisão no periódico:
  - (5) "*A new frequency analysis operator for population improvement in Genetic Algorithms to solve Job Shop Scheduling Problem*".

- Artigo em elaboração:
  - (6) "*A new Genetic Algorithm with Local Search Strategies including the Genetic Diversity as Objective applied to Job Shop Scheduling Problem*".

1.7.2 Publicações indiretamente relacionadas à Tese

Nesta seção, são listados os artigos publicados que resultaram durante o desenvolvimento deste projeto de doutorado e que não são diretamente relacionados à Tese, mas que fazem uso de um material preliminar baseado no *framework* de técnicas do tipo GA proposto para solucionar outros problemas combinatórios, que é o caso dos artigos (7) e (8). Também é apresentado um trabalho resultante da colaboração com membros do laboratório TEAR (9).

(7) "*An Improved Local Search Genetic Algorithm with a New Mapped Adaptive Operator Applied to Pseudo-Coloring Problem*" (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020b).

(8) "*A New Local Search Adaptive Genetic Algorithm for the Pseudo-Coloring Problem*" (CONTRERAS; MORANDIN JUNIOR; VIANA, 2020).

(9) "*A collaborative CPN− Fuzzy modelling strategy for conflict solution in flexible manufacturing systems*" (SANTANA *et al.*, 2018).

1.7.3 Avaliadores das publicações

Nesta seção, são apresentadas medidas de avaliadores das publicações para cada artigo. Na Tabela 1.1, são apresentadas algumas informações a respeito das publicações em periódicos. A saber, destaca-se o ano em que a publicação ocorreu; o veículo de publicação; alguns quantificadores deste periódico tais como o fator de impacto referente ao ano de 2020, sua pontuação no Scopus e sua qualificação de acordo com o qualis da CAPES na área de ciência da computação para os anos de 2013 a 2016; e informações de acesso e uso do trabalho no meio acadêmico tais como o número de visualizações e *downloads* do manuscrito e o número de citações na plataforma Google Scholar.

Tabela 1.1. Publicações em periódicos. O símbolo "-" representa a inexistência da informação. Visualizado em 03 de abril de 2022.

Trabalho	Ano	Periódico - Editora	Fator de Impacto 2020	Scopus	Qualis CC 2013 - 2016	Visualizações do manuscrito	Citações (Google Scholar)
(1)	2020	Sensors - MDPI	3.576	90%	A1	1473	12
(7)	2020	Symmetry – MDPI	2.713	90%	-	749	1
(9)	2018	International Journal of Computer Integrated Manufacturing – Taylor & Francis	3.205	87%	B1	216	1

Fonte: Elaborada pela autora.

Na Tabela 1.2, são apresentados alguns quantificadores a respeito das conferências nas quais resultados preliminares deste trabalho de doutorado foram publicados e apresentados. Especificamente, destacam-se o ano em que a publicação ocorreu; a sigla da conferência; o índice h5 do Google Scholar; na qualificação de acordo com o qualis da CAPES na área de ciência da computação para os anos de 2013 a 2016; a quantidade de visualizações e *downloads* do trabalho; e o número de citações de acordo com a plataforma Google Scholar.

Tabela 1.2. Publicações em conferências. O símbolo "-" representa a inexistência da informação. Visualizado em 03 de abril de 2022.

Trabalho	Ano	Conferência	Índice h5 Google 2021	Qualis CC 2013 - 2016	Visualizações do manuscrito	Citações (Google Scholar)
(2)	2020	ICAART	17	B1	-	2
(3)	2020	ICAISC	15	A2	1388	3
(4)	2021	ICAISC	15	A2	265	0
(8)	2020	ICSI	15	B1	1018	7

Fonte: Elaborada pela autora.

1.8 Organização do trabalho

Este trabalho está dividido em 5 capítulos. Sendo este primeiro capítulo dedicado a uma breve introdução ao texto da tese de doutorado; o próximo capítulo trata de destacar a fundamentação teórica utilizada no desenvolvimento do material proposto assim como avanços correlatos que definem o estado da arte de solução de JSSP com metaheurísticas; no terceiro capítulo, é apresentada a formulação do método proposto; resultados obtidos com diferentes configurações do material proposto é apresentado no quarto capítulo; este trabalho é finalizado com conclusões acerca do material desenvolvido e dos resultados obtidos. Em detalhes, esta tese de doutorado está dividida nos seguintes capítulos:

- **[Capítulo 1: Introdução]** Neste capítulo, o leitor é introduzido ao tema de solução de JSSP com metaheurísticas. As motivações e justificativas do porquê o tema precisa ser investigado são apresentadas. Os objetivos quanto ao material proposto são definidos e as contribuições do trabalho foram apresentadas.
- **[Capítulo 2: Revisão Bibliográfica]** No Capítulo 2, são apresentados os principais conceitos envolvidos na proposta, incluindo uma descrição geral do ambiente de produção, assim como as características do problema em consideração e também uma breve descrição conceitual de metaheurísticas do tipo Algoritmo Genético para solução de JSSP. Os artigos correlatos são apresentados em seguida, buscando-se apresentar como são abordados os trabalhos que buscam minimizar o *makespan* no problema da programação da produção JSSP.
- **[Capítulo 3: Proposta do Trabalho]** O Capítulo 3 é dedicado à apresentação da proposta deste trabalho, no qual está descrito o passo a passo de como se o material foi desenvolvido para solucionar o problema de programação da produção JSSP. Em outras palavras, destaca-se a construção teórica dos operadores dedicados à condução da busca local e do operador de melhoramento genético.
- **[Capítulo 4: Aplicação da Abordagem e Resultados]** No Capítulo 4, são apresentados os resultados obtidos por meio de ensaios em instâncias JSSP tradicionais na literatura especializada e que possuem o objetivo de auxiliar a execução da proposta.

- **[Capítulo 5: Conclusão]** Por fim, no Capítulo 5, são apresentadas as conclusões do trabalho e perspectivas de avanços futuros neste tema de pesquisa.

Capítulo 2

REVISÃO BIBLIOGRÁFICA

2.1 Considerações iniciais

Neste capítulo, são apresentadas e discutidas as principais fundamentações desse trabalho, incluindo a caracterização do ambiente de aplicação e do problema considerado. Em seguida, é introduzida a descrição das principais metaheurísticas do tipo Algoritmo Genético para solução de JSSP, servindo como fundamento geral para a proposta do trabalho. Também são apresentados trabalhos que compõem o estado da arte de solução de JSSP com metaheurísticas.

Em detalhes, este capítulo está dividido da seguinte maneira: na Seção 2.2, são apresentados os fundamentos teóricos do trabalho, sendo estes a definição de JSSP, e o funcionamento detalhado de técnicas e operadores de busca local em GAs; na Seção 2.3, são apresentados trabalhos relacionados à programação da produção do tipo JSSP, visando indicar as principais questões quanto ao uso de métodos para o problema e de como suas especificidades têm sido exploradas para a tentativa de resolução do problema; este capítulo é finalizado na Seção 2.4 com considerações finais acerca do conteúdo apresentado.

2.2 Fundamentação teórica

Nesta seção, serão apresentados alguns conceitos fundamentais à execução do trabalho, tais como: Programação da Produção, *Job Shop Scheduling Problem* e Algoritmo Genético.

2.2.1 Programação da Produção

Segundo Groover (2007), a programação da produção está inserida nas atividades de Planejamento e Controle da Produção. De posse das informações do planejamento da produção, é determinado o delineamento das operações que serão realizadas, tais como: onde as operações dos *jobs* são processadas; quais recursos são utilizados e qual é o tempo de início e término para cada ordem de produção.

De acordo com Xhafa e Abraham (2008), existem diversos ambientes de programação da produção, alguns dos tipos mais conhecidos são:

- *Single Machine Scheduling Problem*: cada *job* (trabalho) possui apenas uma operação a ser executada por uma única máquina.
- *Parallel Machine Scheduling Problem*: cada *job* possui apenas uma operação a ser executada, porém existem M máquinas disponíveis atuando em modo paralelo, no qual cada *job* pode ser operado por quaisquer das M máquinas.
- *Flow Shop Scheduling Problem* (FSSP): existe um conjunto de M máquinas e de N *jobs*, sendo que as operações de todos os *jobs* apresentam a mesma sequência de execução nas máquinas disponíveis.
- *Job Shop Scheduling Problem* (JSSP): existe um conjunto de M máquinas e de N *jobs*, sendo que as operações dos *jobs* podem possuir sequências de execução diferentes.
- *Open Shop Scheduling Problem* (OSSP): os *jobs* não apresentam restrições internas do ordenamento de operações, podendo ser processados em qualquer ordem de máquinas.

Neste trabalho, foi abordado o problema de programação da produção do tipo *Job Shop Scheduling Problem* (JSSP). Na próxima subseção, é descrita detalhadamente a definição de um JSSP.

2.2.2 Definição do problema de programação da produção JSSP

O JSSP é um problema de otimização combinatória pertencente à classe NP-*Hard* de problemas computacionais, isto é, um problema cujo tempo de processamento é não polinomial. De acordo com Xhafa e Abraham (2008), o JSSP pode ser definido como um conjunto $\mathcal{J}$ de N *jobs* $\{J_1, J_2, \ldots, J_N\}$ a serem processados em um conjunto $\mathcal{M}$ de M máquinas $\{m_1, m_2, \ldots, m_M\}$. Especificamente, em uma instância de um JSSP, cada *job* J_i deve ser processado por todas as

M máquinas possuindo, assim, uma sequência de operações $\mathcal{O}_i = (O_{i1}, O_{i2}, \dots, O_{iM})$ distintas entre si e que descrevem a ordem de processamento, consistindo em uma associação a uma permutação pré-definida dos elementos de $\mathcal{M}$. No caso, as sequências de operações geralmente são diferentes para cada *job* e, portanto, para definir uma configuração de uma instância de um JSSP, é preciso estabelecer uma ordem de prioridade para todas as operações de todos os *jobs*. Desta forma, $\mathcal{O} = (O_1, O_2, \dots, O_{NM})$ é a sequência que define a prioridade com a qual o processamento de cada *job* é iniciado sobre as máquinas de seus respectivos roteiros. Sendo $\mathcal{O}$ uma sequência válida de operações se os seguintes requerimentos forem verdadeiros:

1) $O_k \in \{O_{ij} \mid i = 1,2, \dots, N \text{ e } j = 1,2, \dots, M\}$;
2) As coordenadas de $\mathcal{O}$ são permutações das coordenadas de $\mathcal{O}_i$, para todos os i,
3) Para quaisquer operações $O_k = O_{ij_1}$ e $O_r = O_{ij_2}$ de um mesmo *job* J_i, se $k < r$, então $j_1 < j_2$, para todos k, r, i, j_1, j_2 possíveis. Isto é, as operações nas coordenadas de $\mathcal{O}$ devem respeitar a ordem de processamento do roteiro de todos os *jobs*.

Por exemplo, se a primeira operação O_1 for igual a O_{i1}, então o job J_i tem prioridade para ser processado na primeira máquina de seu roteiro e, consequentemente, o processamento deve ser iniciado com esta operação. O conjunto de todas as sequências válidas de operações é chamado de $\mathbb{O}$.

Cada instância de um JSSP é definida pelos tempos de processamento de seus *jobs*. Especificamente, T_{ij} é o tempo necessário para a j-ésima operação do *job* J_i ser concluída. Além disso, de acordo com a sequência de operações definida em $\mathcal{O}$, considera-se que cada *job* J_i leva um tempo $T_i(\mathcal{O})$ para ser processado por todas as máquinas de seu roteiro e, assim, ser considerado finalizado. O tempo necessário para concluir o processamento de todos os *jobs* de J de acordo com a ordem de operações $\mathcal{O}$ é chamado de *makespan* (MKS). Matematicamente,

$$\text{MKS(O)} = \max_{i \in \{1,2,3,\dots,N\}} T_i(\mathcal{O}). \tag{2.1}$$

Assim, o problema de programação da produção do tipo JSSP consiste em encontrar uma ordem de prioridade $\mathcal{O}$ para as operações de todos os *jobs* de $\mathcal{J}$ com o objetivo de otimizar um critério de desempenho específico, que geralmente é o *makespan*. Além disso, algumas restrições são estabelecidas neste problema.

- Cada *job* pode ser processado em uma única máquina por vez;

- Cada máquina pode processar apenas um *job* de cada vez;
- As operações são consideradas não preemptivas, isto é, não podem ser interrompidas,
- Os tempos de configuração são incluídos nos tempos de processamento e são independentes das decisões de sequenciamento.

Existem diversas medidas de desempenho utilizados para aferir uma programação da produção, tais como (*makespan*, *tardiness*, *lateness*, *maximum workload*, *total workload*, *due date*, etc.) (XHAFA; ABRAHAM, 2008). Neste trabalho, é adotada a já mencionada medida de desempenho *makespan*, isto é, o tempo total necessário para terminar a produção de um conjunto de *jobs*. Deste modo, para solucionar uma instância de um JSSP, é preciso determinar uma sequência de operações válida de forma que o tempo necessário para terminar o processamento de todos os *Jobs* seja mínimo. Matematicamente, esta situação é descrita pela Equação (2.2) a seguir.

$$\min_{\mathcal{O}\in\mathbb{O}}\left\{\max_{i\in\{1,2,3,\ldots,N\}} T_i(\mathcal{O})\right\}. \tag{2.2}$$

A formulação da medida de desempenho *makespan* é geralmente utilizada em abordagens JSSP como função objetivo que orienta os algoritmos que utilizam metaheurísticas a procurar por soluções adequadas, como detalhado na Seção 2.2.4.1.

2.2.2.1 Exemplo de uma instância JSSP

Para exemplificar como os componentes de uma instância de JSSP se relacionam, considera-se uma situação na qual é desejável otimizar a produção de um conjunto com $N = 3$ *jobs*, $\mathcal{J} = \{J_1, J_2, J_3\}$, que devem ser processados por $M = 3$ máquinas pertencentes ao conjunto $\mathcal{M} = \{m_1, m_2, m_3\}$. Nesta situação, os roteiros de todos os *jobs* estão definidos na Tabela 2.1.

Tabela 2.1. Sequências de operação pré-definida, ou roteiro, de cada *job*.

Jobs	$\mathcal{O}_i$
J_1	$\mathcal{O}_1 = (O_{11}, O_{12}, O_{13}) = (m_1, m_2, m_3)$
J_2	$\mathcal{O}_2 = (O_{21}, O_{22}, O_{23}) = (m_3, m_2, m_1)$
J_3	$\mathcal{O}_3 = (O_{31}, O_{32}, O_{33}) = (m_1, m_2, m_3)$

Fonte: Elaborada pela autora.

Vale ressaltar que todos os *jobs* devem ser processados uma única vez por cada máquina de $\mathcal{M}$. Na Tabela 2.2, estão apresentados em unidades de tempo os tempos de processamento necessários para que cada *job* J_i seja processado em cada máquina m_j.

Tabela 2.2. Tempo de processamento, em unidades de tempo, que cada *job* necessita para ser processado em cada máquina.

Jobs	$\boldsymbol{m_1}$	$\boldsymbol{m_2}$	$\boldsymbol{m_3}$
J_1	3	2	3
J_2	1	2	3
J_3	3	2	1

Fonte: Elaborada pela autora.

Neste exemplo, é suposto que a sequência de operações seja a sequência $\mathcal{O}$:

$$\mathcal{O} = (O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8, O_9)$$
$$= (O_{11}, O_{21}, O_{31}, O_{32}, O_{12}, O_{22}, O_{13}, O_{23}, O_{33}).$$

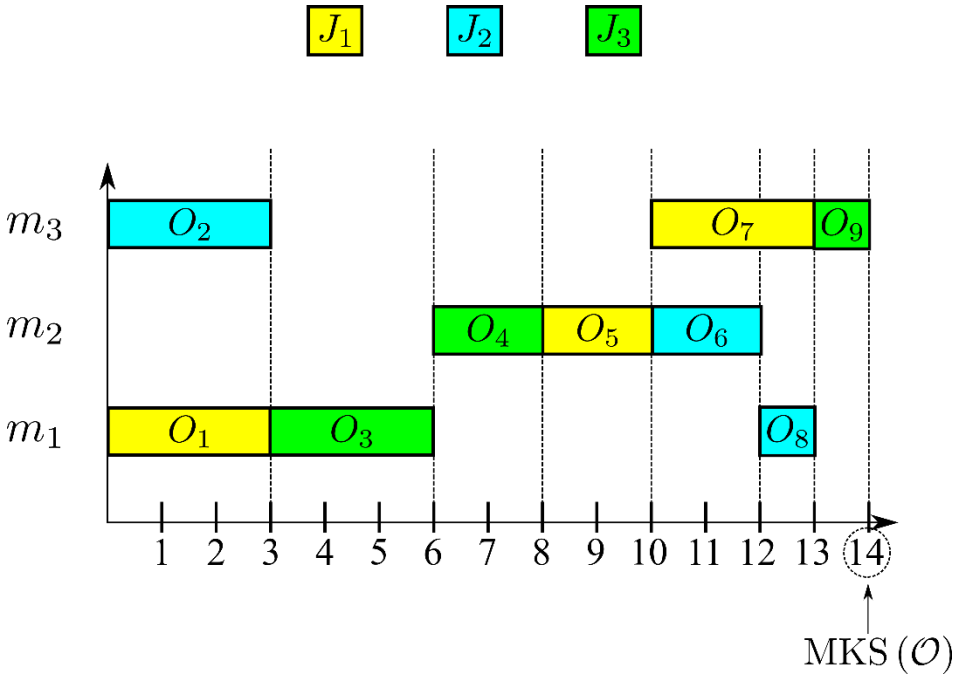
No caso, a ordem de prioridade que o processamento dos *jobs* deve seguir de acordo com $\mathcal{O}$ é:

(1) a primeira operação do *job* J_1 (O_{11});
(2) a primeira operação do *job* J_2 (O_{21});
(3) a primeira operação do *job* J_3 (O_{31});
(4) a segunda operação do *job* J_3 (O_{32});
(5) a segunda operação do *job* J_1 (O_{12});
(6) a segunda operação do *job* J_2 (O_{22});
(7) a terceira operação do *job* J_1 (O_{13});
(8) a terceira operação do *job* J_2 (O_{23}),
(9) e a terceira operação do *job* J_3 (O_{33}).

De acordo com esta configuração, para que todos os *jobs* sejam processados, são necessárias 14 unidades de tempo. Desta forma, o *makespan* desta instância JSSP, seguindo-se a sequência de operações $\mathcal{O}$, é 14. Uma forma bem estabelecida na literatura de conferir visualmente este resultado é a partir do gráfico de Gantt (JIA *et al.*, 2007) da configuração, que

consiste em uma representação da alocação de todos os *jobs* considerando todas as máquinas durante todas as etapas de processamento. Na Figura 2.1, é apresentado o gráfico de Gantt da instância JSSP destacada neste exemplo.

Figura 2.1. Gráfico de Gantt que representa visualmente a alocação dos *Jobs* de $\mathcal{J}$ nas máquinas de $\mathcal{M}$ de acordo com a sequência de operações $\mathcal{O}$.



Fonte: Elaborada pela autora.

2.2.3 Metaheurística: Algoritmo Genético

O Algoritmo Genético (GA) é um método evolucionário de otimização com inspiração no processo evolutivo proposto por Charles Darwin. O GA baseia-se no processo de evolução dos seres vivo, no qual por meio de cruzamentos e mutações, a população inicial tem uma tendência ao melhoramento, pois os seres mais adaptados ao ambiente sobrevivem e os menos adaptados ao ambiente não sobrevivem (MITCHELL, 1998). Os GAs foram idealizados por John Holland na década de 60. Nas pesquisas realizadas sobre os GAs, John Holland publicou um livro sobre este assunto, intitulado: "*Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*" (HOLLAND, J., 1975).

O GA é um método estocástico, ou seja, é possível encontrar soluções diferentes a cada execução do método mesmo que se utilize a mesma população inicial e a mesma parametrização (RAO, 2009). Basicamente, o GA consiste no desenvolvimento de um conjunto de elementos conhecidos como cromossomos (ou indivíduos), que geralmente são vetores pertencentes ao $\mathbb{R}^n$, por meio de um número pré-estabelecido de iterações (ou gerações). Sendo que para cada cromossomo desta simulação é associado um valor numérico por meio de uma função de adaptação conhecida como função *fitness* que representa o quão bem adaptado ao meio o respectivo cromossomo está. Além disso, a formulação desse algoritmo consiste na simulação de dois operadores primordiais: o cruzamento e a mutação. Sendo o "operador de cruzamento" uma técnica que busca combinar possíveis soluções de um problema que sejam consideradas mais adequadas, simulando, assim, o conceito de que indivíduos bem adaptados a um meio devem gerar mais descendentes do que indivíduos menos adaptados. O "operador de mutação" geralmente é responsável por realizar modificações (perturbações) em possíveis soluções de um problema a fim de simular a geração de características diferentes em indivíduos de um ambiente e, assim, ampliar o poder de busca da técnica. Finalmente, técnicas de atualização populacional se fazem necessárias nesse tipo de simulação, uma vez que soluções que não sejam adequadas ao problema tratado devam ser removidas do procedimento dando espaço ao desenvolvimento de boas soluções.

O funcionamento de um GA básico é sumarizado no Algoritmo 2.1.

Algoritmo 2.1.: Algoritmo Genético básico.

Entrada	n: Número de indivíduos na população. n_{Gen}: Número máximo de gerações. $\Delta_{\times}$: Taxa de cruzamento. $f_{\times}$: Função de cruzamento. Δ_m: Taxa de mutação. f_m: Função de mutação. f_{nova}: Função de criação de nova população. F: Função de *fitness*.

Saída	c^*: Indivíduo com melhor[i] valor de *fitness*.

$P_1 \coloneqq \{c_1, c_2, \ldots, c_n\}$; // Gerar população inicial.

PARA $i \coloneqq 1$ **ATÉ** n_{Gen} **FAÇA**

  $\mathcal{F} \coloneqq \{F(c_1), F(c_2), \ldots, F(c_3)\}$; // Avaliar os novos indivíduos.

  $P_{\text{Selecionados}} \coloneqq \{\ \}$;

  PARA CADA $c \in P_i$ **FAÇA**

    $r \coloneqq \text{rand}(0,1)$; // $\text{rand}(a, b)$ é a função que gera um número aleatório entre a e b.

    SE $r < \Delta_\times$ **ENTÃO**

      $P_{\text{Selecionados}} \coloneqq P_{\text{Selecionados}} \cup \{c\}$; // Selecionar indivíduos para cruzamento.

    FIM_SE

  FIM_PARA

  $P_{\text{Filhos}} \coloneqq \{\ \}$;

  PARA CADA $p_1, p_2 \in P_{\text{Selecionados}}$ **FAÇA**

    $P_{\text{Filhos}} \coloneqq P_{\text{Filhos}} \cup \{f_\times(p_1, p_2)\}$; // Realizar cruzamento.

  FIM_PARA

  PARA CADA $c \in P_{\text{Filhos}}$ **FAÇA**

    $r \coloneqq \text{rand}(0,1)$;

    SE $r < \Delta_m$ **ENTÃO**

      $c \coloneqq f_m(c)$; // Realizar mutação.

    FIM_SE

  FIM_PARA

  $P_{i+1} \coloneqq f_{\text{nova}}(P_i \cup P_{\text{Filhos}})$; // Gerar nova população.

FIM_PARA

Fonte: Adaptado de Goldberg e Holland (1988).

Para resolver cada problema estabelecido, é necessário adaptar o GA para isso. Desta forma, na Seção 2.2.4, é apresentado o uso de GAs na resolução de instâncias JSSP, assim como a descrição de seus operadores básicos.

[i] Neste texto, sempre que for afirmado que um indivíduo possui *fitness* "melhor" do que o *fitness* de outros indivíduos, então, na verdade, este possui valor de *fitness* menor do que o apresentado por outros indivíduos, tendo em vista de que, neste trabalho, as metaheurísticas são empregadas para minimizar uma função objetivo.

2.2.4 Algoritmo Genético e suas variações para JSSP

Nesta seção, são detalhadas algumas fundamentações nas quais as técnicas propostas são baseadas, sendo que estas tratam da minimização da medida de desempenho *makespan* em ambientes de testes do tipo JSSP propostos na literatura. Especificamente, são apresentadas as principais estratégias e operadores de métodos baseados no GA cujo funcionamento é especializado na resolução de instâncias JSSP.

GAs são amplamente utilizados na resolução de problemas combinatórios desde sua formulação por J. Holland em 1975 (HOLLAND, J. H.; OTHERS, 1992). Entretanto, seu uso foi amplamente difundido em meados de 1980 (SHL, 1997) e, em 1991, Nakano e Yamada , em seu trabalho intitulado "*Conventional genetic algorithm for job shop problems*", propuseram uma codificação do GA para encontrar soluções para JSSPs, sendo este o primeiro trabalho que aborda JSSPs com uma heurística evolutiva. Os trabalhos que investigam soluções de JSSPs com técnicas heurísticas se tornaram muito numerosos nos últimos anos, em especial os trabalhos que fazem uso de algoritmos genéticos modificados, devido à simplicidade e à aplicabilidade destes.

2.2.4.1 Função objetivo (fitness): Makespan

Como já citado anteriormente neste texto, a otimização de programação da produção pode ser feita com respeito a diversas medidas relacionadas ao tempo gasto para produzir um conjunto de *jobs* ou realizar uma série de tarefas. Sendo a medida conhecida como *makespan* uma das mais utilizadas como padrão de qualidade ao se tratar do tempo gasto em uma programação da produção, uma vez que a mesma representa o tempo total gasto para processar um conjunto de N *jobs* em um conjunto de M máquinas, obedecendo as mesmas restrições já estabelecidas na Seção 2.2.2 para a configuração de uma instância JSSP.

Quanto menor for o valor de *makespan* de uma programação da produção, menos tempo deve ser empreendido para finalizar a produção de um dado conjunto de *jobs*. Desta forma, o GA deve procurar opções de configurações para uma programação da produção de forma a minimizar o tempo gasto na conclusão do processamento do conjunto de *jobs* no conjunto de

máquinas que configuram o JSSP. Sendo assim, é pertinente que a função objetivo que o GA leve em consideração para obter um valor mínimo seja a medida de *makespan*.

Portanto, a função objetivo, ou função *fitness*, dos GAs aqui abordados pode ser modelada de acordo com a função F, definida na Equação (2.3) dada a seguir.

$$\begin{array}{ccccc} F & : & \mathbb{O} & \rightarrow & \mathbb{R} \\ & & \mathcal{O} & \mapsto & F(\mathcal{O}) \coloneqq \mathrm{MKS}(\mathcal{O}), \end{array} \tag{2.3}$$

sendo $\mathbb{O}$ o conjunto de todas as sequências possíveis para o JSSP definido. Isto é, se $\mathcal{O} \in \mathbb{O}$, então $\mathcal{O}$ é uma sequência de operações que define a prioridade de início de processamento dos *jobs* de $\mathcal{J}$ nas máquinas de $\mathcal{M}$. Em outras palavras, $\mathbb{O}$ é o conjunto factível de soluções no qual o GA deve realizar suas buscas.

2.2.4.2 Representação genética

Salvo a presença de operadores específicos de cada trabalho, a estrutura básica de um GA continua sendo formada pelo laço de repetição que envolve dois principais operadores: operador de cruzamento e o operador de mutação. Sendo essa estrutura preservada na grande maioria das técnicas presentes no estado da arte. Enquanto que a codificação utilizada para representar uma possível solução (cromossomo) do problema pode ser feita de várias maneiras diferentes, como destacado por Jorapur *et al.* (2014).

Jorapur *et al.* (2014) também ressaltam que as representações cromossômicas disponíveis são isomorfas e, portanto, devem explorar o espaço de busca com a mesma taxa aproximada de sucesso, uma vez que tais representações não possuem capacidade de apresentar grandes modificações nos resultados obtidos em sua utilização.

Uma das representações mais usuais da literatura é conhecida como “codificação por ordem de operação”, apresentada pela primeira vez por Bierwirth, Mattfeld e Kopfer (1996). Sendo que, nesta representação, o espaço de soluções de um JSSP de N *jobs* e M máquinas é formado por cromossomos $c \in \mathbb{N}^{N \cdot M}$, tais que exatamente M coordenadas de c são iguais a i (representando o *job* de índice i), para todo $i \in \{1,2, \ldots, N\}$. Na Figura 2.2, são apresentados

alguns exemplos de cromossomos (c_1, c_2 e c_3) que obedecem a esta formulação em um JSSP com dois *jobs* ($N = 2$) e três máquinas ($M = 3$). Como requere a formulação, o *job* de índice 1 e o *job* de índice 2 aparecem exatamente 3 vezes, uma vez que 3 é o número de máquinas no problema.

Figura 2.2. Exemplos de cromossomos na representação por operação.

$$c_1 = (1, 1, 2, 1, 2, 2)$$
$$c_2 = (2, 2, 2, 1, 1, 1)$$
$$c_3 = (1, 2, 1, 2, 1, 2)$$

Fonte: Elaborada pela autora.

Esta codificação determina a prioridade que cada operação possui na alocação de máquinas. Como exemplo, seja $c = (1,2,1,1,2,2)$ um cromossomo em um JSSP de dimensões 2×3, isto é, com dois *jobs* e três máquinas. Neste caso, a ordem estabelecida por c define que as seguintes ações devem ser realizadas sequencialmente e apenas devem ser iniciadas se a mesma puder ser realizada paralelamente à ação anterior ou caso a ação anterior já tenha sido finalizada:

1ª) O *job* 1 deve ser processado pela 1ª máquina de seu roteiro.

2ª) O *job* 2 deve ser processado pela 1ª máquina de seu roteiro.

3ª) O *job* 1 deve ser processado pela 2ª máquina de seu roteiro.

4ª) O *job* 1 deve ser processado pela 3ª máquina de seu roteiro.

5ª) O *job* 2 deve ser processado pela 2ª máquina de seu roteiro.

6ª) O *job* 2 deve ser processado pela 3ª máquina de seu roteiro.

Assim, uma forma de gerar a população inicial neste tipo de configuração é criar um grupo de cromossomos iguais a $(1, \dots, 1, 2, \dots, 2, \dots, N, \dots, N)$ em que cada um dos N *jobs* de um JSSP aparece em exatamente M posições e, a seguir, reorganizar aleatoriamente todas as

coordenadas destes cromossomos. Matematicamente, considera-se a função f_{shuffle} que reordena aleatoriamente as coordenadas de um dado vetor, definido na Equação (2.4)

$$\begin{array}{rcccc} f_{\text{shuffle}} & : & \{(1,\dots,1,2,\dots,2,\dots,N,\dots,N)\} & \rightarrow & \{1,2,3,\dots,N\}^{N\cdot M} \\ & & (x_1, x_2, \dots, x_n) & \mapsto & \left(x_{i_1}, x_{i_2}, \dots, x_{i_n}\right), \end{array} \tag{2.4}$$

sendo $i_1, i_2, \dots, i_n$ uma disposição aleatória dos índices $1, 2, \dots, n$.

Então, um possível algoritmo de geração de população inicial para a codificação apresentada é o Algoritmo 2.2:

Algoritmo 2.2. Geração de população inicial.

Entrada	N_{Pop}: Número de indivíduos. $N \times M$: Dimensão do JSSP.
Saída	P_0: População inicial.

$P_0 = \{\ \ \};$

PARA $i := 1$ **ATÉ** N_{Pop} **FAÇA**

$c := (\underbrace{1, 1, \dots, 1}_{M \text{ vezes}}, \underbrace{2, 2, \dots, 2}_{M \text{ vezes}}, \dots, \underbrace{N, N, \dots, N}_{M \text{ vezes}});$

$\hat{c} := f_{\text{shuffle}}(c);$ // f_{shuffle} é definida na Equação (2.4).

$P_0 := P_0 \cup \{\hat{c}\};$

FIM_PARA

Fonte: Elaborado pela autora.

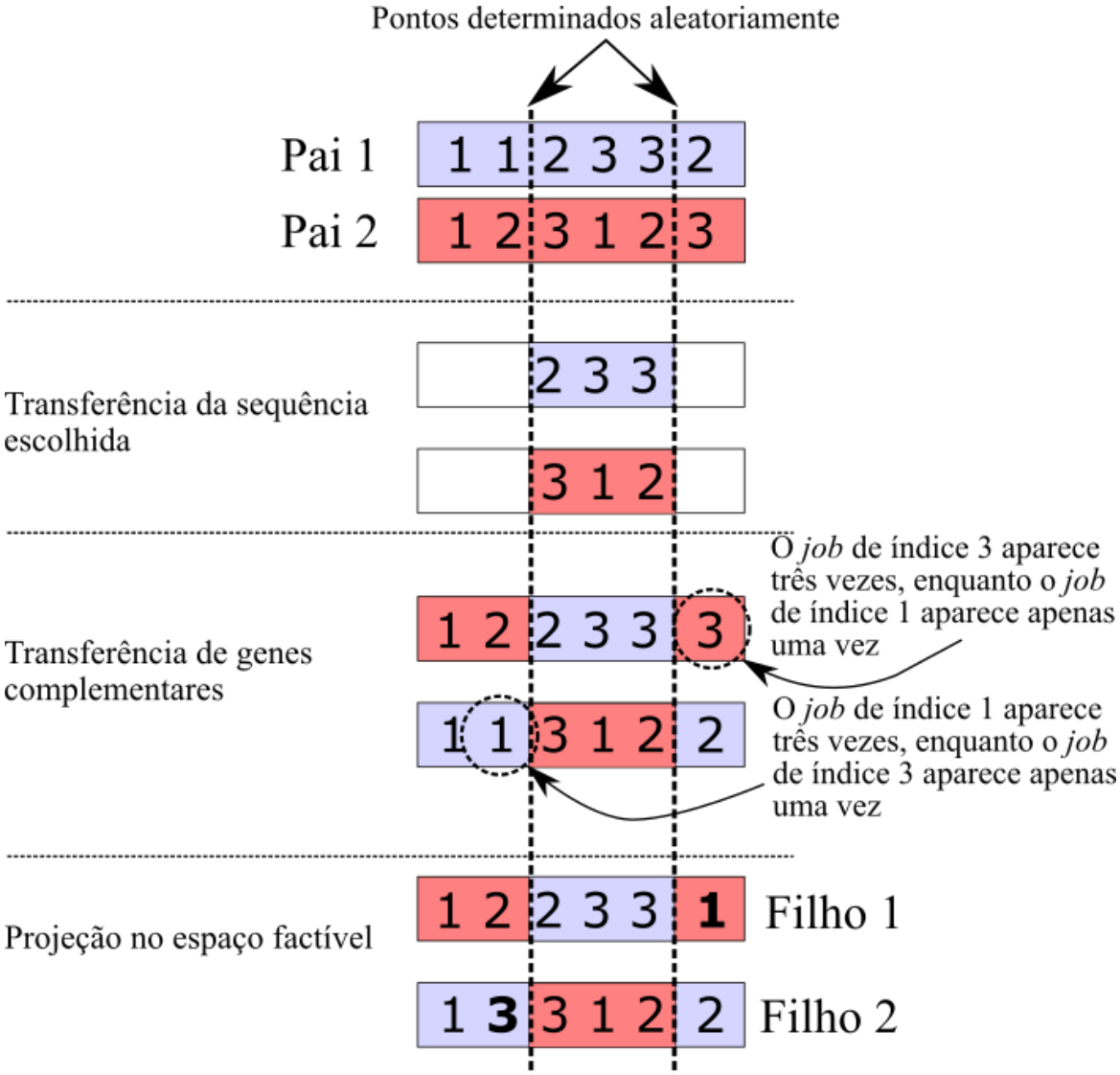
2.2.4.3 Operadores de cruzamento em JSSP

Nesta seção, são abordados os dois tipos de cruzamento mais utilizados em GAs na solução de JSSPs. Desta forma, a fundamentação destes operadores é apresentada de acordo com a codificação cromossômica estabelecida na Seção 2.2.4.2 anterior.

2.2.4.3.1 Operador de cruzamento parcialmente mapeado (PMX)

Nesta técnica, originalmente conhecida como *Partially Mapped Crossover* (PMX) (GOLDBERG; LINGLE; OTHERS, 1985), é feito uma cópia de uma subsequência, a qual é definida por dois pontos de cortes determinados aleatoriamente, de um cromossomo pai para um cromossomo filho respeitando-se as posições ocupadas por essa sequência. Em seguida, transferem-se para as posições vagas no cromossomo filho os genes do segundo cromossomo pai que não ocupam as mesmas posições da sequência já transferida. Finalmente, projeta-se o cromossomo filho gerado no espaço de busca factível do problema abordado, uma vez que neste cruzamento não existe a garantia de que o cromossomo gerado seja uma possível solução do problema, sendo assim, é necessário corrigir alguns genes no cromossomo filho para transformá-lo em uma possível solução. Sendo toda alteração necessária devendo ser realizada nos genes complementares à sequência escolhida inicialmente. Para conduzir esta correção, geralmente conduz-se a projeção para o espaço factível de acordo com a distância de Hamming (WEGNER, 1960). Na Figura 2.3 a seguir, é exemplificado o funcionamento deste cruzamento em dois cromossomos de um JSSP com três *jobs* e duas máquinas (3×2).

Figura 2.3. Exemplo de cruzamento PMX.



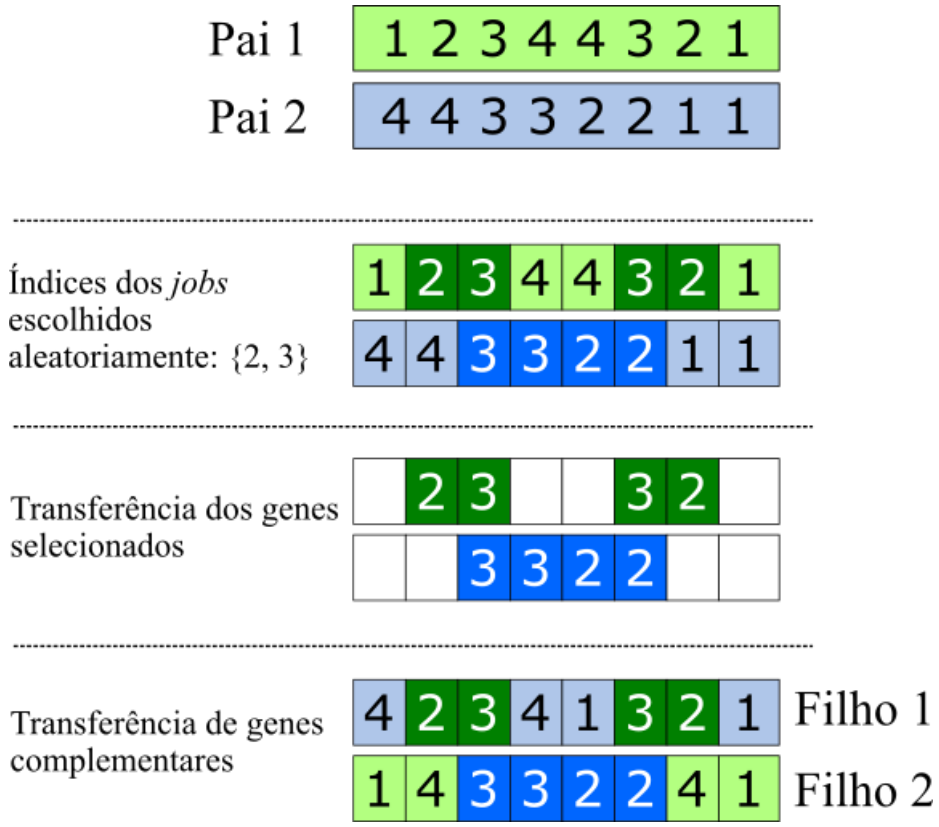
Fonte: Elaborada pela autora.

2.2.4.3.2 Operador de cruzamento baseado em ordem (OX2)

Este cruzamento, conhecido originalmente como *Order-based Crossover* (OX2) (ANAND; PANNEERSELVAM, 2016), é um tipo de cruzamento muito utilizado em modelagens de GA aplicadas em JSSP, uma vez que esta técnica apresenta como resultado cromossomos pertencentes ao conjunto factível do problema tratado. Além disso, o OX2 é um tipo de cruzamento que leva em consideração a ordem com a qual os genes são dispostos nos cromossomos pais.

Especificamente, para realizar o OX2 entre dois cromossomos pais, seleciona-se randomicamente alguns índices de *jobs*. Em seguida, cria-se um cromossomo filho que possua, na mesma posição genética de um dos pais os respectivos índices que foram selecionados inicialmente. Os genes restantes deste cromossomo filho devem ser preenchidos com os índices dos *jobs* complementares, respeitando-se a sequência com a qual estes estejam dispostos no segundo cromossomo pai. Na Figura 2.4 a seguir, um exemplo deste cruzamento em cromossomos de um JSSP 4×2 pode ser visualizado.

Figura 2.4. Exemplo de cruzamento OX2.



Fonte: Elaborada pela autora.

Por construção, todo filho gerado neste cruzamento é uma solução factível, uma vez que a técnica lida com a transferência de todos os *jobs* de um determinado conjunto de índices. Sendo assim, não é necessário realizar nenhuma projeção ou correção nos indivíduos gerados.

2.2.4.4 Operadores de mutação em JSSP

São apresentadas nesta seção as três funções de mutação mais utilizadas em GAs adaptados para solucionar JSSPs: *swap*, *inverse* e *insert* (AMJAD *et al.*, 2018). São técnicas amplamente difundidas na literatura especializada e que também são utilizadas na proposta deste trabalho. De maneira geral, uma função de mutação f deve ser definida de acordo com a dimensão $N \times M$ da instância JSSP e é uma transformação que opera sobre duas coordenadas de um cromossomo no espaço de buscas $\mathbb{O}$ e, matematicamente, obedecem a forma geral da Equação (2.5) a seguir.

$$\begin{array}{ccccc} f & : & \mathbb{O} \times \{1,2,\dots,N \cdot M\}^2 & \rightarrow & \mathbb{O} \\ & & (c,(i,j)) & \mapsto & \hat{c} = f(c,(i,j)), \end{array} \tag{2.5}$$

sendo $\hat{c}$ uma permutação das coordenadas de c e que, portanto, é uma solução factível por construção.

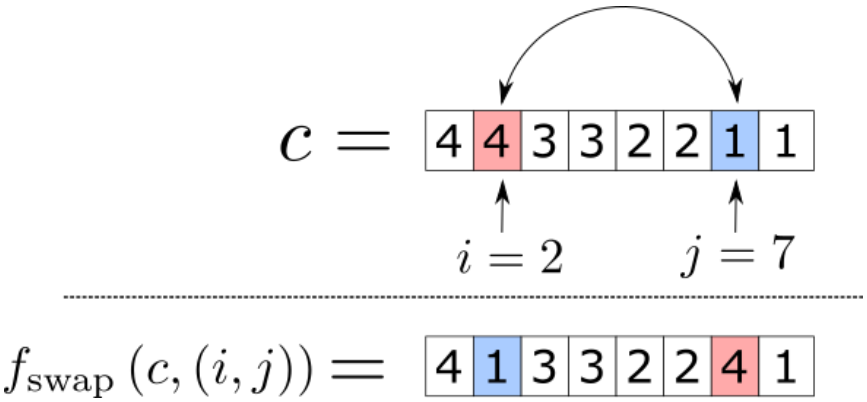
2.2.4.4.1 Mutação de Troca de operações (Swap)

Esta mutação é modelada pela função f_{swap}, definida na Equação (2.6):

$$\begin{array}{ccccc} f_{\text{swap}} & : & \mathbb{O} \times \{1,2,\dots,N \cdot M\}^2 & \rightarrow & \mathbb{O} \\ & & ((x_1,x_2,\dots,x_n),(i,j)) & \mapsto & (x_1,x_2,\dots,x_{i-1},x_j,x_{i+1},\dots,x_{j-1},x_i,x_{j+1},\dots,x_n) \end{array} \tag{2.6}$$

Neste tipo de mutação, são realizadas trocas simultâneas entre duas dadas coordenadas de uma possível solução, como exemplificado na Figura 2.5 a seguir supondo-se uma instância JSSP de dimensão 4×2.

Figura 2.5. Exemplo de mutação *Swap*.



Fonte: Elaborada pela autora.

2.2.4.4.2 Mutação de Inversão de operações (Inverse)

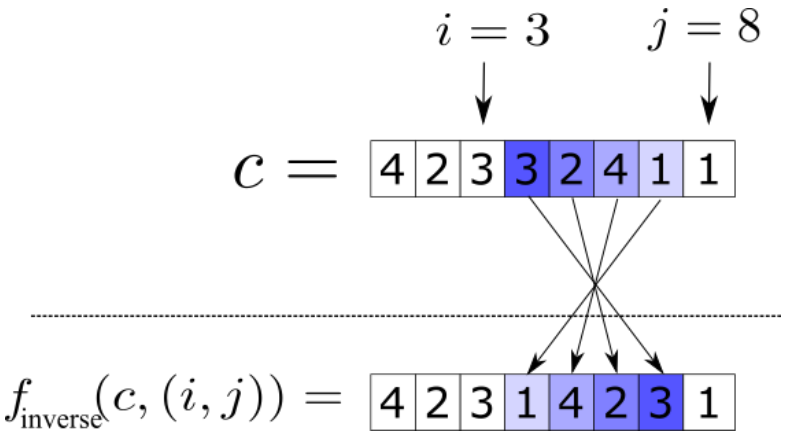
Ao utilizar essa mutação em um cromossomo, uma subsequência, determinada aleatoriamente, desta solução deve ser invertida. Desta forma, modela-se este procedimento pela função f_{inverse} definida na Equação (2.7):

$$f_{\text{inverse}} : \mathbb{O} \times \{1,2,\dots,N \cdot M\}^2 \rightarrow \mathbb{O}$$

$$\big((x_1, x_2, \dots, x_n), (i,j)\big) \mapsto \begin{cases} (x_1, x_2, \dots, x_i, x_{j-1}, x_{j-2}, \dots, x_{i+1}, x_j, x_{j+1}, \dots, x_n), \text{ se } i < j, \\ (x_1, x_2, \dots, x_j, x_{i-1}, x_{i-2}, \dots, x_{j+1}, x_i, x_{i+1}, \dots, x_n), \text{ se } j \leq i. \end{cases} \quad (2.7)$$

Na Figura 2.6 a seguir, é apresentada uma aplicação desta mutação em uma possível solução de um JSSP 4×2.

Figura 2.6. Exemplo de mutação *Inverse*.



Fonte: Elaborada pela autora.

2.2.4.4.3 Mutação de Inserção de operações (Insert)

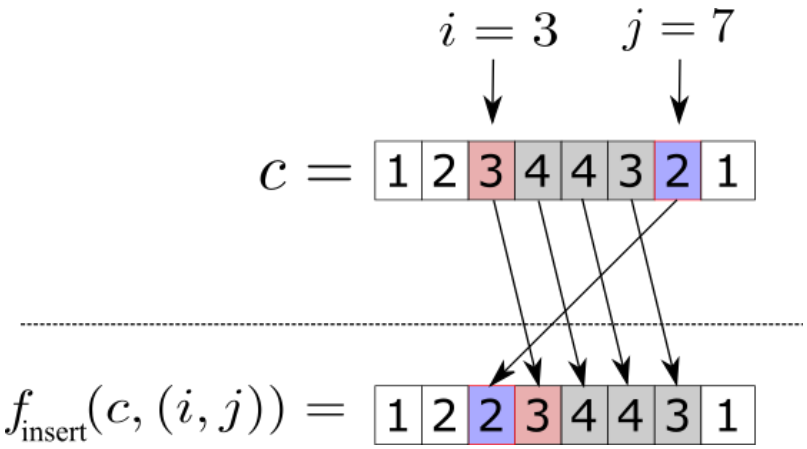
Ao aplicar essa mutação, realiza-se o deslocamento de uma subsequência qualquer de um cromossomo para uma coordenada à direita e, em seguida, o último termo dessa subsequência passa a ocupar a coordenada que fica vaga nesse processo. Esta mutação é modelada pela função f_{insert}, definida na Equação (2.8), a seguir:

$$f_{\text{insert}} : \mathbb{O} \times \{1,2,\dots,N \cdot M\}^2 \rightarrow \mathbb{O}$$

$$\big((x_1, x_2, \dots, x_n), (i,j)\big) \mapsto \begin{cases} (x_1, x_2, \dots, x_{i-1}, x_j, x_i, x_{i+1}, \dots, x_{j-1}, x_{j+1}, \dots, x_n), \text{ se } i < j, \\ (x_1, x_2, \dots, x_{j-1}, x_i, x_j, x_{j+1}, \dots, x_{i-1}, x_{i+1}, \dots, x_n), \text{ se } j \leq i. \end{cases} \quad (2.8)$$

Na Figura 2.7, é apresentado um exemplo deste tipo de mutação em uma possível solução a uma instância JSSP de dimensão 4×2.

Figura 2.7. Exemplo de mutação *Insert*.



Fonte: Elaborada pela autora.

2.2.4.5 Algoritmo genético básico para JSSP

É sumarizado no Algoritmo 2.3 a seguir, a forma padrão de um GA para tratar um JSSP. Sendo que o método utilizado na fase de geração de nova população geralmente é a inserção pela técnica de roleta viciada retendo-se o melhor indivíduo da população (GOLDBERG; HOLLAND, 1988), uma vez que esse método se mostrou superior ao evitar convergência precoce na maioria dos trabalhos atuais (ASADZADEH, 2015).

Algoritmo 2.3. Algoritmo genético básico para resolução de JSSP.

Entrada	N_{Pop}: Número de indivíduos na população. N_{Gen}: Número de gerações. Δ_{cruz}: Taxa de cruzamento. Δ_{mut}: Taxa de mutação. $N \times M$:Dimensão do JSSP. $F:$ Função de fitness.
Saída	c^*: Melhor solução obtida. F^*: $F(c^*)$ (Melhor valor de *fitness* obtido).

P_1:= **Algoritmo** 2.2 ($N_{\text{Pop}}, N \times M$);
$N_{\text{cruz}} = \text{round}(N_{\text{Pop}} \cdot \Delta_{\text{cruz}})$; // round é a função "arredondamento".
$N_{\text{mut}} = \text{round}(N_{\text{Pop}} \cdot \Delta_{\text{mut}})$;
PARA $i =: 1$ **ATÉ** N_{Gen} **FAÇA**
    $P_{\text{filhos}} = \{\ \}$;
    PARA $j \coloneqq 1$ **ATÉ** N_{cruz} **FAÇA**
        $c_1 \coloneqq \text{rand\_set}(P_i)$; /* rand_set é a função que seleciona um indivíduo aleatoriamente em um dado conjunto. */
        $c_2 \coloneqq \text{rand\_set}(P_i)$;
        $c \coloneqq f_{\times}(c_1, c_2)$; /* $f_{\times}$ é a função que realiza um dos cruzamentos descritos na Seção 2.2.4.3. */
        $P_{\text{filhos}} \coloneqq P_{\text{filhos}} \cup \{c\}$;
    FIM_PARA

    $P_{\text{mut}} \coloneqq \{\ \}$;
    PARA $j \coloneqq 1$ **ATÉ** N_{mut} **FAÇA**
        $c \coloneqq \text{rand\_set}(P_{\text{filhos}})$;
        $\hat{c} \coloneqq \text{MUT}(c)$; /* MUT é a função que realiza uma das mutações descritas na Seção 2.2.4.4. */
        $P_{\text{mut}} \coloneqq P_{\text{mut}} \cup \{\hat{c}\}$;
    FIM_PARA

    $P_{i+1} := \text{ROLETA}(P_i)$; /* Criação da nova população pela técnica da roleta viciada retendo-se o melhor indivíduo (GOLDBERG; HOLLAND, 1988). */
FIM_PARA

Fonte: Elaborado pela autora.

2.2.4.6 Operadores especializados em Busca local

De acordo com as inspirações biológicas (HOLLAND, 1992), o operador de cruzamento é idealizado como operador de busca local, uma vez que os indivíduos gerados (filhos) devem ser semelhantes com os indivíduos sobre os quais sua construção foi baseada (pais), pois este

procedimento funciona apenas em função do "conhecimento" armazenado em cada indivíduo a ser cruzado. Ao mesmo tempo em que o operador de mutação foi idealizado como sendo um operador de busca global, uma vez que mesmo a menor das perturbações aos genes de um cromossomo pode gerar grandes modificações no indivíduo, o que é observado em fenômenos tais como o fenômeno conhecido por "Abismo de Hamming" (MITCHELL, 1998).

Tais inspirações estão relacionadas aos termos *exploitation* e *exploration,* os quais fazem referência, respectivamente, à exploração local e à exploração global de um espaço de soluções de um determinado problema a ser resolvido por metaheurística. Em detalhes, uma exploração de caráter global conduz as buscas com respeito a todo espaço considerado, enquanto que a exploração de caráter local conduz as buscas com respeito à vizinhança de um indivíduo em especial (WATANABE; IDA; GEN, 2005). Alguns autores ponderam o uso entre busca local e busca global em técnicas de otimização que fazem uso de metaheurística para aprimorar o funcionamento de suas técnicas e realizar buscas equilibradas e que abrangem satisfatoriamente o espaço de soluções.

Neste trabalho, são abordadas principalmente técnicas que fazem uso de buscas locais aprimoradas e modeladas por meio de operadores específicos para tal finalidade, uma vez que a utilização de tais técnicas em GAs se mostrou muito vantajosa em trabalhos recentes que procuram soluções para JSSPs (ASADZADEH, 2015).

2.2.4.6.1 Busca local em operadores de AG

Watanabe, Ida e Gen (2005) propõem o acréscimo de rotinas de busca local nos operadores de cruzamento e mutação para aprimorar a exploração local (*exploitation*) nas soluções geradas nestes operadores.

Especificamente, o cruzamento é realizado par-a-par entre três indivíduos selecionados aleatoriamente na população e repetido por um número R_c de vezes ou até que seja gerado um indivíduo melhor que o pior indivíduo da população, como esquematizado no Algoritmo 2.4 a seguir.

Algoritmo 2.4. Operador de busca local em cruzamento de Watanabe, Ida e Gen (2005)

Entrada	F: Função *fitness*. p_1, p_2, p_3: Indivíduos distintos selecionados aleatoriamente. f_{worst}: Pior valor de *fitness* apresentado na população.
Saída	c_1, c_2, c_3: Filhos gerados a partir dos pais p_1, p_2 e p_3.

PARA $k \coloneqq 1$ **ATÉ** 3 **FAÇA**
  SE $k == 1$ **ENTÃO**
    $P_1 \coloneqq p_1$;
    $P_2 \coloneqq p_2$;
  SENÃO SE $k == 2$ **ENTÃO**
    $P_1 \coloneqq p_1$;
    $P_2 \coloneqq p_3$;
  SENÃO
    $P_1 \coloneqq p_2$;
    $P_2 \coloneqq p_3$;
  FIM_SE

  PARA $i \coloneqq 1$ **ATÉ** R_c **FAÇA**
    $\hat{c}_i \coloneqq \text{PMX}(P_1, P_2)$; // PMX é a operação de cruzamento descrita na Seção 2.2.4.3.1.
    $F_i \coloneqq F(\hat{c}_i)$;
    SE $F_i < f_{\text{worst}}$ **ENTÃO** "**SAIA DO PARA**";
  FIM_PARA

  $i^* \coloneqq \arg\min_i\{F_i\}$; // Índice referente ao melhor valor de *fitness*.
  $c_k \coloneqq \hat{c}_{i^*}$; // Cromossomo filho que possui o melhor valor de *fitness*.

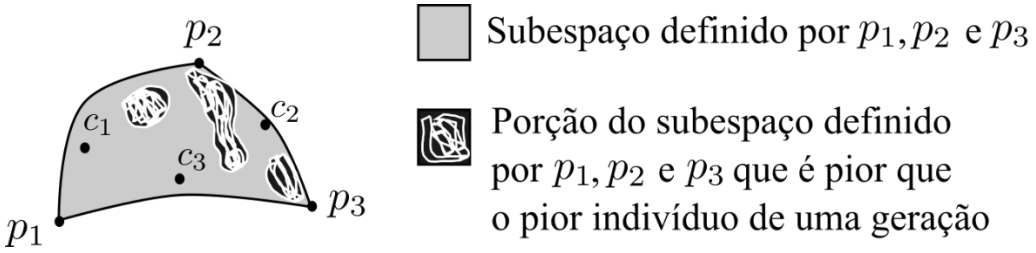
FIM_PARA

Fonte: Adaptado de Watanabe, Ida e Gen (2005).

Desta forma, o Algoritmo 2.4 apresenta um método que procura combinações entre dois indivíduos que sejam boas o suficiente para serem levadas em consideração no decorrer do

método. Isto é, dados três indivíduos, realizam-se buscas no espaço determinado por estes para se encontrar três possíveis soluções que sejam pelo menos melhores que o pior indivíduo de uma geração, como esquematizado na Figura 2.8.

Figura 2.8. Interpretação geométrica do operador de cruzamento de Watanabe, Ida e Gen (2005).



Fonte: Elaborada pela autora.

A mutação no algoritmo de Watanabe, Ida e Gen (2005) tem forte caráter de busca local sendo que, para cada indivíduo gerado no processo de cruzamento do Algoritmo 2.4 são geradas R_m mutações do mesmo e sua melhor mutação, caso apresente *fitness* melhor do que o pior *fitness* disposto na população, substitui o seu respectivo pai na população para a próxima iteração do método. Caso a melhor mutação gerada de um indivíduo não apresente *fitness* melhor do que o pior *fitness* apresentado na população corrente, então é selecionado randomicamente um dentre todos os cromossomos da população corrente e a melhor mutação do indivíduo avaliado. Sendo que o elemento assim selecionado substitui o seu respectivo pai na população. O Algoritmo 2.5 sumariza esta técnica de mutação.

Algoritmo 2.5. Operador de busca local em mutação de Watanabe, Ida e Gen (2005).

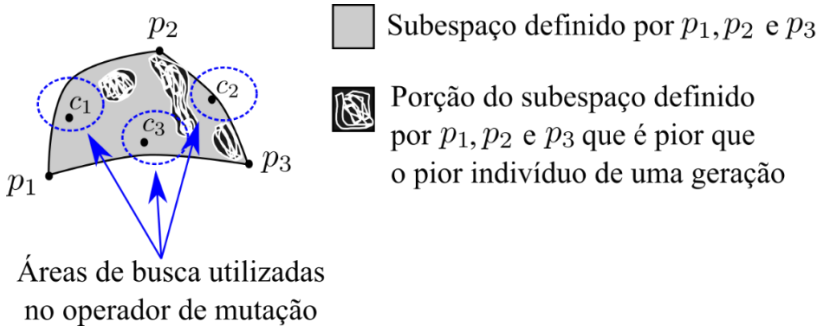
Entrada	P_{filhos}: População de filhos gerada utilizando o Algoritmo 2.4. F: Função $fitness$. f_{worst}: Pior valor de *fitness* apresentado na população corrente. $P_i = \{p_1, p_2, \dots, p_{N_{\text{Pop}}}\}$: População atual. N_{dim}: Número de genes em cada cromossomo.
Saída	P_{i+1}: Nova população.

$P_{i+1} \coloneqq \{\ \ \}$;
PARA CADA $c \in P_{\text{filhos}}$ **FAÇA**
  PARA $j \coloneqq 1$ **ATÉ** R_m **FAÇA**
    $r_1 \coloneqq \text{randi}(1, N_{\text{dim}})$; /* A função $\text{randi}(a, b)$ gera um número inteiro aleatório entre a e b. */
    $r_2 \coloneqq \text{randi}(1, N_{\text{dim}})$;
    $\hat{c}_j \coloneqq f_{\text{swap}}(c, (r_1, r_2))$; // Equação (2.6).
    $F_j \coloneqq F(\hat{c}_j)$;
  FIM_PARA
  $j^* \coloneqq \arg\min_j F_j$;
  $\hat{c}^* \coloneqq \hat{c}_{j^*}$; // Melhor indivíduo gerado.
  SE $F(\hat{c}^*) < f_{\text{worst}}$ **ENTÃO**
    $P_{i+1} \coloneqq P_{i+1} \cup \{\hat{c}^*\}$;
  SENÃO // Caso o melhor indivíduo gerado não seja melhor que o pior indivíduo de P_i.
    $p \coloneqq \text{rand\_set}(\{p_1, p_2, \ldots, p_{N_{\text{Pop}}}, \hat{c}^*\})$; // Toma-se aleatoriamente um indivíduo
                                                // considerando o conjunto formado pelos
                                                // indivíduos de P_i e por $\hat{c}^*$.
    $P_{i+1} \coloneqq P_{i+1} \cup \{p\}$;
  FIM_SE
FIM_PARA

Fonte: Adaptado de Watanabe, Ida e Gen (2005).

O Algoritmo 2.5 descreve uma técnica de busca local que reforça a procura por indivíduos relativamente bons próximos aos indivíduos gerados no operador de cruzamento apresentado no Algoritmo 2.4. Isto é, a técnica procura o melhor indivíduo existente na vizinhança de cada filho gerado no cruzamento do GA, como ilustrado na Figura 2.9.

Figura 2.9. Interpretação geométrica do operador de mutação de Watanabe, Ida e Gen (2005).



Fonte: Elaborada pela autora.

Além disso, o operador de mutação também é responsável pela atualização populacional, fazendo com que o GA de Watanabe, Ida e Gen (2005) seja descrito pelo Algoritmo 2.6 a seguir:

Algoritmo 2.6. Algoritmo genético de Watanabe, Ida e Gen (2005) aplicado em JSSP.

Entrada	N_{Pop}: Número de indivíduos na população. N_{Gen}: Número de gerações do AG. R_c: Número máximo de repetições de cruzamento por indivíduos. R_m: Número exato de repetições de mutação por indivíduos. $N \times M$: Dimensões do JSSP. F: Função *fitness* do problema.
Saída	c^*: Melhor solução obtida. F^*: $F(c^*)$ (Melhor valor de *fitness* obtido).

P_1:= **Algoritmo** 2.2 ($N_{\text{Pop}}, N \times M$); // Gerar população inicial.

PARA $i := 1$ **ATÉ** N_{Gen} **FAÇA**

  PARA $j := 1$ **ATÉ** N_{Pop} **FAÇA**

    $F_j := F(p_j)$;

  FIM_PARA

  $f_{\text{worst}} := \max_j F_j$; // O pior valor de *fitness* (f_{worst}) é o máximo valor de *fitness*.

  $P_{\text{filhos}} = \{\ \}$;

  $k := 0$;

  ENQUANTO $k \leq N_{\text{Pop}}$ **FAÇA** // Operador de cruzamento.

$p_1 \coloneqq \text{rand\_set}(P_i)$; /* rand_set é a função que seleciona um indivíduo aleatoriamente em um dado conjunto. */

$p_2 \coloneqq \text{rand\_set}(P_i - \{p_1\})$; // Seleciona-se um indivíduo diferente de p_1.

$p_3 \coloneqq \text{rand\_set}(P_i - \{p_1, p_2\})$; // Seleciona-se um indivíduo diferente de p_1 e p_2.

$(c_1, c_2, c_3) \coloneqq$ **Algoritmo** 2.4 $(F, p_1, p_2, p_3, f_{\text{worst}})$; // Conduz-se o cruzamento.

$P_{\text{filhos}} \coloneqq P_{\text{filhos}} \cup \{c_1, c_2, c_3\}$;

$P_i \coloneqq P_i - \{p_1, p_2, p_3\}$; // Os indivíduos p_1, p_2, p_3 são removidos da população.

$k \coloneqq k + 3$;

FIM_ENQUANTO

$P_{i+1} \coloneqq$ **Algoritmo** 2.5 $(P_{\text{filhos}}, F, f_{\text{worst}}, P_i, N \cdot M)$; // Conduz-se a mutação.

FIM_PARA

Fonte: Adaptado de Watanabe, Ida e Gen (2005).

Sendo assim, todo o método de Watanabe, Ida e Gen (2005) possui comportamento de busca local, não sendo preservado nenhum operador do GA em sua forma canônica. Entretanto, alguns trabalhos mantêm a forma original do AG e acrescentam a busca local na forma de operadores adicionais, como apresentado na subseção a seguir.

2.2.4.6.2 Operador dedicado a busca local por varredura de vizinhança

Ombuki e Ventresca (2004) propõem o uso de um operador de busca local em um GA básico na resolução de JSSPs. Sendo que este operador possui uma chance de 50% de substituir uma mutação convencional em cada uso do mesmo. O operador realiza substituições entre operações sucessivas em um indivíduo até obter duas substituições que pioram o valor de *fitness* do indivíduo, como esquematizado no Algoritmo 2.7 a seguir.

Algoritmo 2.7. Operador de busca local de Ombuki e Ventresca (2004).

Entrada	c: Indivíduo F: Função *fitness* do problema. N_{dim}: Número de genes no cromossomo.

Saída	c: Indivíduo aprimorado.

$F_c \coloneqq F(c)$;

$N_{\text{Pioras}} \coloneqq 0$; // Variável que controla as pioras do cromossomo durante o processo.

PARA $i =: 1$ **ATÉ** N_{dim} **FAÇA**

$\hat{c} \coloneqq f_{\text{swap}}(c, (i, i+1))$; // A perturbação em c ocorre em duas coordenadas // consecutivas (i e $i+1$).

$F_{\hat{c}} \coloneqq F(\hat{c})$;

SE $F_{\hat{c}} \leq F_c$ **ENTÃO** // Se a perturbação for benéfica, então deve ser mantida.

$c \coloneqq \hat{c}$; // Atualiza-se o cromossomo e seu *fitness*.

$F_c \coloneqq F_{\hat{c}}$;

SENÃO // Senão, acumulam-se as pioras e desconsidera-se a perturbação.

$N_{\text{Pioras}} \coloneqq N_{\text{Pioras}} + 1$;

FIM_SE

SE $N_{\text{Pioras}} == 2$ **ENTÃO** // Se ocorrem duas perturbações, então o procedimento é // finalizado.

"SAIA DO PARA";

FIM_SE

FIM_PARA

Fonte: Adaptado de Ombuki e Ventresca (2004).

Desta forma, o operador de mutação de Ombuki e Ventresca (2004) é generalizado no Algoritmo 2.8.

Algoritmo 2.8. Operador de mutação de Ombuki e Ventresca (2004).

Entrada	c: Indivíduo F: Função *fitness* do problema. N_{dim}: Número de genes no cromossomo.
Saída	c: Indivíduo aprimorado.

$r \coloneqq \text{rand}([0,1])$; // A função $\text{rand}([a, b])$ gera um número aleatório real entre a e b.

SE $r \leq 0.5$ **ENTÃO**

$c \coloneqq$ **Algoritmo** $2.7(c, F, N_{\text{dim}})$; // Operador de busca local de Ombuki e Ventresca (2004).

SENÃO // É feita uma mutação simples com a f_{swap}.

$r_1 \coloneqq \text{rand\_set}(\{1,2,3,\dots,N_{\text{dim}}\})$; // Selecionam-se aleatoriamente duas coordenadas distintas.

$r_2 \coloneqq \text{rand\_set}(\{1,2,3,\dots,r_1-1,r_1+1,\dots,N_{\text{dim}}\})$;

$c \coloneqq f_{\text{swap}}(c,(r_1,r_2))$; // É conduzida uma troca entre as coordenadas selecionadas.

FIM_SE

Fonte: Adaptado de Ombuki e Ventresca (2004).

Então o AG de Ombuki e Ventresca (2004) é um GA básico cujo operador de mutação eventualmente é substituído por um operador de busca local mais específico para busca em problemas combinatórios e mais robusto.

Em 2015, Asadzadeh propôs um aprimoramento ao AG de Ombuki e Ventresca (2004), apresentando um AG híbrido cujo operador de mutação é representado apenas por um operador de busca local que é uma versão mais abrangente do operador descrito no Algoritmo 2.8.

A saber, o operador de busca local, que também é uma espécie de mutação, de Asadzadeh (2015) estabelece randomicamente uma dentre as três funções de mutação mais utilizadas na literatura: as funções $f_{\text{swap}}, f_{\text{inverse}}$ e f_{insert} apresentadas respectivamente nas Equações 2.6, 2.7 e 2.8. Em sequência, são realizadas $N \cdot M$[ii] mutações sucessivas no mesmo indivíduo, mantendo-se as alterações benéficas e continuando o processo a partir delas. Este operador é aplicado a todas as soluções resultantes do operador de cruzamento de seu AG. O procedimento é detalhado no Algoritmo 2.9 a seguir.

Algoritmo 2.9. Operador de busca local de Asadzadeh (2015).

Entrada	P_{filhos} : Indivíduos gerados no cruzamento. F: Função *fitness* do problema. $N \times M$: Dimensões do JSSP tratado.

[ii] $N \times M$ é a dimensão do JSSP abordado, em que N representa o número de *jobs* e M representa o número de máquinas. Sendo assim, a dimensão do cromossomo que modela soluções do problema tem $N \cdot M$ coordenadas.

Saída	P_{mut}: Indivíduos mutantes.

$r \coloneqq \text{rand\_set}(\{1,2,3\})$; // Seleciona-se aleatoriamente uma das três funções de mutação
// apresentadas.

SE $r == 1$ **ENTÃO** $f \coloneqq f_{\text{swap}}$; // (Equação 2.6)

SENAO SE $r == 2$ **ENTÃO** $f \coloneqq f_{\text{inverse}}$; // (Equação 2.7)

SENÃO $f \coloneqq f_{\text{insert}}$; // (Equação 2.8)

$N_{\text{dim}} \coloneqq N \cdot M$; // Número total de perturbações.

$P_{\text{mut}} \coloneqq \{\ \}$;

PARA $c \in P_{\text{filhos}}$ **FAÇA**

PARA $i =: 1$ **ATÉ** N_{dim} **FAÇA**

$r_1 \coloneqq \text{rand\_set}(\{1,2,3,\dots,N_{\text{dim}}\})$; // Selecionam-se aleatoriamente duas coordenadas
// distintas.

$r_2 \coloneqq \text{rand\_set}(\{1,2,3,\dots,r_1-1,r_1+1,\dots,N_{\text{dim}}\})$;

$\hat{c} \coloneqq f\big(c,(r_1,r_2)\big)$; // $\hat{c}$ é o indivíduo mutante.

$F_{\hat{c}} \coloneqq F(\hat{c})$; // Avalia-se $\hat{c}$.

SE $F_{\hat{c}} \leq F_c$ **ENTÃO** // No caso de haver melhora, mantém-se a perturbação.

$c \coloneqq \hat{c}$; // Para isso, atualiza-se o cromossomo

$F_c \coloneqq F_{\hat{c}}$; // e o valor de *fitness*.

FIM_SE // Caso não haja melhora, deve-se avaliar a próxima perturbação.

FIM_PARA

$P_{\text{mut}} \coloneqq P_{\text{mut}} \cup \{c\}$;

FIM_PARA

Fonte: Adaptado de Asadzadeh (2015).

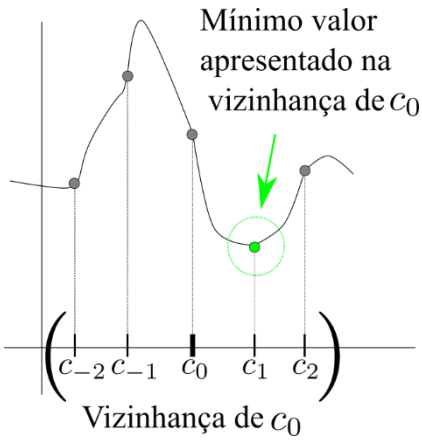
Vale ressaltar que o operador de mutação de Asadzadeh (2015) se assemelha com o operador de mutação de Watanabe, Ida e Gen (2005) para $R_m = N \cdot M$. A principal diferença se dá ao fato de que diferentes funções de mutação podem ser aplicadas eventualmente no operador de Asadzadeh (2015). Além disso, na técnica de Watanabe, Ida e Gen (2005), a função f_{swap} é sempre aplicada a partir do mesmo indivíduo, enquanto que na mutação de Asadzadeh (2015), o indivíduo é substituído no decorrer do método em toda ocasião em que há melhora em sua versão perturbada.

Asadzadeh (2015) também propõe o uso de um operador de busca local massiva em torno de soluções que se destacam na população, como detalhado na seção a seguir.

2.2.4.6.3 Operador de busca local elite

Operadores de busca local elite têm como principal finalidade realizar uma busca local mais elaborada em torno de uma solução com bom valor de *fitness*. Em outras palavras, dada uma solução c_0 e uma vizinhança finita desta solução, um operador de busca local elite deve realizar sucessivas avaliações no maior número possível de elementos dessa vizinhança a fim de encontrar uma solução melhor do que c_0. Como ilustrado na Figura 2.10.

Figura 2.10. Busca massiva em uma vizinhança de c_0.



Fonte: Elaborada pela autora.

Esta categoria de operador costuma apresentar comportamento semi-guloso, uma vez que realizam avaliações no maior número possível de elementos da vizinhança de uma solução. Desta forma, é recomendável que estes operadores sejam utilizados em poucos indivíduos de uma população em cada iteração do método.

Asadzadeh (2015) propõe em seu trabalho que seja realizada uma busca local elite no melhor indivíduo de cada geração de um algoritmo genético híbrido ao buscar soluções de JSSPs. A autora leva em consideração que o melhor indivíduo pode ser aprimorado com a

aplicação da perturbação correta, uma vez que a sequência de operações definida por ele já é a melhor de uma população.

Inicialmente, Asadzadeh (2015) propõe que a função f_{swap} (Equação (2.6)) seja aplicada sucessivas vezes no melhor indivíduo a fim de substituir toda coordenada do mesmo em toda outra coordenada também do mesmo indivíduo mantendo-se as perturbações que o aprimoram, melhorando seu valor de *fitness*, e dando continuidade às buscas a partir destas soluções aprimoradas. Este processo é apresentado no Algoritmo 2.10 e é chamado de "Operador de Busca Local no indivíduo de Elite[iii]".

Algoritmo 2.10. Operador de busca local massiva de Asadzadeh (2015).

Entrada	c_{Elite} : Melhor indivíduo presente na população F: Função *fitness* do problema. $N \times M$: Dimensões do JSSP tratado.
Saída	$\hat{c}_{\text{Elite}}$: Indivíduo aprimorado.

$N_{\text{dim}} \coloneqq N \cdot M$; // Número total de coordenadas no cromossomo.
$F_{\text{Elite}} \coloneqq F(c_{\text{Elite}})$; // Calcula-se o *fitness* do melhor indivíduo.
PARA $i \coloneqq 1$ **ATÉ** N_{dim} **FAÇA** // Todas as coordenadas de c_{Elite} são consideradas.
  PARA $j =: 1$ **ATÉ** N_{dim} **FAÇA**
    $\hat{c} \coloneqq f_{\text{swap}}(c_{\text{Elite}}, (i,j))$; // Perturbação em c_{Elite} com respeito às suas coordenadas i // e j por meio da função f_{swap}.
    $F_{\hat{c}} \coloneqq F(\hat{c})$; // Avalia-se a perturbação de c_{Elite}.
    SE $F_{\hat{c}} \leq F_{\text{Elite}}$ **ENTÃO** // No caso da perturbação ser benéfica, a mesma deve ser // mantida.
      $c_{\text{Elite}} \coloneqq \hat{c}$; // Atualiza-se o melhor indivíduo com sua versão perturbada.
      $F_{\text{Elite}} \coloneqq F_{\hat{c}}$; // Um melhor valor de *fitness* foi encontrado.
    FIM_SE
  FIM_PARA
FIM_PARA
$\hat{c}_{\text{Elite}} \coloneqq c_{\text{Elite}}$;

Fonte: Adaptado de Asadzadeh (2015).

[iii] Chama-se de "indivíduo Elite" o melhor indivíduo de uma população. Isto é, o indivíduo que apresenta o menor valor de *fitness* em uma população.

No Algoritmo 2.10 é feito uma busca para se obter um melhor sequenciamento a partir do melhor sequenciamento disponível de uma geração do GA por meio de substituições sucessivas de operações, mantendo-se as substituições que foram benéficas ao sequenciamento inicial.

Desta forma, o algoritmo genético com operadores de busca local de Asadzadeh (2015) pode ser interpretado e esquematizado de acordo com o Algoritmo2.11. Vale ressaltar que este GA cria a nova população com a técnica de roleta viciada com retenção do melhor indivíduo (GOLDBERG; HOLLAND, 1988).

Algoritmo 2.11. Algoritmo genético de Asadzadeh (2015) aplicado em JSSP.

Entrada	N_{Pop}: Número de indivíduos na população. N_{Gen}: Número de gerações do AG. $N \times M$: Dimensões do JSSP. F: Função *fitness* do problema.
Saída	c^*: Melhor solução obtida. F^*: $F(c^*)$ (Melhor valor de *fitness* obtido).

P_1:= **Algoritmo** 2.2 ($N_{\text{Pop}}, N \times M$); // Gera-se uma população inicial.

PARA $i =: 1$ **ATÉ** N_{Gen} **FAÇA**

$P_{\text{filhos}} = \{\ \}$;

$k \coloneqq 0$;

ENQUANTO $k \leq N_{\text{Pop}}$ **FAÇA** // Operador de cruzamento.

$p_1 \coloneqq \text{rand\_set}(P_i)$; /* rand_set é a função que seleciona um indivíduo aleatoriamente em um dado conjunto. */

$p_2 \coloneqq \text{rand\_set}(P_i - \{p_1\})$; // Selecionam-se p_1 e p_2 dois pais distintos.

$(c_1, c_2) \coloneqq \text{PMX}(p_1, p_2)$; // Conduz-se o cruzamento através da técnica PMX.

$P_{\text{filhos}} \coloneqq P_{\text{filhos}} \cup \{c_1, c_2\}$;

$P_i \coloneqq P_i - \{p_1, p_2\}$; // Remove-se os pais p_1 e p_2 da população.

$k \coloneqq k + 2$;

FIM_ENQUANTO

$P_{\text{mut}} \coloneqq$ **Algoritmo** 2.9($P_{\text{filhos}}, F, N \times M$); // Operador de mutação.

$c_{\text{Elite}} \coloneqq \underset{c \in P_{\text{mut}} \cup P_{\text{filhos}} \cup P_i}{\arg\min} F(c);$ // c_{Elite} é o melhor indivíduo da população.

$\hat{c}_{\text{Elite}} \coloneqq$ **Algoritmo** 2.10 $(c_{\text{Elite}}, F, N \times M)$; // Operador de busca local massiva.

$P_{i+1} \coloneqq \text{ROLETA}(P_i \cup P_{\text{mut}} \cup \{\hat{c}_{\text{Elite}}\})$; // Criação da nova população por meio da técnica
// roleta viciada, assegurando a presença do melhor
// indivíduo da população.

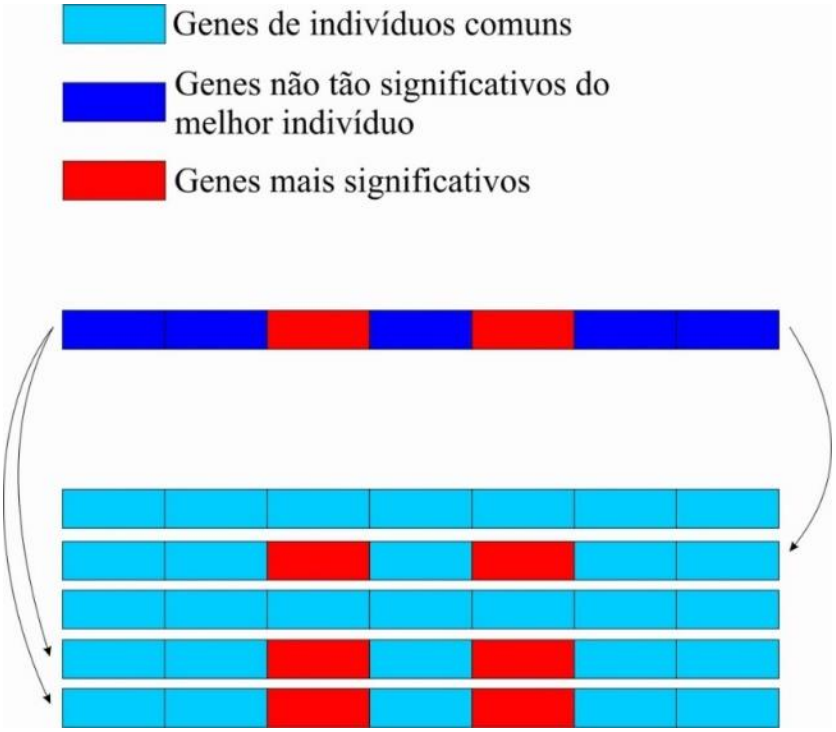
FIM_PARA

Fonte: Adaptado de Asadzadeh (2015).

2.2.4.6.4 Operador de transgenia

O operador Transgênico proposto por Viana, Morandin Junior e Contreras (2020d) é uma modificação e adaptação do operador de Transgenia proposto por Amaral e Hruschka (2014). Tal operador foi desenvolvido com a inspiração da engenharia genética, na qual se encontra a possibilidade de manipular o material genético de indivíduos adicionando-se características das quais se julga serem as mais importantes. Esse operador é acoplado ao GA, mas para um bom funcionamento é necessário identificar qual o gene ou quais os genes são os mais significativos do problema, isto é, quais genes levam a obter melhores resultados. Após a definição dos genes mais significativos, é selecionado o melhor indivíduo da população, isto é, indivíduo com o melhor *fitness* para transferir seus genes mais significativos para um determinado número de outros indivíduos da população, como pode ser observado na Figura 2.11, onde há 6 indivíduos, sendo que o primeiro é classificado como melhor e seus genes mais significativos, representados na cor vermelha, são passados para outros 3 indivíduos.

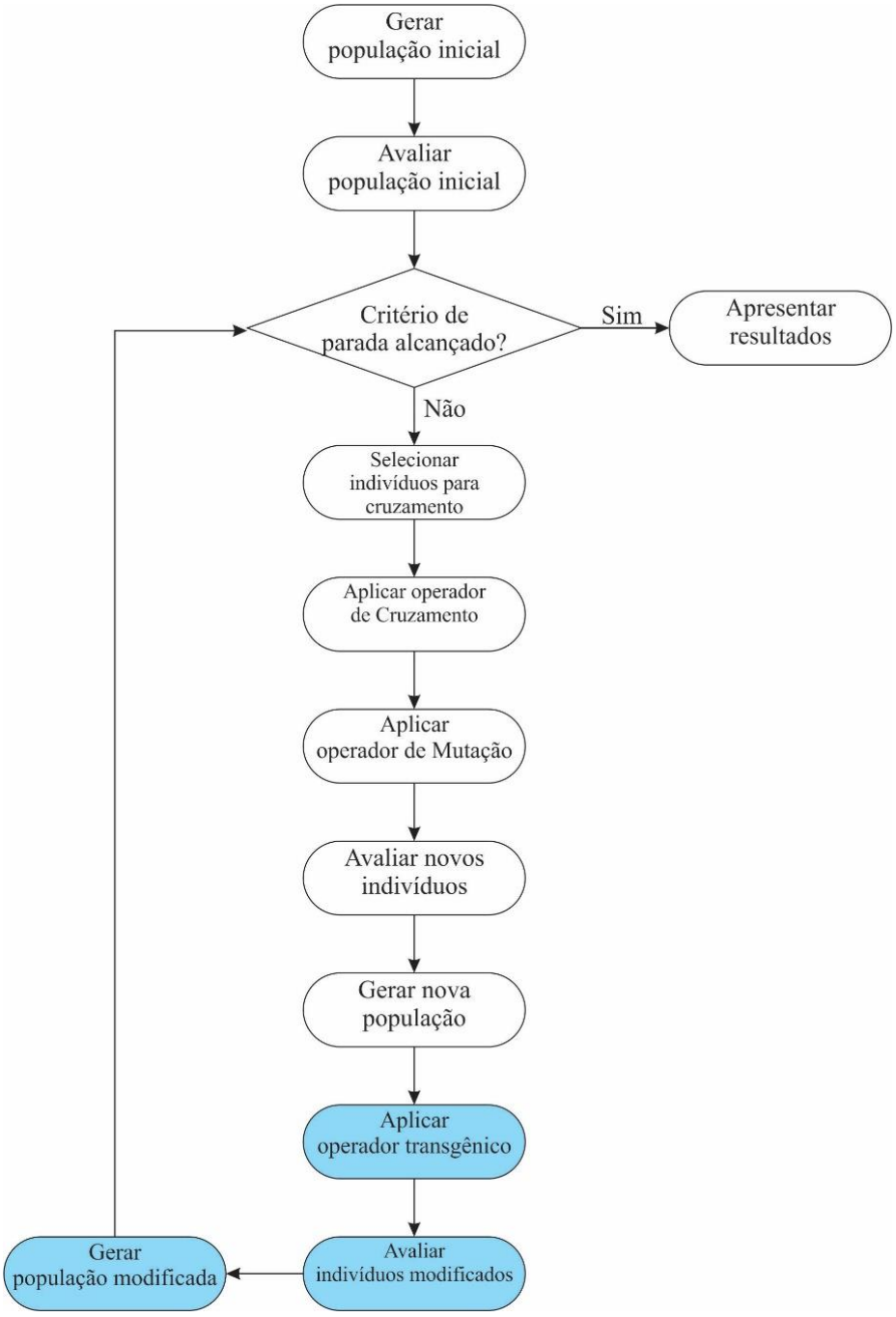
Figura 2.11. Processo de transgenia.



Fonte: Dissertação de Viana (2016).

Os passos do algoritmo genético com operador transgênico são semelhantes aos do GA tradicional, exceto pelo processo de transgenia que é adicionado à técnica canônica. No fluxograma da Figura 2.12 são descritas as etapas do GA transgênico, sendo que as atividades em azul são referentes ao operador transgênico.

Figura 2.12. Fluxograma do GA com operador Transgênico.



Fonte: Dissertação de Viana (2016).

Para determinar quais genes são os mais significativos, isto é, quais genes levam ao melhor direcionamento de busca ao serem transmitidos para uma parcela da população, a autora Viana, Morandin Junior e Contreras (2020d) propôs um método que consiste na simulação do comportamento do algoritmo genético com operador transgênico, no qual cada gene separadamente é simulado como sendo o mais significativo pelo operador transgênico e a

simulação é realizada em um certo número de gerações N_g. A cada geração é salva a melhoria dos indivíduos modificados, que é a diferença entre a aptidão de um indivíduo antes da utilização do transgênico e depois da utilização do transgênico. Assim, na i-ésima geração é construído um vetor $g_i \in \mathbb{R}^{N_g}$ que armazena a melhoria média entre os valores dos *fitness* dos N_C indivíduos antes e depois da aplicação de transgenia. Ao fim de todas as gerações da simulação do gene fixado, define-se a matriz de diferenças G, descrita na Equação (2.9) a seguir.

$$G = \begin{matrix} \text{geração } 1 \rightarrow \\ \vdots \\ \text{geração } N_g \rightarrow \end{matrix} \begin{bmatrix} | & | & \cdots & | \\ g_1 & g_2 & \cdots & g_n \\ | & | & \cdots & | \end{bmatrix}. \tag{2.9}$$

De posse da matriz de dados apresentada na Equação (2.9) é feita uma média ponderada para determinar quais são os genes mais significativos. O método de média ponderada consiste em extrair de cada coluna g_i a média (μ), o desvio padrão (σ) e a maior diferença (amplitude λ) dentre os valores de tal coluna. Esses valores são incluídos em uma média ponderada (M_p), dada pela Equação (2.10), que é um vetor que apresenta uma coordenada para cada gene presente no cromossomo. Assim, é feita uma classificação dos genes mais importantes do problema abordado.

$$M_p(g_i) = \frac{\sigma(g_i) + (\mu(g_i) \cdot 1{,}5) + (\lambda(g_i) \cdot 2)}{4{,}5}. \tag{2.10}$$

O operador proposto por Viana, Morandin Junior e Contreras (2020d) possui um maior custo computacional devido ao pré-processamento necessário, onde para determinar os genes mais significativos é executado o GA com operador transgênico simulando gene-a-gene como o mais significativo. O novo operador de transgenia proposto neste trabalho não possui essa etapa de pré-processamento de escolha de genes, com isso é possível diminuir o tempo computacional total.

2.3 Trabalhos correlatos

Nesta seção, está o mapeamento das pesquisas feitas na área de programação da produção JSSP e a revisão da literatura atual, assim como a análise dos dados relevantes encontrados.

2.3.1 Trabalhos relacionados ao problema de programação da produção JSSP

O problema de programação JSSP é classificado como NP-*Hard*, por isso é um problema que possui uma alta complexidade para resolução, o que torna difícil o cálculo de uma solução ideal. Abordagens diretas, heurísticas e metaheurísticas são os principais métodos utilizados para resolver o problema de programação da produção JSSP. Em muitas pesquisas foram utilizadas as abordagens diretas, tais como métodos de busca enumerativos, *branch and bound* e regras de prioridade de despacho para resolver problemas de programação, alguns destes trabalhos foram o de Ashour e Hiremath (1973), Stinson, Davis e Khumawala (1978), Barker e Mcmahon (1985), Adams, Balas e Zawack (1988), Carlier e Pinson (1989), Brucker, Jurisch e Sievers (1994) e o de Artigues e Feillet (2008). Abordagens diretas possuem uma tendência de alto consumo de tempo e normalmente não são capazes de produzirem soluções de boa qualidade para problemas de larga escala. Por conseguinte, o esforço tem levado a procura e desenvolvimento de heurísticas e metaheurísticas eficientes para a resolução de problemas de alta complexidade (UDHAYAKUMAR; KUMANAN, 2010).

Nos parágrafos a seguir serão detalhados determinados trabalhos presentes na literatura que tratam o problema de programação da produção JSSP por meio de diversas metaheurísticas. Estes trabalhos foram escolhidos por possuírem um grande impacto na literatura especializado e/ou serem trabalhos atuais. Os trabalhos detalhados a seguir são: Ombuki e Ventresca (2004), Watanabe, Ida e Gen (2005), Asadzadeh (2015), Jorapur et al. (2016), Wang, Cai e Li (2016), Kurdi (2016), Sayoti, Riffi e Labani (2016), Díaz et al. (2017), Jiang (2018), Wang et al. (2018), Dao et al. (2018), Jiang e Zhang (2018), Semlali, Riffi e Chebihi (2019), Kurdi (2019), Hamzadayi, Baykasoğlu e Akpinar (2020), Yu *et al.* (2020) e Alkhateeb, Abed-Alguni e Al-Rousan (2021).

Os autores Ombuki e Ventresca (2004) propuserem a metaheurística *Local Search Genetic Algorithm* (LSGA) para tratar o problema de programação da produção do tipo *Job*

shop com objetivo de minimizar o *makespan*, isto é, o tempo máximo de conclusão de um conjunto de *jobs*. O LSGA proposto é um *Genetic Algorithm* (GA) com busca local, o qual possui um operador semelhante à mutação que é voltado para a busca local, com o objetivo de melhorar ainda mais a qualidade da solução. No LSGA, foi proposto um operador de cruzamento simples que garante que as soluções geradas pela evolução genética sejam todas viáveis, assim, não é necessário um mecanismo de reparo das soluções infactíveis. O operador de cruzamento proposto é uma melhoria do operador UOX de Ombuki, Nakamura e Onaga (1998), na nova versão do operador é preservada a ordenação de cada tarefa e, portanto, com isso é mantida a factibilidade após a aplicação nos cromossomos. Para a mutação no LSGA é utilizado um simples *swap*, que consiste em escolher dois trabalhos aleatoriamente e trocar a posição que é ocupada no cromossomo entre os mesmos. No LSGA, a busca local é aplicada probabilisticamente, isto é, aplica-se a mutação simples ou a mutação de busca local selecionadas dinamicamente em cada geração do GA. Em uma simples mutação, exatamente uma troca é permitida, isto é, as posições de dois trabalhos consecutivos selecionados aleatoriamente são trocadas. Em uma mutação de busca local uma abordagem sistemática é utilizada para consecutivamente testar vários *swaps* em termos de comprimentos do cromossomo, a melhor melhoria obtida pelo *swap* é salva, mas se nenhuma melhoria foi encontrada, nenhum *swap* será salvo na solução original e o processo será finalizado. O LSGA proposto por Ombuki e Ventresca (2004) foi testado em 52 instâncias de JSSP que possuem variáveis graus de complexidade, 3 instâncias FT de Fisher e Thompson (1963), 40 instâncias LA de Lawrence (1984) e em 9 instâncias ORB de Applegate e Cook (1991). As configurações do algoritmo foram: tamanho da população = 200, número de gerações = 550, taxa de cruzamento = 0,95 e taxa de mutação = 0,15. Cada instância foi executada 10 vezes e o melhor resultado obtido foi armazenado. O LSGA conseguiu encontrar soluções cujos resultados de *makespan* foram iguais os melhores valores conhecidos ou quando piores, ficaram dentro do intervalo de erro de 10%. Das 52 instâncias JSSP testadas, o LSGA encontrou a melhor solução conhecida em apenas 16 dessas instâncias. O trabalho comparou os resultados do LSGA com um GA canônico, e apesar do LSGA ter obtido a melhor solução conhecida em poucas instâncias, ficou evidente nos resultados obtidos, que o LSGA obteve um melhor comportamento de busca, então é possível afirmar que a estratégia de busca local incluída no GA melhorou sua técnica de busca e com isso suas soluções obtidas.

No artigo de Watanabe, Ida e Gen (2005), foi proposta a metaheurística GA com adaptação de área de busca (GSA) para resolver o problema de programação da produção do

tipo JSSP. O GSA proposto possui uma adaptação da área de busca com capacidade de se adaptar à estrutura do espaço da solução e de controlar o equilíbrio entre buscas globais e locais. No trabalho, são consideradas algumas restrições pertinentes ao problema JSSP, sendo estas: 1 - Um trabalho não visita a mesma máquina duas vezes, 2- Não há restrições de precedência entre as operações de trabalhos diferentes, 3 - Cada máquina pode processar apenas um trabalho de cada vez, 4 - Cada trabalho pode ser processado por apenas uma máquina de cada vez, 5 - Os tempos de liberação e as datas de vencimento não são especificados. A operação de cruzamento do GSA consiste em executar o cruzamento várias vezes em todos os pares de pais, sendo que todas as vezes é sorteado um novo ponto de corte. O cruzamento é repetido até que seja encontrado um filho melhor que o pior indivíduo da população ou até que um determinado número de iterações seja alcançado. A operação de mutação do GSA consiste em executar perturbações várias vezes em todos os filhos, realizando vários *swaps*. A mutação é repetida até que seja encontrado um filho mutante melhor que o pior indivíduo da população ou até que um determinado número de iterações seja alcançado. O GSA foi testado em 2 instâncias FT de Fisher e Thompson (1963). As configurações do algoritmo foram: tamanho da população = 400, critério de parada por número de novas soluções geradas = 3.000.000, taxa de cruzamento = 0,8 e taxa de mutação = 0,2. Cada instância foi executada 100 vezes e o melhor resultado obtido foi armazenado. O GSA foi comparado com um GA canônico e apresentou melhores resultados. O GSA alcançou uma maior frequência em encontrar soluções mais próximas às soluções ótimas. Apesar do algoritmo não ter encontrado a melhor solução conhecida nas instâncias FT, o mesmo apresentou uma melhoria no GA canônico e melhorou significativamente os resultados obtidos no estado da arte até então.

No artigo de Asadzadeh (2015), foi proposta a metaheurística *Local Search Genetic Algorithm* (aLSGA) para resolver o problema de programação da produção do tipo JSSP com objetivo de minimizar o *makespan*. O aLSGA proposto é um GA com busca local com a inclusão de agentes inteligentes. O método é composto por um sistema multiagente, nos quais cada agente possui um comportamento especializado para implementar a busca local. O aLSGA combina heurísticas de busca local com operadores de cruzamento e mutação. A autora Asadzadeh (2015), para implementar o modelo baseado em agente, utilizou o JADE de Bellifemine, Poggi e Rimassa (2001) como uma plataforma e os agentes foram construídos no mesmo. O JADE permite que cada agente descubra dinamicamente outros agentes e se comunique com eles de maneira *peer-to-peer*. O aLSGA contém quatro agentes, sendo estes o agente de gerenciamento (MA), o agente processador (PA), o agente de busca local (LSA) e o

agente de busca local elite (ELSA). O MA tem a responsabilidade de criar a população inicial para o algoritmo genético, computando o valor de *fitness* dos cromossomos e corrigindo os cromossomos ilegais e, também, controla os comportamentos de outros agentes na plataforma. O PA executa o algoritmo genético na população genética. O LSA executa o procedimento de busca local nos cromossomos da população. ELSA executa o procedimento de busca de local no cromossomo elite da população. O aLSGA possui dois procedimentos de busca local, o primeiro é intitulado de LSA e o segundo de ELSA. O primeiro é responsável por explorar a vizinhança de cada cromossomo produzido pelo operador de cruzamento e o segundo é responsável por melhorar a qualidade do melhor cromossomo da população. O procedimento LSA foi projetado para funcionar da seguinte maneira: depois de criar um cromossomo utilizando o operador de cruzamento, o agente PA o envia ao agente LSA para executar o procedimento de busca local. O procedimento LSA consiste em três operadores de mutação: *Swap*, *Insert* e *Inverse* que são aplicados aleatoriamente no cromossomo selecionado. O novo cromossomo substitui o cromossomo antigo se ele tiver um melhor valor de *fitness*. O procedimento ELSA foi projetado para funcionar da seguinte maneira: é selecionado o melhor cromossomo da população em cada geração genética e enviado para o ELSA para a aplicação do procedimento de busca local. A busca local do ELSA consiste em trocar as ordens de operação de trabalho de duas posições selecionadas em uma solução e comparar o valor de *fitness* anterior e o novo valor de *fitness*. Se o novo valor for melhor que o valor anterior, o novo indivíduo será aceito. Este método examina cada intercâmbio de pares possível da operação de trabalho na primeira posição. Em seguida, outras posições são verificadas usando a mesma operação. Sempre que houver uma melhoria na função de *fitness*, as ordens das operações de trabalho são intercambiadas. Asadzadeh (2015) utilizou no método aLSGA a modelagem por operações para o JSSP. A modelagem proposta é similar à de Cheng, Gen e Tsujimura (1996), na qual cada trabalho tem um número distinto para indicar suas operações. No problema de N trabalhos e M máquinas, o número máximo de genes é $N \times M$. Um exemplo de cromossomo do problema 4×4, isto é, $N = 4$ e $M = 4$, seria $(3\ 4\ 2\ 3\ 1\ 4\ 1\ 2\ 4\ 2\ 3\ 1\ 1\ 3\ 2\ 4)$. Nesse cromossomo, 1 significa o trabalho J_1, 2 significa trabalho J_2, e assim por diante. O trabalho J_1 tem quatro operações, sendo assim, o número de vezes que 1 aparece no cromossomo é quatro. O aLSGA foi testado em 47 instâncias de JSSP, 3 instâncias FT de Fisher e Thompson (1963), 35 instâncias LA de Lawrence (1984) e em 9 instâncias ORB de Applegate e Cook (1991). A proposta foi comparada com os seguintes métodos: *Greedy Randomized Adaptive Search Procedure* (GRASP) de Binato et al. (2002), *Guided Local Search* (GLS) de Aarts et al. (1994), *Parallel Agent-based Genetic Algorithm*

(PAGA) de Asadzadeh e Zamanifar (2010), *Hybrid Genetic Algorithm* (HGA) de Gonçalves, Mendes e Resende (2005), *Local Search Genetic Algorithm (LSGA) de* Ombuki e Ventresca (2004) *e Adaptive* de Gabel e Riedmiller (2007). As configurações do algoritmo aLSGA foram: tamanho da população = 100, número de gerações = 100, taxa de cruzamento = 0,95. Não é informado o número de vezes em que cada instância foi executada. O aLSGA encontrou a melhor solução conhecida para 3 instâncias FT e para 22 instâncias LA, ou seja, em 100% e 63% dos casos de problemas, respectivamente. O aLSGA não obteve a solução mais conhecida para instâncias de ORB, mas a qualidade das soluções encontradas foi melhor do que quase todos os outros algoritmos, exceto o algoritmo GRASP de Binato et al. (2002), que encontrou uma melhor qualidade de solução em algumas instâncias ORB. Os resultados computacionais mostraram que o aLSGA proposto é eficaz em encontrar soluções ótimas e quase ótimas para várias instâncias do problema JSSP. Nota-se que ao fazer inclusão de uma heurística de busca local no GA, é obtido um melhoramento no seu desempenho e em sua velocidade de convergência.

Os autores Jorapur et al. (2016) propuseram a metaheurística *Promising Initial Population Based Genetic Algorithm* (IPBGA) para tratar o problema de programação da produção do tipo *Job shop*. O algoritmo IPBGA é a combinação do GA com uma nova modelagem baseada em tarefa (*job*) para a construção da população inicial. O objetivo do trabalho foi apresentar uma modelagem de população alternativa para o GA e, também, apresentar o impacto que esse tipo de alteração pode obter em relação à eficácia do GA. O IPBGA foi testado em 64 instâncias de JSSP que possuem variáveis graus de complexidade, 3 instâncias FT de Fisher e Thompson (1963), 40 instâncias LA de Lawrence (1984), 8 instâncias ORB de Applegate e Cook (1991), 5 instâncias ABZ de Adams, Balas e Zawack (1988) e em 8 instâncias CAR de Carlier (1978). As configurações do algoritmo foram: tamanho da população = 1000, número de gerações = 50, taxa de cruzamento = 0,1 e taxa de mutação $\in [0.1, 0.9]$. Cada instância foi executada 20 vezes e o melhor resultado obtido foi armazenado. O IPBGA foi testado em 64 instâncias e conseguiu encontrar a melhor solução conhecida em apenas 25 dessas instâncias. A proposta não foi comparada com outros métodos presentes na literatura, com isso os autores não apresentaram se houve ganhos em relação ao estado da arte. Além do IPBGA ter alçando a melhor solução conhecida em pouquíssimas instâncias, nas demais os resultados obtidos ficaram significativamente distantes da solução ótima. Dado os resultados apresentados por Jorapur et al. (2016), é possível concluir que o método IPBGA não tem um desempenho significativo para instâncias JSSP.

No artigo de Wang, Cai e Li (2016), foi proposta a metaheurística *Adaptive Multi-population Genetic Algorithm* (AMGA) para resolver o problema de programação da produção do tipo JSSP com objetivo de minimizar o *makespan*. O AMGA proposto é um AG que faz uso de multipopulação e que tem probabilidade adaptativa de cruzamento e de mutação, com o objetivo de ampliar o escopo da pesquisa e melhorar seu desempenho. O trabalho possui alguns pontos que o difere de outros trabalhos que tratam o problema JSSP com GA. O primeiro ponto é a inserção de multipopulação no GA, o segundo ponto é possuir uma probabilidade adaptativa de cruzamento e de mutação e o terceiro ponto é que os indivíduos de elite (indivíduos com melhor *fitness*) de cada população são evoluídos diretamente para a próxima geração. O AMGA foi testado em 39 instâncias de JSSP, 3 instâncias de Fisher e Thompson (1963) e 36 instâncias de Lawrence (1984). A proposta foi comparada com os seguintes métodos: GA de Croce, Tadei e Volta (1995), *Simulated Annealing* (SA) de Van, Aarts e Lenstra (1992), *Tabu Search* (TS) de Dell'amico e Trubian (1993), *Modified Genetic Algorithm* (MGA) de Wang e Zheng (2001), *Parallel Agent-based Genetic Algorithm* (PAGA) de Asadzadeh e Zamanifar (2010) e *New Island Model Genetic Algorithm* (NIMGA) de Kurdi (2016). As configurações do algoritmo AMGA foram: tamanho da população = 100, número de populações = 4, número de gerações = 50, taxa de cruzamento = 0,8 e taxa de mutação = 0,05. Cada instância foi executada 100 vezes. O AMGA foi testado em 39 instâncias e conseguiu encontrar a melhor solução conhecida em 38 dessas instâncias. Os resultados computacionais mostraram que o AMGA proposto pode produzir valores ótimos ou quase ótimos em quase todas as instâncias de benchmark testadas, porém nas instâncias de Lawrence (1984) nem todas foram testadas e as instâncias que ficaram sem avaliação são justamente as instâncias que apresentam uma maior complexidade, então não é possível concluir que o método se adaptaria também a tais instâncias. O trabalho obteve melhores resultados do que aqueles apresentados nos artigos comparados e, também, expôs por meio dos testes que o AG multipopulação proposto tem um comportamento de busca mais adequado do que outras versões modificadas de AG, como o MGA, PAGA e NIMGA.

O autor Kurdi (2016) propôs uma melhoria na metaheurística *Island Model Genetic Algorithm* (IMGA) para tratar o problema de programação da produção do tipo *Job shop* com o objetivo de minimizar o *makespan*. Para melhorar a eficácia do IMGA foi proposto no trabalho de Kurdi (2016) o NIMGA, que é o IMGA com um novo modelo de evolução de inspiração natural e um novo mecanismo de seleção de migração de inspiração natural, sendo que esses novos modelos são capazes de melhorar a busca de diversificação e retardar a convergência prematura. No novo modelo de evolução proposto por Kurdi (2016), as ilhas

empregam diferentes métodos de mutação durante suas fases de autoadaptação, em vez de empregar os mesmos métodos como no IMGA, assim as ilhas exploram novas regiões no espaço de busca e isso amplia a cobertura do processo de busca. No novo mecanismo de seleção de migração proposto por Kurdi (2016), os indivíduos menos adaptados aos seus ambientes migram primeiro, esperando encontrar uma melhor chance de viver em um ambiente mais adequado que imponha um método de autoadaptação mais adequado a eles. O objetivo da adaptação proposta Kurdi (2016) foi o de melhorar a eficácia do IMGA, aprimorando sua capacidade de diversificação de busca e evitando a convergência prematura e mantendo sua pureza, ou seja, sem hibridizar com qualquer outro método, ou incorporando qualquer conhecimento específico do problema em seus operadores genéticos. O IMGA diferentemente do GA canônico que funciona em uma única população que executa um processo de evolução, trabalha em várias ilhas independentes, que também são chamadas de subpopulações, que evoluem isoladas umas das outras, e apenas ocasionalmente indivíduos migram de suas ilhas para outras ilhas. Isso pode ajudar a atrasar a convergência prematura e a aumentar a diversificação de busca. O NIMGA foi testado em 52 instâncias de JSSP, 3 instâncias de Fisher e Thompson (1963), 40 instâncias LA de Lawrence (1984) e em 9 instâncias de Applegate e Cook (1991). A proposta foi comparada com os seguintes métodos: PAGA de Asadzadeh e Zamanifar (2010), *Hybrid Parallel* GA de Yusof et al. (2011), *Multiple Colony Ant Algorithm (*MAnt) de Udomsakdigool e Kachitvichyanukul (2008) e *Particle Swarm Optimization* (PSO) de Yen e Ivers (2009). As configurações do algoritmo foram: número de ilhas = 3, tamanho da população de cada ilha = 100, número de gerações = 1000, taxa de cruzamento = 0,8 e taxa de mutação = 0,2, frequência de migração = 10 gerações futuras sem qualquer melhoria, taxa de imigração = 1 indivíduo. Cada instância foi executada 4 vezes. O NIMGA foi testado em 52 instâncias e conseguiu encontrar a melhor solução conhecida em 30 dessas instâncias. O algoritmo proposto teve a mesma dificuldade que a maioria dos trabalhos presentes na literatura, que é a falta de adaptabilidade do método às instâncias de maior complexidade. Dentre as instâncias avaliadas, as de Applegate e Cook (1991) são as que mais apresentam dificuldade de solução aos métodos presentes na literatura. Das 9 instâncias de Applegate e Cook (1991) que foram testadas pelo NIMGA de Sayoti, Riffi e Labani (2016), apenas em 2 instâncias foram encontradas a melhor solução conhecida. A proposta NIMGA obteve melhores resultados que outras versões de GA, como o PAGA de Asadzadeh e Zamanifar (2010) e o *Hybrid Parallel* GA de Yusof et al. (2011), então o método apresenta-se como uma contribuição e melhoria no

domínio de pesquisa de metaheurísticas do tipo GA e no domínio de pesquisa de problemas JSSP.

No artigo de Sayoti, Riffi e Labani (2016), foi proposto uma adaptação da metaheurística *Golden Ball Algorithm* (GBA) para tratar o problema de otimização JSSP. O GBA foi criado por Osaba et al. (2014) e é inspirado nos conceitos do futebol. O GBA é composto por quatro fases principais: a fase de inicialização, a fase de treinamento, a fase de competição e a fase de transferência. No algoritmo GBA, as funções de treinamento convencionais foram definidas utilizando as técnicas: 2-opt de Croes (1958), método *Insert de* Fischetti, González e Toth (1997) e a técnica *Swap* de Tarantilis (2005). Foi utilizado o *Ordered Crossover* (OX) de Davis (1985) como uma função de treinamento personalizada. Na fase de competição, cada solução do grupo deverá ser comparada com outra existente em outro grupo escolhido aleatoriamente, o grupo que tiver a melhor solução, isto é, solução com menor resultado de *makespan*, recebe três pontos e caso as duas possíveis soluções forem iguais, os dois grupos recebem um ponto. Na fase de transferência, as possíveis soluções e as funções de treinamento são trocadas entre os grupos. O método foi testado em 36 instâncias de JSSP, 2 instâncias FT de Fisher e Thompson (1963), 23 instâncias LA de Lawrence (1984), 2 instâncias ABZ de Adams, Balas e Zawack (1988) e em 9 instâncias ORB de Applegate e Cook (1991). A proposta foi comparada com os seguintes métodos: *Artificial Immune Systems* (AIS) de Bondal (2008), AIS de Mahapatra (2012), *Multilayer Perception* (MLP) de Chaudhuri e De (2010), IAS de Luh e Chueh (2007), *Multiple Colony Ant Algorithm* (MAnt) de Udomsakdigool e Kachitvichyanukul (2008) e *Threshold Accepting* (TA) de Käschel et al. (1999). Cada instância foi executada 10 vezes. O GBA foi testado em 36 instâncias e conseguiu encontrar a melhor solução conhecida em 21 dessas instâncias. A proposta obteve bons resultados em instâncias menores, uma vez que conforme as instâncias foram aumentando o tamanho e a complexidade, o algoritmo teve uma decaída nos resultados. Nota-se que o algoritmo funciona muito bem para instâncias de baixa e média complexidade, mas para as de alta complexidade a proposta obteve qualidade de resultados abaixo dos resultados alcançados pelos trabalhados comparados.

Os autores Díaz et al. (2017) propuseram a metaheurística *Memetic Algorithm* (MA) para tratar o problema de programação da produção do tipo *Job shop*. O algoritmo MA combina a exploração global do espaço de busca por GA e a exploração local do espaço de busca por busca local baseada na estrutura de vizinhança de Nowicki e Smutnicki (1996). O método foi testado em 30 instâncias de JSSP, 3 instâncias FT de Fisher e Thompson (1963) e em 27

instâncias LA de Lawrence (1984). A proposta foi comparada com os seguintes métodos: GA de Tsai et al. (2008), MA de González Fernández (2011) e LSGA de Essafi, Mati e Dauzère-Pérès (2008). Os experimentos computacionais foram divididos em duas partes, a primeira parte corresponde às instâncias de complexidade baixa e média (FT06, LA01, LA06, LA09, LA11, LA14, LA17) e a segunda parte corresponde às instâncias de complexidade alta (FT10, FT20, LA16, LA21, LA24, LA25, LA38). Para as instâncias de complexidade baixa e média foram executados 8 vezes cada uma e para as instâncias de complexidade alta foram executadas 5 vezes cada uma. As configurações do algoritmo para as instâncias de complexidade baixa e média foram: tamanho da população $= \{10, 30\}$, número de gerações = $\{20, 30\}$, probabilidade de seleção = $\{0{,}7, ; 0{,}9\}$ e probabilidade de mutação = $\{0{,}05; 0{,}1\}$. Para as instâncias de complexidade alta foram: tamanho da população $= \{80, 150\}$, número de gerações = $\{100, 170\}$, probabilidade de seleção = $\{0{,}8; 0{,}9\}$ e probabilidade de mutação = $\{0{,}05;\ 0{,}1\}$. O método MA foi testado em 30 instâncias e conseguiu encontrar a melhor solução conhecida em 19 dessas instâncias. Em 10 instâncias o MA proposto encontrou piores resultados que as outras abordagens comparadas, mas mesmo assim apresentou resultados competitivos. Nota-se que o MA proposto não oferece uma boa busca em instâncias de complexidade alta, a proposta não conseguiu encontrar a melhor solução conhecida em nenhuma das instâncias de alta complexidade.

No artigo de Jiang (2018), foi proposto a metaheurística *Hybrid Grey Wolf Optimization* (HGWO) para tratar o problema de programação da produção do tipo JSSP. O HGWO é composto pela combinação do algoritmo GWO com o algoritmo de busca local VNS e também a adição de operadores genéticos (cruzamento e mutação) com a finalidade de equilibrar a capacidade de exploração local e global do algoritmo. Na proposta foram utilizadas três estruturas de vizinhança: *Swap*, *Insert* e *Inverse*. O método foi testado em 23 instâncias de JSSP, 3 instâncias de Fisher e Thompson (1963) e em 20 instâncias de Lawrence (1984). A proposta foi comparada com os seguintes métodos: PSO de Sha e Hsu (2006), *Multiple Colony Ant Algorithm* (MAnt) de Udomsakdigool e Kachitvichyanukul (2008), *Memetic algorithms* (MA) de Hasan et al. (2009), *Ant Colony Optimization* (ACO) de Seo e Kim (2010), *Differential Evolution* (DE) de Wisittipanich e Kachitvichyanukul (2012), *Shuffled Complex Evolution Algorithm* de Zhao, Zhang et al. (2015), GA de Jorapur et al. (2016), *Local Search Genetic Algorithm* (LSGA) de Asadzadeh (2015) e *Bacterial Foraging Algorithm* (BFO) de Zhao, Jiang et al. (2015). As configurações do algoritmo foram: tamanho da população $= 30$, número de iterações = 1000, máximo de iterações do VNS $p_{\max} = 10$, taxa de cruzamento = $0{,}8$ e taxa de

mutação = 0,1. Cada instância foi executada 10 vezes. O HGWO foi testado em 23 instâncias e conseguiu obter resultados melhores ou iguais aos algoritmos comparados em 17 dessas instâncias. O algoritmo proposto obteve resultados competitivos ao ser comparado com trabalhos relevantes da literatura, mas das 40 instâncias propostas por Lawrence (1984), apenas as 20 menores foram testadas, com isso não há como avaliar o comportamento do algoritmo em instâncias com maior complexidade de busca.

Os autores Wang et al. (2018) propuseram a metaheurística TSAUN, que é um algoritmo híbrido de busca local para tratar o problema de otimização combinatória *Job shop* e minimizar o *makespan*. O TSAUN é composto pela combinação de método *Simulated Annealing* (SA) e o método *Tabu Search* (TS). A estrutura do TSAUN executa um núcleo de SA e aplica a técnica TS na busca local. Este algoritmo híbrido aproveita as vantagens do SA estocástico para escapar dos mínimos locais e, ao mesmo tempo, melhora o desempenho da pesquisa por meio de um TS. O método foi testado em 1 instância de Fisher e Thompson (1963) e em 3 instâncias de Lawrence (1984). Cada instância foi executada 20 vezes e os resultados obtidos pela proposta foram comparados com dois possíveis algoritmos alternativos. O TSAUN não alcançou os melhores resultados conhecidos nas instâncias testadas, mas o método mostrou-se competitivo em relação a outros trabalhos presentes no estado da arte. O trabalho apresenta contribuição na área de pesquisa sobre algoritmos híbridos com a inserção de técnicas de busca local e através dos resultados obtidos é reforçada a melhoria que essas combinações de técnicas podem alcançar.

No artigo de Dao et al. (2018), foi proposto a metaheurística *Parallel Bat Algorithm* (PBA) que é composta da metaheurística *Bat Algorithm* (BA) com a inclusão de processamento paralelo. A proposta foi utilizada para tratar o problema de programação da produção do tipo JSSP. O objetivo da adição do processamento paralelo ao BA foi de que com estratégias de comunicação é possível correlacionar indivíduos em cada enxame e compartilhar informações entre os mesmos, as comunicações fornecem diversidades aprimoradas e aceleram a busca por soluções satisfatórias. Na proposta de Dao et al. (2018), também foram inseridos operadores de vizinhança do tipo *Swap*, *Insert* e *Inverse*. O BA foi criado por Yang (2010), sendo que este algoritmo simula partes de características de ecolocalização dos morcegos. Três características dos morcegos foram empregadas para a construção da estrutura básica do BA. A primeira característica é o comportamento da ecolocalização. A segunda característica é a frequência que o morcego envia, sendo frequência f com um comprimento de onda variável λ. A terceira característica é o volume que o morcego usa para procurar por presas. No BA com estrutura

paralela proposto no artigo de Dao et al. (2018), vários grupos são criados dividindo a população em subpopulações para construir o processamento paralelo. Cada uma das subpopulações evolui independentemente em iterações regulares. Uma boa solução baseada no melhor valor de *fitness* é selecionada para continuar nos próximos períodos. Depois que um esquema de comunicação é acionado, áreas ruins do espaço da solução são eliminadas e a exploração de regiões promissoras é realizada. Os melhores morcegos dentre todos os morcegos de um grupo são migrados para outros grupos para substituir os morcegos mais inferiores daquele grupo e atualizar para cada grupo após executar cada número fixo de iterações. O método PBA proposto por Dao et al. (2018) foi testado em 43 instâncias JSSP, 3 instâncias de Fisher e Thompson (1963) e 40 instâncias de Lawrence (1984). A proposta foi comparada com os seguintes métodos: *Particle Swarm Optimization* (PSO) de Ge et al. (2008) e um BA básico. As configurações do algoritmo foram: tamanho da população $= 30$ e número de iterações = 200. Cada instância foi executada 5 vezes. O PBA foi testado em 43 instâncias e conseguiu obter o melhor resultado conhecido em 31 dessas instâncias. O algoritmo PBA ao ser comparado com um BA, encontrou resultados melhores ou iguais em todas as instâncias testada. Ao comparar o PBA com PSO, a proposta obteve resultados piores em apenas 3 instâncias. Nota-se que o algoritmo BA com a inclusão de processamento paralelo consegue atingir melhores soluções em problemas JSSP do que um BA básico, isso pode ser devido ao compartilhamento de informações entre os indivíduos em cada enxame, melhorando o espaço de busca e tendo uma maior diversidade na população. É possível que em metaheurísticas similares o comportamento de melhorias seja repetido ao incluir o processamento paralelo.

Os autores Jiang e Zhang (2018) propuseram utilizar a metaheurística *Grey Wolf Optimization* (GWO) para tratar o problema de otimização combinatória JSSP e minimizar o *makespan*. O algoritmo GWO foi criado por Mirjalili, Mirjalili e Lewis (2014) e foi inspirado na hierarquia e no comportamento de caça dos lobos cinzento na natureza. No GWO como outros algoritmos de enxame, ele também começa com um tamanho pré-definido de população, no qual os lobos são hierarquicamente classificados em quatro tipos de acordo com seus valores de *fitness*. O melhor lobo é denotado como alfa (α), o segundo melhor é denotado como beta (β), o terceiro melhor é denotado como delta (δ) e todos os outros indivíduos são considerados omega (ω). O comportamento de busca é principalmente guiado por α, β e δ. No algoritmo GWO, o operador de busca é projetado com base na operação de cruzamento para manter o trabalho do algoritmo diretamente em um domínio discreto. Em seguida, um método de mutação adaptativa é introduzido para manter a diversidade da população e evitar a

convergência prematura. Além disso, um método de busca de vizinhança variável é incorporado para aprimorar ainda mais a exploração. Jiang e Zhang (2018) incorporaram o método *Variable Neighborhood Search* (VNS) ao GWO, realizando uma busca local nos três melhores indivíduos. Diversos trabalhos correlatos fazem uso da estratégia da integração de metaheurísticas de comportamento de busca global com métodos de busca local, com o objetivo de melhorar a pesquisa de vizinhança em soluções mais aptas. O algoritmo GWO foi testado nas instâncias JSSP de Fisher e Thompson (1963) e de Lawrence (1984). A proposta foi comparada com os seguintes métodos: PAGA de Asadzadeh e Zamanifar (2010), LSGA de Asadzadeh (2015), *Teaching-Learning Based Optimization* (TLBO) de Baykasoglu, Hamzadayi e Köse (2014), Differential Evolution (DE) de Zhao et al. (2016) e Bacterial Foraging Algorithm (*BFO)* de Zhao et al. (2015). As configurações do algoritmo foram: tamanho da população $= 200$, número de iterações = $5 \times m \times n$, $q_{\max} = 10$ e $p_{\max} = 30$. A variável n representa o número de *jobs* e m o número de máquinas. Cada instância foi executada 10 vezes e o melhor resultado obtido foi levado em consideração para as comparações. O método GWO foi testado em 39 instâncias e conseguiu obter o melhor resultado conhecido em 23 dessas instâncias. Nota-se que o algoritmo encontrou os melhores resultados em instâncias menores, como as 10×5, 10×15 e 20×5, em instâncias maiores o método não conseguiu encontrar as melhores soluções e apresentou resultados inferiores em algumas instâncias em relação a outros trabalhos da literatura que foram utilizados para comparação. O método proposto não foi testado em todas as instâncias Lawrence (1984), deixando 4 instâncias maiores 15×15 fora da comparação e com isso a análise da comparação dos resultados foi de certa forma prejudicada. Nas instâncias comparadas, o GWO proposto obteve resultados competitivos ao ser comparado com métodos mais tradicionais ao problema tratado, tais como GA e DE.

No artigo de Semlali, Riffi e Chebihi (2019), foi proposta a metaheurística *Memetic Chicken Swarm Optimization* (MeCSO) para resolver o problema de programação da produção do tipo JSSP, o objetivo do trabalho foi fazer a minimização do *makespan*. A proposta do trabalho foi a de fazer a integração do algoritmo *Chicken swarm optimization* (CSO) com o método de busca local 2-opt de Croes (1958). O algoritmo CSO foi estabelecido por Meng et al. (2014) e foi inspirado no comportamento de um enxame de frangos enquanto procurava por comida. O algoritmo consiste em um enxame dividido em vários grupos, no qual cada um possui galo, galinhas e pintinhos. A ordem hierárquica nesse enxame é estabelecida pelo valor de *fitness*. O algoritmo MeCSO foi testado nas instâncias JSSP de Fisher e Thompson (1963),

Lawrence (1984), Applegate e Cook (1991) e Adams, Balas e Zawack (1988). As configurações do algoritmo foram: tamanho da população = 500, número de galos = 12%, número de galinhas = 25%, números de pintinhos = 63%, número de iterações = 10, fator de aprendizagem = 0,5, fator de aprendizagem das galinhas mães = 0,4 e fator de aprendizagem do galo = 0,65. O método MeCSO encontrou em 51,08% das instâncias avaliadas a melhor solução conhecida, alcançando resultados melhores que os encontrados por AIS de Bondal (2008). Nota-se que o algoritmo teve uma boa eficiência em instâncias de tamanhos menores, como as 10×5 e 10×10, mas em instâncias maiores como a de dimensão 15×10 o método encontrou soluções piores. Observando os resultados, é possível notar que o algoritmo em problemas maiores tem uma tendência a ficar preso em ótimos locais e não consegue ultrapassar certos pontos. Os autores Semlali, Riffi e Chebihi (2019) não avaliaram a técnica em todas as instâncias disponíveis em cada base da literatura, uma vez que, por exemplo, a base proposta por Lawrence (1984) contém 40 instâncias para teste, e os autores apenas testaram em 21 instâncias, deixando de verificar as instâncias de tamanho maiores, isso prejudicou uma melhor avaliação do comportamento do algoritmo.

No artigo de Kurdi (2019), foi proposto GA-CPG-GT, que é um Algoritmo Genético com operador de cruzamento guiado por caminhos críticos combinado com o algoritmo de Giffler e Thompson (1960). O algoritmo proposto foi aplicado em 55 instâncias de JSSP para a validação do método e teve como objetivo a minimização do *makespan*. Muitos trabalhos da literatura que fazem uso do GA, normalmente utilizam operadores de cruzamento que fazem a combinação de materiais genéticos de forma aleatória, isto é, sem ter um conhecimento prévio de quais materiais genéticos contém traços mais importantes a serem passados para as próximas gerações. O autor Kurdi (2019) propôs em seu artigo uma forma de cruzamento no qual existe uma troca seletiva dos materiais genéticos. O algoritmo segue os seguintes passos: No primeiro passo é criada uma população de indivíduos usando o algoritmo Giffler e Thompson, cada indivíduo deve possuir duas representações chamadas de fenótipo e genótipo, no qual o fenótipo representa os traços comportamentais deste indivíduo em seu ambiente, ou seja, representa uma solução potencial para o JSSP, o genótipo representa a composição genética deste indivíduo sobre a forma de um cromossomo. O segundo passo envolve avaliar o *fitness* de cada indivíduo para o ambiente, isto é, a quão boa é a solução representada. Os passos seguintes são referentes às fases de evolução da cooperação e às fases de autoadaptação. Enquanto as fases de cooperação proporcionam exploração do espaço de busca através do operador de cruzamento, as fases de autoadaptação fornecem exploração do espaço de busca através do operador de

mutação. O algoritmo foi avaliado nas instâncias JSSP de Fisher e Thompson (1963), Lawrence (1984), Applegate e Cook (1991) e Adams, Balas e Zawack (1988). As configurações do algoritmo foram: tamanho da população = 100, máximo de gerações = 1000, taxa de cruzamento = 0,8 e taxa de mutação 0,05. Cada instância JSSP foi testada 10 vezes e o menor valor obtido das 10 execuções foi apresentado. A proposta foi comparada com os seguintes métodos: PAGA de Asadzadeh e Zamanifar (2010), ACO de Seo e Kim (2010), PSO de Yen e Ivers (2009), *Hybrid Parallel* GA de Yusof et al. (2011) e *Clonal Selection Algorithm* (CSA) de Atay e Kodaz (2014). O GA-CPG-GT obteve resultados melhores do que os outros artigos comparados, isso mostra que uma escolha prévia e boa de quais materiais genéticos são trocados tem forte impacto no resultado obtido. O objetivo do estudo de Kurdi (2019) foi investigar os impactos de selecionar os materiais genéticos trocados durante o cruzamento com informações prévias sobre os caminhos críticos que existem no domínio ao invés de selecioná-los aleatoriamente. Por meio dos resultados apresentados, o autor conseguiu apresentar o impacto que traz essa área de estudo. De acordo com Kurdi (2019), a ideia básica que foi proposta para a identificação dos genes que seguram as características mais importantes é uma área muito promissora de pesquisa e merece uma investigação mais aprofundada, uma vez que esta produz melhorias significativas quando aplicada no JSSP.

Os autores Hamzadayi, Baykasoğlu e Akpinar (2020) propuseram adaptar alguns componentes da metaheurística *Single Seekers Society* (SSS) para lidar com problemas de otimização combinatória. O SSS proposto foi aplicado em dois tipos de programação da produção presentes na literatura, o *Flow Shop Scheduling Problem* e o *Job Shop Scheduling Problem*. O algoritmo SSS foi elaborado por Baykasoğlu, Hamzadayi e Akpinar (2019) e consiste em uma nova metaheurística que permite a cooperação entre diferentes heurísticas de busca. O SSS incorpora diversas metaheurísticas, como o *Simulated Annealing*, *Threshold Accepting*, *Greedy Search* (GS) e todas as informações de cada método funcionam de modo integrado. Para gerar novas soluções, o SSS compartilha informações via *crossover* e manipula a busca integrando as informações via estrutura de vizinhança. O SSS proposto por Hamzadayi, Baykasoğlu e Akpinar (2020) foi testado nas instâncias JSSP de Fisher e Thompson (1963) e Lawrence (1984) e nas instâncias de FSSP. As configurações do algoritmo foram: tamanho da população = 100 e máximo de gerações = 1000. Cada instância JSSP foi testada 10 vezes e o menor valor obtido das 10 execuções foi apresentado. A proposta foi comparada com os seguintes métodos: TLBO de Baykasoglu, Hamzadayi e Köse (2014), HGA de Qing-dao-er-ji e Wang (2012), MA de Gao *et al.* (2011), F&F de Rego e Duarte (2009), GRASP de Binato *et*

al. (2002a) e *Beam Search* (BS) de Sabuncuoglu e Bayiz (1999). O SSS não foi o método que obteve melhores resultados para instâncias JSSP, porém obteve resultados competitivos e conseguiu encontrar o melhor valor conhecido em 14 instâncias das 20 instâncias que foram testadas. O método comprovou ser capaz de manter sua efetividade em problemas similares de otimização combinatória obtendo resultados satisfatórios nos dois problemas de programação da produção testados, o JSSP e FSSP.

No artigo de Yu *et al.* (2020), foi proposto um PSO híbrido aprimorado com peso de inércia não linear e mutação gaussiana (NGPSO) para resolver instâncias de JSSP. O peso da inércia não linear foi adicionado ao método com o objetivo de melhorar a capacidade de busca local e a mutação gaussiana foi adicionada com o objetivo de melhorar a capacidade de busca global. O método busca manter um equilíbrio entre fazer buscas locais e garantir a diversidade da população, reduzindo a probabilidade de o algoritmo cair em uma solução ótima local. O NGPSO foi testado nas instâncias JSSP de Fisher e Thompson (1963), Lawrence (1984), Adams, Balas e Zawack (1988) e Applegate e Cook (1991). As configurações do algoritmo foram: tamanho da população = 100 e máximo de gerações = 1000 para instâncias de menor complexidade e tamanho da população = 150 e máximo de gerações = 1000 para instâncias de maior complexidade. Cada instância JSSP foi testada 30 vezes e o menor valor obtido das 30 execuções foi apresentado. No caso, a proposta de Yu *et al.* (2020) foi comparada com os seguintes métodos: PSO de Zhang *et al.* (2011), PSO de Meng, Zhang e Fan (2016) e com os métodos canônicos de CSA, GA, DE e ABC. Os resultados experimentais indicaram que o algoritmo NGPSO possui desempenho satisfatório e alta capacidade na resolução de JSSP, sendo que conseguiu encontrar o melhor valor conhecido em 38 instâncias das 62 instâncias que foram testadas. Nas instâncias de menor complexidade foram encontrados o melhor valor conhecido em 26 casos de 33 instâncias, mas nas instâncias de maior complexidade o desempenho do método declinou e foi encontrado o melhor valor conhecido em apenas 12 casos de 29 instâncias. O NGPSO apresentou uma melhor convergência ao ser comparado com outros PSO de Zhang *et al.* (2011) e de Meng, Zhang e Fan (2016). As técnicas que foram adicionadas para aprimoramento de busca local e de busca global melhoraram significativamente o PSO, mas para instâncias de maior complexidade é necessário um melhor balanceamento entre as buscas.

Os autores Alkhateeb, Abed-Alguni e Al-Rousan (2021) propuseram um método híbrido discreto de *Cuckoo Search* (CS) e *Simulated Annealing*, intitulado DCSA, para tratar instâncias de JSSP. O DCSA incorpora os operadores de otimização do SA no algoritmo de

pesquisa do CS. É utilizado uma combinação dos métodos VNS e voo de Lévy para uma melhor exploração do espaço de pesquisa. O DCSA foi testado nas instâncias JSSP de Fisher e Thompson (1963) e Lawrence (1984). As configurações do algoritmo foram: tamanho da população = 50 e máximo de gerações = 1500. Cada instância JSSP foi testada 10 vezes e o menor valor obtido das 30 execuções foi apresentado. A proposta foi comparada com os seguintes métodos: ABC discreto (DABC) de Yin, Li e Zhou (2011), Híbrido de PSO e VNS de Tasgetiren *et al.* (2006), CS discreto (DCS) de Ouaarab, Ahiod e Yang (2015), GWO de Jiang e Zhang (2018), WPA discreto (DWPA) de Wang, Tian e Wang (2019) e GA de Kalshetty, Adamuthe e Kumar (2020). Nos testes realizados, o DCSA proposto apresentou uma convergência mais rápida que os demais métodos comparados e conseguiu encontrar a melhor solução conhecida em 29 instâncias das 34 que foras selecionadas para teste. O DCSA também apresentou um menor custo computacional que os demais métodos comparados. O DCSA apresentou um desempenho superior que o DCS, essa melhoria deve ser devida a integração do SA ao CS e do uso de diferentes métodos de exploração, como o VNS e voo de Lévy. No artigo não foram incluídas instâncias mais complexas do problema. Por exemplo, no caso das instâncias Lawrence (1984) foram testadas apenas 31 das 40 instâncias disponíveis, não sendo abordadas as instâncias que são consideradas mais difíceis e que normalmente a maioria dos métodos da literatura fica presa em mínimos locais.

Diversas abordagens aplicadas em JSSP foram propostas, algumas incluíram agentes inteligentes, populações paralelas ou hibridização de metaheurísticas com técnicas de busca local. Constata-se por meio dos trabalhos relatados que a hibridização é uma maneira eficaz de melhorar o desempenho e a eficácia de diversas metaheurísticas. As técnicas de busca local são a forma mais comum de hibridização e são utilizadas com a finalidade de melhorar o desempenho desses algoritmos.

2.4 Considerações finais

Neste capítulo, foram apresentados os conceitos de Programação da Produção, definição do problema JSSP e a fundamentação teórica de Algoritmos Genéticos híbridos especializados em solucionar JSSPs, sendo necessários para o entendimento deste trabalho. Constata-se por meio da revisão bibliográfica que os problemas de programação da produção do tipo JSSP têm

atraído a atenção de vários pesquisadores pelo fato de ter um comportamento combinatório classificado como sendo *NP-Hard* e por ser equivalente a vários problemas da atualidade.

Os algoritmos híbridos de metaheurística estão se destacando entre os vários métodos para JSSP porque eles podem combinar os méritos de diferentes algoritmos que utilizam metaheurísticas. Os algoritmos híbridos geralmente apresentam um desempenho notável, incorporando a busca local para obter uma combinação adequada de esforços de busca de diversificação e intensificação. Entre a grande diversidade de estudos, é possível constatar a superioridade das metaheurísticas de comportamento de busca global com inclusão de técnicas de busca local, como pode ser verificado nos trabalhos de Ombuki e Ventresca (2004), Watanabe, Ida e Gen (2005), Asadzadeh (2015), Sayoti, Riffi e Labani (2016), Díaz *et al.* (2017), Jiang (2018), Wang, Bing *et al.* (2018), Dao *et al.* (2018), Jiang e Zhang (2018), Semlali, Riffi e Chebihi (2019), Kurdi (2019), Hamzadayi, Baykasoğlu e Akpinar (2020), Yu *et al.* (2020) e Alkhateeb, Abed-Alguni e Al-Rousan (2021).

Na Seção 2.3, foram apresentadas diversas abordagens que utilizam variados algoritmos, tais como AG, LSGA, GSA, aLSGA, IPBGA, AMGA, NIMGA, GBA, MA, HGWO, TSAUN, PBA, GWO, MeCSO, SSS, NGPSO e DCSA, para tratar o problema de programação da produção do tipo JSSP e constata-se que estas estratégias obtiveram sucesso.

No próximo capítulo, é apresentada a proposta deste trabalho, a qual é baseada nas fundamentações apresentadas neste capítulo de revisão.

Capítulo 3

PROPOSTA DO TRABALHO

3.1 Considerações iniciais

Neste capítulo, são apresentados os métodos que foram propostos e que compõem o *framework* de métodos do tipo GA para JSSP. O *framework* proposto possui a finalidade de cumprir os objetivos gerais deste trabalho, os quais tratam da minimização da medida de desempenho *makespan* em ambientes de testes do tipo JSSP propostos na literatura; e os objetivos específicos, os quais estão compreendidos entre desenvolver melhorias em busca local para Algoritmos Genéticos, desenvolver novas estratégias de controle de convergência prematura e desenvolver novos operadores para o GA que direcionem e melhorem os resultados alcançados pela literatura. Em resumo, as contribuições deste trabalho são abrangidas pelos seguintes tópicos:

- Um operador de crossover aprimorado (Seção 3.2.1) baseado na versão do GSA de Watanabe, Ida e Gen (2005), incluindo uma estratégia multi-*crossover* com o objetivo de aumentar a capacidade de pesquisa do método usando um framework baseado em um conjunto de funções de crossover.
- Uma técnica de busca local aprimorada (Seção 3.2.2) em união com uma versão generalizada do operador de mutação proposto na versão do aLSGA de Asadzadeh (2015) e na versão do LSGA de Ombuki e Ventresca (2004), incluindo uma parametrização variável.

- Uma versão melhorada do operador de busca local elite (Seção 3.2.3) de Asadzadeh (2015), expandindo o espaço de busca utilizando um conjunto de funções de mutação.
- Um novo operador de orientação populacional para GAs (Seção 3.2.4): o Operador de Melhoramento Genético baseado em Análise de Frequência, cuja sigla do inglês é GIFA. O método consiste em uma nova forma de determinar a relevância genética com base na análise de frequência dos genes de indivíduos que apresentam bons valores de *fitness* na população. Também é proposta a construção de um indivíduo representativo que represente este grupo de bons indivíduos e que seja utilizado no processo de manipulação genética para orientar os piores indivíduos para boas soluções e, possivelmente, que estas se tornem destaques positivos na população.
- Um *framework* de métodos do tipo GA contendo métodos tradicionais da literatura especializada de GA para JSSP, melhorias de métodos da literatura e novos operadores que podem melhorar o desempenho do GA.

Desta forma, este capítulo está seccionado da seguinte maneira: na Seção 3.2, são apresentados os operadores de busca local e melhoramento genético propostos; na Seção 3.3, é definido os componentes do *framework* proposto e semelhanças e diferenças do material proposto em comparação com trabalhos da literatura especializada são destacadas; o capítulo é finalizado com algumas conclusões na Seção 3.4.

3.2 Operadores propostos: Busca local e Melhoramento genético

Neste trabalho, são propostos três operadores de busca local, sendo um deles embutido em operador de cruzamento; um como operador de mutação; e outro de caráter massivo. Tais procedimentos possuem inspiração nos trabalhos descritos no Capítulo 2. Também é apresentada uma nova forma de melhoramento genético a partir da análise de frequência de determinados genes em indivíduos bem adaptados na população.

Especificamente, são detalhadas nesta seção as seguintes propostas: o uso de busca local de forma embutida nos operadores de cruzamento (Seção 3.2.1) e de mutação (Seção 3.2.2) e de forma massiva em torno de um conjunto de indivíduos bem adaptados da população (Seção 3.2.3). Também é proposto um novo operador de direcionamento populacional para sugerir áreas de buscas aos indivíduos de uma população (Seção 3.2.4).

3.2.1 Operador de cruzamento acoplado com busca local

O operador de cruzamento proposto consiste na estratégia de fazer uso de mais de uma função de cruzamento para a condução da adaptação de área de busca, o que representa a generalização do operador de cruzamento de Watanabe, Ida e Gen (2005). Assim, o operador proposto é apresentado na forma de um framework que considera $n_{\times}$ funções de cruzamento a fim de recombinar os mesmos cromossomos de maneiras diferentes. Para isso, é definido um conjunto $\mathcal{F}_{\times}$ como apresentado na Equação (3.1).

$$\mathcal{F}_{\times} = \{f_{\times,1}, f_{\times,2}, \dots, f_{\times,\mathrm{n}_{\times}}\}. \tag{3.1}$$

Para a modelagem do operador, considera-se, sem perda de generalidade, que cada função $f_{\times} \in \mathcal{F}_{\times}$ é uma função que combina dois pais do conjunto factível da instância JSSP resultando em dois filhos também factíveis. Isto é, cada função $f_{\times} \in \mathcal{F}_{\times}$ assume a forma da Equação (3.2) a seguir.

$$f_{\times}: \mathbb{O} \times \mathbb{O} \to \mathbb{O} \times \mathbb{O}. \tag{3.2}$$

Essencialmente, o operador de cruzamento proposto deve agir como uma versão mais abrangente e rigorosa do operador de cruzamento de Watanabe, Ida e Gen (2005). No caso, é válido supor que o uso de técnicas distintas de cruzamento aumenta o poder de exploração local, uma vez que estas definem duas estratégias diferentes de conduzir a recombinação dos mesmos

indivíduos e, portanto, dão margem à exploração de diferentes áreas de busca. Sendo assim, o operador de cruzamento funciona a partir de três indivíduos selecionados aleatoriamente na população e aplica-se ocasionalmente diferentes funções de $\mathcal{F}_\times$ em todos os pares possíveis desse trio até que sejam encontradas três soluções que superem os seus respectivos progenitores, no sentido de apresentarem valores de *fitness* melhores, ou até que cada par tenha realizado R_c cruzamentos. A esquematização detalhada deste operador é apresentada no Algoritmo 3.1.

Algoritmo 3.1. Operador de multi-*crossover* proposto.

Entrada	$\mathcal{F}_\times = \{f_{\times,1}, f_{\times,2}, \dots, f_{\times,\mathrm{n}_\times}\}$: Conjunto de funções de cruzamento. p_1, p_2, p_3: Três indivíduos selecionados aleatoriamente. F: Função *fitness*. R_c: Número máximo de cruzamentos permitidos para cada par de indivíduos.
Saída	c_1, c_2, c_3: Indivíduos gerados.

PARA $k \coloneqq 1$ **ATÉ** 3 **FAÇA** // Avaliar todos os casais possíveis considerando três pais.
  SE $k == 1$ **ENTÃO** // Na primeira iteração, considera-se como pais os indivíduos p_1 e // p_2.
    $(P_1, P_2) \coloneqq (p_1, p_2)$
  SENÃO SE $k == 2$ **ENTÃO**
    $(P_1, P_2) \coloneqq (p_1, p_3)$ // Na segunda iteração, considera-se como pais os indivíduos p_1 e // p_3.
  SENÃO
    $(P_1, P_2) \coloneqq (p_2, p_3)$ // Na terceira iteração, considera-se como pais os indivíduos p_2 e // p_3.
  FIM_SE

  $F_{P_1} \coloneqq F(P_1)$; // Avalia-se o primeiro progenitor.
  $F_{P_2} \coloneqq F(P_2)$; // Avalia-se o segundo progenitor.
  PARA $i \coloneqq 1$ **ATÉ** R_c **FAÇA**
    $f_\times \coloneqq \mathrm{rand\_set}(\mathcal{F}_\times)$; // Uma função de cruzamento é tomada aleatoriamente em $\mathcal{F}_\times$.
    $(\hat{c}_{i,1}, \hat{c}_{i,2}) \coloneqq f_\times(P_1, P_2)$ // Dois filhos são gerados utilizando-se a função $f_\times$.
    $F_{i,1} \coloneqq F(\hat{c}_{i,1})$; // Avalia-se o primeiro filho.

$F_{i,2} \coloneqq F(\hat{c}_{i,2})$; // Avalia-se o segundo filho.

SE $F_{i,1} < F_{i,2}$ **ENTÃO** // Se o *fitness* do primeiro filho for melhor do que o *fitness* do // segundo filho,

$(\hat{c}_i,\ F_i) \coloneqq (\hat{c}_{i,1},\ F_{i,1})$ // então considera-se apenas o primeiro filho e seu respectivo // valor de *fitness.*

SENÃO // Caso contrário,

$(\hat{c}_i,\ F_i) \coloneqq (\hat{c}_{i,2},\ F_{i,2})$ // considera-se apenas o segundo filho e seu respectivo valor // de *fitness.*

FIM_SE

SE $F_i < F_{P_1}$ **OU** $F_i < F_{P_2}$ **ENTÃO** "**SAIA DO PARA**"; // Se o fitness do melhor filho // gerado é melhor que o fitness de um dos pais, então não é preciso gerar mais filhos para // este casal.

FIM_PARA

$i^* \coloneqq \arg\min_i\{F_i\}$; // Seleciona-se o índice do melhor filho gerado para o casal fixado.

$c_k \coloneqq \hat{c}_{i^*}$; // O melhor filho gerado para o casal fixado.

FIM_PARA

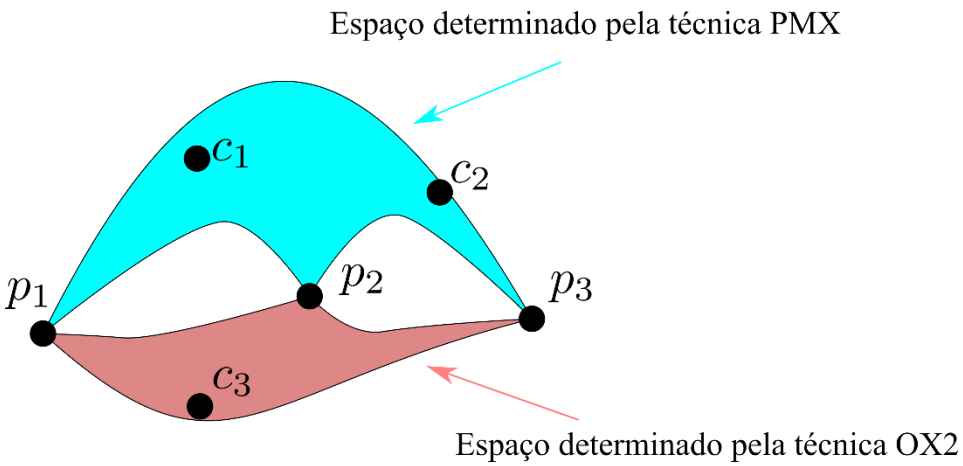
Fonte: Elaborado pela autora.

Desta forma, em todos os pares possíveis de três indivíduos tomados aleatoriamente na população são realizadas operações distintas de cruzamento até que seja gerada uma solução que possua um valor de *fitness* menor do que o valor de *fitness* de um dos pais ou até que sejam realizados R_c cruzamentos.

O critério de busca deste operador é bem mais rigoroso do que o critério de busca dos operadores do GA de Watanabe, Ida e Gen (2005), uma vez que o operador destes autores realiza cruzamentos até que seja encontrada uma solução cujo valor de *fitness* seja melhor do que o pior valor de *fitness* apresentado na população toda, sendo que o valor de *fitness* dos pais não são necessariamente levados em consideração. Sendo assim, o operador proposto deve ser capaz de encontrar boas soluções com mais frequência que o operador de cruzamento de Watanabe, Ida e Gen (2005), uma vez que é realizada uma busca mais criteriosa. Além disso, a utilização de várias técnicas de cruzamento amplia o espaço de busca percorrido, pois as

soluções devem ser geradas por lógicas distintas e, portanto, a área de busca pode ser representada por regiões distintas, como ilustrado na Figura 3.1.

Figura 3.1. Áreas de busca definidas pelos cruzamentos de p_1, p_2 e p_3 considerando como funções de cruzamento as técnicas PMX e OX2.



Fonte: Elaborada pela autora.

3.2.2 Operador de mutação com rotina de busca local

De maneira análoga à modelagem do operador de cruzamento, o operador de mutação proposto é definido como um *framework* que funciona de acordo com um conjunto de n_{mut} funções de mutação. Isto é, durante o processo de mutação executado no método proposto, um cromossomo pode ser transformado com uma mutação definida em um conjunto $\mathcal{F}_{\text{mut}}$, apresentado na Equação (3.3).

$$\mathcal{F}_{\text{mut}} = \{f_{\text{mut},1}, f_{\text{mut},2}, \dots, f_{\text{mut},n_{\text{mut}}}\}, \tag{3.3}$$

na qual, cada função $f_{\text{mut}} \in \mathcal{F}_{\text{mut}}$ é uma função de mutação que opera com respeito a duas coordenadas i e j de um dado cromossomo $c \in \mathbb{O}$ resultando em um cromossomo $\hat{c} \in \mathbb{O}$, tal como apresentado na Equação (2.5).

Neste trabalho, é proposto como operador de mutação a generalização do operador de busca local de Asadzadeh (2015) como sendo uma variação do operador de mutação de

Watanabe, Ida e Gen (2005) (Algoritmo 2.8). Desta forma, para cada indivíduo gerado no operador de cruzamento, uma f_{mut} é escolhida randomicamente em $\mathcal{F}_{\text{mut}}$ e aplicada sucessivamente R_m vezes mantendo-se as modificações benéficas e prosseguindo com o método a partir das mesmas, dando origem a uma população mutante com o mesmo número de indivíduos que a população de filhos. Entretanto, a fim de manter parte do comportamento tradicional de um operador de mutação, que consiste em causar transformações no cromossomo desconsiderando se esta é benéfica ou não ao indivíduo, é acrescentada a possibilidade de ser aplicada apenas uma execução de uma função de mutação sobre um pequeno grupo de cromossomos, o que corresponde a um processo de mutação simples. Para controlar esta rotina, considera-se que uma porcentagem ϵ_{LS} de cromossomos devem passar pelo processo de busca local e, de maneira complementar, uma porcentagem $1 - \epsilon_{\text{LS}}$ de cromossomos devem passar pelo processo de mutação simples. A esquematização é apresentada no Algoritmo 3.2.

Algoritmo 3.2. Operador de mutação proposto.

Entrada	$\mathcal{F}_{\text{mut}} = \{f_{\text{mut},1}, f_{\text{mut},2}, \cdots, f_{\text{mut},\text{n}_{\text{mut}}}\}$: Conjunto de funções de mutação. P_{filhos} : Indivíduos gerados no cruzamento. Δ_{mut}: Taxa de mutação. F: Função *fitness* do problema. $N \times M$: Dimensões do JSSP tratado. R_m: Número total de mutações realizadas em cada indivíduo. ϵ_{LS}: Porcentagem dos indivíduos de P_{filhos} que passarão pelo processo de busca local.
Saída	P_{mut}: Indivíduos mutantes.

$f_{\text{mut}} \coloneqq \text{rand\_set}(\mathcal{F}_{\text{mut}})$; // Seleciona-se aleatoriamente uma função de mutação de $\mathcal{F}_{\text{mut}}$.

$P_{\text{mut}} \coloneqq \{\ \}$;

PARA $c \in P_{\text{filhos}}$ **FAÇA** // O processo é conduzido sobre todos os indivíduos da população // de filhos.

$s_1 = \text{rand}([0,1])$ // Seleciona-se aleatoriamente um valor entre 0 e 1.

$s_2 = \text{rand}([0,1])$

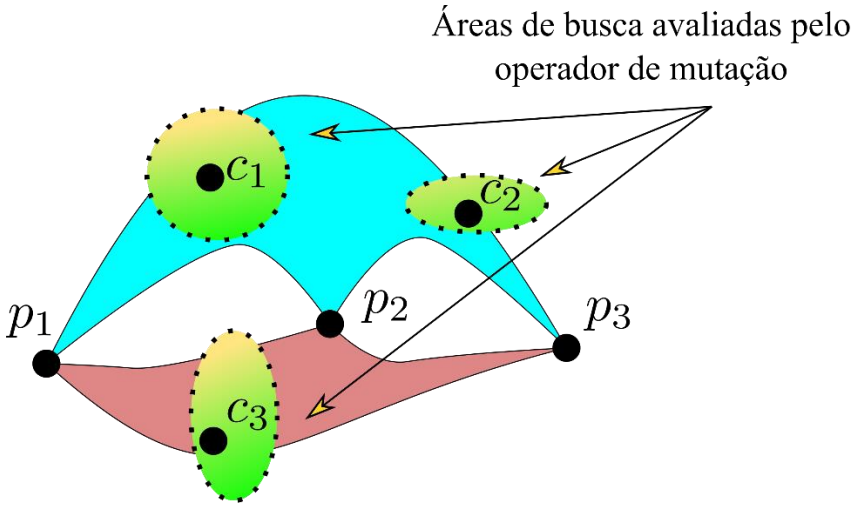
SE $s_1 < \Delta_{\text{mut}}$ **ENTÃO** // Neste caso, haverá mutação.

SE $s_2 < \epsilon_{\text{LS}}$ **ENTÃO** // Caso o número gerado seja menor a ϵ_{LS}, então
// deve haver busca local sobre c.
$F_c \coloneqq F(c)$; // Avalia-se o valor de *fitness* inicial de c.
PARA $i =: 1$ **ATÉ** R_m **FAÇA** // O processo de busca é repetido R_m vezes.
// Selecionam-se aleatoriamente duas coordenadas r_1 e r_2 distintas:
$r_1 \coloneqq \text{rand\_set}(\{1,2,3,\dots,N \cdot M\})$;
$r_2 \coloneqq \text{rand\_set}(\{1,2,3,\dots,r_1 - 1, r_1 + 1,\dots,N \cdot M\})$;
$\hat{c} \coloneqq f_{\text{mut}}(c,(r_1,r_2))$; // Aplica-se a função de mutação f_{mut} em c.
$F_{\hat{c}} \coloneqq F(\hat{c})$; // Calcula-se o valor de *fitness* da versão mutante de c.
SE $F_{\hat{c}} \leq F_c$ **ENTÃO** // Avalia-se se houve melhora no valor de *fitness*.
// No caso de $\hat{c}$ apresentar melhor valor de *fitness* que c, então o processo deve
// continuar a partir de $\hat{c}$.
$c \coloneqq \hat{c}$; // Atualiza-se c como sendo sua versão mutante.
$F_c \coloneqq F_{\hat{c}}$; // Atualiza-se o valor de *fitness* como sendo o *fitness* de $\hat{c}$.
FIM_SE
FIM_PARA
SENÃO // Caso o número r gerado seja maior que ϵ_{LS}, então c deve passar por um
// processo de mutação simples.
$r_1 \coloneqq \text{rand\_set}(\{1,2,3,\dots,N \cdot M\})$;
$r_2 \coloneqq \text{rand\_set}(\{1,2,3,\dots,r_1 - 1, r_1 + 1,\dots,N \cdot M\})$;
$c \coloneqq f_{\text{mut}}(c,(r_1,r_2))$; // Aplica-se somente uma vez a função de mutação f_{mut} em
// c.
FIM_SE
FIM_SE
$P_{\text{mut}} \coloneqq P_{\text{mut}} \cup \{c\}$;
FIM_PARA

Fonte: Elaborado pela autora.

Este procedimento tem como principal intuito realizar minuciosas buscas em regiões próximas a soluções sabidamente boas, uma vez que tais soluções foram determinadas de forma que fossem melhores que os seus respectivos pais e que provavelmente aprimoram as soluções de gerações anteriores. A ilustração desta busca local é apresentada na Figura 3.2.

Figura 3.2. Áreas de busca avaliadas pelo operador de mutação em torno de c_1, c_2 e c_3.



Fonte: Elaborada pela autora.

Vale ressaltar que o operador de mutação e busca local de Asadzadeh (2015) é um caso especial do operador de mutação e busca local proposto no caso em que $R_m = N \cdot M$, $\mathcal{F}_{\text{mut}} = \{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$, $\epsilon_{\text{LS}} = 100\%$ e com taxa de mutação (Δ_{mut}) igual a 100%, pois todos os filhos passam pelo processo de mutação que possui caráter de busca local. Assim, o operador proposto nessa seção também pode ser interpretado como sendo uma generalização do operador de mutação de Asadzadeh (2015).

3.2.3 Operador de busca local massiva

Neste trabalho, é proposto como operador de busca local massiva um aprimoramento da busca local elite de Asadzadeh (2015), discutida neste texto na Seção 2.2.4.6.3. Este aprimoramento se dá através do uso de mais de uma função de perturbação na técnica, fazendo com que o operador faça uso eventual de diferentes funções de mutação ao invés de apenas uma.

Um operador de busca local massiva tem como objetivo principal avaliar todas as perturbações possíveis com respeito às coordenadas de uma dada solução e preservar as modificações que aprimoram o seu valor de *fitness*. A principal proposta desta rotina é efetuar

buscas mais elaboradas em regiões próximas de boas soluções, uma vez que estas representam boas características do espaço de busca e, intuitivamente, podem estar próximas de indivíduos com características semelhantes, mas com melhor valor de *fitness*. Esta categoria de busca é conduzida levando-se em consideração todas as combinações possíveis dentre as perturbações de coordenadas de um cromossomo por meio de diferentes estratégias de perturbação. Asadzadeh (2015) faz uso apenas da função de mutação f_{swap} em torno do melhor indivíduo em seu operador de busca local elite. Neste trabalho, é proposto que esta rotina ocorra utilizando-se diferentes funções de perturbação dadas por um conjunto $\mathcal{F}_{\text{pert}}$ com n_{pert} funções de perturbação, o qual é definido na Equação (3.4).

$$\mathcal{F}_{\text{pert}} = \left\{ f_{\text{pert},1}, f_{\text{pert},2}, \dots, f_{\text{pert},\text{n}_{\text{pert}}} \right\}. \tag{3.4}$$

No caso, o uso da nomenclatura "funções de perturbação" será empregado nesta seção para facilitar a apresentação do método, entretanto, as funções de perturbação utilizadas no operador proposto são definidas da mesma forma que as funções de mutação de acordo com a Equação (2.5).

Para o funcionamento do método, é proposto que uma função seja selecionada aleatoriamente em $\mathcal{F}_{\text{pert}}$ e, na sequência, sejam conduzidas todas as perturbações sobre as coordenadas de um indivíduo utilizando-se esta função, de maneira que apenas as perturbações que melhore os valores de *fitness* do indivíduo sejam preservadas. Com isso, no decorrer das gerações, o uso de várias funções de perturbação ao invés de apenas uma pode aprimorar a capacidade de busca do operador. Além disso, é proposto que esta busca local de caráter massivo seja realizada em um dado conjunto P_{melhores} de melhores indivíduos que sejam distintos na população ao invés de conduzi-la apenas sobre o melhor indivíduo, uma vez que este procedimento promove uma exploração mais elaborada em mais de uma região do espaço de busca, preservando mais do que uma "boa linhagem" na população. Sendo assim, é apresentado o "Operador de Busca Local Massivo" nos indivíduos de P_{melhores} no Algoritmo 3.3.

Algoritmo 3.3. Operador de busca local massiva proposto.

Entrada	$\mathcal{F}_{\text{pert}} = \{f_{\text{pert},1}, f_{\text{pert},2}, \ldots, f_{\text{pert},n_{\text{pert}}}\}$: Conjunto de funções de perturbação. P_{melhores} : Melhores indivíduos distintos presentes na população. F: Função *fitness* do problema. $N \times M$: Dimensões do JSSP tratado.
Saída	P_{massivo}: Aprimoramento dos indivíduos de P_{melhores}.

$f_{\text{pert}} \coloneqq \text{rand\_set}(\mathcal{F}_{\text{pert}})$; // Seleciona-se aleatoriamente uma função de perturbação.

$P_{\text{massivo}} \coloneqq \{\ \}$;

$N_{\text{dim}} \coloneqq N \cdot M$;

PARA $c \in P_{\text{melhores}}$ **FAÇA** // O procedimento é efetuado para todos os integrantes do
// grupo de melhores indivíduos.

$F_c \coloneqq F(c)$; // Avalia-se o cromossomo c.

PARA $i =: 1$ **ATÉ** N_{dim} **FAÇA** // Destaca-se que todas as combinações possíveis dos
// índices i e j são consideradas.

PARA $j \coloneqq 1$ **ATÉ** N_{dim} **FAÇA**

$\hat{c} \coloneqq f_{\text{pert}}(c, (i,j))$; // Realiza-se a perturbação.

$F_{\hat{c}} \coloneqq F(\hat{c})$; // Calcula-se o valor do *fitness* da versão perturbada de c.

SE $F_{\hat{c}} < F_c$ **ENTÃO** // Caso houve melhora no valor do fitness do cromossomo,

$c \coloneqq \hat{c}$; // atualiza-se o cromossomo, mantendo a perturbação benéfica
// e conduzindo-se as próximas a partir desta nova versão.

$F_c \coloneqq F_{\hat{c}}$;

FIM_SE

FIM_PARA

FIM_PARA

$P_{\text{massivo}} \coloneqq P_{\text{massivo}} \cup \{c\}$; // Atualiza-se o conjunto de indivíduos aprimorados.

FIM_PARA

Fonte: Elaborado pela autora.

3.2.4 Operador de melhoramento genético via análise de frequências (GIFA)

Operadores de melhoramento genético possuem como principal objetivo fornecer a indivíduos que não conseguem se destacar em uma população o reforço de um ou mais indivíduos que obtiveram sucesso no processo de adaptação. Em outras palavras, operadores de transgenia direcionam os piores indivíduos de uma população a áreas sabidamente boas do espaço de busca.

Amaral e Hruschka JR (2014) apresentam um operador de transgenia que simula o processo de melhoramento genético. Para realizar tal procedimento, os autores propõem que no decorrer do GA, a população do mesmo seja copiada para quatro populações e, em cada uma dessas quatro populações, os melhores indivíduos transfiram aleatoriamente até 4 genes para os piores indivíduos. Em seguida, mantém-se apenas a população que apresenta o melhor indivíduo dentre as quatro comparadas.

Viana, Morandin Junior e Contreras (2020d) especializaram o operador de transgenia de Amaral e Hruschka JR (2014) para a solução de instâncias de JSSP com GA. Os autores propõem a identificação de relevância genética nos genes utilizados no processo de transgenia e definem um pré-processamento que determina quais genes são mais relevantes para serem utilizados no operador transgênico proposto. Entretanto, tal pré-processamento consome muito tempo computacional e pode ser inviável em JSSPs de grande porte.

Neste trabalho, propõe-se uma nova maneira de se determinar a relevância genética com base na frequência genética de indivíduos com bons valores de *fitness*. O operador, de sigla GIFA (*Genetic improvement based on frequency analysis*), calcula quais genes de um cromossomo podem ser utilizados para orientar indivíduos mal adaptados a melhores espaços de busca. Também é proposta a construção de um indivíduo que represente os bons indivíduos de uma população e seja utilizado no processo de melhoramento genético para “orientar” os piores indivíduos de uma população a fim de que os mesmos possam se destacar positivamente na população.

3.2.4.1 Confecção do indivíduo modelo

Inicialmente, seleciona-se uma parcela da população que apresente os melhores valores de *fitness*. Suponha-se que sejam selecionados $N_{\text{Exemplares}}$ indivíduos que sejam considerados bons exemplos de cromossomos da população considerada. Em seguida, para cada *job* de índice i, associa-se um vetor de frequências $\vec{v}_i \in \mathbb{Z}_+^{N \cdot M}$, no qual em cada coordenada é armazenado o número de vezes em que o *job* i aparece exatamente na posição desta coordenada nos cromossomos selecionados para comparação. No Algoritmo 3.4, é apresentada a rotina necessária para o cálculo dos vetores $\vec{v}_i$.

Algoritmo 3.4. Construção dos vetores de frequência genética.

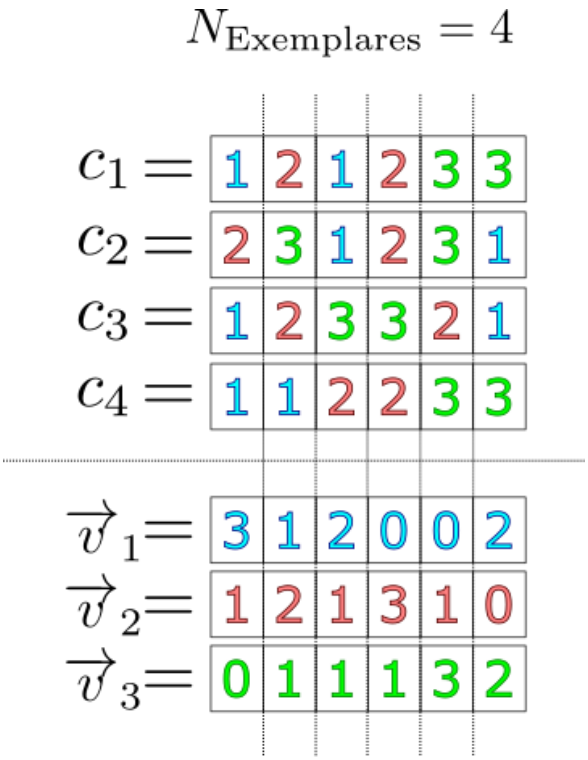
Entrada	$\{c_1, c_2, \dots, c_{N_{\text{Exemplares}}}\}$: Conjunto dos melhores indivíduos da população. $N \times M$: Dimensões do JSSP tratado.
Saída	$\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_N\}$: Vetores de frequência

PARA $i \coloneqq 1$ **ATÉ** N **FAÇA**
  $\vec{v}_i \coloneqq \underbrace{(0,0,\dots 0)}_{N \cdot M \text{ coordenadas}} \in \mathbb{R}^{N \cdot M}$; // Inicialização dos vetores de frequência.
FIM_PARA
PARA $i \coloneqq 1$ **ATÉ** N **FAÇA** // É confeccionado um vetor de frequências por *job*.
  PARA $j =: 1$ **ATÉ** $N \cdot M$ **FAÇA** // Todos os genes dos cromossomos são considerados.
    PARA $k \coloneqq 1$ **ATÉ** $N_{\text{Exemplares}}$ **FAÇA** // Todos os indivíduos exemplares são
                                                // considerados.
      SE $c_{k,j} == i$ **ENTÃO** // O j-ésimo gene de c_k representa o *job* J_i? Caso positivo,
        $v_{ij} \coloneqq v_{ij} + 1$; // acrescenta-se 1 na j-ésima coordenada de $\vec{v_i}$.
      FIM_SE
    FIM_PARA
  FIM_PARA
FIM_PARA

Fonte: Elaborado pela autora.

Na Figura 3.3, é apresentado um exemplo do cálculo de $\vec{v}_i$ ao se considerar 4 indivíduos exemplares em um JSSP 3 × 2.

Figura 3.3. Exemplo de cálculo de frequências dos *jobs* em cada coordenada dos cromossomos tomados como "Exemplares".



Fonte: Elaborada pela autora.

Uma vez confeccionado o vetor $\vec{v}_i$ para todo *job* de índice i, define-se como **indivíduo modelo** um cromossomo cujas coordenadas sejam determinadas pelo índice do *job* que apresente maior frequência na mesma coordenada. Isto é, cada gene (coordenada) do indivíduo modelo vai ser definido como sendo o índice do *job* que tenha se manifestado mais frequentemente nesta respectiva coordenada nos indivíduos tomados como "exemplares".

Também é possível estabelecer de acordo com os vetores de frequência $\vec{v}_i$ uma ordem de relevância genética. Ou seja, é possível definir quais genes do indivíduo modelo são mais adequados para serem transferidos no processo de transgenia. Tal relevância também é definida de acordo com a frequência que os *jobs* apresentam em cada coordenada dos indivíduos exemplares. Sendo que os genes que apresentam o mesmo *job* em muitos bons indivíduos podem categorizar uma "tendência" que acarreta em bons valores de *fitness*. Então tais genes

devem ser considerados como sendo relevantes, uma vez que descrevem uma característica positiva nos indivíduos.

Especificamente, o indivíduo modelo e sua relevância genética são confeccionados de acordo com o seguinte procedimento:

1. Sejam c o indivíduo modelo e w um vetor que representa uma pontuação para cada coordenada do indivíduo modelo, inicialmente nulos. Nos itens a seguir, as coordenadas de c e de w são confeccionadas.

2. Define-se $I_1 \coloneqq \arg \max_{i\in\{1,2,\dots,N\}} \{\vec{v}_{i,1}\}_{i=1}^{N}$.[iv]

   Isto é, I_1 é o índice do *job* que apresenta maior frequência na primeira coordenada dos indivíduos exemplares. Sendo assim, a primeira coordenada do indivíduo modelo é definida como sendo I_1. Matematicamente,

   $$c_1 \coloneqq I_1.$$

   Além disso, associa-se à primeira coordenada de c uma pontuação w_1 definida como sendo a máxima frequência apresentada na primeira coordenada dos indivíduos exemplares. Isto é,

   $$w_1 \coloneqq \max_{i \in \{1,2,\dots,N\}} \{\vec{v}_{i,1}\}_{i=1}^{N} = \vec{v}_{I_1,1}.$$

3. Atribui-se o valor 2 a j.
4. Define-se $I_j \coloneqq \arg \max_{i \in \{1,2,\dots,N\}} \{\vec{v}_{i,j}\}_{i=1}^{N}$ [v], isto é, I_j é o índice do *job* mais frequente na coordenada j nos indivíduos exemplares. Entretanto, para garantir a factibilidade do indivíduo modelo, é necessário estabelecer mais duas restrições:

   4.1. Caso o *job* I_j não esteja em M coordenadas de c, então define-se como I_j a $j-$ésima coordenada do indivíduo modelo. Isto é,

   $$c_j \coloneqq I_j.$$

[iv] $\{\vec{v}_{i,1}\}_{i=1}^{N} = \{\vec{v}_{1,1}, \vec{v}_{2,1}, \dots, \vec{v}_{N,1}\}$.
[v] $\{\vec{v}_{i,j}\}_{i=1}^{N} = \{\vec{v}_{1,j}, \vec{v}_{2,j}, \dots, \vec{v}_{N,j}\}$

Neste caso, associa-se à j −ésima coordenada do indivíduo modelo sua respectiva pontuação como sendo o máximo valor possível apresentado na j −ésima coordenada dos indivíduos exemplares. Ou seja,

$$w_j \coloneqq \max_{i \in \{1,2,\dots,N\}} \{\vec{v}_{i,j}\}_{i=1}^{N} = \vec{v}_{I_j,j}.$$

4.2. Caso contrário, para garantir a factibilidade de c, desconsidera-se as frequências do *job* de índice I_j, uma vez que o mesmo já se encontra disposto em M coordenadas de c e, portanto, não pode mais ocupar nenhuma coordenada de c. Para tal, deve-se anular seu respectivo vetor de frequências, ou seja,

$$\vec{v}_{I_j} \coloneqq \vec{0}.$$

Neste caso, para continuar a preencher as coordenadas do indivíduo modelo, deve-se retornar ao início do item 3.

5. Se $j \neq N \cdot M$ faça $j \coloneqq j + 1$ e retorne ao item 4. Caso contrário, o procedimento é finalizado, obtendo-se o indivíduo modelo e as pontuações de relevância para suas coordenadas: (c, w).

Repare que não é necessário projetar o indivíduo modelo no espaço factível do problema abordado, uma vez que por sua própria construção, em especial ao item 4 do procedimento anterior, o mesmo já é factível. No Algoritmo 3.5, são apresentados detalhes do método de cálculo de frequências e relevância genética propostos.

Algoritmo 3.5. Construção do indivíduo modelo.

Entrada	$\{\vec{v}_1, \vec{v}_2, \dots, \vec{v}_N\}$: Vetores de frequência. $N \times M$: Dimensões do JSSP tratado.
Saída	c: Indivíduo modelo. w: Vetores de pesos e relevâncias genéticas.

$c \coloneqq \underbrace{(0,0,\dots 0)}_{N \cdot M \text{ coordenadas}} \in \mathbb{R}^{N \cdot M}$; // Inicialização do indivíduo modelo como sendo o vetor nulo.

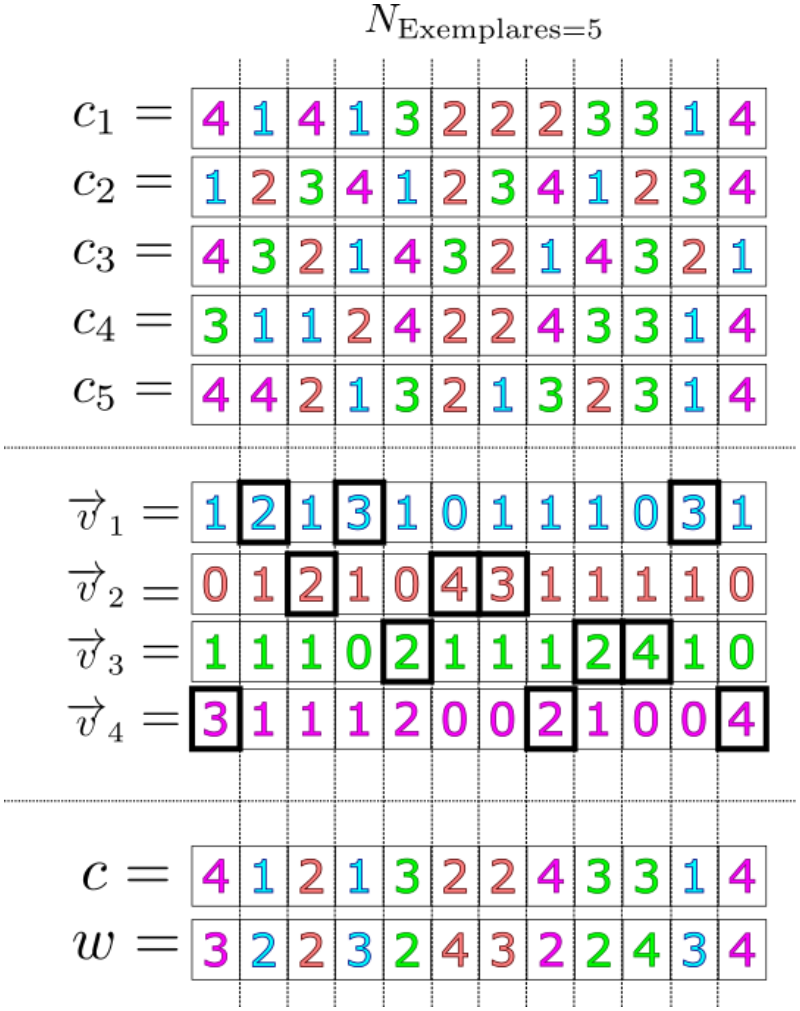
$I_1 \coloneqq \arg\max_{i \leq i \leq N} \{\vec{v}_{i,1}\}$; // Índice do *Job* mais frequente na primeira coordenada dos // indivíduos exemplares.

$w_1 \coloneqq \vec{v}_{I_1,1}$; // Número de vezes em que o *job* de índice 1 ocupou a primeira
// coordenada dos indivíduos exemplares.
$c_1 \coloneqq I_1$; // A primeira coordenada do indivíduo modelo é dada por I_1.
$j = 2$; // Para as demais coordenadas de c será conduzido um processo semelhante, com a
// ressalva de que deverá haver garantia de factibilidade.
ENQUANTO $j \leq N \cdot M$ **FAÇA** // É confeccionado um vetor de frequências por *job*.
$I_j \coloneqq \arg \max_{i \leq i \leq N} \{\vec{v}_{i,j}\}_{i=1}^{N}$; // Calcula-se o índice do *job* mais recorrente na j-ésima
// coordenada dos cromossomos exemplares.
contJob $\coloneqq 0$; // Para garantir factibilidade, é preciso garantir que o índice de cada *job*
// esteja em exatas M coordenadas de c.
PARA $k \coloneqq 1$ **ATÉ** $N \cdot M$ **FAÇA**
SE $c_k == I_j$ **ENTÃO**
contJob $\coloneqq$ contJob $+ 1$; // Contabiliza-se o número de coordenadas de c nas
// quais o *job* de índice I_j está presente.
FIM_SE
FIM_PARA
SE contJob $< M$ **ENTÃO** // No caso do *job* I_j ainda não estar presente em M
// coordenadas de c, então ele pode assumir sua j-ésima
// coordenada.
$w_j \coloneqq \vec{v}_{I_j,j}$;
$c_j \coloneqq I_j$;
$j \coloneqq j + 1$;
SENÃO // Caso contrário, o próximo *job* mais frequente nos cromossomos
// exemplares deve ser avaliado.
$\vec{v}_{I_j} \coloneqq (0,0, \ldots 0) \in \mathbb{R}^{N \cdot M}$; // Para isso, o vetor de frequências associado ao *job* de
// índice I_j deve ser anulado e, consequentemente,
// desconsiderado nas próximas iterações.
FIM_SE
FIM_ENQUANTO

Fonte: Elaborado pela autora.

Na Figura 3.4, é apresentado um exemplo do cálculo do indivíduo modelo e da relevância genética em um JSSP 4×3, tomando como indivíduos exemplares os 5 melhores indivíduos disponíveis na população.

Figura 3.4. Exemplo do cômputo do indivíduo modelo (*c*) e da relevância genética (*w*).



Fonte: Elaborada pela autora.

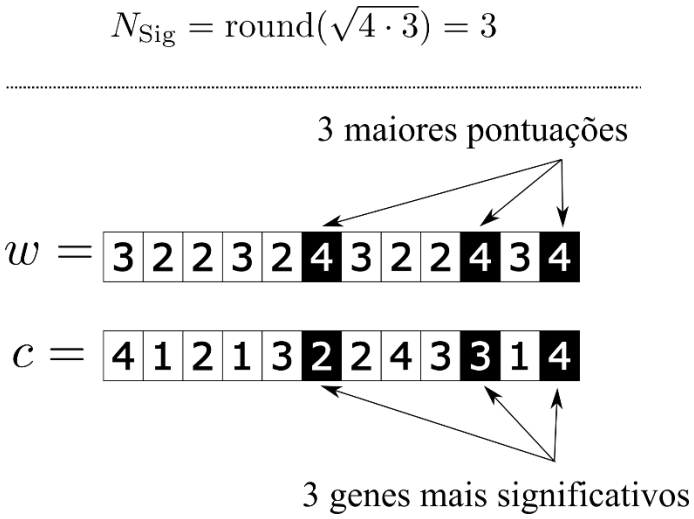
3.2.4.2 Uso do indivíduo modelo em melhoramento genético

Uma vez calculados o indivíduo modelo e a relevância genética de cada um de seus genes, então é proposto que seus genes mais relevantes sejam transferidos para os piores indivíduos da população, simulando, assim, um mecanismo de transgenia.

Especificamente, toma-se $P_{\text{piores}} = \left\{x_1, x_2, \dots, x_{N_{\text{piores}}}\right\}$ como sendo o conjunto dos N_{piores} piores indivíduos dispostos em uma população. Na sequência, transfere-se para todos os indivíduos de P_{piores} os N_{Sig} genes mais significativos, ou com maior relevância, do indivíduo modelo. Finalmente, os indivíduos gerados neste procedimento são projetados no conjunto factível do problema abordado.

Viana, Morandin Junior e Contreras (2020d) determinam empiricamente para seu operador de transgenia que devem ser transferidos aproximadamente $\sqrt{N \cdot M}$[vi] genes para que o processo seja vantajoso e não gere convergência precoce na população, sendo $N \times M$ as dimensões do JSSP abordado. Desta forma, levando em conta que o processo de transgenia é um processo de melhoramento genético, neste trabalho define-se N_{Sig} como sendo $\text{round}\left(\sqrt{N \cdot M}\right)$. Na Figura 3.5, é apresentado um exemplo da determinação dos genes mais significativos de um indivíduo modelo c dada as pontuações w de seus genes ao abordar um JSSP de dimensões 4×3.

Figura 3.5. Determinação dos genes mais significativos de um indivíduo modelo.



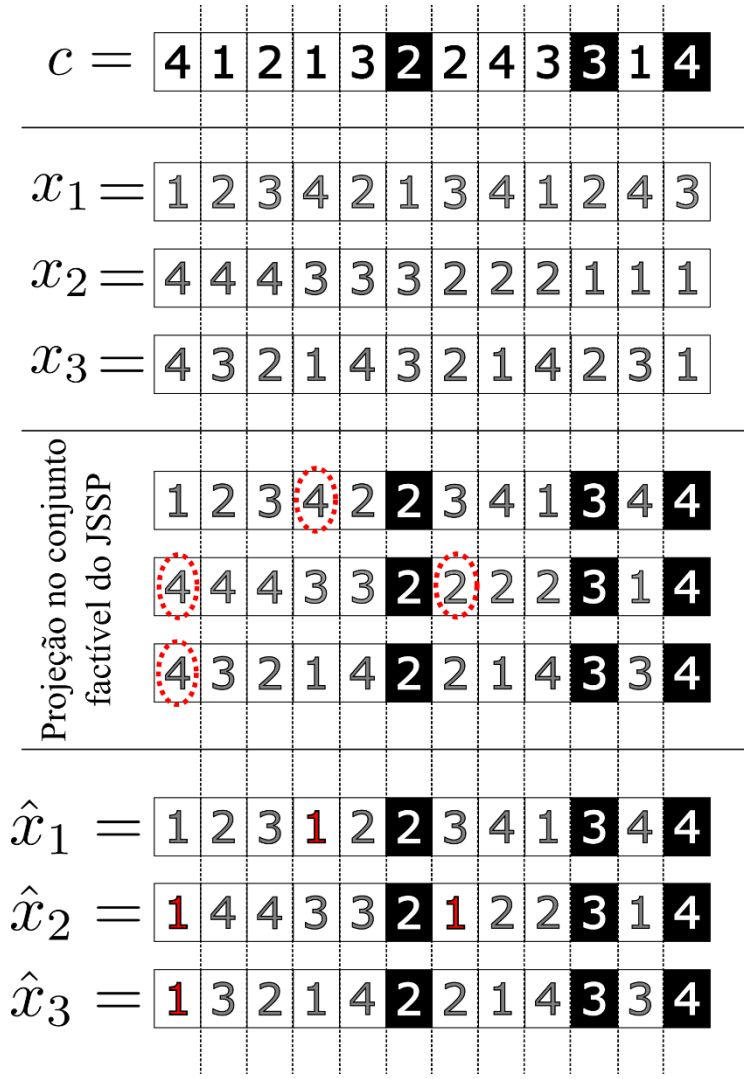
Fonte: Elaborada pela autora.

Supondo $N_{\text{piores}} = 3$ e $P_{\text{piores}} = \{x_1, x_2, x_3\}$ como sendo o conjunto dos 3 piores indivíduos de uma população, é apresentado na Figura 3.6 o processo de melhoramento

[vi] Este valor é definido de acordo com o número de genes (coordenadas) dos cromossomos que modelam as soluções do problema, o qual no caso é $N \cdot M$.

genético que transfere os N_{Sig} melhores genes do indivíduo modelo c da Figura 3.4 para todos os indivíduos do conjunto P_{piores}. Ressaltando-se que o processo de projeção utilizado neste procedimento é semelhante ao procedimento utilizado no operador de cruzamento PMX, discutido na Seção 3.2.3.1, no caso, a projeção de Hamming (WEGNER, 1960).

Figura 3.6. Processo de melhoramento genético proposto.



Fonte: Elaborada pela autora.

Para utilizar o operador de melhoramento genético em um método do tipo GA, é preciso que o mesmo seja aplicado em N_{piores} piores indivíduos da população. Além disso, é possível que seja necessário conduzir um procedimento de correção genética a fim de projetar os indivíduos geneticamente modificados para o espaço factível do problema. Esta projeção deve

ser conduzida de acordo com a projeção de Hamming (WEGNER, 1960) já discutida na Seção 2.2.4.3.1. No Algoritmo 3.6, este processo é detalhado.

Algoritmo 3.6. Condução do melhoramento genético nos piores indivíduos de uma população.

Entrada	c: Indivíduo modelo. w: Vetor de pesos para relevância genética. $P_{\text{piores}} = \{x_1, x_2, \dots, x_{N_{\text{piores}}}\}$: Conjunto dos N_{piores} piores indivíduos da população. $N \times M$: Dimensões do JSSP tratado.
Saída	$P_{\text{aprimorado}} = \left\{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{N_{\text{piores}}}\right\}$: Cromossomos aprimorados.

$N_{\text{genes}} \coloneqq \text{round}\left(\sqrt{N \cdot M}\right)$; // Número de genes a serem transferidos.

$K \coloneqq \{\ \}$; // Conjunto das coordenadas dos genes mais relevantes.

PARA $i \coloneqq 1$ **ATÉ** N_{genes}**FAÇA**

$K_i \coloneqq \arg\max_k\{w_k\}$; // i-ésimo gene mais relevante.

$K \coloneqq K \cup \{K_i\}$; // Atualiza-se o conjunto K.

$w_{K_i} \coloneqq 0$; // Desconsidera-se o gene em questão nas próximas iterações.

FIM_PARA

PARA $i \coloneqq 1$ **ATÉ** N_{genes}**FAÇA**

PARA $k =: 1$ **ATÉ** N_{piores} **FAÇA**

$x_{k,i} \coloneqq c_{K_i}$; // Transfere-se o gene da coordenada K_i de c para a i-ésima coordenada // de x_k.

FIM_PARA

FIM_PARA

$P_{\text{aprimorado}} := \{\ \}$;

PARA $k =: 1$ **ATÉ** N_{piores} **FAÇA**

$\hat{x}_k \coloneqq \text{proj}_{\text{Hamming}}(x_k)$; // Corrige-se a possível infactibilidade do processo com a // projeção de Hamming.

$P_{\text{aprimorado}} := P_{\text{aprimorado}} \cup \{\hat{x}_k\}$;

FIM_PARA

Fonte: Elaborado pela autora.

3.2.4.3 O operador de melhoramento genético proposto

A estratégia de melhoramento genético proposta foi desenvolvida a fim de ser a mais versátil possível no sentido de que seja possível anexá-la em qualquer método do tipo GA. Assim, o operador proposto (GIFA) deve ser utilizado depois da execução dos operadores originais do algoritmo considerado a fim de guiar as soluções que não foram capazes de se destacar utilizando tais operadores. Além disso, o uso do operador de melhoramento genético proposto deve ser conduzido juntamente a alguma estratégia de manutenção de diversidade genética, a fim de não corroborar para a estagnação genética prematura da população. Uma das rotinas mais utilizadas na literatura para este fim são as substituições de indivíduos da população por novos indivíduos gerados a partir da distribuição de Lévy. De forma intuitiva (YANG; DEB, 2013), esta distribuição está associada a caminhadas aleatórias cujos passos são definidos por distribuições exponenciais, isto é, $\text{Lévy}(s) \sim |s|^{-1-\beta}$, com $\beta \in (0,2]$. Matematicamente (YANG, 2010b), um número aleatório gerado por uma distribuição de Lévy, obedece a seguinte distribuição:

$$\text{Lévy}(s,\gamma,\mu) = \begin{cases} \sqrt{\frac{\gamma}{2\pi}}\, e^{-\frac{\gamma}{2(s-\mu)}} \frac{1}{(s-\mu)^{\frac{3}{2}}}, & \text{caso } 0 < \mu < s < \infty, \\ 0, & \text{caso } s \leq 0, \end{cases} \tag{3.5}$$

na qual, μ é o passo mínimo da caminhada aleatória e γ é um fator de escala.

Neste operador, para gerar novos indivíduos, é utilizada uma função $\hat{f}_{\text{shuffle}}$, definida tal como f_{shuffle} com a ressalva de gerar números aleatórios com uma distribuição de Lévy. Especificamente, é necessário avaliar os indivíduos que devem receber melhoramento genético antes e depois do procedimento e, aqueles que não forem capazes de apresentar melhora devem ser substituídos por novos indivíduos gerados com a $\hat{f}_{\text{shuffle}}$. Em detalhes, os passos que definem o operador de melhoramento genético são apresentados no Algoritmo 3.7 a seguir.

Algoritmo 3.7. Operador de melhoramento genético proposto (GIFA).

Entrada	c: Indivíduo modelo. w: Vetor de pesos para relevância genética.

	$P_{\text{piores}} = \{x_1, x_2, \dots, x_{N_{\text{piores}}}\}$: Conjunto dos N_{piores} piores indivíduos da população. $N \times M$: Dimensões do JSSP tratado. F: Função *fitness* do problema.
Saída	$P_{\text{aprimorado}} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_{N_{\text{piores}}}\}$: Cromossomos aprimorados.

$P_{\text{aprimorado}} \coloneqq$ Algoritmo 3.6$(c, w, P_{\text{piores}}, N, M)$; // Obtém-se o conjunto de indivíduos
// aprimorados pelo processo de melhoramento
// genético por meio do Algoritmo 3.6.

PARA $i \coloneqq 1$ **ATÉ** N_{piores} **FAÇA** // Todos os indivíduos aprimorados devem ser avaliados
// para assegurar se houve melhoramento no *fitness*.

$F_{\text{sem melhoramento}} \coloneqq F(x_i)$; // x_i é o indivíduo sem aprimoramento.

$F_{\text{com melhoramento}} \coloneqq F(\hat{x}_i)$; // $\hat{x}_i \in P_{\text{aprimorado}}$ é a versão geneticamente aprimorada de x_i.

SE $F_{\text{sem melhoramento}} \leq F_{\text{com melhoramento}}$ **ENTÃO** // No caso de não haver melhora, o
// indivíduo em questão deve ser
// substituído por um novo indivíduo
// gerado a partir da permutação
// aleatória, com esta sendo definida
// pela distribuição de Lévy, de uma
// solução factível.

$P_{\text{aprimorado}} \coloneqq P_{\text{aprimorado}} - \{\hat{x}_i\}$; // Remove-se o indivíduo de $P_{\text{aprimorado}}$.

$\hat{x}_i \coloneqq \hat{f}_{\text{shuffle}}((1, \dots, 1, 2, \dots, 2, \dots, N, \dots, N))$; // Gera-se o novo indivíduo
// utilizando-se a distribuição de Lévy.

$P_{\text{aprimorado}} \coloneqq P_{\text{aprimorado}} \cup \{\hat{x}_i\}$; // O indivíduo gerado é alocado em $P_{\text{aprimorado}}$.

FIM_SE

FIM_PARA

Fonte: Elaborado pela autora.

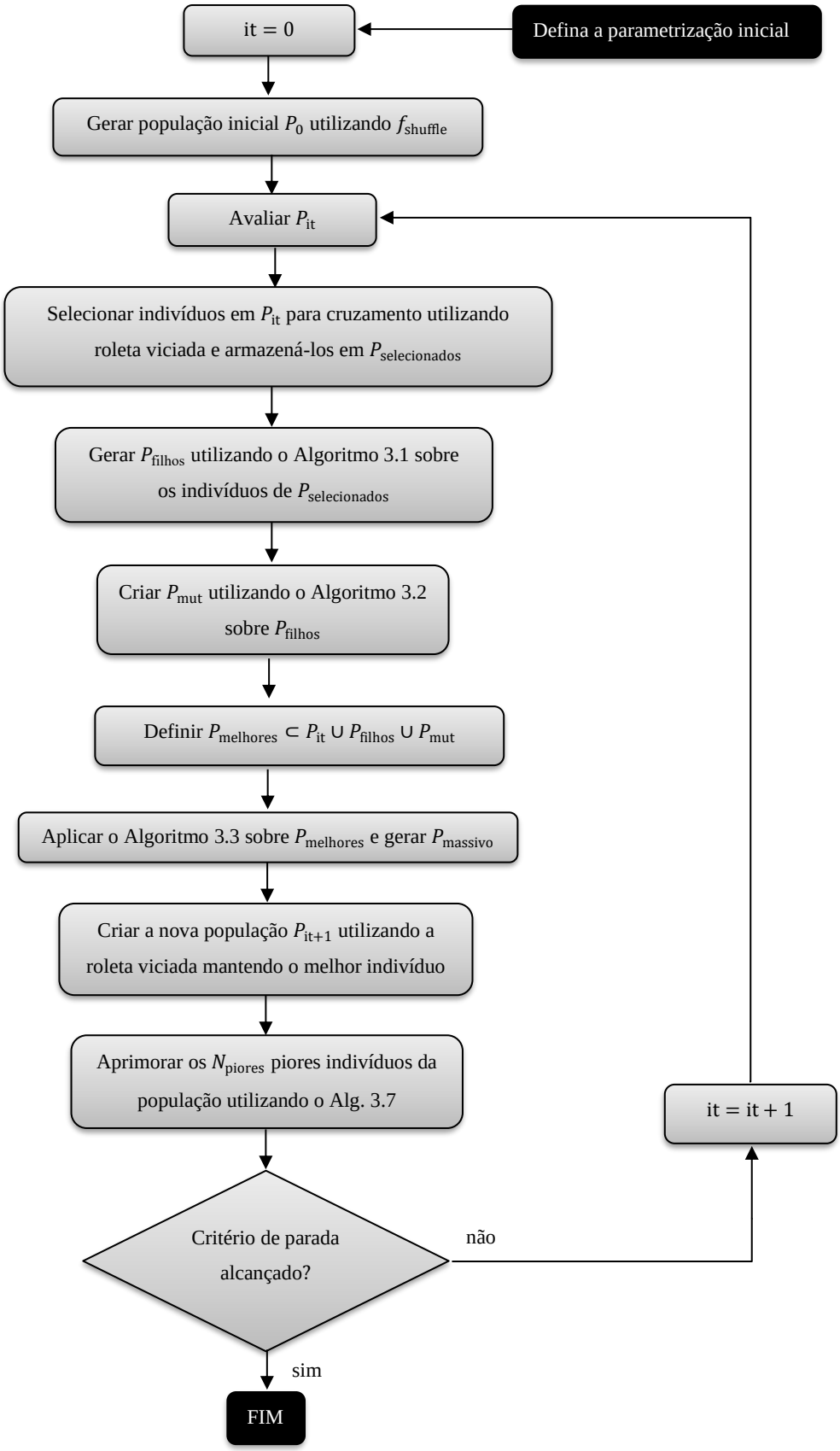
3.3 Esquema de uso para os operadores propostos: estrutura do algoritmo

Todos os operadores foram desenvolvidos para serem o mais versátil possível no sentido de que possam ser anexados em todo método do tipo GA. A utilização de todos os operadores em conjunto segue uma metodologia semelhante à utilizada por Asadzadeh (2015), que consiste nas seguintes etapas centrais:

- Etapa 1: Iniciar a configuração do método, que inclui a escolha da definição padrão dos parâmetros e a determinação das funções de *crossover*, mutação e perturbação;
- Etapa 2: Gerar a população inicial usando o Algoritmo 2.2 e a função f_{shuffle} definida na Equação (2.4)
- Etapa 3: Selecionar os indivíduos e executar o operador de multi-*crossover* do Algoritmo 3.1;
- Etapa 4: Executar o operador de busca local e mutação do Algoritmo 3.2 em todos os descendentes gerados na etapa anterior;
- Etapa 5: Executar o operador de busca local de caráter massivo do Algoritmo 3.3 para procurar melhores soluções em torno de um grupo de bons indivíduos;
- Etapa 7: Gerar uma nova população usando a estratégia da roleta viciada, retendo o melhor indivíduo;
- Etapa 6: Executar o operador de Melhoramento Genético (GIFA) do Algoritmo 3.7,
- Etapa 8: Finalizar a execução do método se o critério de parada for alcançado ou voltar para a Etapa 3 no caso contrário.

O método deve ser utilizado para resolver uma instância JSSP por vez. A Figura 3.7 apresenta o fluxograma contendo todos os detalhes sobre as etapas do método proposto, que é a metaheurística proposta para aplicação em instâncias JSSP.

Figura 3.7. Fluxograma do método proposto.



Fonte: Elaborada pela autora.

Observando o fluxograma do método proposto, é possível perceber algumas diferenças entre a técnica proposta e as técnicas que serviram de inspiração para sua confecção. É possível constatar que o uso de estratégias multi-*crossover* com várias funções de crossover e o operador de melhoramento genético proposto são únicos com respeito à metodologia proposta. Além disso, apenas o método proposto utiliza mais de uma função de perturbação no operador de busca local massiva e as aplica a um conjunto de indivíduos. Em detalhes, todos os métodos semelhantes ao GA influenciaram a modelagem da proposta, no entanto, também existem diferenças significativas, conforme destacado na sequência:

- **GA básico**:
    - Inspiração: A sequência de operações.
    - Diferenças: O método proposto utiliza mais operadores, como o operador de busca local massiva e o operador de melhoramento genético.
- **GSA** (WATANABE; IDA; GEN, 2005):
    - Inspiração: Aplica apenas uma função de cruzamento mais de uma vez ou até que o procedimento encontre um indivíduo melhor do que o pior indivíduo da população. Aplicações sucessivas de apenas uma função de mutação.
    - Diferenças: O método proposto aplica várias funções de cruzamento mais de uma vez ou até que o método encontre um indivíduo melhor do que um dos pais. Aplicações sucessivas de várias funções de mutação.
- **LSGA** (OMBUKI; VENTRESCA, 2004):
    - Inspiração: O operador de mutação é composto por dois procedimentos possíveis: o primeiro realizando uma busca local com aplicações sucessivas de uma variante da função de mutação f_{swap}; e o segundo sendo uma mutação simples na forma de uma única perturbação.
    - Diferenças: O método proposto também possui esses dois procedimentos possíveis em seu operador de mutação. No entanto, o método dispõe de um grupo de funções de mutação que utiliza durante suas iterações nesses procedimentos.
- **aLSGA** (ASADZADEH, 2015):
    - Inspiração: Possui operadores de busca local elite e busca local no operador mutação. O operador de mutação realiza aplicações sucessivas de um grupo de funções de mutação em um cromossomo. O operador de

busca local elite executa perturbações sucessivas com f_{swap} considerando todas as combinações possíveis entre as coordenadas do melhor cromossomo da população.

- Diferenças: O operador de mutação do método proposto é dividido em duas sub-rotinas possíveis: a primeira com o comportamento de busca local aplicando perturbações sucessivas com um grupo de funções de mutação, como o aLSGA; e o segundo sendo formado a partir de uma mutação simples de forma que apenas uma perturbação seja aplicada ao cromossomo. Ao contrário do operador de busca local elite do aLSGA, o operador de busca local massivo do método proposto utiliza um conjunto de funções de perturbação, e não apenas a função f_{swap}, em um conjunto de indivíduos e não apenas no melhor indivíduo.

Em resumo, na Tabela 3.1 é apresentado como cada uma das técnicas do tipo GA utiliza as estratégias mencionadas em comparação com o método proposto.

Tabela 3.1: Comparação entre o uso de diferentes estratégias por cada método. "Sim" significa que o método utiliza a estratégia correspondente e "Não" significa que não utiliza.

Estratégia	Método				
	GA	GSA	LSGA	aLSGA	Proposta
Conjunto de funções de cruzamento (multi-*crossover*)	Não	Não	Não	Não	Sim
Adaptação da área de busca no operador de cruzamento	Não	Sim	Não	Não	Sim
Operador de mutação com busca local	Não	Sim	Sim	Sim	Sim
Conjunto de funções de mutação	Não	Não	Não	Sim	Sim
Mutação simples (apenas uma aplicação)	Sim	Não	Sim	Não	Sim
Busca local massiva	Não	Não	Não	Sim	Sim
Conjunto de funções de perturbação	Não	Não	Não	Não	Sim
Operador de Melhoramento Genético	Não	Não	Não	Não	Sim

Fonte: Elaborada pela autora.

3.4 Considerações finais

Neste capítulo, foram apresentados todos os componentes do material proposto. No caso, foram fundamentados os aperfeiçoamentos e as generalizações das estratégias de busca local inicialmente idealizadas pelos principais métodos do tipo GA dedicados a solucionar JSSP presentes na literatura especializada. Também foi apresentada a estratégia de melhoramento genético baseada na análise de frequência de genes pertencentes a indivíduos bem adaptados na população. Finalmente, as inspirações e diferenças do material proposto com respeito às técnicas presentes na literatura especializada foram destacadas. Uma proposta de uso na forma de um *framework* do método foi detalhada.

No próximo capítulo, são definidos os ambientes de testes em que o ferramental proposto foi avaliado e os trabalhos disponíveis no estado da arte que foram considerados a fins de comparações para inferir a qualidade do funcionamento dos métodos propostos. Os resultados obtidos são comparados a estas técnicas em três estudos de caso diferentes, que demonstram a competitividade do material.

Capítulo 4

APLICAÇÃO DA ABORDAGEM E RESULTADOS

4.1 Considerações iniciais

Neste capítulo, são apresentados os resultados alcançados e que fazem parte do projeto de tese. Tais resultados corroboram para a validação das hipóteses aqui propostas. Especificamente, o material proposto é avaliado diante de três estudos de casos: o primeiro consistindo na análise dos operadores de busca local propostos e como cada um deles influenciam na capacidade de busca do método; o segundo consistindo na análise do operador de melhoramento genético quando anexado a métodos do tipo GA presentes na literatura; e o terceiro consistindo na comparação da capacidade do método proposto em obter as melhores soluções conhecidas com respeito a diferentes metaheurísticas do estado da arte que se propõem a solucionar instâncias de JSSP.

Em todos os testes conduzidos, subconjuntos das mesmas instâncias foram considerados. Em detalhes, utiliza-se cenários das instâncias FT (FISHER; THOMPSON, 1963), LA (LAWRENCE, 1984), ORB (APPLEGATE; COOK, 1991) e ABZ (ADAMS; BALAS; ZAWACK, 1988). De forma que todas essas instâncias representam cenários de JSSP contendo dados dos *jobs* que devem ser processados: seus roteiros contendo a ordem de processamento sobre as máquinas do cenário e a quantidade de tempo em que cada *job* leva para ser processado em cada máquina. Ressaltando-se que essas bases compõem o *benchmark* atual para o tema de pesquisa em questão, apresentando cenários de testes de complexidade variada. Entretanto, são cenários simulados e que, consequentemente, podem não representar, necessariamente, ambientes reais de produção.

O material proposto foi codificado na linguagem Matlab e os testes foram realizados em um computador com CPU Intel (R) Core i7 5500U de 2,4 GHz e 16 GB de RAM. Ressalta-se que foram utilizadas apenas as funções padrões da IDE do Matlab para conduzir os

experimentos computacionais e, consequentemente, não foram utilizados nenhum pacote de *software* de otimização pré-existente.

4.2 Estudo de caso I: Análise dos operadores de busca local

Neste primeiro estudo de caso, o principal objetivo consiste em avaliar o funcionamento dos operadores que, de certa forma, possuem inspiração em métodos existentes na literatura. No caso, estes se tratam dos operadores de busca local anexados no cruzamento e na mutação e aqueles dedicados a busca local massiva. Neste estudo de caso, é comprovado um dos principais avanços do trabalho: a melhoria de desempenho dos operadores de busca local por meio das generalizações propostas. Especificamente, demonstra-se a eficácia do método proposto por meio de duas diferentes situações: a primeira dedicada a analisar os operadores do método isoladamente para avaliar a influência de cada um deles durante a solução de instâncias JSSP; e a segunda dedicada a comparar o método proposto com outras técnicas semelhantes ao GA ao tratar o JSSP.

4.2.1 Situação I: Análise de operadores isolados

Na primeira situação, foram avaliadas diferentes configurações para investigar como cada um dos operadores propostos neste trabalho atuam no método. Especificamente, o objetivo foi observar quais operadores influenciam mais significativamente o método proposto. Para isso, foram avaliadas diferentes configurações da técnica proposta em três instâncias LA de Lawrence (1984). Estas configurações foram estipuladas para analisar a influência de cada um dos operadores separadamente. Ou seja, foi definido uma configuração de um GA básico, outra de um GA básico com o operador multi-*crossover*, outra de um GA básico com o operador de busca local proposto, e assim por diante. Para isso, foi investigado o funcionamento das seguintes configurações:

- **GA**: GA básico;
- **GA + mX**: GA básico com operador multi-*crossover* proposto na Seção 3.2.1;
- **GA + LS***: GA básico com operador de busca local proposto na Seção 3.2.2 utilizando $\Delta_{\text{mut}} = 1$ e $\epsilon_{\text{LS}} = 1$, isto é, executando busca local em todos os descendentes que são gerados no operador de cruzamento;

- **GA + LS****: GA básico com operador de busca local proposto na Seção 3.2.2 utilizando $\Delta_{\text{mut}} = 1$ e ϵ_{LS} = 0.8, isto é, executando busca local em 80% dos descendentes que são gerados no operador de cruzamento;
- **GA+ELSA**: GA básico com mutação simples (apenas um uso de f_{swap}) e utilizando o operador de busca local massivo proposto com $\mathcal{F}_{\text{pert}} = \{f_{\text{swap}}\}$ (busca local elite de Asadzadeh (2015));
- **GA+MLS**: GA básico com mutação simples (apenas um uso de f_{swap}) e utilizando o operador de busca local massivo proposto com $\mathcal{F}_{\text{pert}} = \{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$;
- **GA+mX+LS****: Utilizando o operador multi-*crossover* na configuração anterior GA + LS **;
- **mXLSGA**: Método com todos os operadores de busca local propostos com exceção do operador de melhoramento genético.

Os detalhes da configuração de cada um dos algoritmos avaliados são apresentados na Tabela 4.1.

Tabela 4.1. Configuração para a situação I do estudo de caso I. O c_{best} é o melhor indivíduo na população e $c_{\text{best},1}$ e $c_{\text{best},2}$ são os dois melhores indivíduos diferentes na população. (continua)

	GA	**GA+mX**	**GA+LS***	**GA+LS***	**GA+ELSA**
Número de cromossomos	100	100	100	100	100
Número máximo de iterações	100	100	100	100	100
Taxa de cruzamento ($\Delta_{\times}$)	0,95	0,95	0,95	0,95	0,95
Taxa de mutação (Δ_{mut})	0,05	0,05	1	1	0,05
ϵ_{LS}	0	0	1	0,8	0
R_c	1	10	1	1	1
R_m	1	1	$N \cdot M$	$N \cdot M$	1
$\mathcal{F}_{\times}$	{PMX}	{PMX,OX2}	{PMX}	{PMX}	{PMX}
$\mathcal{F}_{\text{mut}}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$	$\{f_{\text{swap}}\}$
$\mathcal{F}_{\text{pert}}$	{ }	{ }	{ }	{ }	$\{f_{\text{swap}}\}$
P_{melhores}	{ }	{ }	{ }	{ }	$\{c_{\text{best}}\}$

Fonte: Elaborada pela autora.

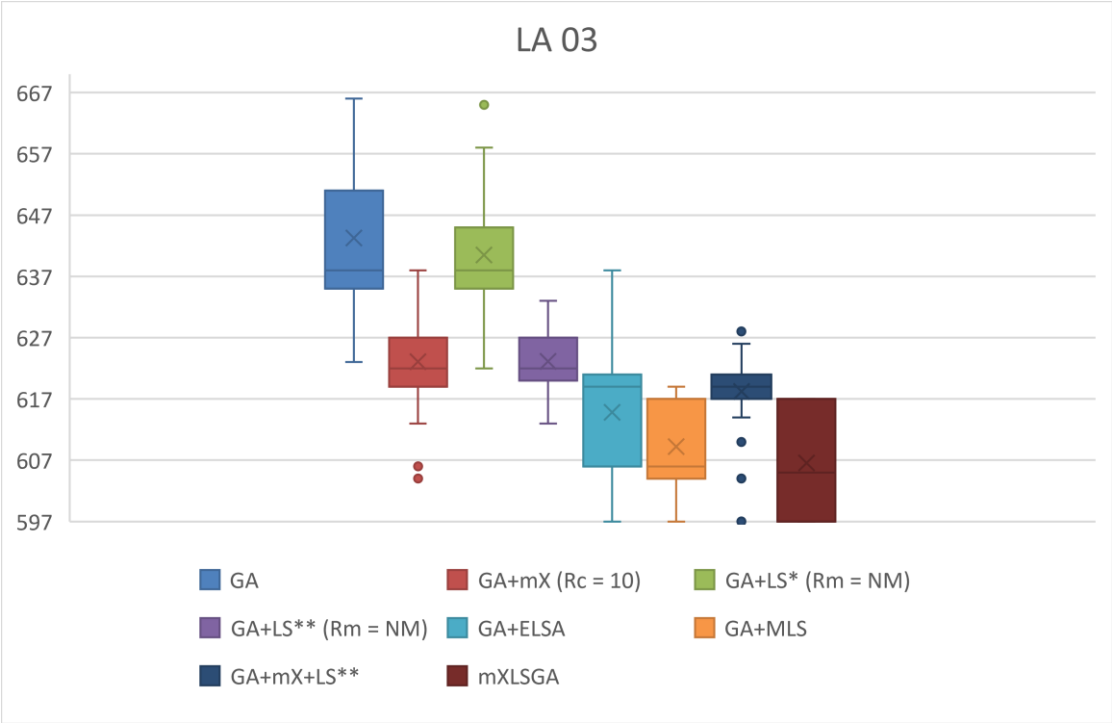
Tabela 4.1. Configuração para a situação I do estudo de caso I. O c_{best} é o melhor indivíduo na população e $c_{\text{best,1}}$ e $c_{\text{best,2}}$ são os dois melhores indivíduos diferentes na população. (conclusão)

	GA+MLS	**GA+mX+LS****	**mXLSGA**
Número de cromossomos	100	100	100
Número máximo de iterações	100	100	100
Taxa de cruzamento ($\Delta_{\times}$)	0,95	0,95	0,95
Taxa de mutação (Δ_{mut})	0,05	1	1
ϵ_{LS}	0	0,8	0,8
R_c	1	10	10
R_m	1	$N \cdot M$	$N \cdot M$
$\mathcal{F}_{\times}$	{PMX}	{PMX,OX2}	{PMX,OX2}
$\mathcal{F}_{\text{mut}}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
$\mathcal{F}_{\text{pert}}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$	{ }	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
P_{melhores}	$\{c_{\text{best}}\}$	{ }	$\{c_{\text{best,1}}, c_{\text{best,2}}\}$

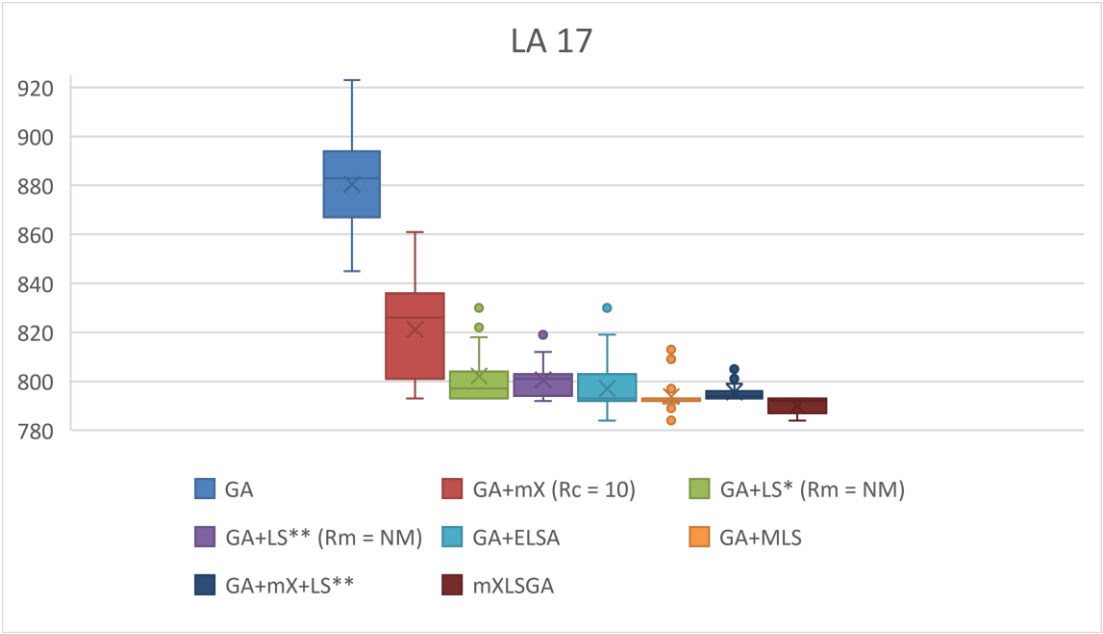
Fonte: Elaborada pela autora.

Cada uma das configurações apresentadas neste estudo de caso foi executada 35 vezes em três instâncias de LA de tamanhos diferentes: LA 03, com uma dimensão de 20×5, considerada de complexidade baixa; LA 17, de tamanho 10×10 e complexidade média; e LA 30, com um tamanho de 20×10 e de complexidade alta. Essas instâncias foram definidas de forma que pelo menos uma instância de cada um dos três principais grupos de dimensões fosse considerada para verificar a eficiência do algoritmo e seu comportamento em instâncias de dimensões variadas, das mais simples às mais complexas. Os valores de *fitness* para cada configuração foram utilizados para construir os gráficos de caixa (*boxplot*) na Figura 4.1.

Figura 4.1. Gráficos de caixa dos valores de *fitness* de 35 execuções das diferentes configurações do *framework* proposto. (continua)

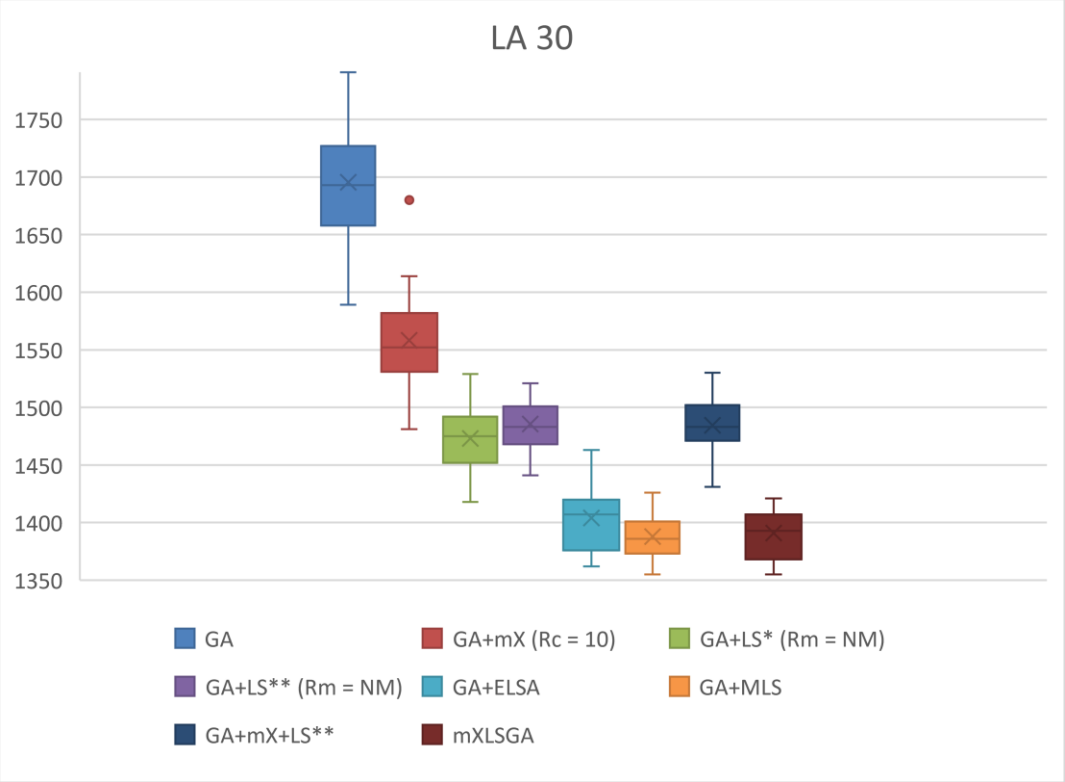


(a) LA 03: 20 × 5.



(b) LA 17: 10 × 10.

Figura 4.1. Gráficos de caixa dos valores de *fitness* de 35 execuções das diferentes configurações do *framework* proposto. (conclusão)



(c) LA 30: 20 × 10.

Fonte: Elaborada pela autora.

Analisando os gráficos de caixa apresentados na Figura 4.1, é possível observar que a adição de qualquer um dos operadores propostos no GA foi muito benéfica. No entanto, alguns operadores têm maior influência em certos casos. Por exemplo, o operador multi-*crossover* tem maior influência em bases menos complexas, sendo que na Figura 4.1 (a), a técnica GA+mX funciona tão bem ou até melhor do que as configurações GA + LS * e GA + LS **, enquanto que em bases mais complexas (Figura 4.1 (b) e (c)), a configuração GA + mX apresenta resultados estatisticamente inferiores aos resultados de GA + LS * e GA + LS **.

Em relação aos operadores de mutação que utilizam estratégias de busca local, é possível notar que conforme a complexidade da instância analisada aumenta, o desempenho do GA + LS * melhora em relação à configuração do GA + LS **. Isso porque, quanto maior a complexidade da instância, o uso de estratégias de busca local torna-se mais vantajoso do que a variabilidade genética garantida pelo operador com o uso de $\epsilon_{\text{LS}} \neq 1$. Entretanto, é possível notar na Figura 4.1 que em instâncias de menor complexidade, a configuração GA + LS ** apresenta resultados muito melhores e mais estáveis do que os apresentados por GA + LS *.

Portanto, é uma boa estratégia usar um valor entre 0.8 e 1 para ϵ_{LS} nos próximos estudos de caso.

É possível observar também que o operador de busca local massiva é o operador que apresenta melhor qualidade e maior estabilidade nos resultados. A adição de operadores de busca local massiva no GA básico resulta em melhores valores de *fitness* em comparação com a adição de dois operadores distintos, como é o caso da configuração GA + mX + LS **, que resulta em valores de *fitness* mais altos do que GA + ELSA e configurações do GA + MLS nos gráficos de caixa da Figura 4.1 (a) e (c). Além disso, o uso de mais de uma função de perturbação em F_{pert} resultou em GA + MLS apresentando resultados melhores e mais estáveis do que os resultados de GA + ELSA, que usa apenas a função f_{swap} em F_{pert} , em todos os casos avaliados. Assim, é possível concluir que o operador com maior influência é o operador de busca local massiva.

A utilização de dois operadores de busca local no GA + mX + LS ** tornou esta configuração mais estável, isto é, com menor desvio padrão, do que as configurações mais simples que utilizam apenas um operador, no caso as configurações GA + mX, GA + LS * e GA + LS **. Além disso, também em comparação com essas configurações, GA + mX + LS ** apresenta estatisticamente melhores valores de *fitness* nas instâncias LA03 (Figura 4.1 (a)) e LA17 (Figura 4.1 (b)).

Por fim, a configuração da metodologia proposta que corresponde ao uso conjunto de todos os operadores discutidos é o método mXLSGA, que apresenta os melhores resultados em todos os casos avaliados, com maior intensidade nas instâncias LA03 (Figura 4.1 (a)) e LA17 (Figura 4.1 (b)). Isso confirma que o uso de todos os operadores concomitantemente é a melhor configuração para a metodologia, pois é nesta configuração que os melhores resultados são obtidos.

Na Tabela 4.2, são apresentados os valores estatísticos obtidos pelas configurações de métodos. O melhor valor obtido (Melhor), o pior valor obtido (Pior), a média dos *fitness* (Média), desvio padrão (DP) e quantidade de ótimo alcançados. A configuração mXLSGA obtém os melhores valores para as medidas de melhor valor encontrado e de pior valor encontrado. Ademais, o mXLSGA obtém a menor média nas instâncias LA03 e LA17 e obtém a segunda menor média na LA30. Além disso, é a técnica que encontra a maior quantidade de valores ótimo em todos os casos.

Tabela 4.2. Informações estatísticas de cada configuração do método proposto para resolver as instâncias JSSP consideradas durante 35 execuções independentes.

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos
LA 03	GA	623	689	643,2857	13,44831	0
	GA+mX*	604	638	623,0571	7,951851	0
	GA+LS*	622	665	640,4857	9,635631	0
	GA+LS**	613	633	623,1143	4,638531	0
	GA+ELSA	597	638	614,8286	9,817409	2
	GA+MLS	597	619	609,2286	6,975106	3
	GA+mX+LS**	597	628	618,2286	5,594565	1
	mXLSGA	597	617	606,5429	8,430836	12
LA 17	GA	845	923	880,5143	17,78693	0
	GA+mX*	793	861	821,2	20,38136	0
	GA+LS*	793	830	802,1143	11,06869	0
	GA+LS**	792	822	800,5714	8,092428	0
	GA+ELSA	784	830	797,0857	10,27095	2
	GA+MLS	784	816	793,7714	7,380761	2
	GA+mX+LS**	793	805	795,6571	3,596917	0
	mXLSGA	784	793	789,9714	3,584983	6
LA 30	GA	1589	1791	1695,343	52,28651	0
	GA+mX*	1481	1680	1558,229	37,44493	0
	GA+LS*	1418	1529	1472,943	27,90524	0
	GA+LS**	1441	1521	1485,657	21,44753	0
	GA+ELSA	1362	1463	1403,886	27,91349	0
	GA+MLS	1355	1426	1387,743	18,66948	3
	GA+mX+LS**	1431	1530	1484,429	23,33677	0
	mXLSGA	1355	1421	1390,8	20,60954	4

Fonte: Elaborada pela autora.

Na Tabela 4.3, é apresentada a média de tempo necessária para todas as configurações do método proposto solucionarem as instâncias JSSP consideradas neste estudo de caso. Como pode-se observar, o GA básico é a técnica mais veloz dentre todas as comparadas. Entretanto, nota-se que a adição do operador de multi-*crossover* (GA+mX) não compromete muito o tempo computacional, mas, como apresentado nos gráficos de caixa das Figuras 4.1 (a) e (c), este acréscimo melhora consideravelmente a performance do método em obter bons resultados. No caso das versões que usam apenas operadores de busca local massiva (GA+ELSA e GA+MLS), fica evidenciado que a performance é aprimorada e que, em instâncias menores tais como a LA03 e a LA17, o tempo computacional não tem um aumento significativo. Entretanto, no caso

da instância LA30, o tempo computacional tem um grande aumento. Isso ocorre devido ao comportamento quadrático destes operadores, uma vez que, como pode ser observado em sua formulação no Algoritmo 3.3, em cada iteração do método ocorrem $(N \cdot M)^2$ perturbações nos cromossomos de P_{massivo}. Algo similar ocorre com as configurações que fazem uso apenas de estratégias de busca local no operador mutação (GA+LS* e GA+LS**). Especificamente, é possível observar que estes são os operadores isolados mais custosos em tempo no caso de instâncias menores (LA03 e LA17) e este custo aumenta proporcionalmente de acordo com a complexidade e dimensão da instância. Estes fatos são refletidos na configuração GA+mX+LS**, cujo tempo médio em todas as instâncias é aproximadamente a soma dos tempos da configuração GA+mX e GA+LS**. Em todas as instâncias, a configuração que obtém melhor performance, no caso a mXLSGA, também é a configuração mais custosa em tempo computacional, mas isto é um resultado esperado, uma vez que esta técnica utiliza todos os operadores descritos neste trabalho. Desta forma, nos estudos de caso a seguir, as análises e comparações serão direcionadas para o mXLSGA.

Tabela 4.3. Tempo médio gasto em segundos por cada configuração do método proposto para resolver cada instância do JSSP durante 35 execuções independentes.

	LA03	**LA17**	**LA30**
GA	3,6	4,09	5,95
GA+mX*	5,40	6,21	9,62
GA+LS*	20,01	37,70	115,78
GA+LS**	16,68	30,23	100,50
GA+ELSA	9,45	28,35	177,28
GA+MLS	9,62	29,23	184,76
GA+mX+LS**	20,34	34,87	106,14
mXLSGA	32,42	70,29	236,32

Fonte: Elaborada pela autora.

4.2.2 Situação II: Comparação a técnicas do tipo GA

Esta situação foi planejada e executada para verificar a eficácia do método mXLSGA ao ser comparado com os principais métodos do tipo GA que compõem a literatura especializada: GA básico, GSA de Watanabe, Ida e Gen (2005), LSGA de Ombuki e Ventresca (2004) e aLSGA de Asadzadeh (2015). Esta eficácia deve ser garantida através da análise de diversas medidas estatísticas e do tempo de processamento de cada método. Na verdade, para efetuar as análises, todas as técnicas comparadas foram programadas neste trabalho. De forma que todos os métodos foram programados da forma mais fiel possível às descrições presentes nos seus respectivos artigos, porém, ressalta-se que a codificação pode ter diferenças da codificação original devido a vários motivos, tais como: técnicas de programação diferentes; algum detalhe do método que não esteja presente no texto do trabalho original ou que foi interpretado de outra forma; parametrizações do método, como, quantidade de vezes que o método foi executado, etc. Os métodos selecionados para comparação foram GA, GSA, LSGA e aLSGA.

Foi seguida a parametrização mais fiel possível à parametrização original de cada técnica, entretanto, foram utilizados 100 indivíduos e 100 iterações em todos os métodos. Em detalhes, a configuração utilizada se encontra na Tabela 4.4.

Tabela 4.4. Configurações para a situação II do estudo de caso I. Onde c_{best} é o melhor indivíduo na população e $c_{\text{best},1}$ e $c_{\text{best},2}$ são os dois melhores indivíduos diferentes da população.

	GA	GSA	LSGA	aLSGA	mXLSGA
Número de cromossomos	100	100	100	100	100
Número máximo de iterações	100	100	100	100	100
Taxa de cruzamento ($\Delta_{\times}$)	0,95	0,95	0,95	0,95	0,95
Taxa de mutação (Δ_{mut})	0,05	0,05	1	1	1
ϵ_{LS}	0	0	0,5	1	0,95
R_c	1	10	1	1	10
R_m	1	140	$N \cdot M$	$N \cdot M$	$N \cdot M$
$\mathcal{F}_{\times}$	{PMX}	{PMX}	{PMX}	{PMX}	{PMX,OX2}
$\mathcal{F}_{\text{mut}}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
$\mathcal{F}_{\text{pert}}$	{ }	{ }	{ }	$\{f_{\text{swap}}\}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
P_{melhores}	{ }	{ }	{ }	$\{c_{\text{best}}\}$	$\{c_{\text{best},1}, c_{\text{best},2}\}$

Fonte: Elaborada pela autora.

As técnicas comparadas foram executadas 35 vezes em instâncias de diferentes dimensões nas quais o método proposto foi capaz de encontrar a melhor solução conhecida de *makespan*. Na Tabela 4.5, é possível observar algumas informações estatísticas sobre essas execuções, a saber: o melhor valor de *fitness* alcançado pela técnica (Melhor); o pior valor de *fitness* alcançado (Pior); a média dos valores de *fitness* das execuções de cada técnica (Média); o desvio padrão desses valores (DP); o número de vezes que o método alcançou a melhor solução conhecida (Quantidade de ótimos); o número de iterações necessárias para atingir a melhor solução conhecida (Iteração do ótimo); e o tempo médio (Tempo (s)) em segundos que a técnica leva para realizar 100 iterações.

Tabela 4.5. Estatísticas de métodos do tipo GA. O símbolo "-" significa que o método não foi capaz de alcançar a melhor solução conhecida. (continua)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Iteração do ótimo	Tempo (s)
FT06	GA	55	57	55,45	0,85	27	52	2,4
	GSA	55	55	55	0	35	8	39,78
	LSGA	55	59	57,68	1,43	6	27	7,95
	aLSGA	55	55	55	0	35	11	11,51
	mXSLGA	55	55	55	0	35	5	22,11
LA01	GA	666	712	679,02	9,98	6	76	2,65
	GSA	666	715	677,8	13,61	13	10	51,79
	LSGA	666	726	697	16,65	1	61	12,15
	aLSGA	666	666	666	0	35	11	20,07
	mXSLGA	666	666	666	0	35	5	38,06
LA06	GA	926	938	927,4	2,87	24	79	3,34
	GSA	926	935	926,31	1,54	33	7	67,31
	LSGA	926	970	935,8	13,15	17	55	19,87
	aLSGA	926	926	926	0	35	3	38,07
	mXSLGA	926	926	926	0	35	2	71,98
LA11	GA	1222	1256	1235,97	10,81	5	58	3,97
	GSA	1222	1276	1232,14	14,94	20	19	81,59
	LSGA	1222	1299	1251,6	19,62	2	32	31,33
	aLSGA	1222	1222	1222	0	35	5	60,82
	mXSLGA	1222	1222	1222	0	35	3	116,57
LA16	GA	982	1100	1045,6	26,4	0	–	2,89
	GSA	994	1110	1046,77	26,37	0	–	55,31
	LSGA	1016	1148	1084,25	32,27	0	–	20,62
	aLSGA	959	985	980,51	4,48	0	–	38,75
	mXSLGA	945	982	972,25	13,3	2	96	66,25

Tabela 4.5. Estatísticas de métodos do tipo GA. O símbolo "-" significa que o método não foi capaz de alcançar a melhor solução conhecida. (conclusão)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Iteração do ótimo	Tempo (s)
LA23	GA	1189	1336	1271,71	34,44	0	–	3,78
	GSA	1148	1347	1214,08	43,85	0	–	73,19
	LSGA	1214	1419	1295,34	43,7	0	–	38,39
	aLSGA	1035	1115	1078	16,34	0	–	75,62
	mXSLGA	1032	1093	1060,45	17,96	1	51	123,64
LA26	GA	1525	1699	1619,51	39,5	0	–	4,44
	GSA	1433	1586	1512,22	36,47	0	–	91,67
	LSGA	1517	1665	1597,94	36,67	0	–	60,84
	aLSGA	1302	1384	1343,28	19,69	0	–	124,93
	mXSLGA	1218	1371	1300,85	41,88	6	85	203,95
LA31	GA	2120	2326	2223,14	48,45	0	–	6,23
	GSA	1943	2142	2050,17	63,09	0	–	130,65
	LSGA	2005	2336	2177	66,29	0	–	123,23
	aLSGA	1808	1897	1843,51	21,17	0	–	258,53
	mXSLGA	1784	1845	1807,71	19,2	5	80	424,56

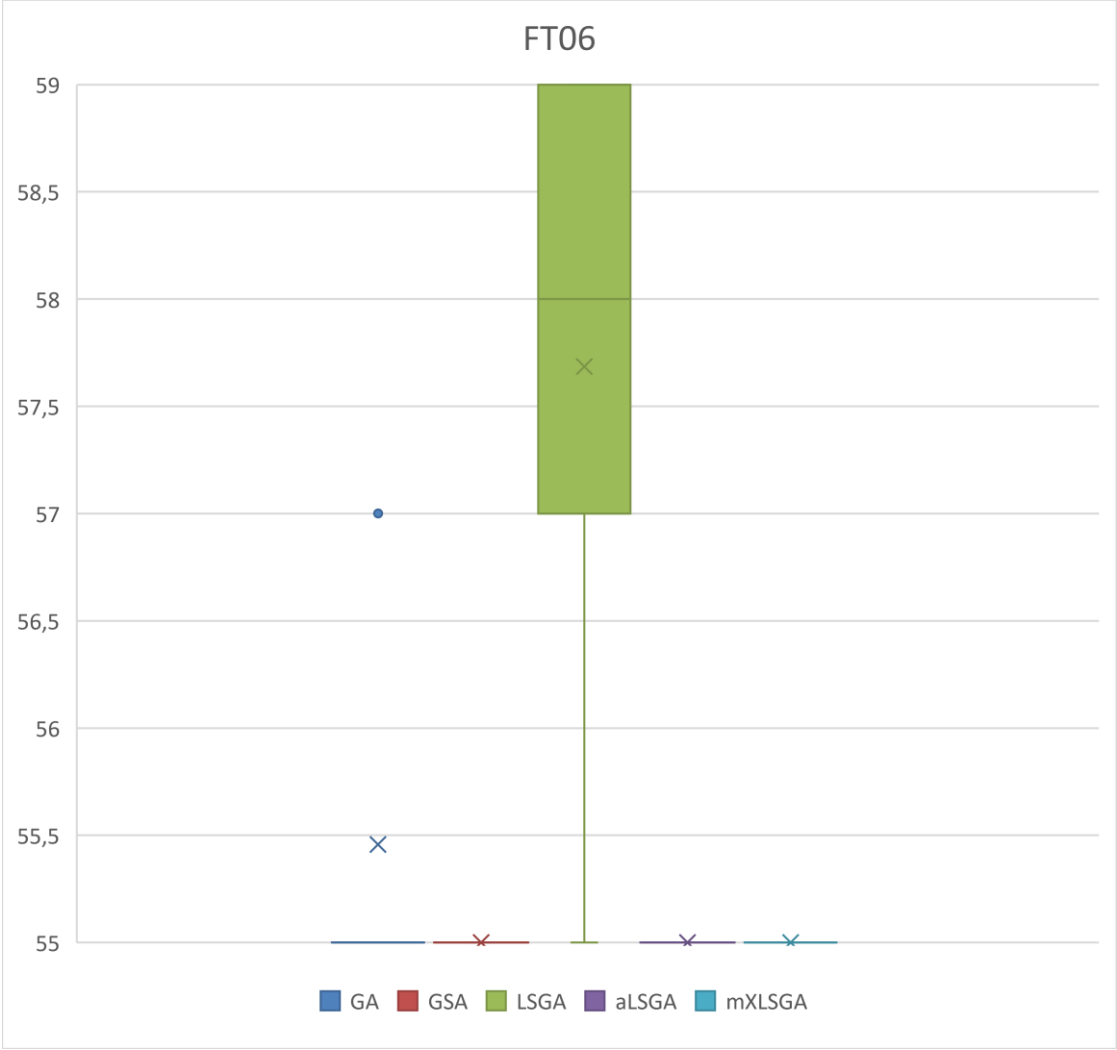
Fonte: Elaborada pela autora.

Conforme apresentado na Tabela 4.5, é notável que mXLSGA encontrou o melhor valor de *fitness* conhecido em todas as instâncias analisadas, e nas instâncias FT 06, LA 01, LA 06 e LA 11, o método proposto encontrou o a melhor solução conhecida em todas as execuções. O mXLSGA apresentou o menor pior valor e a menor média em todas as instâncias. Nas instâncias LA 16, LA 23 e LA 26, o mXLSGA não obteve o menor valor de desvio padrão, porém foi o único método que encontrou a melhor solução conhecida em todas essas instâncias. O método proposto foi o método que exigiu o maior tempo médio de execução, mas como será melhor explicado nos parágrafos a seguir, isso não é de forma alguma uma deficiência do mXLSGA, já que o mesmo não necessita de todas as 100 gerações para convergir.

Para visualizar o desempenho estatístico de cada método, foram confeccionados os gráficos de caixa dos valores de *fitness* alcançados nesta situação, os quais são apresentados na Figura 4.2. Conforme destacado na situação I anterior (Seção 4.2.1), os gráficos deixam claro como a presença de um operador de busca local massivo torna o método muito mais robusto em comparação com os demais, uma vez que em todos os casos o aLSGA e os métodos

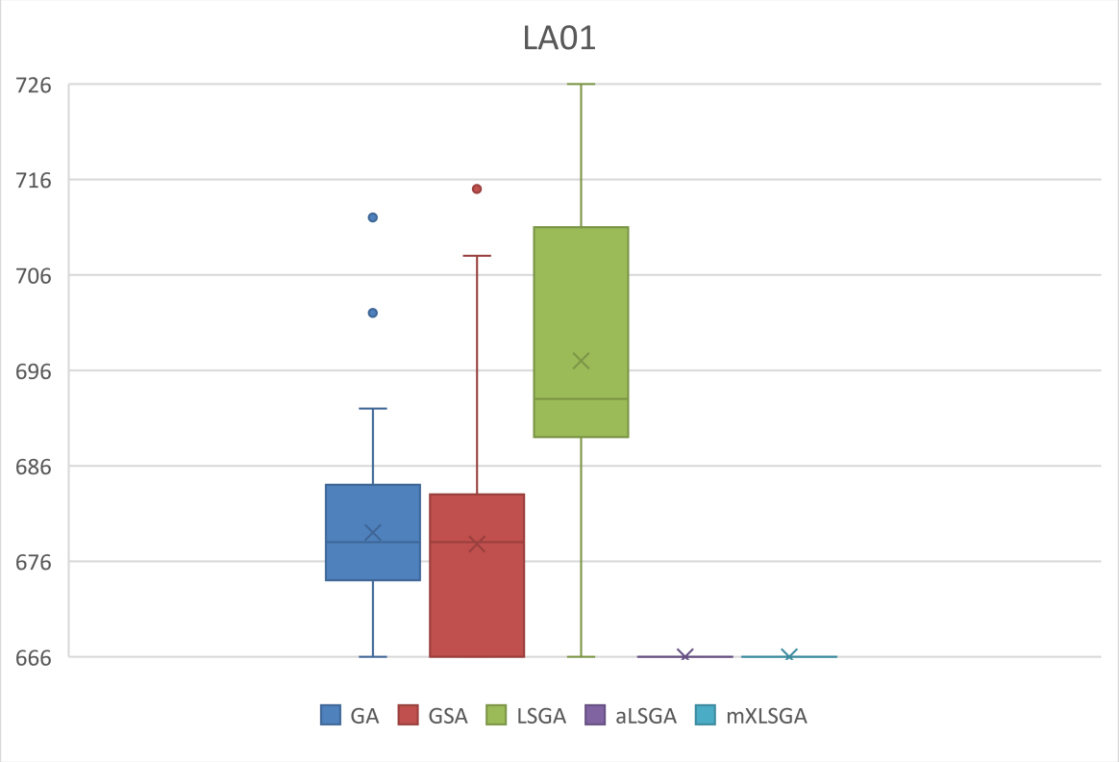
mXLSGA são os mais estáveis e têm os melhores valores de *fitness*. Essa diferença é ainda mais acentuada conforme a complexidade da instância aumenta. Além disso, os gráficos refletem os valores apresentados na Tabela 4.5, visto que mXLSGA é a caixa abaixo das outras caixas em todas as avaliações. Também se notam alguns detalhes, tais como o desvio padrão de mXLSGA é o maior quando executado na instância LA 26, já que isso ocorre porque a técnica encontra a melhor solução conhecida 6 vezes como uma discrepância. Também fica claro no gráfico que, estatisticamente, o método proposto é o método que encontra as melhores configurações de *makespan* com mais frequência.

Figura 4.2. Gráficos de caixa dos valores de *fitness* de 35 execuções dos métodos do tipo GA. (continua)

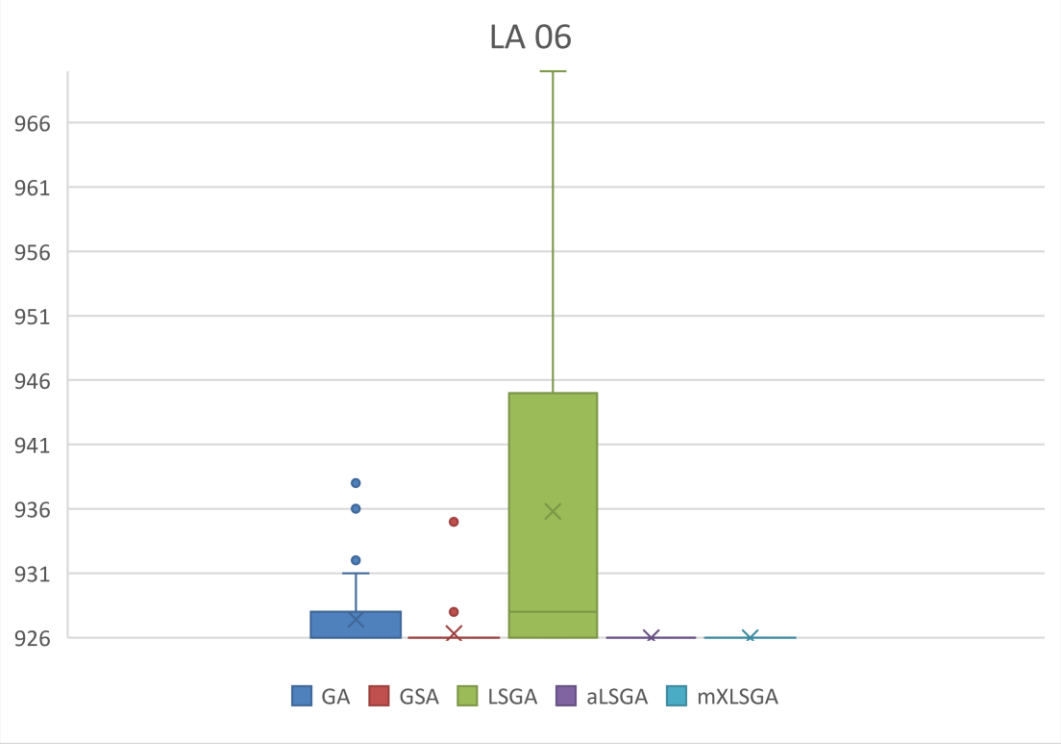


(a) FT 06: 6 × 6.

Figura 4.2. Gráficos de caixa dos valores de *fitness* de 35 execuções dos métodos do tipo GA. (continua)

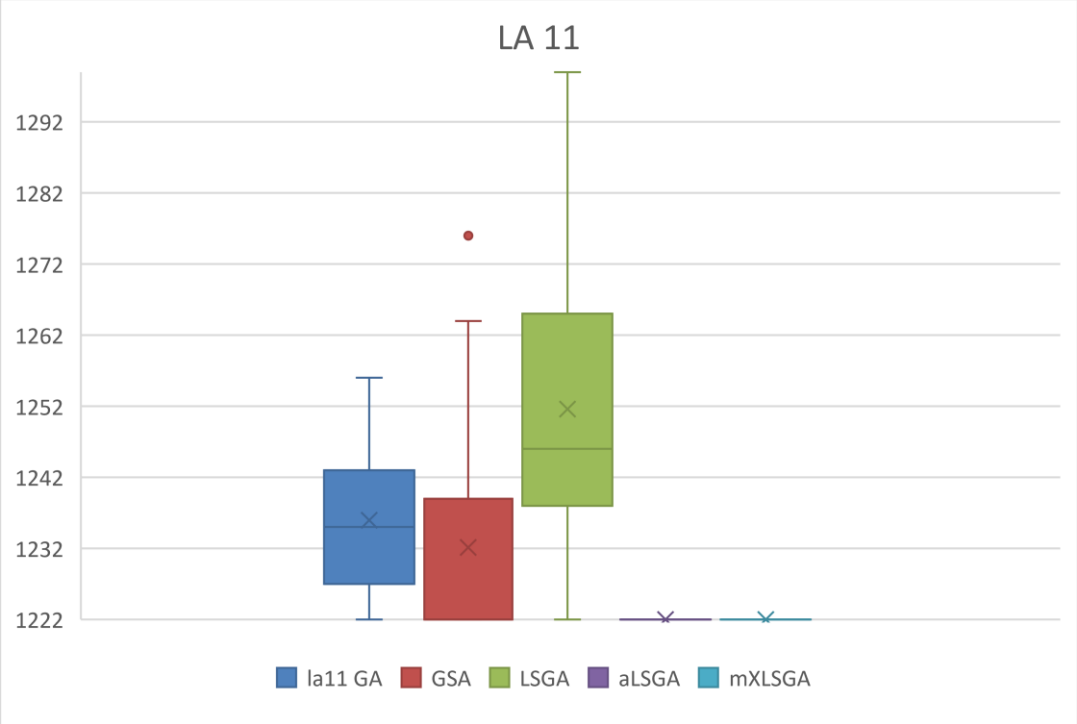


(b) LA 01: 10 × 5.

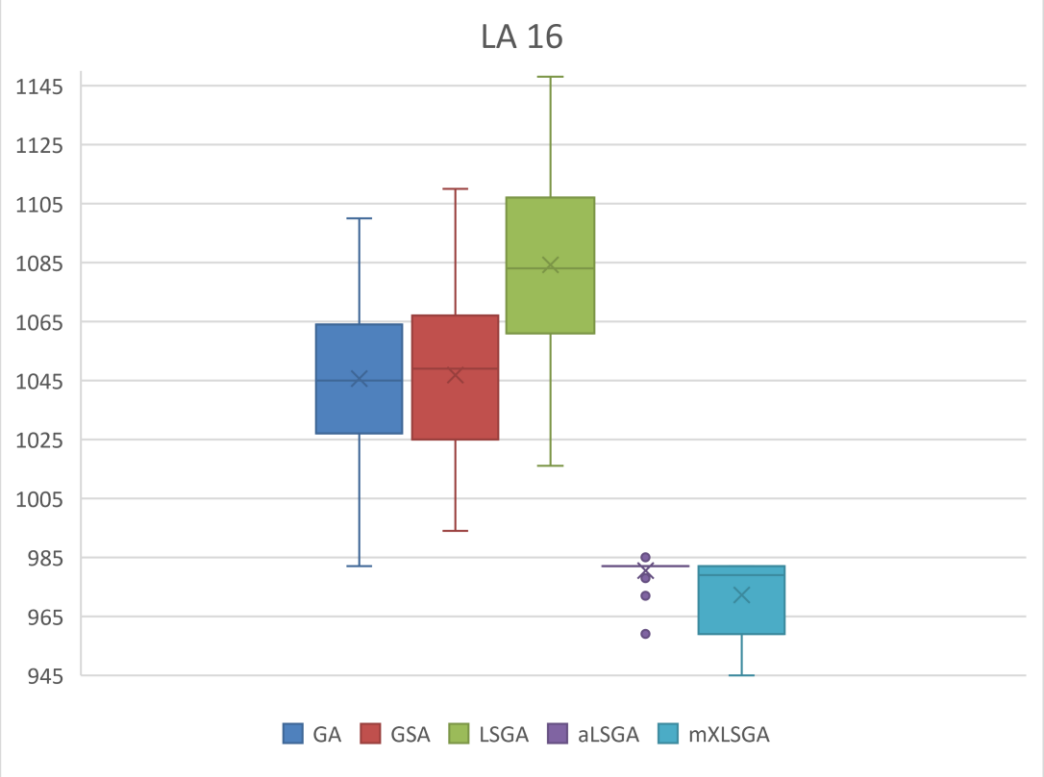


(c) LA 06: 15 × 5.

Figura 4.2. Gráficos de caixa dos valores de *fitness* de 35 execuções dos métodos do tipo GA. (continua)

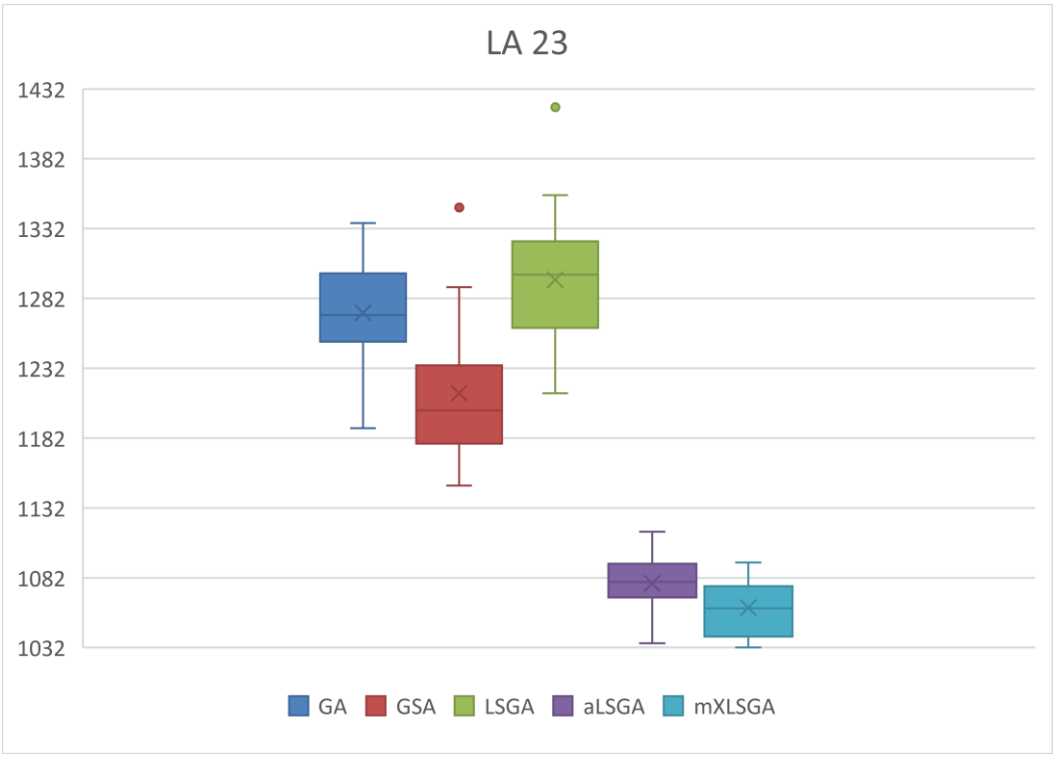


(d) LA 11: 20 × 5.

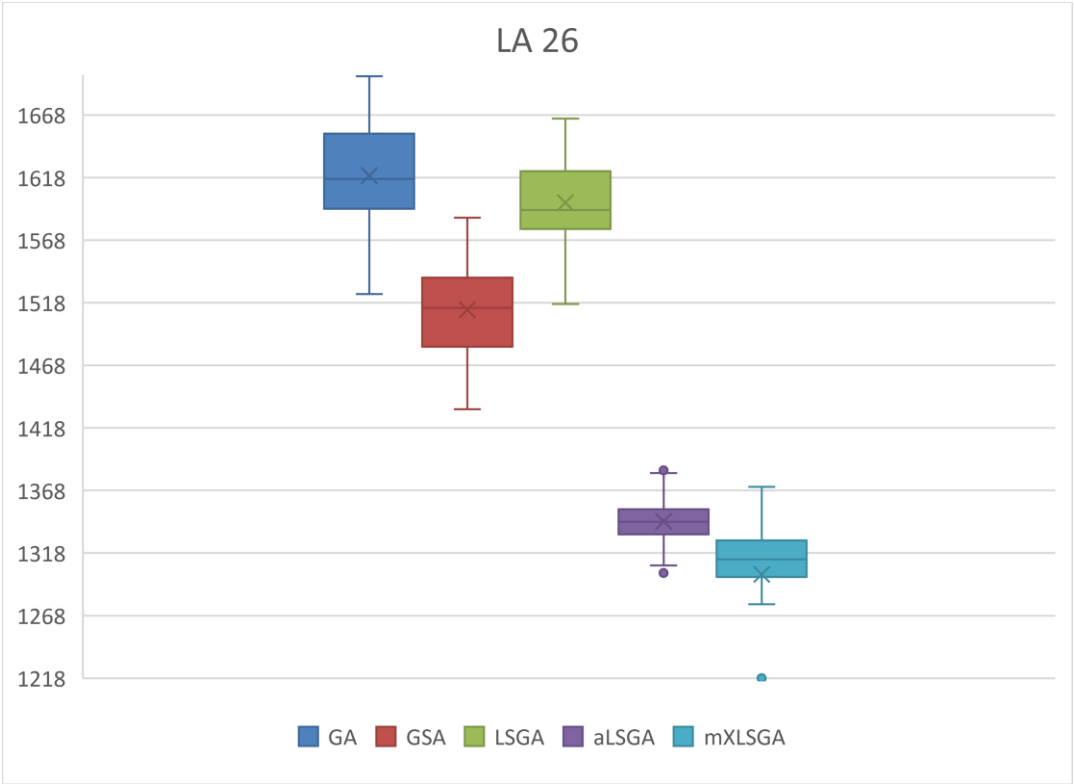


(e) LA 16: 10 × 10.

Figura 4.2. Gráficos de caixa dos valores de *fitness* de 35 execuções dos métodos do tipo GA. (continua)

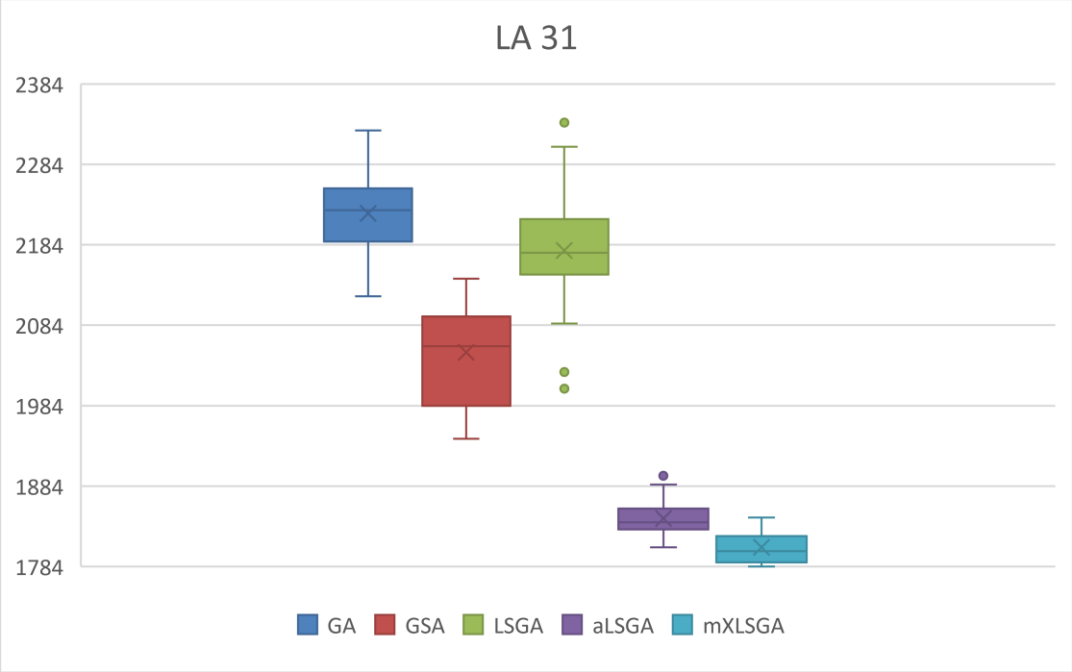


(f) LA 23: 15 × 10.



(g) LA 26: 20 × 10.

Figura 4.2. Gráficos de caixa dos valores de *fitness* de 35 execuções dos métodos do tipo GA. (conclusão)



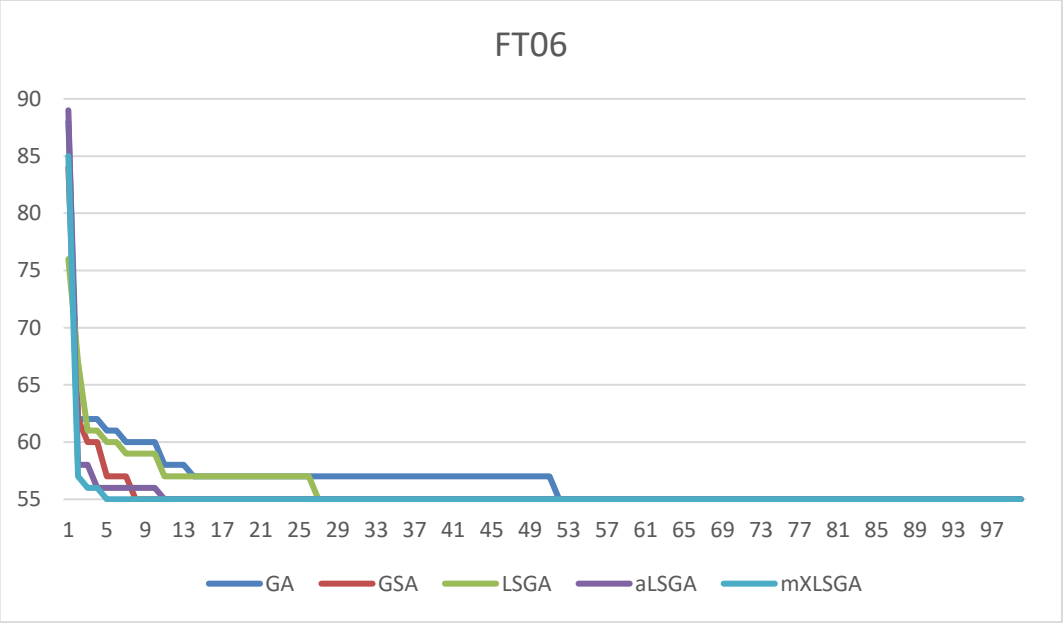
(h) LA 31: 30 × 10.

Fonte: Elaborada pela autora.

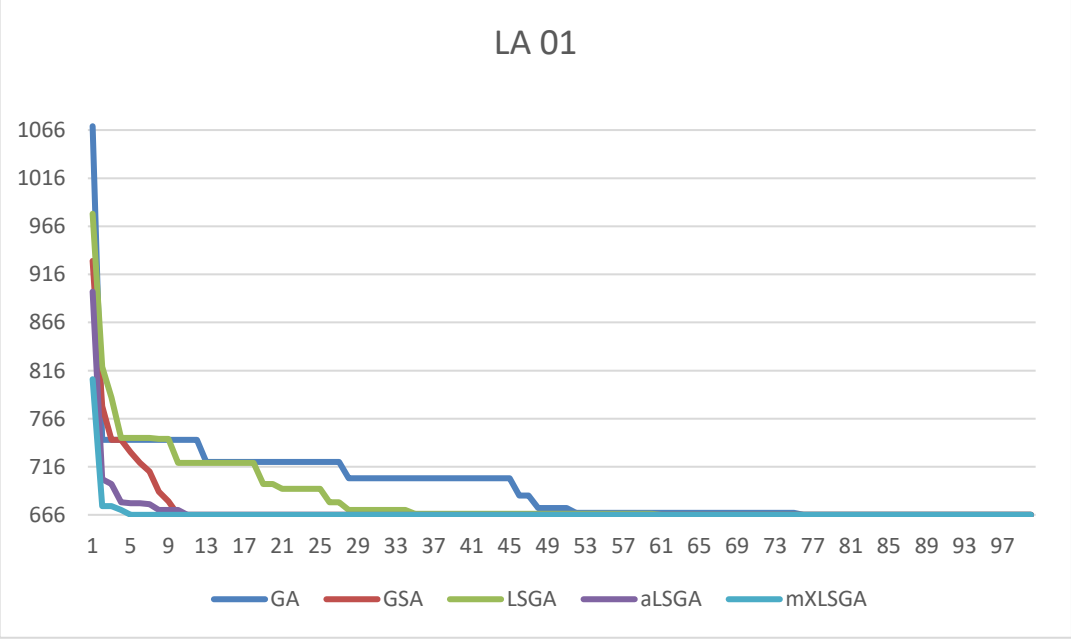
Na Figura 4.3, destacam-se as curvas de convergência do valor de *fitness* do melhor indivíduo das 35 execuções de cada método em cada instância durante as 100 gerações. Conforme a complexidade da instância avaliada aumenta, o número de iterações que cada método precisa para atingir o melhor valor também aumenta. No entanto, em nenhuma das situações o método proposto requer todas as 100 gerações para encontrar o melhor valor. No caso, o mXLSGA encontra o valor ideal de *makespan*, ou muito próximo da melhor solução conhecida, com aproximadamente 50 gerações em instâncias mais difíceis. Considerando que, no caso de instâncias mais simples, como FT 06 (Figura 4.3(a)), LA 01 (Figura 4.3 (b)), LA 06 (Figura 4.3 (c)) e LA 11 (Figura 4.3 (d)), percebe-se que o método proposto atinge o *fitness* ideal antes da 5ª iteração. Este fato não ocorre com as demais técnicas avaliadas, que precisam de mais iterações para alcançar a melhor solução conhecida ou, com a configuração utilizada, não conseguiram atingir nenhuma das melhores soluções conhecidas, como acontecia com todas as técnicas com exceção do mXLSGA nas instâncias LA 16 (Figura 4.3 (e)), LA 23 (Figura 4.3 (f)), LA 26 (Figura 4.3 (g)) e LA 31 (Figura 4.3 (h)). Desta forma, o tempo computacional não é uma grande preocupação para o método proposto, uma vez que este requer apenas 50% das iterações para alcançar bons resultados em instâncias maiores, o que reduz o

tempo gasto pela metade. Além Disso, o mesmo só precisa de 5% de iterações para atingir a melhor solução conhecida em instâncias mais simples.

Figura 4.3. Curvas de convergência de métodos do tipo GA ao longo de 100 gerações. (continua)

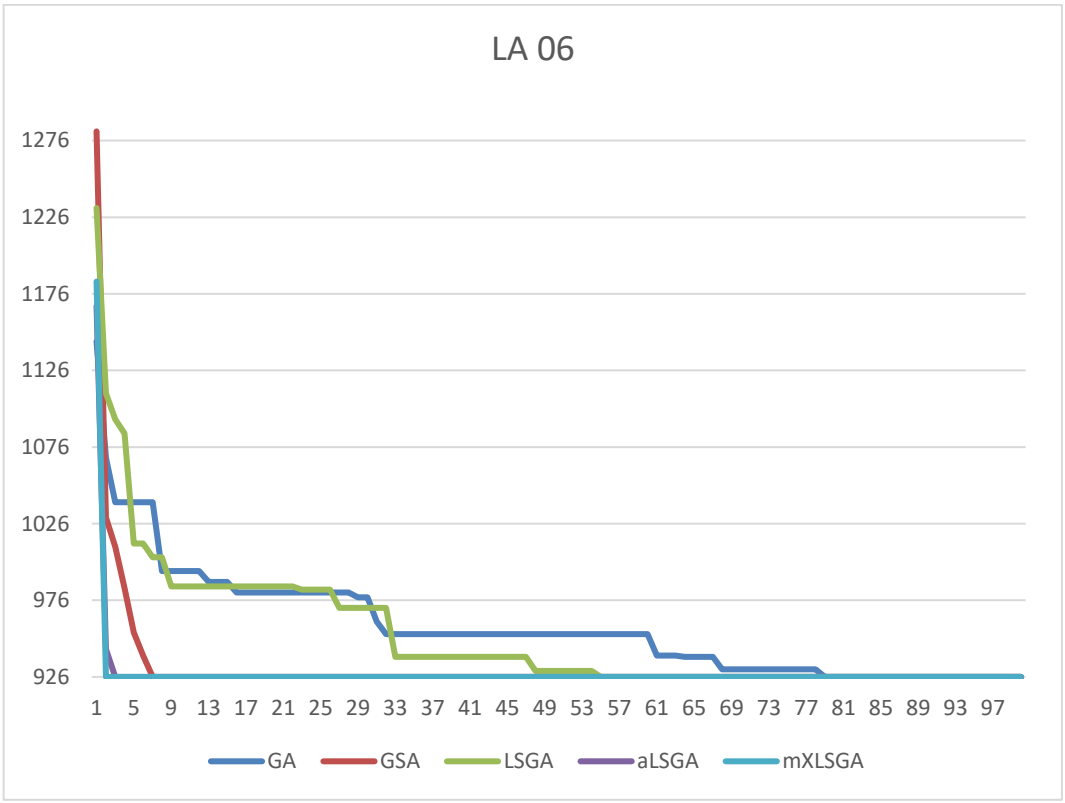


(a) FT 6: 6 × 6.

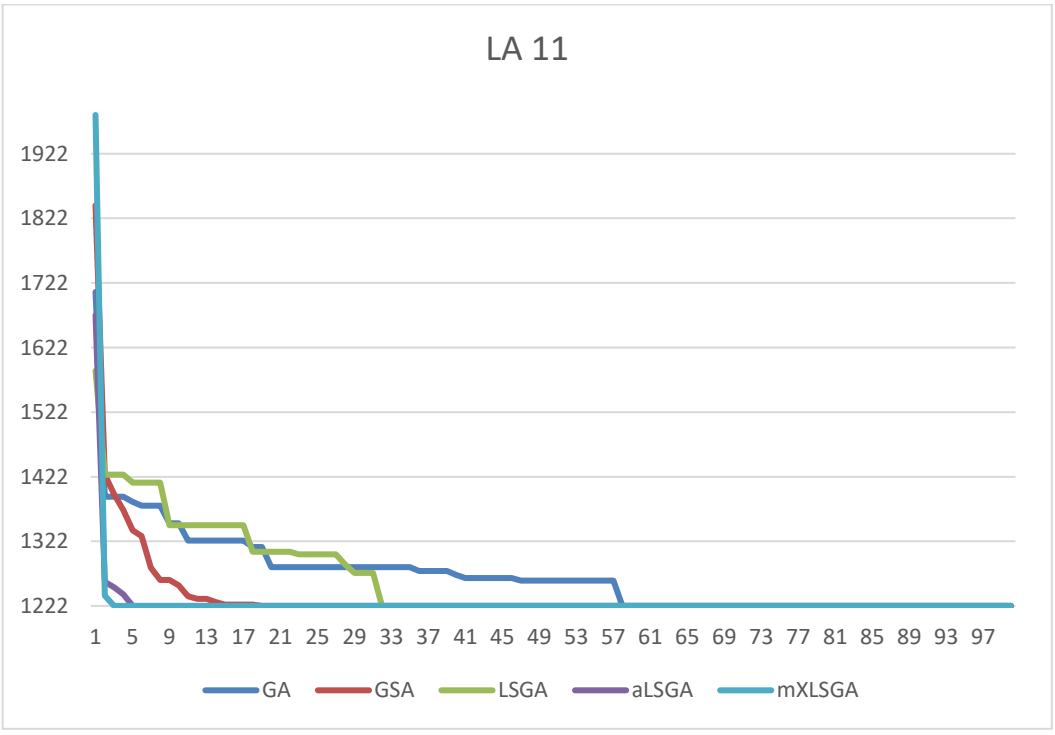


(b) LA 01: 10 × 5.

Figura 4.3: Curvas de convergência de métodos do tipo GA ao longo de 100 gerações. (continua)

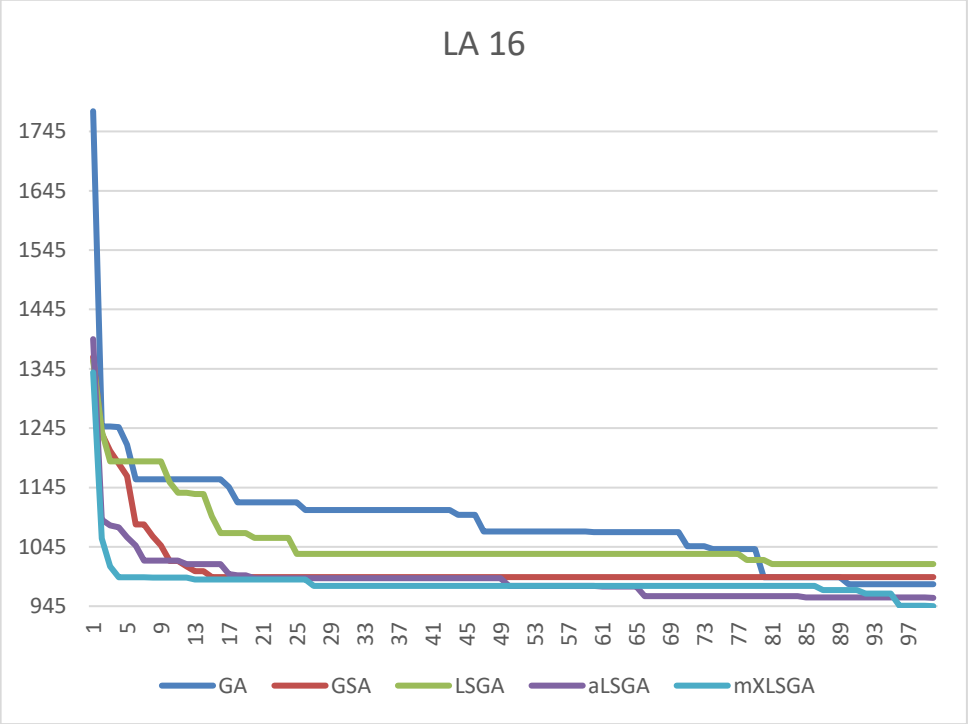


(c) LA 06: 10 × 10.

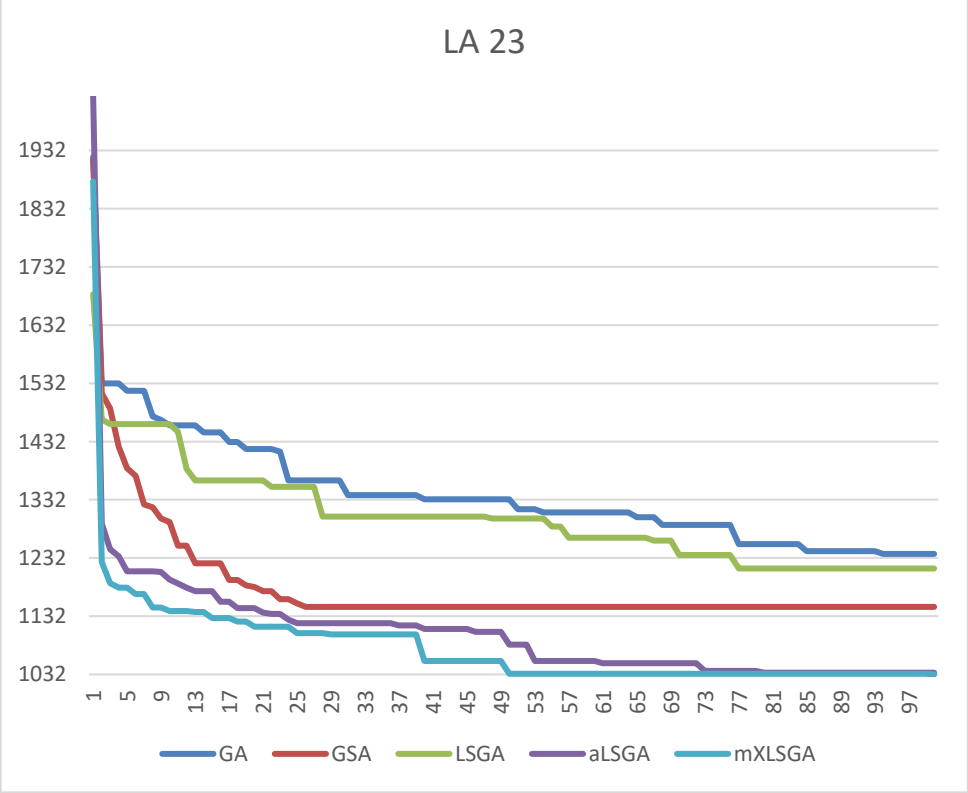


(d) LA 11: 20 × 5.

Figura 4.3: Curvas de convergência de métodos do tipo GA ao longo de 100 gerações. (continua)

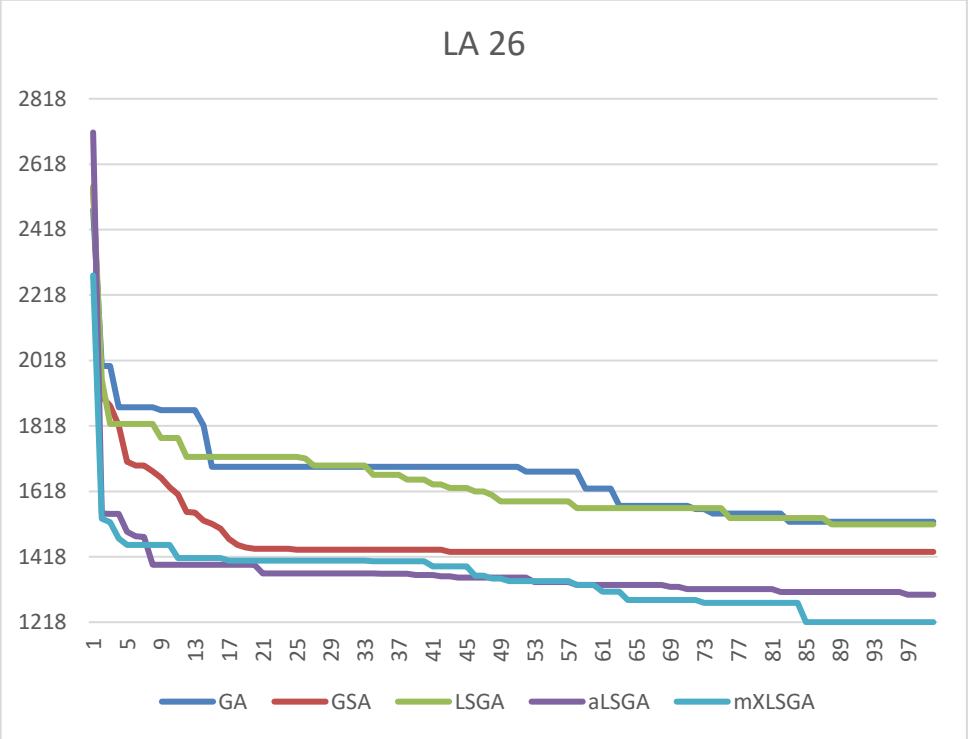


(e) LA 16: 10 × 10.

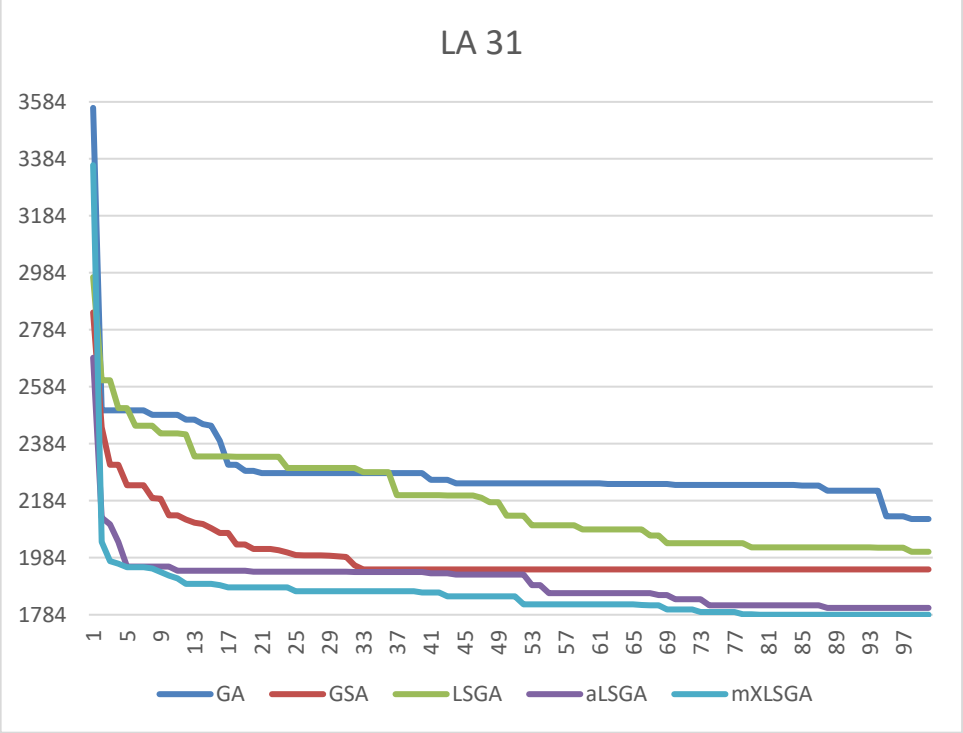


(f) LA 23: 15 × 10.

Figura 4.3: Curvas de convergência de métodos do tipo GA ao longo de 100 gerações. (conclusão)



(g) LA 26: 20 × 10.



(h) LA 31: 30 × 10.

Fonte: Elaborada pela autora.

Na sequência, ainda no que diz respeito à análise do tempo computacional, avalia-se a seguinte situação: em cada instância considerada neste estudo de caso executam-se todas as técnicas tomando como critério de parada o tempo computacional em vez do número máximo de iterações. Ou seja, todas as técnicas serão executadas durante o mesmo período de tempo. Desta forma, para cada instância, é tomado como limite de tempo para todas as técnicas o tempo que levou o método mais demorado para completar 100 iterações em relação a cada instância de acordo com a Tabela 4.5. Por exemplo, para este experimento, todos os métodos têm 39,78 segundos para pesquisar boas soluções para a instância FT-06, já que este período de tempo é o mesmo que a técnica GSA, que é a técnica mais demorada com respeito a esta instância, leva para executar 100 iterações. Todas as técnicas foram executadas independentemente 35 vezes seguindo-se a estratégia descrita. Os resultados estão resumidos na Tabela 4.6.

Tabela 4.6. Resultado de 35 execuções independentes de cada método considerando-se como critério de parada o tempo de execução. Nesse caso, a cada execução, todas as técnicas funcionam por, no mínimo, o mesmo tempo que a técnica que mais demora na instância considerada. (continua)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Tempo (s)
FT06	GA	55	55	55	0	35	39,78
	GSA	55	55	55	0	35	39,78
	LSGA	55	55	55	0	35	39,78
	aLSGA	55	55	55	0	35	39,78
	mXSLGA	55	55	55	0	35	39,78
LA01	GA	666	688	668,8	6,13	24	51,79
	GSA	666	715	677,8	13,61	13	51,79
	LSGA	666	731	692,11	13,66	2	51,79
	aLSGA	666	666	666	0	35	51,79
	mXSLGA	666	666	666	0	35	51,79
LA06	GA	926	926	926	0	35	71,98
	GSA	926	937	926,37	1,88	33	71,98
	LSGA	926	926	926	0	35	71,98
	aLSGA	926	926	926	0	35	71,98
	mXSLGA	926	926	926	0	35	71,98

Tabela 4.6. Resultado de 35 execuções independentes de cada método considerando-se como critério de parada o tempo de execução. Nesse caso, a cada execução, todas as técnicas funcionam por, no mínimo, o mesmo tempo que a técnica que mais demora na instância considerada. (conclusão)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Tempo (s)
LA11	GA	1222	1222	1222	0	35	116,57
	GSA	1222	1267	1226,54	9,49	25	116,57
	LSGA	1222	1222	1222	0	35	116,57
	aLSGA	1222	1222	1222	0	35	116,57
	mXSLGA	1222	1222	1222	0	35	116,57
LA16	GA	982	1035	997,94	18,36	0	66,25
	GSA	993	1103	1044,51	28,94	0	66,25
	LSGA	1003	1138	1081,48	31,66	0	66,25
	aLSGA	959	985	979,2	3,57	0	66,25
	mXSLGA	945	982	972,25	13,3	2	66,25
LA23	GA	1042	1180	1104,68	27,51	0	123,64
	GSA	1133	1262	1201,65	31,33	0	123,64
	LSGA	1214	1353	1276,02	35,1	0	123,64
	aLSGA	1037	1106	1069,97	19,53	0	123,64
	mXSLGA	1032	1093	1060,45	17,96	1	123,64
LA26	GA	1309	1449	1382,97	33,59	0	203,95
	GSA	1421	1599	1515,91	49,36	0	203,95
	LSGA	1496	1690	1591,2	48,03	0	203,95
	aLSGA	1285	1374	1332,11	22,97	0	203,95
	mXSLGA	1218	1371	1300,85	41,88	6	203,95
LA31	GA	1810	2022	1886,34	48,18	0	424,56
	GSA	1974	2176	2056,8	51,09	0	424,56
	LSGA	2005	2261	2155,54	51,48	0	424,56
	aLSGA	1808	1866	1821,8	20,9	0	424,56
	mXSLGA	1784	1845	1807,71	19,2	5	424,56

Fonte: Elaborada pela autora.

Na Tabela 4.6, é possível observar que houve melhora em determinados indicadores de todas as técnicas que tiveram mais tempo para serem executadas. Um exemplo é o GA básico, que mostrou uma melhoria de desempenho em praticamente todas as medidas considerando todas as instâncias. No entanto, a complexidade da instância avaliada permanece um fator

predominante no que diz respeito ao desempenho da técnica neste experimento, uma vez que, no caso de uma instância menos complexa como a FT06, todos os métodos avaliados foram capazes de encontrar a melhor solução conhecida (55) em todas as 35 execuções. No caso de instâncias mais complexas como as instâncias LA16, LA23, LA26 e LA31 apenas o método proposto foi capaz de encontrar a melhor solução conhecida, mesmo com todas as técnicas podendo funcionar durante o mesmo tempo. Além disso, também nestas instâncias, o método proposto apresentou sem empate as melhores medidas estatísticas do melhor valor de *fitness*, pior valor de *fitness* e média de *fitness*. Em outras palavras, a metodologia proposta apresentou o melhor desempenho em instâncias mais complexas e empatou essas medidas em instâncias mais simples neste experimento. Isso serve como um indicativo de que a metodologia proposta neste trabalho fornece melhor capacidade de busca para a técnica, tornando-a mais eficiente e superando outros algoritmos do tipo GA presentes na literatura.

4.3 Estudo de caso II: Uso do operador de melhoramento genético

Neste estudo de caso, o principal objetivo consiste em avaliar o funcionamento do operador de melhoramento genético e os efeitos de seu uso em diferentes técnicas do tipo GA. Para isso, avalia-se o impacto do operador proposto ao ser anexado em cinco configurações especiais do *framework* proposto: um GA básico (**GA**); o GA com adaptação de área de busca (**GSA**) de Watanabe; Ida; Gen (2005); o GA com Busca Local (**LSGA**) de Ombuki; Ventresca (2004), o GA com Busca Local de Elite e ajuste por agentes (**aLSGA**) de Asadzadeh (2015); e o GA com multi-*crossover* e busca local massiva tratado nas seções anteriores, que consiste no **mXLSGA** proposto. Em cada uma dessas versões que representam o estado-da-arte em técnicas do tipo GA para solução de JSSP acrescenta-se o operador de aprimoramento genético proposto, o **GIFA**, e conduz-se as avaliações sobre oito instâncias JSSP de complexidades variadas que compõem o benchmark da área, sendo 1 de Fisher e Thompson (1963) (FT) e 7 de Lawrence (1984) (LA): a FT 06, de dimensão 6×6, e melhor solução conhecida (BKS) igual a 55; a LA 01, de dimensão 10×5, e BKS igual a 666; a LA 06, de dimensão 15×5, e BKS igual a 926; a LA 11, de dimensão 20×5, e BKS igual a 1222; a LA 16, de dimensão 10×10, e BKS igual a 945; a LA 23, de dimensão 15×10, e BKS igual a 1032; a LA 26, de dimensão 20×10, e BKS igual a 1218; e a LA 31, de dimensão 30×10, e BKS igual a 1784. Desta

forma, cada método do tipo GA considerado possui uma versão com o operador proposto, representado pela sigla **GIFA** junto à sua sigla padrão.

O principal intuito nesta situação é avaliar o impacto do uso do GIFA em cada um dos métodos do tipo GA, por isso mantem-se a melhor configuração possível de cada um dos métodos disponíveis em seus trabalhos originais com a ressalva de que todos tenham 100 indivíduos em suas populações e que sejam executados por 100 gerações. Além disso, acrescenta-se a cada um deles a configuração referente ao GIFA, a qual é definida como segue: $N_{\text{Exemplares}} = N_{\text{piores}} = 10$. No caso, são apresentados na Tabela 4.7 o melhor valor de *fitness* obtido (Melhor), o pior valor de *fitness* obtido (Pior), a média dos *fitness* obtidos (Média), e o desvio padrão (DP) dos valores de *fitness* calculados em 35 execuções independentes de cada método sobre as três instâncias JSSP consideradas. Também são apresentados o número de vezes em que o método atingiu a melhor solução conhecida (Quantidade de ótimos); o número de iterações (Iteração do ótimo) necessárias para atingir a melhor solução conhecida; e o tempo médio (Tempo (s)) em segundos que a técnica leva para realizar 100 iterações.

Tabela 4.7. Resultado de 35 execuções independentes de cada método considerando-se como critério de parada o número de iterações. (continua)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Iteração do ótimo	Tempo (s)
FT06	GA	55	57	55,45	0,85	27	52	2,4
	GIFA-GA	55	56	55,14	0,35	30	38	2,63
	GSA	55	55	55	0	35	8	39,78
	GIFA-GSA	55	55	55	0	35	8	40,02
	LSGA	55	59	57,68	1,43	6	27	7,95
	GIFA-LSGA	55	56	55,83	0,38	6	20	8,17
	aLSGA	55	55	55	0	35	11	11,51
	GIFA-aLSGA	55	55	55	0	35	6	13,01
	mXLSGA	55	55	55	0	35	5	22,11
	GIFA-mXLSGA	55	55	55	0	35	5	22,37
LA01	GA	666	712	679,02	9,98	6	76	2,65
	GIFA-GA	666	678	669,37	5,17	15	26	2,94
	GSA	666	715	677,8	13,61	13	10	51,79
	GIFA-GSA	666	687	672,34	7,43	17	10	51,96
	LSGA	666	726	697	16,65	1	61	12,15
	GIFA-LSGA	666	707	688,67	15,59	8	23	12,42
	aLSGA	666	666	666	0	35	11	20,07
	GIFA-aLSGA	666	666	666	0	35	6	20,29
	mXLSGA	666	666	666	0	35	5	38,06
	GIFA-mXLSGA	666	666	666	0	35	4	38,34
LA06	GA	926	938	927,4	2,87	24	79	3,34
	GIFA-GA	926	936	926,86	2,26	30	54	3,57
	GSA	926	935	926,31	1,54	33	7	67,31
	GIFA-GSA	926	926	926	0	35	7	67,56
	LSGA	926	970	935,8	13,15	17	55	19,87
	GIFA-LSGA	926	952	932,28	9,08	20	41	20,13
	aLSGA	926	926	926	0	35	3	38,07
	GIFA-aLSGA	926	926	926	0	35	3	38,31
	mXLSGA	926	926	926	0	35	2	71,98
	GIFA-mXLSGA	926	926	926	0	35	2	72,21

Tabela 4.7. Resultado de 35 execuções independentes de cada método considerando como critério de parada o número de iterações. (continua)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Iteração do ótimo	Tempo (s)
LA11	GA	1222	1256	1235,97	10,81	5	58	3,97
	GIFA-GA	1222	1253	1233,48	9,52	6	51	4,18
	GSA	1222	1276	1232,14	14,94	20	19	81,59
	GIFA-GSA	1222	1263	1231,24	13,56	26	15	81,7
	LSGA	1222	1299	1251,6	19,62	2	32	31,33
	GIFA-LSGA	1222	1278	1250,17	11,09	4	26	31,59
	aLSGA	1222	1222	1222	0	35	5	60,82
	GIFA-aLSGA	1222	1222	1222	0	35	4	61,03
	mXLSGA	1222	1222	1222	0	35	3	116,57
	GIFA-mXLSGA	1222	1222	1222	0	35	3	116,84
LA16	GA	982	1100	1045,6	26,4	0	–	2,89
	GIFA-GA	982	1061	1022,89	20,51	0	–	3,12
	GSA	994	1110	1046,77	26,37	0	–	55,31
	GIFA-GSA	994	1021	1017,38	15,49	0	–	55,63
	LSGA	1016	1148	1084,25	32,27	0	–	20,62
	GIFA-LSGA	1016	1077	1037,11	26,62	0	–	20,83
	aLSGA	959	985	980,51	4,48	0	–	38,75
	GIFA-aLSGA	956	982	975,12	2,36	0	–	38,98
	mXLSGA	945	982	972,25	13,3	2	96	66,25
	GIFA-mXLSGA	945	979	959,93	6,37	7	49	66,51
LA23	GA	1189	1336	1271,71	34,44	0	–	3,78
	GIFA-GA	1151	1324	1269,51	30,95	0	–	4
	GSA	1148	1347	1214,08	43,85	0	–	73,19
	GIFA-GSA	1121	1339	1191,25	30,27	0	–	73,42
	LSGA	1214	1419	1295,34	43,7	0	–	38,39
	GIFA-LSGA	1115	1369	1278,36	34,79	0	–	38,63
	aLSGA	1035	1115	1078	16,34	0	–	75,62
	GIFA-aLSGA	1032	1098	1067,58	15,06	2	75	75,79
	mXLSGA	1032	1093	1060,45	17,96	1	51	123,64
	GIFA-mXLSGA	1032	1093	1039,8	16,54	2	34	123,87

Tabela 4.7. Resultado de 35 execuções independentes de cada método considerando como critério de parada o número de iterações. (conclusão)

Instância	Método	Melhor	Pior	Média	DP	Quantidade de ótimos	Iteração do ótimo	Tempo (s)
LA26	GA	1525	1699	1619,51	39,5	0	–	4,44
	GIFA-GA	1485	1667	1614,11	32,59	0	–	4,65
	GSA	1433	1586	1512,22	36,47	0	–	91,67
	GIFA-GSA	1420	1523	1503,73	17,99	0	–	91,94
	LSGA	1517	1665	1597,94	36,67	0	–	60,84
	GIFA-LSGA	1492	1537	1534,11	28,74	0	–	61,07
	aLSGA	1302	1384	1343,28	19,69	0	–	124,93
	GIFA-aLSGA	1273	1382	1332,68	13,28	0	–	125,12
	mXLSGA	1218	1371	1300,85	41,88	6	85	203,95
	GIFA-mXLSGA	1218	1371	1258,51	32,32	11	67	204,3
LA31	GA	2120	2326	2223,14	48,45	0	–	6,23
	GIFA-GA	2111	2322	2197,31	45,19	0	–	6,51
	GSA	1943	2142	2050,17	63,09	0	–	130,65
	GIFA-GSA	1919	2101	1964,2	56,94	0	–	130,89
	LSGA	2005	2336	2177	66,29	0	–	123,23
	GIFA-LSGA	1986	2301	2119,88	62,49	0	–	123,57
	aLSGA	1808	1897	1843,51	21,17	0	–	258,53
	GIFA-aLSGA	1784	1861	1841,25	19,13	2	80	258,81
	mXLSGA	1784	1845	1807,71	19,2	5	80	424,56
	GIFA-mXLSGA	1784	1845	1805,98	18,96	7	71	424,77

Fonte: Elaborada pela autora.

Na Tabela 4.7, é possível observar que o operador tornou todos os métodos mais estáveis, reduzindo a amplitude da média e do desvio padrão em todas as situações nas quais eram possíveis haver melhora. Além disso, na maioria dos casos, a adição do operador GIFA resultou na diminuição do pior valor de *makespan* encontrado. De fato, o operador não foi capaz de melhorar este indicador apenas com respeito ao método mXLSGA e considerando três instâncias: LA 23, LA 26 e LA 31. Um fenômeno análogo pode ser observado com respeito ao melhor valor obtido por cada técnica, uma vez que, na maioria dos casos, o uso do operador GIFA faz com que a técnica original seja capaz de alcançar um valor mais próximo da melhor solução conhecida para a instância avaliada. No caso, é possível observar que, o uso do operador GIFA aumentou o número de melhores soluções conhecidas encontradas pelas técnicas em

todas as instâncias. Este fato é observado principalmente nas instâncias de menor complexidade. Entretanto, em instâncias de maior complexidade, especificamente a partir da instância LA 16, o operador proposto foi capaz de auxiliar uma técnica base a encontrar a melhor solução conhecida apenas nos casos das técnicas **aLSGA** e **mXLSGA**, sendo que esta última já é capaz de encontrar estes valores sem o uso do operador de melhoramento genético. Isto serve de indicativo de que o operador proposto oferece um aumento considerável na estabilidade do método, mas a capacidade de exploração do espaço de busca ainda possui forte dependência da técnica original utilizada. Isto ocorre, pois o GIFA orienta a população no sentido de guiar indivíduos com valores de *makespan* considerados ruins em regiões nas quais sabe-se da existência de indivíduos com bons valores de *fitness* para, assim, aumentar a exploração local e, portanto, encontrar boas soluções, mas cabe à técnica original indicar boas regiões de busca.

É válido destacar que a melhora que o operador GIFA proporciona a uma técnica base não possui muita relação com o tempo computacional necessário para sua execução, uma vez que este está definido entre 0.2 e 0.3 segundos, diferente do operador transgênico de Viana; Morandin Junior e Contreras (2020d) que demanda uma custosa etapa de pré-processamento e simulação para determinar relevância genética.

4.4 Estudo de caso III: Uso dos operadores de busca local e melhoramento genético propostos em comparação com metaheurísticas gerais

Nesta situação, é avaliada a capacidade dos operadores propostos em procurar soluções ótimas no espaço de buscas. Em detalhes, pretende-se demonstrar a eficácia do método ao ser aplicado em instâncias JSSP presentes na literatura especializada e comparando os seus resultados com aqueles obtidos por metaheurísticas de diferentes inspirações quando aplicados no mesmo problema. Em detalhes, para avaliar a abordagem, os experimentos foram realizados em 58 cenários de instância JSSP, sendo estes: 3 instâncias FT (FISHER; THOMPSON, 1963), 40 instâncias LA (LAWRENCE, 1984), 10 instâncias ORB (APPLEGATE; COOK, 1991) e 5 instâncias ABZ (ADAMS; BALAS; ZAWACK, 1988).

Também é esperado com este experimento que seja possível avaliar a capacidade do operador de melhoramento genético proposto em aumentar o poder de busca e exploração de

uma dada técnica do tipo GA. Para isso, são consideradas duas versões do método proposto: o **mXLSGA** e o **GIFA-mXLSGA**. Assim, o operador GIFA será avaliado com respeito à melhor versão do método proposto, uma vez que este apresentou os melhores resultados nos estudos de caso anteriores.

Os resultados obtidos na execução dos testes foram comparados com trabalhos da literatura específica. No caso, os artigos determinados para cada comparação foram selecionados por se tratarem de trabalhos relevantes na literatura, os quais tratam do JSSP com as mesmas instâncias. Especificamente, os artigos selecionados para comparação dos resultados foram os seguintes: **IHHO** (LIU, 2021), **DCSA** (ALKHATEEB; ABED-ALGUNI; AL-ROUSAN, 2021), **SSO** (ZHOU; ZHOU; ZHAO, 2021), **LSO** (HUANG *et al.*, 2021), **HIGA** (LU; HUANG; CAO, 2020), **NGPSO** (YU *et al.*, 2020), **SSS** (HAMZADAYI; BAYKASOĞLU; AKPINAR, 2020), **NAGA** (CHEN; ZHANG; GAO, 2019), **GA-CPG-GT** (KURDI, 2019), **DWPA** (WANG; TIAN; WANG, 2019), **GWO** (JIANG; ZHANG, 2018), **HGWO** (JIANG, 2018), **MA** (LAMOS-D'IAZ *et al.*, 2017), **NIMGA** (KURDI, 2016), **IPB-GA** (JORAPUR *et al.*, 2016), **aLSGA** (ASADZADEH, 2015), **PaGA** (ASADZADEH; ZAMANIFAR, 2010), **GSA** (WATANABE; IDA; GEN, 2005) e **LSGA** (OMBUKI; VENTRESCA, 2004).

A configuração dos parâmetros para o mXLSGA e o GIFA-mXLSGA foi estabelecida por meio de testes e também levando-se em consideração, quando possível, uma parametrização mais próxima dos trabalhos que foram utilizados para comparação. Desta forma, os parâmetros foram definidos conforme apresentado na Tabela 4.8.

Tabela 4.8. O c_{best} é o melhor indivíduo na população e $c_{\text{best},1}$ e $c_{\text{best},2}$ são os dois melhores indivíduos diferentes na população.

Número de cromossomos	100
Número máximo de iterações	100
Taxa de cruzamento ($\Delta_{\times}$)	0.95
Taxa de mutação (Δ_{mut})	0.95
ϵ_{LS}	0.95
R_c	10
R_m	$2 \cdot N \cdot M$
$\mathcal{F}_{\times}$	{OX2, PMX}
$\mathcal{F}_{\text{mut}}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
$\mathcal{F}_{\text{pert}}$	$\{f_{\text{swap}}, f_{\text{inverse}}, f_{\text{insert}}\}$
P_{melhores}	$\{c_{\text{best},1}, c_{\text{best},2}\}$
$N_{\text{Exemplares}}$	10
N_{piores}	10

Fonte: Elaborada pela autora.

Os métodos mXLSGA e GIFA-mXLSGA propostos foram executados 10 vezes para cada instância JSSP e o melhor valor obtido foi usado para comparação com outros artigos. Na maioria dos trabalhos comparativos, os autores não mencionam o tempo de processamento de suas técnicas e apenas apresentam o melhor resultado. Nesta situação específica, as avaliações serão conduzidas da mesma forma.

São apresentados nas Tabelas 4.9, 4.10, 4.11 e 4.12 os resultados derivados dos testes das instâncias LA, FT, ORB e ABZ. As colunas indicam, respectivamente, a instância que foi avaliada, o tamanho da instância (número de *jobs* pelo número de máquinas), a melhor solução conhecida para cada instância, os resultados alcançados por cada método (melhor solução encontrada), o erro relativo (Equação (4.1)), e a média do erro relativo para cada *benchmark* (MErr).

$$\textbf{Erro} = 100 \cdot \frac{\textbf{Melhor} - \textbf{BKS}}{\textbf{BKS}}, \tag{4.1}$$

na qual **Erro** é o erro relativo, **BKS** é a melhor solução conhecida e **Melhor** é o melhor valor obtido executando o algoritmo para cada instância.

Tabela 4.9. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias FT. O símbolo "-" significa "não avaliado nessa instância". (continua)

			GIFA-mXLSGA		mXLSGA		IHHO		DCSA		SSO		LSO	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
FT06	6 × 6	55	55	0	55	0	55	0	55	0	–		55	0
FT10	10 × 10	930	930	0	930	0	1053	13,22	930	0	–		1159	24,62
FT20	20 × 5	1165	1165	0	1165	0	1352	16,05	1174	0,77	1374	17,93	1429	22,66
MErr				**0**		**0**		**9,75**		**0,25**		**17,93**		**15,76**

Fonte: Elaborada pela autora.

Tabela 4.9. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias FT. O símbolo "-" significa "não avaliado nessa instância". (continua)

			SSS		NAGA		GA-CPG-GT		MeCSO		GWO		HGWO	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
FT06	6 × 6	55	55	0	55	0	55	0	55	0	55	0	55	0
FT10	10 × 10	930	936	0,64	962	3,44	935	0,53	939	0,96	940	1,07	951	2,25
FT20	20 × 5	1165	1165	0	1216	4,37	1180	1,28	–		1178	1,11	1178	1,11
MErr				**0,21**		**2,60**		**0,60**		**0,48**		**0,73**		**1,12**

Fonte: Elaborada pela autora.

Tabela 4.9. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias FT. O símbolo "-" significa "não avaliado nessa instância". (continua)

			MA		IPB-GA		NIMGA		aLSGA		PaGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
FT06	6 × 6	55	55	0	55	0	55	0	55	0	55	0
FT10	10 × 10	930	937	0,75	960	3,22	930	0	930	0	997	7,20
FT20	20 × 5	1165	1182	1,45	1192	2,31	1173	0,68	1165	0	1196	2,66
MErr				**0,73**		**1,84**		**0,22**		**0**		**3,28**

Tabela 4.9. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias FT. O símbolo "-" significa "não avaliado nessa instância". (conclusão)

			GSA		LSGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**
FT06	6 × 6	55	–		55	0
FT10	10 × 10	930	937	0,0075	976	4,94
FT20	20 × 5	1165	1174	0,0077	1209	3,77
MErr				**0,0076**		**2,90**

Fonte: Elaborada pela autora.

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

			GIFA-mXLSGA		**mXLSGA**		IHHO		DCSA		SSO		LSO	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA01	10 × 5	666	666	0,00	666	0,00	666	0,00	666	0,00	–		666	0,00
LA02	10 × 5	655	655	0,00	655	0,00	671	2,44	655	0,00	–		–	
LA03	10 × 5	597	597	0,00	597	0,00	626	4,86	597	0,00	–		628	5,19
LA04	10 × 5	590	590	0,00	590	0,00	616	4,41	590	0,00	–		–	
LA05	10 × 5	593	593	0,00	593	0,00	593	0,00	593	0,00	–		–	
LA06	15 × 5	926	926	0,00	926	0,00	926	0,00	926	0,00	–		–	
LA07	15 × 5	890	890	0,00	890	0,00	902	1,35	890	0,00	–		–	
LA08	15 × 5	863	863	0,00	863	0,00	863	0,00	863	0,00	–		–	
LA09	15 × 5	951	951	0,00	951	0,00	951	0,00	951	0,00	–		–	
LA10	15 × 5	958	958	0,00	958	0,00	958	0,00	958	0,00	–		–	
LA11	20 × 5	1222	1222	0,00	1222	0,00	1222	0,00	1222	0,00	–		–	
LA12	20 × 5	1039	1039	0,00	1039	0,00	1039	0,00	1039	0,00	–		–	
LA13	20 × 5	1150	1150	0,00	1150	0,00	1150	0,00	1150	0,00	–		–	
LA14	20 × 5	1292	1292	0,00	1292	0,00	1292	0,00	1292	0,00	–		–	
LA15	20 × 5	1207	1207	0,00	1207	0,00	1262	4,56	1207	0,00	–		–	
LA16	10 × 10	945	945	0,00	945	0,00	1027	8,68	945	0,00	–		–	
LA17	10 × 10	784	784	0,00	784	0,00	838	6,89	784	0,00	–		–	
LA18	10 × 10	848	848	0,00	848	0,00	907	6,96	848	0,00	–		–	
LA19	10 × 10	842	842	0,00	842	0,00	926	9,98	842	0,00	–		–	
LA20	10 × 10	902	902	0,00	902	0,00	967	7,21	907	0,55	–		–	
LA21	15 × 10	1046	1052	0,57	1059	1,24	1292	23,52	1061	1,43	–		–	

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

			GIFA-mXLSGA		**mXLSGA**		IHHO		DCSA		SSO		LSO	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA22	15 × 10	927	927	0,00	935	0,86	1111	19,85	935	0,86	–		–	
LA23	15 × 10	1032	1032	0,00	1032	0,00	1231	19,28	1032	0,00	–		–	
LA24	15 × 10	935	940	0,53	946	1,18	1106	18,29	946	1,18	–		–	
LA25	15 × 10	977	984	0,72	986	0,92	1163	19,04	–		–		–	
LA26	20 × 10	1218	1218	0,00	1218	0,00	1511	24,06	1218	0,00	–		–	
LA27	20 × 10	1235	1261	2,11	1269	2,75	1546	25,18	–		–		–	
LA28	20 × 10	1216	1239	1,89	1239	1,89	1509	24,10	–		–		–	
LA29	20 × 10	1152	1190	3,30	1201	4,25	1498	30,03	–		–		–	
LA30	20 × 10	1355	1355	0,00	1355	0,00	1592	17,49	1355	0,00	–		–	
LA31	30 × 10	1784	1784	0,00	1784	0,00	2085	16,87	1784	0,00	–		–	
LA32	30 × 10	1850	1850	0,00	1850	0,00	2168	17,19	1850	0,00	–		–	
LA33	30 × 10	1719	1719	0,00	1719	0,00	1975	14,89	1719	0,00	–		–	
LA34	30 × 10	1721	1721	0,00	1721	0,00	2034	18,19	1721	0,00	–		–	
LA35	30 × 10	1888	1888	0,00	1888	0,00	2230	18,11	1888	0,00	–		–	
LA36	15 × 15	1268	1295	2,13	1295	2,13	1541	21,53	–		–		–	
LA37	15 × 15	1397	1407	0,72	1415	1,29	1770	26,70	–		–		–	
LA38	15 × 15	1196	1246	4,18	1246	4,18	1555	30,02	–		–		–	
LA39	15 × 15	1233	1258	2,03	1258	2,03	1585	28,55	–		–		–	
LA40	15 × 15	1222	1243	1,72	1243	1,72	1562	27,82	–		1528	25,04	–	
MErr				**0,50**		**0,61**		**12,45**		**0,13**		**25,04**		**2,60**

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

		HIGA		NGPSO		SSS		NAGA		GA-CPG-GT		DWPA	
Problema	**Tamanho**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA01	10 × 5	666	0,00	666	0,00	666	0,00	666	0,00	666	0,00	666	0,00
LA02	10 × 5	655	0,00	655	0,00	655	0,00	655	0,00	655	0,00	655	0,00
LA03	10 × 5	597	0,00	597	0,00	597	0,00	603	1,01	597	0,00	614	2,85
LA04	10 × 5	590	0,00	590	0,00	590	0,00	–		590	0,00	598	1,36
LA05	10 × 5	593	0,00	593	0,00	593	0,00	–		593	0,00	593	0,00
LA06	15 × 5	–		926	0,00	926	0,00	926	0,00	926	0,00	926	0,00
LA07	15 × 5	–		890	0,00	890	0,00	890	0,00	890	0,00	890	0,00
LA08	15 × 5	–		863	0,00	863	0,00	863	0,00	863	0,00	863	0,00
LA09	15 × 5	–		951	0,00	951	0,00	–		951	0,00	951	0,00
LA10	15 × 5	–		958	0,00	958	0,00	–		958	0,00	958	0,00
LA11	20 × 5	–		1222	0,00	1222	0,00	1222	0,00	1222	0,00	1222	0,00
LA12	20 × 5	–		1039	0,00	–		1039	0,00	1039	0,00	1039	0,00
LA13	20 × 5	–		1150	0,00	–		–		1150	0,00	1150	0,00
LA14	20 × 5	–		1292	0,00	–		–		1292	0,00	1292	0,00
LA15	20 × 5	–		1207	0,00	–		–		1207	0,00	1273	5,47
LA16	10 × 10	964	2,01	945	0,00	947	0,21	–		946	0,11	993	5,08
LA17	10 × 10	785	0,13	794	1,28	–		–		784	0,00	793	1,15
LA18	10 × 10	848	0,00	848	0,00	–		–		848	0,00	861	1,53
LA19	10 × 10	852	1,19	842	0,00	–		–		842	0,00	888	5,46
LA20	10 × 10	907	0,55	908	0,67	–		–		907	0,55	934	3,55
LA21	15 × 10	–		1183	13,10	1076	2,87	1046	0,00	1090	4,21	1105	5,64
LA22	15 × 10	–		927	0,00	–		–		954	2,91	989	6,69
LA23	15 × 10	–		1032	0,00	–		–		1032	0,00	1051	1,84
LA24	15 × 10	–		968	3,53	–		–		974	4,17	988	5,67
LA25	15 × 10	–		977	0,00	–		–		999	2,25	1039	6,35
LA26	20 × 10	–		1218	0,00	–		1218	0,00	1237	1,56	1303	6,98
LA27	20 × 10	–		1394	12,87	1256	1,70	–		1313	6,32	1346	8,99
LA28	20 × 10	–		1216	0,00	–		–		1280	5,26	1291	6,17
LA29	20 × 10	–		1280	11,11	–		–		1247	8,25	1275	10,68
LA30	20 × 10	–		1355	0,00	–		–		1367	0,89	1389	2,51
LA31	30 × 10	–		1784	0,00	1784	0,00	1793	0,50	1784	0,00	1784	0,00

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

		HIGA		NGPSO		SSS		NAGA		GA-CPG-GT		DWPA	
Problema	**Tamanho**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA32	30 × 10	–		1850	0,00	–		–		1850	0,00	1850	0,00
LA33	30 × 10	–		1719	0,00	–		–		1719	0,00	1719	0,00
LA34	30 × 10	–		1721	0,00	–		–		1725	0,23	1788	3,89
LA35	30 × 10	–		1888	0,00	–		–		1888	0,00	1947	3,13
LA36	15 × 15	–		1408	11,04	1304	2,84	–		1308	3,15	1388	9,46
LA37	15 × 15	–		1515	8,45	–		–		1489	6,59	1486	6,37
LA38	15 × 15	–		1196	0,00	–		–		1275	6,61	1339	11,96
LA39	15 × 15	–		1662	34,79	–		–		1290	4,62	1334	8,19
LA40	15 × 15	–		1222	0,00	1252	2,45	–		1252	2,45	1347	10,23
MErr			**0,39**		**2,42**		**0,59**		**0,14**		**1,50**		**3,53**

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

			MeCSO		GWO		HGWO		MA		IPB-GA		NIMGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA01	10 × 5	666	666	0,00	666	0,00	666	0,00	666	0,00	666	0,00	666	0,00
LA02	10 × 5	655	655	0,00	655	0,00	655	0,00	655	0,00	655	0,00	655	0,00
LA03	10 × 5	597	599	0,34	597	0,00	597	0,00	597	0,00	599	0,34	597	0,00
LA04	10 × 5	590	590	0,00	590	0,00	590	0,00	590	0,00	590	0,00	590	0,00
LA05	10 × 5	593	593	0,00	593	0,00	593	0,00	593	0,00	593	0,00	593	0,00
LA06	15 × 5	926	926	0,00	926	0,00	926	0,00	926	0,00	926	0,00	926	0,00
LA07	15 × 5	890	890	0,00	890	0,00	890	0,00	890	0,00	890	0,00	890	0,00
LA08	15 × 5	863	863	0,00	863	0,00	863	0,00	863	0,00	863	0,00	863	0,00
LA09	15 × 5	951	951	0,00	951	0,00	951	0,00	951	0,00	951	0,00	951	0,00
LA10	15 × 5	958	958	0,00	958	0,00	958	0,00	958	0,00	958	0,00	958	0,00
LA11	20 × 5	1222	1222	0,00	1222	0,00	1222	0,00	1222	0,00	1222	0,00	1222	0,00
LA12	20 × 5	1039	1039	0,00	1039	0,00	1039	0,00	1039	0,00	1039	0,00	1039	0,00
LA13	20 × 5	1150	1150	0,00	1150	0,00	1150	0,00	1150	0,00	1150	0,00	1150	0,00
LA14	20 × 5	1292	1292	0,00	1292	0,00	1292	0,00	1292	0,00	1292	0,00	1292	0,00
LA15	20 × 5	1207	1207	0,00	1207	0,00	1207	0,00	1207	0,00	1207	0,00	1207	0,00

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

			MeCSO		GWO		HGWO		MA		IPB-GA		NIMGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA16	10 × 10	945	950	0,53	956	1,16	959	1,48	946	0,11	946	0,11	946	0,11
LA17	10 × 10	784	784	0,00	790	0,77	784	0,00	784	0,00	784	0,00	784	0,00
LA18	10 × 10	848	851	0,35	859	1,30	857	1,06	858	1,18	853	0,59	848	0,00
LA19	10 × 10	842	850	0,95	845	0,36	845	0,36	–		866	2,85	842	0,00
LA20	10 × 10	902	911	1,00	937	3,88	946	4,88	–		913	1,22	907	0,55
LA21	15 × 10	1046	1085	3,73	1090	4,21	–		1081	3,35	1081	3,35	1058	1,15
LA22	15 × 10	927	–		970	4,64	–		954	2,91	970	4,64	937	1,08
LA23	15 × 10	1032	–		1032	0,00	–		1032	0,00	1032	0,00	1032	0,00
LA24	15 × 10	935	–		982	5,03	–		976	4,39	1002	7,17	947	1,28
LA25	15 × 10	977	–		1008	3,17	–		999	2,25	1023	4,71	992	1,54
LA26	20 × 10	1218	–		1239	1,72	–		–		1273	4,52	1218	0,00
LA27	20 × 10	1235	–		1290	4,45	–		–		1317	6,64	1269	2,75
LA28	20 × 10	1216	–		1263	3,87	–		–		1288	5,92	1253	3,04
LA29	20 × 10	1152	–		1244	7,99	–		–		1233	7,03	1247	8,25
LA30	20 × 10	1355	–		1355	0,00	–		–		1377	1,62	1355	0,00
LA31	30 × 10	1784	–		1784	0,00	–		1784	0,00	1784	0,00	1784	0,00
LA32	30 × 10	1850	–		1850	0,00	–		1868	0,97	1851	0,05	1850	0,00
LA33	30 × 10	1719	–		1719	0,00	–		–		1719	0,00	1719	0,00
LA34	30 × 10	1721	–		1721	0,00	–		–		1749	1,63	1721	0,00
LA35	30 × 10	1888	–		1888	0,00	–		1901	0,69	1888	0,00	1888	0,00
LA36	15 × 15	1268	–		1311	3,39	–		–		1334	5,21	1293	1,97
LA37	15 × 15	1397	–		–		–		–		1467	5,01	1439	3,01
LA38	15 × 15	1196	–		–		–		1258	5,18	1278	6,86	1222	2,17
LA39	15 × 15	1233	–		–		–		–		1296	5,11	1259	2,11
LA40	15 × 15	1222	–		–		–		–		1284	5,07	1246	1,96
MErr				**0,33**		**1,28**		**0,39**		**0,78**		**1,99**		**0,77**

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (continua)

			aLSGA		PaGA		LSGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA01	10 × 5	666	666	0,00	666	0,00	666	0,00
LA02	10 × 5	655	655	0,00	655	0,00	655	0,00
LA03	10 × 5	597	606	1,51	617	3,35	597	0,00
LA04	10 × 5	590	593	0,51	607	2,88	590	0,00
LA05	10 × 5	593	593	0,00	593	0,00	593	0,00
LA06	15 × 5	926	926	0,00	926	0,00	926	0,00
LA07	15 × 5	890	890	0,00	890	0,00	890	0,00
LA08	15 × 5	863	863	0,00	863	0,00	863	0,00
LA09	15 × 5	951	951	0,00	951	0,00	951	0,00
LA10	15 × 5	958	958	0,00	958	0,00	958	0,00
LA11	20 × 5	1222	1222	0,00	1223	0,08	1222	0,00
LA12	20 × 5	1039	1039	0,00	1039	0,00	1039	0,00
LA13	20 × 5	1150	1150	0,00	1150	0,00	1150	0,00
LA14	20 × 5	1292	1292	0,00	1292	0,00	1292	0,00
LA15	20 × 5	1207	1207	0,00	1273	5,47	1207	0,00
LA16	10 × 10	945	946	0,11	994	5,19	959	1,48
LA17	10 × 10	784	784	0,00	793	1,15	792	1,02
LA18	10 × 10	848	848	0,00	860	1,42	857	1,06
LA19	10 × 10	842	852	1,19	873	3,68	860	2,14
LA20	10 × 10	902	907	0,55	912	1,11	907	0,55
LA21	15 × 10	1046	1068	2,10	1146	9,56	1114	6,50
LA22	15 × 10	927	956	3,13	1007	8,63	989	6,69
LA23	15 × 10	1032	1032	0,00	1033	0,10	1035	0,29
LA24	15 × 10	935	966	3,32	1012	8,24	1032	10,37
LA25	15 × 10	977	1002	2,56	1067	9,21	1047	7,16
LA26	20 × 10	1218	1223	0,41	1323	8,62	1307	7,31
LA27	20 × 10	1235	1281	3,72	1359	10,04	1350	9,31
LA28	20 × 10	1216	1245	2,38	1369	12,58	1312	7,89
LA29	20 × 10	1152	1230	6,77	1322	14,76	1311	13,80
LA30	20 × 10	1355	1355	0,00	1437	6,05	1451	7,08
LA31	30 × 10	1784	1784	0,00	1844	3,36	1784	0,00

Tabela 4.10. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias LA. O símbolo "-" significa "não avaliado nessa instância". (conclusão)

			aLSGA		PaGA		LSGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
LA32	30 × 10	1850	1850	0,00	1907	3,08	1850	0,00
LA33	30 × 10	1719	1719	0,00	–		1745	1,51
LA34	30 × 10	1721	1721	0,00	–		1784	3,66
LA35	30 × 10	1888	1888	0,00	–		1958	3,71
LA36	15 × 15	1268	–		–			
LA37	15 × 15	1397	–		–			
LA38	15 × 15	1196	–		–			
LA39	15 × 15	1233	–		–			
LA40	15 × 15	1222	–		–			
MErr				**0,81**		**3,70**		**2,62**

Fonte: Elaborada pela autora.

Tabela 4.11. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias ORB. O símbolo "-" significa "não avaliado nessa instância". (continua)

			GIFA-mXLSGA		**mXLSGA**		IHHO		SSO		MeCSO		IPB-GA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
ORB01	10 × 10	1059	1068	0,85	1068	0,85	1282	21,06	–		1093	3,21	1099	3,78
ORB02	10 × 10	888	889	0,11	889	0,11	984	10,81	–		907	2,14	906	2,03
ORB03	10 × 10	1005	1023	1,79	1023	1,79	1201	19,50	–		1011	0,60	1056	5,07
ORB04	10 × 10	1005	1005	0,00	1005	0,00	1147	14,13	–		1024	1,89	1032	2,69
ORB05	10 × 10	887	887	0,00	889	0,23	1013	14,21	–		891	0,45	909	2,48
ORB06	10 × 10	1010	1013	0,30	1019	0,89	1192	18,02	–		1016	0,59	1038	2,77
ORB07	10 × 10	397	397	0,00	397	0,00	453	14,11	–		402	1,26	411	3,53
ORB08	10 × 10	899	907	0,89	907	0,89	1046	16,35	–		907	0,89	917	2,00
ORB09	10 × 10	934	940	0,64	940	0,64	1045	11,88	–		944	1,07	–	
ORB10	10 × 10	944	944	0,00	944	0,00	1053	11,55	1114	18,01	–		–	
MErr				**0,46**		**0,54**		**15,16**		**18,01**		**1,34**		**3,04**

Tabela 4.11. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias ORB. O símbolo "-" significa "não avaliado nessa instância". (conclusão)

			NIMGA		aLSGA		PaGA		LSGA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
ORB01	10 × 10	1059	1059	0,00	1092	3,12	1149	8,50	1088	2,74
ORB02	10 × 10	888	890	0,23	894	0,68	929	4,62	921	3,72
ORB03	10 × 10	1005	1026	2,09	1029	2,39	1129	12,34	1041	3,58
ORB04	10 × 10	1005	1019	1,39	1016	1,09	1062	5,67	1052	4,68
ORB05	10 × 10	887	893	0,68	901	1,58	936	5,52	903	1,80
ORB06	10 × 10	1010	1012	0,20	1028	1,78	1060	4,95	1062	5,15
ORB07	10 × 10	397	397	0,00	405	2,02	416	4,79	408	2,77
ORB08	10 × 10	899	909	1,11	914	1,67	1010	12,35	908	1,00
ORB09	10 × 10	934	942	0,86	943	0,96	994	6,42	980	4,93
ORB10	10 × 10	944	–				–		–	
MErr				**0,73**		**1,70**		**7,24**		**3,37**

Fonte: Elaborada pela autora.

Tabela 4.12. Comparação de resultados computacionais entre o material proposto e outros algoritmos considerando-se as instâncias ABZ. O símbolo "-" significa "não avaliado nessa instância".

			GIFA-mXLSGA		**mXLSGA**		GA-CPG-GT		MeCSO		IPB-GA	
Problema	**Tamanho**	**Solução ótima**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**	**Melhor**	**Erro**
ABZ05	10 × 10	1234	1234	0,00	1234	0,00	1238	0,32	1236	0,16	1241	0,57
ABZ06	10 × 10	943	943	0,00	943	0,00	947	0,42	949	0,64	964	2,23
ABZ07	20 × 15	656	657	0,15	695	5,95	–	–			719	9,60
ABZ08	20 × 15	648	713	10,03	713	10,03	–	–			738	13,89
ABZ09	20 × 15	679	680	0,15	721	6,19	–	–			742	9,28
MErr				**2,07**		**4,43**		**0,37**		**0,40**		**7,11**

Fonte: Elaborada pela autora.

Conforme apresentado nas Tabelas 4.9-4.12, mXLSGA encontrou a melhor solução conhecida em 100% de instâncias FT, 70% de instâncias LA, 30% de instâncias ORB e 40% de instâncias ABZ. A proposta mXLSGA alcançou em 28 instâncias LA a melhor solução conhecida e obteve um erro relativo médio (MErr) de 0.61. Alguns métodos tais como o SSS e

o HGWO obtiveram um erro médio inferior ao mXLSGA, assumindo 0.59 e 0.38, respectivamente, mas estes não avaliaram suas abordagens em todas as instâncias LA. Se consideradas apenas as instâncias que foram avaliadas por estes métodos, o mXLSGA apresenta MErr menor. Por exemplo, o MErr apresentado pelo mXLSGA sobre as mesmas instâncias LA avaliadas pelo SSS é igual a 0.46. Analogamente, se consideradas apenas as instâncias LA avaliadas pelo HGWO, o método proposto obteria 0 de MErr. Especificamente, em instâncias FT, o mXLSGA alcançou em 3 instâncias a melhor solução conhecida e obteve um MErr de 0.00. Em instâncias ORB, o mXLSGA alcançou em 3 instâncias a melhor solução conhecida e obteve um MErr de 0.54. Nas instâncias ABZ, o mXLSGA alcançou em 2 instâncias a melhor solução conhecida e obteve um MErr de 4.46. O método que alcançou o menor erro relativo foi GA-CPG-GT, mas o trabalho em questão não avaliou todas as instâncias ABZ, e se forem consideradas apenas as instâncias em que GA-CPG-GT foi avaliado, o mXLSGA obteve erro relativo médio 0.00, ou seja, o mXLSGA alcançou a melhor solução conhecida nas primeiras 2 instâncias de ORB.

Em particular, o método proposto superou a técnica na qual foi baseado, que neste caso é o aLSGA. Em casos de LA, o mXLSGA encontrou 6 melhores soluções conhecidas a mais do que aLSGA. O aLSGA obteve nas instâncias LA um MErr de 0.80 e o mXLSGA obteve um MErr de 0.61, mas o aLSGA foi testado apenas nas primeiras 35 instâncias LA. Assim, se considerado o MErr apenas para as primeiras 35 instâncias LA, o mXLSGA obtém um MErr de 0.37, que é inferior à metade do valor obtido pelo aLSGA. Para instâncias ORB, o mXLSGA obteve um MErr menor que um terço do obtido pelo aLSGA. A melhoria alcançada pelo mXLSGA é uma consequência da inserção dos operadores propostos.

Com respeito ao GIFA, ao analisar as Tabelas 4.9-4.12, pode-se verificar que o operador de melhoramento genético proposto foi capaz de melhorar a capacidade de busca do método mXLSGA. Especificamente, considerando apenas as instâncias LA, o uso do operador proposto foi capaz de reduzir a magnitude do erro relativo médio em 0.12, o que corresponde a uma redução de 19.67% de seu valor. Em outras palavras, o operador GIFA tornou o método mXLSGA capaz de encontrar o melhor $makespan$ conhecido em 72.5% das instâncias LA, obtendo erro relativo médio de 0.50, o menor entre todos os métodos. Com respeito às instâncias FT, o operador proposto GIFA não comprometeu a capacidade de busca do mXLSGA, fazendo com que fossem encontradas as melhores soluções conhecidas em todas as instâncias. No caso das instâncias ORB, o GIFA melhorou o desempenho do mXLSGA na

instância ORB05 e tornou o método capaz de encontrar a melhor solução conhecida na ORB06, reduzindo o erro médio da técnica de 0.54 para 0.46. Além disso, com respeito às instâncias ABZ, o erro médio do GIFA-mXLSGA é menor que a metade do erro do mXLSGA, uma vez que o operador de melhoramento genético proposto melhorou os resultados da técnica base nas instâncias ABZ07 e ABZ09. Em resumo, pode-se ressaltar alguns pontos ao analisar os resultados referentes às Tabelas 4.9-4.12:

- Não houve piora dos resultados em nenhuma instância com a utilização do operador GIFA;
- O método GIFA-mXLSGA obteve o menor erro relativo;
- O operador GIFA tornou o mXLSGA capaz de encontrar a melhor solução conhecida na instância LA 22;
- O operador GIFA melhorou os resultados do mXLSGA em 7 instâncias LA;
- O operador GIFA tornou o mXLSGA capaz de encontrar a melhor solução conhecida na instância ORB06 e melhorou a solução obtida na ORB05,
- O operador GIFA reduziu o erro médio do mXLSGA em 53% nas instâncias ABZ.

De posse destes resultados, pode-se ver que nas instâncias JSSP testadas, o mXLSGA proposto resulta em soluções melhores ou iguais àquelas obtidas pelos algoritmos de última geração comparados. Além disso, nota-se que o operador de melhoramento genético proposto é eficaz em aumentar a eficiência da técnica base mXLSGA em encontrar boas soluções.

4.5 Considerações finais

Neste capítulo, foram apresentados e discutidos os resultados alcançados e que fazem parte da validação da proposta da tese. No caso, o material foi avaliado em três diferentes estudos de caso: o primeiro dedicado à análise dos operadores de busca local propostos funcionando de maneira isolada e conjunta; o segundo dedicado à análise do efeito causado pela adição do operador de melhoramento genético em técnicas do tipo GA; e o terceiro dedicado a averiguar a capacidade de busca por boas soluções considerando todo o material proposto.

De acordo com os resultados obtidos, pôde-se confirmar que o *framework* proposto supera os atuais métodos do tipo GA na maioria das situações estipuladas, o que torna o material desenvolvido competitivo às técnicas desta categoria. Além disso, a capacidade de busca de boas soluções para o JSSP do *framework* proposto foi comparada com uma grande variedade de metaheurísticas que compõem a literatura especializada atual neste tema e observou-se que o método proposto também se mostra competitivo a estas estratégias de otimização. Tais resultados corroboraram para a validação das hipóteses aqui propostas.

Capítulo 5

CONCLUSÃO

O objetivo principal deste trabalho de doutorado foi desenvolver uma abordagem baseada em um algoritmo do tipo GA para redução de *makespan* em uma programação da produção JSSP. Para atingir este objetivo, foi apresentada uma profunda discussão sobre técnicas existentes no tema, bem como o detalhamento da fundamentação daquelas que serviram de inspiração para a confecção do material que compõe os avanços apresentados neste texto.

Para obter avanços na solução de instâncias de JSSP, foi proposto neste trabalho o desenvolvimento de uma nova metaheurística do tipo GA. No caso, foram conduzidos progressos em duas frentes de pesquisa: no aprimoramento e na generalização de operadores de busca local e no desenvolvimento de um operador de melhoramento genético baseado na análise de frequência de genes presentes em indivíduos bem adaptados da população. A junção dessas técnicas configurou um *framework* especializado na solução de instâncias de JSSP e que se mostrou promissor no tema.

Para observar o funcionamento de todos os componentes do material proposto, três estudos de caso diferentes foram considerados. Em resumo, destacam-se algumas constatações:

- No primeiro estudo de caso, as análises foram divididas em duas situações, sendo que na primeira foram avaliados de forma individual todos os operadores de busca local que compõem o método. Assim, constatou-se que todos os operadores corroboram de forma conjunta no método para a melhoria de resultados obtidos. Isto é, nenhum operador de forma isolada obteve resultados melhores que o método formado por todos os operadores de busca local. Além disso, foi possível observar nesta situação que o operador mais influente é o

operador de busca massiva, o qual possui poder de busca maior que os propostos operadores de mutação e multi-*crossover* modificados. Na segunda situação considerada neste estudo de caso, o mXLSGA foi comparado com as técnicas nas quais sua modelagem foi baseada. No caso, as técnicas do tipo GA que compõem o estado da arte do tema. De acordo com os resultados obtidos, o mXLSGA obteve as melhores medidas estatísticas em comparação com as demais técnicas. Entretanto, o tempo computacional utilizado pela técnica para obter estes resultados considerando-se 100 iterações para cada método foi o maior. A análise foi concluída mostrando que, de acordo com as curvas de convergência, o mXLSGA precisa de apenas 5% de todas as iterações para alcançar a melhor solução conhecida em instâncias mais simples, e de apenas metade das iterações para obter a melhor solução conhecida ou muito próxima desta em instâncias maiores. Fato que não ocorre com as demais técnicas consideradas. Isto demonstrou que o tempo não é um problema para o mXLSGA, uma vez que este funciona bem mesmo com o uso de configurações mais restritas.

- No segundo estudo de caso, o operador de melhoramento genético proposto, o GIFA, foi anexado a cinco diferentes metaheurísticas do tipo GA que compõem o estado da arte da literatura especializada, sendo o mXLSGA uma destas. Todas as técnicas e suas versões com o GIFA foram executadas 35 vezes e alguns fatos puderam ser observados. No caso, durante as avaliações, constatou-se que o GIFA tornou todas metaheurísticas mais estáveis, uma vez que reduziu a média e o desvio padrão em todos os casos em que era possível. Além disso, o pior valor apresentado por cada técnica durante suas execuções também foi reduzido na maioria dos casos. Algo semelhante ocorreu com o indicador de melhor solução encontrada com cada técnica. Estes fatos corroboram com a suposição de que o GIFA auxilia as metaheurísticas do tipo GA a encontrar melhores soluções.
- No último estudo de caso, foi avaliada a capacidade do material proposto em calcular boas soluções para instâncias JSSP e foi feita a comparação dos resultados obtidos com metaheurísticas dos mais variados tipos e inspirações. Nesta situação, pôde-se observar que o mXLSGA apresenta um poder de busca muito competitivo em comparação com os trabalhos que compõem o estado da arte no tema. No caso, o mXLSGA apresentou o segundo menor erro relativo

médio na maioria das situações consideradas, tendo superado todas as técnicas nas quais suas componentes foram baseadas. Na verdade, o único método que apresentou erro relativo médio menor foi a versão do mXLSGA com o GIFA, servindo também de confirmação para a suposição de que o GIFA é capaz de auxiliar metaheurísticas do tipo GA a ampliar seu poder de busca.

Analisando os três casos de estudos apresentados neste trabalho, conclui-se que o material proposto consiste de um método robusto, com a capacidade de obter bons resultados em instâncias de complexidades variadas e que possui uma taxa de convergência mais rápida ao ser comparado com outros métodos do tipo GA. O *framework* apresenta resultados melhores ou no mínimo competitivos ao ser comparado com outros métodos presentes na literatura especializada.

Desta forma, como principais contribuições deste trabalho destacam-se:

- Um novo *framework* de técnicas do tipo GA com estratégias de busca local aprimoradas;
- Um novo operador de melhoramento genético eficaz e com menor custo computacional,
- O avanço no problema de programação da produção JSSP.

É válido ressaltar que as contribuições deste trabalho foram e continuam sendo divulgadas em conferências e periódicos de elevada qualificação nacional e internacional. Em detalhes, a teoria que compôs o mXLSGA foi publicada em seus estudos preliminares na *International Conference on Artificial Intelligence and Soft Computing* (ICAISC) de 2020 (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020c) e em sua forma estendida no periódico *Sensors* da editora MDPI em 2020 (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020a). A teoria que compõe o operador de melhoramento genético deu origem a dois trabalhos preliminares que foram publicados na *International Conferente on Agents and Artificial Intelligence* (ICAART) de 2020 (VIANA; MORANDIN JUNIOR; CONTRERAS, 2020d) e na ICAISC de 2021 (VIANA; CONTRERAS; MORANDIN JUNIOR, 2021), sendo que a versão estendida desta teoria se encontra em processo de revisão por pares na revista *Sensors* da MDPI.

Para trabalhos futuros, outras estratégias devem ser consideradas para a composição do *framework* proposto. Especificamente, uma maior variedade de funções de cruzamento e mutação devem ser avaliadas. Além disso, outras técnicas utilizadas para aprimorar busca local, tais como aprendizado profundo por reforço, podem substituir a etapa de busca local massiva. Ademais, o uso de técnicas de aprendizado profundo para detectar relevância e conduzir o

processo de melhoramento genético deve ser investigado. Outras categorias de GA, tais como GA multi-população e GA adaptativo, devem ser consideradas para servir de método básico para o *framework* proposto. Ainda, uma etapa de controle de diversidade populacional, considerando a diversidade como uma função objetivo, será acrescentada ao *framework*, sendo que a confecção da teoria está nas etapas de análise e implementação e conta com a parceria de um pesquisador da *Università di Padova*. Também será avaliado o funcionamento do método em problemas de otimização combinatória semelhantes, tais como problema de programação do tipo *job shop* flexível, problema de programação do tipo *flow shop*, entre outros.

REFERÊNCIAS

AARTS, Emile H L *et al*. A computational study of local search algorithms for job shop scheduling. **ORSA Journal on Computing**, [*s. l.*], v. 6, n. 2, p. 118–125, 1994.

ADAMS, Joseph; BALAS, Egon; ZAWACK, Daniel. The shifting bottleneck procedure for job shop scheduling. **Management science**, [*s. l.*], v. 34, n. 3, p. 391–401, 1988.

AHMADIZAR, Fardin; FARAHANI, Mehdi Hosseinabadi. A novel hybrid genetic algorithm for the open shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 62, n. 5–8, p. 775–787, 2012.

AKRAM, Kashif; KAMAL, Khurram; ZEB, Alam. Fast simulated annealing hybridized with quenching for solving job shop scheduling problem. **Applied Soft Computing**, [*s. l.*], v. 49, p. 510–523, 2016.

ALKHATEEB, Faisal; ABED-ALGUNI, Bilal H; AL-ROUSAN, Mohammad Hani. Discrete hybrid cuckoo search and simulated annealing algorithm for solving the job shop scheduling problem. **The Journal of Supercomputing**, [*s. l.*], 2021. Disponível em: https://doi.org/10.1007/s11227-021-04050-6

AMARAL, Laurence Rodrigues; HRUSCHKA JR, Estevam Rafael. Transgenic: An evolutionary algorithm operator. **Neurocomputing**, [*s. l.*], v. 127, p. 104–113, 2014.

AMJAD, Muhammad Kamal *et al*. Recent research trends in genetic algorithm based flexible job shop scheduling problems. **Mathematical Problems in Engineering**, [*s. l.*], v. 2018, 2018.

ANAND, Ellur; PANNEERSELVAM, Ramasamy. A study of crossover operators for genetic algorithm and proposal of a new crossover operator to solve open shop scheduling problem. **American Journal of Industrial and Business Management**, [*s. l.*], v. 6, n. 06, p. 774, 2016.

APPLEGATE, David; COOK, William. A computational study of the job-shop scheduling problem. **ORSA Journal on computing**, [*s. l.*], v. 3, n. 2, p. 149–156, 1991.

ARTIGUES, Christian; FEILLET, Dominique. A branch and bound method for the job-shop problem with sequence-dependent setup times. **Annals of Operations Research**, [*s. l.*], v. 159, n. 1, p. 135–159, 2008.

ASADZADEH, Leila. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. **Computers & Industrial Engineering**, [*s. l.*], v. 85, p. 376–383, 2015.

ASADZADEH, Leila; ZAMANIFAR, Kamran. An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. **Mathematical and Computer Modelling**, [*s. l.*], v. 52, n. 11–12, p. 1957–1965, 2010.

ASHOUR, Said; HIREMATH, S R. A branch-and-bound approach to the job-shop scheduling problem. **International Journal of Production Research**, [*s. l.*], v. 11, n. 1, p. 47–58, 1973.

ATAY, Yilmaz; KODAZ, Halife. Optimization of job shop scheduling problems using modified clonal selection algorithm. **Turkish Journal of Electrical Engineering & Computer Sciences**, [*s. l.*], v. 22, n. 6, p. 1528–1539, 2014.

BANHARNSAKUN, Anan; SIRINAOVAKUL, Booncharoen; ACHALAKUL, Tiranee. Job shop scheduling with the best-so-far ABC. **Engineering Applications of Artificial Intelligence**, [*s. l.*], v. 25, n. 3, p. 583–593, 2012.

BARKER, Jeffrey R; MCMAHON, Graham B. Scheduling the general job-shop. **Management Science**, [*s. l.*], v. 31, n. 5, p. 594–598, 1985.

BAYKASOĞLU, Adil; HAMZADAYI, Alper; AKPINAR, Sener. Single Seekers Society (SSS): Bringing together heuristic optimization algorithms for solving complex problems. **Knowledge-Based Systems**, [*s. l.*], v. 165, p. 53–76, 2019. Disponível em: https://doi.org/https://doi.org/10.1016/j.knosys.2018.11.016

BAYKASOĞLU, Adil; HAMZADAYI, Alper; KÖSE, Simge Yelkenci. Testing the performance of teaching-learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. **Information Sciences**, [*s. l.*], v. 276, p. 204–218, 2014. Disponível em: https://doi.org/10.1016/j.ins.2014.02.056

BELLIFEMINE, Fabio; POGGI, Agostino; RIMASSA, Giovanni. Developing multi-agent systems with a FIPA-compliant agent framework. **Software: Practice and Experience**, [*s. l.*], v. 31, n. 2, p. 103–128, 2001.

BIERWIRTH, Christian; MATTFELD, Dirk C; KOPFER, Herbert. On permutation representations for scheduling problems. *In*: , 1996, Berlin, Heidelberg. (Hans-Michael Voigt et al., Org.)**Parallel Problem Solving from Nature --- PPSN IV**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. p. 310–318.

BINATO, S *et al.* A GRASP for job shop scheduling. *In*: ESSAYS AND SURVEYS IN METAHEURISTICS. [*S. l.*]: Springer, 2002. p. 59–79.

BONDAL, Akshata A. **Artificial immune systems applied to job shop scheduling**. 2008. - Ohio University, [*s. l.*], 2008.

BRUCKER, Peter; JURISCH, Bernd; SIEVERS, Bernd. A branch and bound algorithm for the job-shop scheduling problem. **Discrete applied mathematics**, [*s. l.*], v. 49, n. 1–3, p. 107–127, 1994.

ÇALI\CS, Banu; BULKAN, Serol. A research survey: review of AI solution strategies of job shop scheduling problem. **Journal of Intelligent Manufacturing**, [*s. l.*], v. 26, n. 5, p. 961–973, 2015.

CARLIER, Jacques. Ordonnancements a contraintes disjonctives. **RAIRO-Operations Research**, [*s. l.*], v. 12, n. 4, p. 333–350, 1978.

CARLIER, Jacques; PINSON, Éric. An algorithm for solving the job-shop problem. **Management science**, [*s. l.*], v. 35, n. 2, p. 164–176, 1989.

CHAUDHURI, Arindam; DE, Kajal. Job Scheduling Problem Using Rough Fuzzy Multilayer Perception Neural Networks. **Journal of Artificial Intelligence: Theory & Application**, [*s. l.*], v. 1, n. 1, 2010.

CHEGINI, Saeed Nezamivand; BAGHERI, Ahmad; NAJAFI, Farid. PSOSCALF: A new hybrid PSO based on Sine Cosine Algorithm and Levy flight for solving optimization problems. **Applied Soft Computing**, [*s. l.*], v. 73, p. 697–726, 2018.

CHEN, Xiaohan; ZHANG, Beike; GAO, Dong. Algorithm Based on Improved Genetic Algorithm for Job Shop Scheduling Problem. *In*: , 2019. **2019 IEEE International Conference on Mechatronics and Automation (ICMA)**. [*S. l.: s. n.*], 2019. p. 951–956. Disponível em: https://doi.org/10.1109/ICMA.2019.8816334

CHENG, Runwei; GEN, Mitsuo; TSUJIMURA, Yasuhiro. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. **Computers & industrial engineering**, [*s. l.*], v. 30, n. 4, p. 983–997, 1996.

CHONG, Chin Soon *et al.* A bee colony optimization algorithm to job shop scheduling. *In*: , 2006. **Proceedings of the 38th conference on Winter simulation**. [*S. l.: s. n.*], 2006. p. 1954–1961.

CONTRERAS, Rodrigo Colnago; MORANDIN JUNIOR, Orides; VIANA, Monique Simplicio. A New Local Search Adaptive Genetic Algorithm for the Pseudo-Coloring Problem. *In*: , 2020, Cham. **Advances in Swarm Intelligence**. Cham: Springer International Publishing, 2020. p. 349–361.

CROES, Georges A. A method for solving traveling-salesman problems. **Operations research**, [*s. l.*], v. 6, n. 6, p. 791–812, 1958.

DAO, Thi-Kien *et al.* Parallel bat algorithm for optimizing makespan in job shop scheduling problems. **Journal of Intelligent Manufacturing**, [*s. l.*], v. 29, n. 2, p. 451–462, 2018.

DAVIS, Lawrence. Applying adaptive algorithms to epistatic domains. *In*: , 1985. **IJCAI**. [*S. l.: s. n.*], 1985. p. 162–164.

DELL'AMICO, Mauro; TRUBIAN, Marco. Applying tabu search to the job-shop

scheduling problem. **Annals of Operations research**, [*s. l.*], v. 41, n. 3, p. 231–252, 1993.

DELLA CROCE, Federico; TADEI, Roberto; VOLTA, Giuseppe. A genetic algorithm for the job shop problem. **Computers & Operations Research**, [*s. l.*], v. 22, n. 1, p. 15–24, 1995.

DÍAZ, Henry *et al.* A memetic algorithm for minimizing the makespan in the Job Shop Scheduling problem. **Revista Facultad de Ingenier{\'\i}a**, [*s. l.*], v. 26, n. 44, p. 113–123, 2017.

ESSAFI, Imen; MATI, Yazid; DAUZÈRE-PÉRÈS, Stéphane. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. **Computers & Operations Research**, [*s. l.*], v. 35, n. 8, p. 2599–2616, 2008.

ESWARAMURTHY, V P; TAMILARASI, A. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 40, n. 9–10, p. 1004–1015, 2009.

FISCHETTI, Matteo; SALAZAR GONZÁLEZ, Juan José; TOTH, Paolo. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. **Operations Research**, [*s. l.*], v. 45, n. 3, p. 378–394, 1997.

FISHER, C.; THOMPSON, G.L. Probabilistic learning combinations of local job-shop scheduling rules. **Industrial Scheduling**, [*s. l.*], p. 225–251, 1963.

GABEL, Thomas; RIEDMILLER, Martin. Scaling adaptive agent-based reactive job-shop scheduling to large-scale problems. *In*: , 2007. **2007 IEEE Symposium on Computational Intelligence in Scheduling**. [*S. l.: s. n.*], 2007. p. 259–266.

GAO, Liang *et al.* An efficient memetic algorithm for solving the job shop scheduling problem. **Computers & Industrial Engineering**, [*s. l.*], v. 60, n. 4, p. 699–705, 2011. Disponível em: https://doi.org/10.1016/J.CIE.2011.01.003

GE, Hong-Wei *et al.* An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. **IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans**, [*s. l.*], v. 38, n. 2, p. 358–368, 2008.

GIFFLER, Bernard; THOMPSON, Gerald Luther. Algorithms for solving production-scheduling problems. **Operations research**, [*s. l.*], v. 8, n. 4, p. 487–503, 1960.

GOLDBERG, David E; HOLLAND, John H. Genetic algorithms and machine learning. **Machine learning**, [*s. l.*], v. 3, n. 2, p. 95–99, 1988.

GOLDBERG, David E; LINGLE, Robert; OTHERS. Alleles, loci, and the traveling salesman problem. *In*: , 1985. **Proceedings of an international conference on genetic algorithms and their applications**. [*S. l.: s. n.*], 1985. p. 154–159.

GONÇALVES, José Fernando; DE MAGALHÃES MENDES, Jorge José; RESENDE, Maur\ício G C. A hybrid genetic algorithm for the job shop scheduling problem. **European journal of operational research**, [*s. l.*], v. 167, n. 1, p. 77–95, 2005.

GONZÁLEZ FERNÁNDEZ, Miguel Ángel. Soluciones metaheur{\'\i}sticas al" job-shop scheduling problem with sequence-dependent setup times". [*s. l.*], 2011.

GROOVER, Mikell P. **Fundamentals of modern manufacturing: materials processes, and systems**. [*S. l.*]: John Wiley & Sons, 2007.

GU, Wenbin; TANG, Dunbing; ZHENG, Kun. Minimizing makespan in job-shop scheduling problem using an improved adaptive particle swarm optimization algorithm. *In*: , 2012. **2012 24th Chinese Control and Decision Conference (CCDC)**. [*S. l.: s. n.*], 2012. p. 3189–3193.

GUO, Pengfei; WANG, Xuezhi; HAN, Yingshi. The enhanced genetic algorithms for the optimization design. *In*: , 2010. **2010 3rd International Conference on Biomedical Engineering and Informatics**. [*S. l.: s. n.*], 2010. p. 2990–2994.

HAMZADAYI, Alper; BAYKASOĞLU, Adil; AKPINAR, Şener. Solving combinatorial optimization problems with single seekers society algorithm. **Knowledge-Based Systems**, [*s. l.*], v. 201–202, p. 106036, 2020. Disponível em: https://doi.org/https://doi.org/10.1016/j.knosys.2020.106036

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

HASAN, S.M.K.; SARKER, R.; ESSAM, D. Evolutionary scheduling with rescheduling option for sudden machine breakdowns. *In*: , 2010. **IEEE Congress on Evolutionary Computation (CEC)**. [*S. l.: s. n.*], 2010. p. 1–8.

HASAN, S M Kamrul *et al.* Memetic algorithms for solving job-shop scheduling problems. **Memetic Computing**, [*s. l.*], v. 1, n. 1, p. 69–83, 2009.

HOLLAND, John. Adaptation in natural and artificial systems: an introductory analysis with application to biology. **Control and artificial intelligence**, [*s. l.*], 1975.

HOLLAND, John Henry; OTHERS. **Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence**. [*S. l.*]: MIT press, 1992.

HUANG, Cheng *et al.* Job Shop Scheduling in Discrete Manufacturing Based on Improved Hybrid Lion Swarm Optimization. *In*: , 2021. **2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)**. [*S. l.: s. n.*], 2021. p. 438–443. Disponível em: https://doi.org/10.1109/ICIEA51954.2021.9516353

HUANG, Kuo-Ling; LIAO, Ching-Jong. Ant colony optimization combined with taboo

search for the job shop scheduling problem. **Computers & operations research**, [*s. l.*], v. 35, n. 4, p. 1030–1046, 2008.

JIA, H Z *et al.* Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. **Computers \& Industrial Engineering**, [*s. l.*], v. 53, n. 2, p. 313–320, 2007.

JIANG, Tianhua. A Hybrid Grey Wolf Optimization for Job Shop Scheduling Problem. **International Journal of Computational Intelligence and Applications**, [*s. l.*], v. 17, n. 03, p. 1850016, 2018.

JIANG, Tianhua; ZHANG, Chao. Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. **IEEE Access**, [*s. l.*], v. 6, p. 26231–26240, 2018.

JORAPUR, Vedavyasrao *et al.* Comparative Study of Different Representations in Genetic Algorithms for Job Shop Scheduling Problem. **Journal of Software Engineering and Applications**, [*s. l.*], v. 7, n. 07, p. 571, 2014.

JORAPUR, Vedavyasrao S *et al.* A promising initial population based genetic algorithm for job shop scheduling problem. **Journal of Software Engineering and Applications**, [*s. l.*], v. 9, n. 05, p. 208, 2016.

KALANTARI, Somayeh; SANIEEABADEH, Mohamad. An effective multi-population based hybrid genetic algorithm for job shop scheduling problem. **Bulletin of Electrical Engineering and Informatics**, [*s. l.*], v. 2, n. 1, p. 59–64, 2013.

KALSHETTY, Yoginath R; ADAMUTHE, Amol C; KUMAR, S Phani. Genetic algorithms with feasible operators for solving job shop scheduling problem. **Journal of Scientific Research**, [*s. l.*], v. 64, n. 1, 2020.

KÄSCHEL, Joachim *et al.* Algorithms for the job shop scheduling problem: A comparison of different methods. *In*: , 1999. **European Symposium on Intelligent Techniques**. [*S. l.: s. n.*], 1999. p. 3–4.

KATO, Edilson R R; MORANDIN, O; FONSECA, M A S. A Max-Min Ant System modeling approach for production scheduling in a FMS. *In*: , 2010. **2010 IEEE International Conference on Systems, Man and Cybernetics**. [*S. l.: s. n.*], 2010. p. 3977–3982.

KATO, Edilson R R; MORANDIN, O; FONSECA, M A S. Ant colony optimization algorithm for reactive production scheduling problem in the job shop system. *In*: , 2009. **2009 IEEE International Conference on Systems, Man and Cybernetics**. [*S. l.: s. n.*], 2009. p. 2199–2204.

KUMAR, Pardeep *et al.* An efficient genetic algorithm approach for minimising the makespan of job shop scheduling problems. **International Journal of Science, Engineering and Technology Research (IJSETR)**, [*s. l.*], v. 5, n. 5, 2016.

KURDI, Mohamed. A new hybrid island model genetic algorithm for job shop scheduling problem. **Computers & Industrial Engineering**, [*s. l.*], v. 88, p. 273–283, 2015.

KURDI, Mohamed. An effective genetic algorithm with a critical-path-guided Giffler and Thompson crossover operator for job shop scheduling problem. **International Journal of Intelligent Systems and Applications in Engineering**, [*s. l.*], v. 7, n. 1, p. 13–18, 2019.

KURDI, Mohamed. An effective new island model genetic algorithm for job shop scheduling problem. **Computers & operations research**, [*s. l.*], v. 67, p. 132–142, 2016.

LAMOS-D\'\IAZ, Henry *et al.* A memetic algorithm for minimizing the makespan in the Job Shop Scheduling problem. **Revista Facultad de Ingenier{\'\i}a**, [*s. l.*], v. 26, n. 44, p. 113–123, 2017.

LAWRENCE, S. Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). **Graduate School of Industrial Administration, Carnegie-Mellon University**, [*s. l.*], 1984.

LIAN, Zhigang; JIAO, Bin; GU, Xingsheng. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. **Applied mathematics and computation**, [*s. l.*], v. 183, n. 2, p. 1008–1017, 2006.

LIN, Lin; GEN, Mitsuo. Hybrid evolutionary optimisation with learning for production scheduling: state-of-the-art survey on algorithms and applications. **International Journal of Production Research**, [*s. l.*], v. 56, n. 1–2, p. 193–223, 2018.

LIN, Tsung-Lieh *et al.* An efficient job-shop scheduling algorithm based on particle swarm optimization. **Expert Systems with Applications**, [*s. l.*], v. 37, n. 3, p. 2629–2636, 2010.

LIU, Chang. An improved Harris hawks optimizer for job-shop scheduling problem. **The Journal of Supercomputing**, [*s. l.*], 2021. Disponível em: https://doi.org/10.1007/s11227-021-03834-0

LIU, Tung-Kuan; TSAI, Jinn-Tsong; CHOU, Jyh-Horng. Improved genetic algorithm for the job-shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 27, n. 9–10, p. 1021–1029, 2006.

LU, Yunpeng; HUANG, Zongnan; CAO, Lian. Hybrid immune genetic algorithm with neighborhood search operator for the Job Shop Scheduling Problem. **{IOP} Conference Series: Earth and Environmental Science**, [*s. l.*], v. 474, p. 52093, 2020.

LUH, Guan-Chun; CHUEH, Chung-Huei. Job shop scheduling optimization using multi-modal immune algorithm. *In*: , 2007. **International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems**. [*S. l.: s. n.*], 2007. p. 1127–1137.

MAHAPATRA, Deepak Kumar. **Job shop scheduling using artificial immune system**. 2012. [*s. l.*], 2012.

MENG, Qiaofeng; ZHANG, Linxuan; FAN, Yushun. A Hybrid Particle Swarm Optimization Algorithm for Solving Job Shop Scheduling Problems. *In*: , 2016, Singapore. (Lin Zhang, Xiao Song, & Yunjie Wu, Org.)**Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems**. Singapore: Springer Singapore, 2016. p. 71–78.

MENG, Xianbing *et al.* A new bio-inspired algorithm: chicken swarm optimization. *In*: , 2014. **International conference in swarm intelligence**. [*S. l.: s. n.*], 2014. p. 86–94.

MIRJALILI, Seyedali; MIRJALILI, Seyed Mohammad; LEWIS, Andrew. Grey wolf optimizer. **Advances in engineering software**, [*s. l.*], v. 69, p. 46–61, 2014.

MITCHELL, M. **An introduction to genetic algorithms**. Fifthed. London - England: MIT Press, 1998.

MORANDIN, O. *et al.* **A search method using Genetic Algorithm for production reactive scheduling of manufacturing systems**. [*S. l.: s. n.*], 2008. Disponível em: https://doi.org/10.1109/ISIE.2008.4677072

MORANDIN, O *et al.* An adaptive genetic algorithm based approach for production reactive scheduling of manufacturing systems. *In*: , 2008. **2008 34th Annual Conference of IEEE Industrial Electronics**. [*S. l.: s. n.*], 2008. p. 1461–1466.

MORANDIN, Orides *et al.* PSO in 2D-space to Solve Reactive Scheduling Problems in FMS to Reduce the Makespan. **2013 IEEE International Symposium on Industrial Electronics (ISIE)**, [*s. l.*], p. 1–6, 2013.

NAKANO, Ryohei; YAMADA, Takeshi. Conventional genetic algorithm for job shop problems. *In*: , 1991. **ICGA**. [*S. l.: s. n.*], 1991. p. 474–479.

NOWICKI, Eugeniusz; SMUTNICKI, Czeslaw. A fast taboo search algorithm for the job shop problem. **Management science**, [*s. l.*], v. 42, n. 6, p. 797–813, 1996.

NOWICKI, Eugeniusz; SMUTNICKI, Czesław. An advanced tabu search algorithm for the job shop problem. **Journal of Scheduling**, [*s. l.*], v. 8, n. 2, p. 145–159, 2005.

OMAR, Mahanim; BAHARUM, Adam; HASAN, Yahya Abu. A Job-shop Scheduling Problem (JSSP) Using Genetic Algorithm (GA). *In*: , 2006. **Proceedings of the 2nd im TG T Regional Conference on Mathematics, Statistics and Applications Universiti Sains Malaysia**. [*S. l.: s. n.*], 2006.

OMBUKI, Beatrice M; NAKAMURA, Morikazu; ONAGA, Kenji. An evolutionary scheduling scheme based on gkGA approach to the job shop scheduling problem. **IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer**

Sciences, [*s. l.*], v. 81, n. 6, p. 1063–1071, 1998.

OMBUKI, Beatrice M; VENTRESCA, Mario. Local search genetic algorithms for the job shop scheduling problem. **Applied Intelligence**, [*s. l.*], v. 21, n. 1, p. 99–109, 2004.

OSABA, Eneko *et al.* Focusing on the golden ball metaheuristic: an extended study on a wider set of problems. **The Scientific World Journal**, [*s. l.*], v. 2014, 2014.

OUAARAB, Aziz; AHIOD, Belaïd; YANG, Xin-She. Discrete Cuckoo Search Applied to Job Shop Scheduling Problem BT - Recent Advances in Swarm Intelligence and Evolutionary Computation. *In*: YANG, Xin-She (org.). Cham: Springer International Publishing, 2015. p. 121–137. Disponível em: https://doi.org/10.1007/978-3-319-13826-8_7

PIROOZFARD, Hamed; WONG, Kuan Yew; HASSAN, Adnan. A hybrid genetic algorithm with a knowledge-based operator for solving the job shop scheduling problems. **Journal of Optimization**, [*s. l.*], v. 2016, 2016.

PONNAMBALAM, S G; ARAVINDAN, P; RAJESH, S V. A tabu search algorithm for job shop scheduling. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 16, n. 10, p. 765–771, 2000.

PONSICH, Antonin; COELLO, Carlos A Coello. A hybrid differential evolution—tabu search algorithm for the solution of job-shop scheduling problems. **Applied Soft Computing**, [*s. l.*], v. 13, n. 1, p. 462–474, 2013.

QING-DAO-ER-JI, Ren; WANG, Yuping. A new hybrid genetic algorithm for job shop scheduling problem. **Computers & Operations Research**, [*s. l.*], v. 39, n. 10, p. 2291–2299, 2012. Disponível em: https://doi.org/https://doi.org/10.1016/j.cor.2011.12.005

RAO, Singiresu S. **Engineering optimization: theory and practice**. [*S. l.*]: John Wiley & Sons, 2009.

REGO, César; DUARTE, Renato. A filter-and-fan approach to the job shop scheduling problem. **European Journal of Operational Research**, [*s. l.*], v. 194, n. 3, p. 650–662, 2009. Disponível em: https://doi.org/10.1016/J.EJOR.2007.12.035

SABUNCUOGLU, I.; BAYIZ, M. Job shop scheduling with beam search. **European Journal of Operational Research**, [*s. l.*], v. 118, n. 2, p. 390–412, 1999. Disponível em: https://doi.org/10.1016/S0377-2217(98)00319-1

SANDERS, Peter. Algorithm engineering--an attempt at a definition. *In*: EFFICIENT ALGORITHMS. [*S. l.*]: Springer, 2009. p. 321–340.

SANTANA, Maykon Rocha *et al.* A collaborative CPN-Fuzzy modelling strategy for conflict solution in flexible manufacturing systems. **International Journal of Computer Integrated Manufacturing**, [*s. l.*], v. 31, n. 3, p. 289–295, 2018.

SASTRY, Kumara; GOLDBERG, David; KENDALL, Graham. Genetic algorithms. *In*: SEARCH METHODOLOGIES. [*S. l.*]: Springer, 2005. p. 97–125.

SAYOTI, Fatima; RIFFI, Mohammed Essaid; LABANI, Halima. Optimization of Makespan in Job Shop Scheduling Problem by Golden Ball Algorithm. **Indonesian Journal of Electrical Engineering and Computer Science**, [*s. l.*], v. 4, n. 3, p. 542–547, 2016.

SEMLALI, Soukaina Cherif Bourki; RIFFI, Mohammed Essaid; CHEBIHI, Fayçal. Memetic chicken swarm algorithm for job shop scheduling problem. **International Journal of Electrical and Computer Engineering**, [*s. l.*], v. 9, n. 3, p. 2075, 2019.

SEO, Minseok; KIM, Daecheol. Ant colony optimisation with parameterised search space for the job shop scheduling problem. **International Journal of Production Research**, [*s. l.*], v. 48, n. 4, p. 1143–1154, 2010.

SHA, D Y; HSU, Cheng-Yu. A hybrid particle swarm optimization for job shop scheduling problem. **Computers & Industrial Engineering**, [*s. l.*], v. 51, n. 4, p. 791–808, 2006.

SHL, GUOYONG. A genetic algorithm applied to a classic job-shop scheduling problem. **International Journal of Systems Science**, [*s. l.*], v. 28, n. 1, p. 25–32, 1997.

STINSON, Joel P; DAVIS, Edward W; KHUMAWALA, Basheer M. Multiple resource--constrained scheduling using branch and bound. **AIIE Transactions**, [*s. l.*], v. 10, n. 3, p. 252–259, 1978.

TARANTILIS, Christos D. Solving the vehicle routing problem with adaptive memory programming methodology. **Computers & Operations Research**, [*s. l.*], v. 32, n. 9, p. 2309–2327, 2005.

TASGETIREN, M Fatih *et al.* A particle swarm optimization and differential evolution algorithms for job shop scheduling problem. **International Journal of Operations Research**, [*s. l.*], v. 3, n. 2, p. 120–135, 2006.

TSAI, Jinn-Tsong *et al.* An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 38, n. 9–10, p. 987–994, 2008.

UDHAYAKUMAR, P; KUMANAN, S. Sequencing and scheduling of job and tool in a flexible manufacturing system using ant colony optimization algorithm. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 50, n. 9–12, p. 1075–1084, 2010.

UDOMSAKDIGOOL, A; KACHITVICHYANUKUL, V. Multiple colony ant algorithm for job-shop scheduling problem. **International Journal of Production Research**, [*s. l.*], v. 46, n. 15, p. 4155–4175, 2008.

VAESSENS, Robert Johannes Maria; AARTS, Emile H L; LENSTRA, Jan Karel. Job

shop scheduling by local search. **Informs Journal on computing**, [*s. l.*], v. 8, n. 3, p. 302–317, 1996.

VAN LAARHOVEN, Peter J M; AARTS, Emile H L; LENSTRA, Jan Karel. Job shop scheduling by simulated annealing. **Operations research**, [*s. l.*], v. 40, n. 1, p. 113–125, 1992.

VIANA, Monique Simplicio. **Algoritmo genético com operador de transgenia para minimização de makespan da programação reativa da produção.** Dissertação (Mestrado em Ciência da Computação) - Centro de Ciências Exatas e de Tecnologia, Universidade Federal de São Carlos. São Carlos. [*s. l.*], 2016.

VIANA, Monique Simplicio; CONTRERAS, Rodrigo Colnago; MORANDIN JUNIOR, Orides. A new genetic improvement operator based on frequency analysis for Genetic Algorithms applied to Job Shop Scheduling Problem. *In*: , 2021. **International Conference on Artificial Intelligence and Soft Computing**. [*S. l.: s. n.*], 2021. p. 434–450.

VIANA, Monique Simplicio; MORANDIN JUNIOR, Orides; CONTRERAS, Rodrigo Colnago. A Modified Genetic Algorithm With Local Search Strategies And Multi-Crossover Operator For Job Shop Scheduling Problem. **Sensors**, [*s. l.*], v. 20, p. 5440, 2020a. Disponível em: https://doi.org/10.3390/s20185440

VIANA, Monique Simplicio; MORANDIN JUNIOR, Orides; CONTRERAS, Rodrigo Colnago. An Improved Local Search Genetic Algorithm with a New Mapped Adaptive Operator Applied to Pseudo-Coloring Problem. **Symmetry**, [*s. l.*], v. 12, n. 10, p. 1684, 2020b.

VIANA, Monique Simplicio; MORANDIN JUNIOR, Orides; CONTRERAS, Rodrigo Colnago. An Improved Local Search Genetic Algorithm with Multi-Crossover for Job Shop Scheduling Problem. *In*: , 2020c. **International Conference on Artificial Intelligence and Soft Computing**. [*S. l.: s. n.*], 2020. p. 464–479.

VIANA, Monique Simplicio; MORANDIN JUNIOR, Orides; CONTRERAS, Rodrigo Colnago. Transgenic Genetic Algorithm to Minimize the Makespan in the Job Shop Scheduling Problem. *In*: , 2020d. **Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,**. [*S. l.*]: SciTePress, 2020. p. 463–474.

WANG, Bing *et al.* A hybrid local-search algorithm for robust job-shop scheduling under scenarios. **Applied Soft Computing**, [*s. l.*], v. 62, p. 259–271, 2018.

WANG, Feng; TIAN, Yunna; WANG, Xiaodong. A Discrete Wolf Pack Algorithm for Job Shop Scheduling Problem. *In*: , 2019. **2019 5th International Conference on Control, Automation and Robotics (ICCAR)**. [*S. l.: s. n.*], 2019. p. 581–585.

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

WANG, Ling; ZHENG, D-Z. A modified genetic algorithm for job shop scheduling.

The International Journal of Advanced Manufacturing Technology, [*s. l.*], v. 20, n. 1, p. 72–76, 2002.

WANG, Ling; ZHENG, Da-Zhong. An effective hybrid optimization strategy for job-shop scheduling problems. **Computers & Operations Research**, [*s. l.*], v. 28, n. 6, p. 585–596, 2001.

WANG, Yuping; OTHERS. A new hybrid genetic algorithm for job shop scheduling problem. **Computers & Operations Research**, [*s. l.*], v. 39, n. 10, p. 2291–2299, 2012.

WATANABE, Masato; IDA, Kenichi; GEN, Mitsuo. A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. **Computers & Industrial Engineering**, [*s. l.*], v. 48, n. 4, p. 743–752, 2005.

WEGNER, Peter. A Technique for Counting Ones in a Binary Computer. **Commun. ACM**, New York, NY, USA, v. 3, n. 5, p. 322, 1960. Disponível em: https://doi.org/10.1145/367236.367286

WISITTIPANICH, Warisa; KACHITVICHYANUKUL, Voratas. Two enhanced differential evolution algorithms for job shop scheduling problems. **International Journal of Production Research**, [*s. l.*], v. 50, n. 10, p. 2757–2773, 2012.

XHAFA, Fatos; ABRAHAM, Ajith. **Metaheuristics for scheduling in industrial and manufacturing applications**. [*S. l.*]: Springer, 2008. v. 128

YANG, Xin-She. A new metaheuristic bat-inspired algorithm. *In*: NATURE INSPIRED COOPERATIVE STRATEGIES FOR OPTIMIZATION (NICSO 2010). [*S. l.*]: Springer, 2010a. p. 65–74.

YANG, Xin-She. **Engineering optimization: an introduction with metaheuristic applications**. [*S. l.*]: John Wiley \& Sons, 2010b.

YANG, Xin-She; DEB, Suash. Multiobjective cuckoo search for design optimization. **Computers \& Operations Research**, [*s. l.*], v. 40, n. 6, p. 1616–1624, 2013.

YEN, Gary G; IVERS, Brian. Job shop scheduling optimization through multiple independent particle swarms. **International Journal of Intelligent Computing and Cybernetics**, [*s. l.*], v. 2, n. 1, p. 5–33, 2009.

YIN, Minghao; LI, Xiangtao; ZHOU, Junping. An efficient job shop scheduling algorithm based on artificial bee colony. **Scientific Research and Essays**, [*s. l.*], v. 6, n. 12, p. 2578–2596, 2011.

YU, Hongli *et al.* A Hybrid Particle Swarm Optimization Algorithm Enhanced with Nonlinear Inertial Weight and Gaussian Mutation for Job Shop Scheduling Problems. **Mathematics**, [*s. l.*], v. 8, n. 8, 2020. Disponível em: https://doi.org/10.3390/math8081355

YUSOF, Rubiyah *et al.* Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm. **Applied soft computing**, [*s. l.*], v. 11, n. 8, p. 5782–5792, 2011.

ZHANG, Chaoyong; RAO, Yunqing; LI, Peigen. An effective hybrid genetic algorithm for the job shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, [*s. l.*], v. 39, n. 9–10, p. 965, 2008.

ZHANG, Rui; WU, Cheng. A hybrid immune simulated annealing algorithm for the job shop scheduling problem. **Applied Soft Computing**, [*s. l.*], v. 10, n. 1, p. 79–89, 2010.

ZHANG, Xue-Feng *et al.* An efficient hybrid particle swarm optimization for the Job Shop Scheduling Problem. *In*: , 2011. **2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)**. [*S. l.: s. n.*], 2011. p. 622–626. Disponível em: https://doi.org/10.1109/FUZZY.2011.6007385

ZHAO, Fuqing *et al.* A chemotaxis-enhanced bacterial foraging algorithm and its application in job shop scheduling problem. **International Journal of Computer Integrated Manufacturing**, [*s. l.*], v. 28, n. 10, p. 1106–1121, 2015a.

ZHAO, Fuqing *et al.* A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems. **International Journal of Production Research**, [*s. l.*], v. 54, n. 4, p. 1039–1060, 2016.

ZHAO, Fuqing *et al.* An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems. **Expert Systems with Applications**, [*s. l.*], v. 42, n. 8, p. 3953–3966, 2015b.

ZHOU, Guo; ZHOU, Yongquan; ZHAO, Ruxin. Hybrid social spider optimization algorithm with differential mutation operator for the job-shop scheduling problem. **Journal of Industrial & Management Optimization**, [*s. l.*], v. 17, n. 2, p. 533–548, 2021.

ZHOU, Hong; FENG, Yuncheng; HAN, Limin. The hybrid heuristic genetic algorithm for job shop scheduling. **Computers & Industrial Engineering**, [*s. l.*], v. 40, n. 3, p. 191–200, 2001.