UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS PARA O PROBLEMA DE PROGRAMAÇÃO JOB-SHOP FLEXÍVEL MULTIOBJETIVO

CLEVERSON MOREIRA DE SOUZA

ORIENTADOR: PROF. DR. EDILSON REIS RODRIGUES KATO

São Carlos - SP
Maio/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS PARA O PROBLEMA DE PROGRAMAÇÃO JOB-SHOP FLEXÍVEL MULTIOBJETIVO

CLEVERSON MOREIRA DE SOUZA

Dissertação 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 Mestre em Ciência da Computação, área de concentração: Inteligência Artificial.

Orientador: Prof. Dr. Edilson Reis Rodrigues Kato

São Carlos - SP
Maio/2018

UNIVERSIDADE FEDERAL DE SÃO CARLOS

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

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Cleverson Moreira de Souza, realizada em 10/05/2018:

Prof. Dr. Edilson Reis Rodrigues Kato
UFSCar

Prof. Dr. Mário Luiz Tronco
USP

Profa. Dra. Kelen Cristiane Teixeira Vivaldini
UFSCar

Dedico este trabalho à minha filha e minha esposa, por fazerem parte da minha vida hoje e sempre.

AGRADECIMENTO

Agradeço primeiramente a Deus. Acredito que a força divina seja a maior inspiração para nossas conquistas.

A toda minha família, em especial à minha filha, minha esposa, meus pais e minhas irmãs pelo amor, carinho, compreensão e atenção. Tenho o conceito de que a família é à base de tudo, pois o amor inspira a felicidade e aumenta a motivação do trabalho em busca dos objetivos na expectativa de melhorar a qualidade de vida.

Ao meu orientador, Prof. Dr. Edilson Reis Rodrigues Kato pela confiança, paciência, dedicação, atenção, incentivo, apoio e ensinamentos oferecidos na orientação da pesquisa e desenvolvimento deste trabalho.

Ao Prof. Dr. Daniel Lucrédio, coordenador do PPGCC da UFSCar pela atenção, prestatividade e orientação em todos os momentos.

Aos professores do PPGCC da UFSCar que ofereceram as disciplinas cursadas, em especial à Prof.ª Dra. Sandra Fabbri pelo apoio e atenção.

Aos professores Dr. Mário Luiz Tronco e Dra. Kelen Cristiane Teixeira Vivaldini pela participação da banca de defesa e apontamentos construtivos que contribuíram para melhoria deste trabalho.

A minha amiga do coração Monique Simplicio Viana pelo carinho, motivação, apoio e atenção em todos os momentos.

A minha amiga Joyce Helena Ferreira dos Santos, professora de línguas do Instituto Federal, Câmpus Birigui pela atenção e revisão do texto desta dissertação.

A empresa SmartBio Tecnologia, em especial o gestor Cristiano Fagundes pelo apoio, incentivo e intercâmbio de conhecimentos durante o período em que estudamos e trabalhamos juntos.

Aos meus colegas e amigos Gabriel Diego de A. Aranha, Marcela Ap. Aniceto dos Santos, Maykon R. Santana, Rafael Francisco V. Sanches e Rafael Stofalette João pelo apoio, motivação e intercâmbio de conhecimentos durante as atividades acadêmicas.

Lembre-se que as pessoas podem tirar tudo de você, menos o seu conhecimento.

Albert Einstein

RESUMO

A atividade de programação da produção do tipo *job-shop* encontra-se no nível mais detalhado e complexo de um sistema de planejamento e controle da produção. O problema de programação *job-shop* flexível (do inglês *Flexible Job-shop Scheduling Problem* - FJSP) é uma extensão do problema de programação *job-shop* (do inglês *Job-shop Scheduling Problem* - JSP) e desempenha um papel importante em sistemas reais de produção devido a sua natureza combinatória que permite uma operação ser processada em mais de uma máquina alternativa do conjunto de recursos disponíveis. Os problemas de programação *job-shop* pertencem à classe dos problemas do tipo NP-completo devido à dificuldade em obter uma solução ótima por abordagens tradicionais. A metaheurística Otimização por Colônia de Formigas (do inglês *Ant Colony Optimization* - ACO) tem se mostrado eficiente para resolver problemas de otimização combinatória. O ACO consiste em um algoritmo inspirado no comportamento das colônias de formigas, que funciona como um método probabilístico e constrói soluções por meio da inteligência coletiva. Desta maneira, usa a experiência adquirida durante o processo de busca de forma adaptativa. Neste trabalho é apresentado um Sistema de Formigas (do inglês *Ant System* - AS) em conjunto com a regra do Tempo de Processamento mais Curto (do inglês *Shortest Processing Time* - SPT) para resolução do FJSP multiobjetivo. Tradicionalmente, as decisões de atribuição e programação são realizadas separadamente no gerenciamento de produção. Desta forma, a abordagem proposta emprega a regra SPT para alocação de recursos e o algoritmo AS para o sequenciamento das operações atribuídas, onde cada formiga constrói um cronograma de programação viável conforme as restrições que se aplicam ao problema. O objetivo desta pesquisa é encontrar soluções ótimas para o FJSP, considerando a minimização do tempo de conclusão, minimização da carga da máquina mais crítica e a minimização da carga total de todas as máquinas como critérios de otimização. A combinação de vários critérios de otimização induz complexidade adicional e novos problemas. No entanto, os resultados do estudo comparativo com outras abordagens em instâncias conhecidas na literatura mostraram que o algoritmo proposto é viável e eficaz para resolver o FJSP multiobjetivo, especialmente em problemas de grande escala.

Palavras-chave: Otimização por Colônia de Formigas, Problema de Programação Job-shop Flexível, Sistema de Formigas, Problema de Programação Job-shop.

ABSTRACT

The job-shop production scheduling activity is at the most detailed and complex level of a production planning and control system. The Flexible Job-shop Scheduling Problem (FJSP) is an extension of the job-shop scheduling problem (JSP) and plays an important role in because of its combinatorial nature that allows an operation to be processed in more than one alternative machine of the set of available resources. Problems of job-shop programming belong to the class of NP-complete problems due to the difficulty in obtaining an optimal solution by traditional approaches. The metaheuristic Ant Colony Optimization (ACO) has proved to be efficient in solving combinatorial optimization problems. The ACO consists of an algorithm inspired by the behavior of the ant colonies, which functions as a probabilistic method and constructs solutions through collective intelligence. In this way, it uses the experience gained during the search process adaptively. In this work an Ant System (AS) is presented along with the Shortest Processing Time (SPT) rule for multiobjective FJSP resolution. Traditionally, allocation and scheduling decisions are made separately in production management. Thus, the proposed approach employs the SPT rule for resource allocation and the AS algorithm for the sequencing of assigned operations, where each ant constructs a viable scheduling according to the constraints that apply to the problem. The objective of this research is to find optimal solutions for the FJSP, considering minimization of completion time, minimization of the most critical machine load and minimization of the total load of all machines as optimization criteria. The combination of several optimization criteria induces additional complexity and new problems. However, the results of the comparative study with other approaches in instances known in the literature have shown that the proposed algorithm is feasible and effective for solving multiobjective FJSP, especially in large scale problems.

Keywords: Ant Colony Optimization, Flexible Job-shop Scheduling Problem, Ant System, Job-shop Scheduling Problem.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACO	*Ant Colony Optimization*
AL	*Approach by Localization*
ACS	*Ant Colony System*
AGV	*Automated Guided Vehicles*
AIA	*Artificial Immune Algorithm*
ANTS	*Approximate Nondeterministic Tree Search*
AS	*Ant System*
ASrank	*Rank-Based Ant System*
BEG-NSGA-II	*Bee Evolutionary Guiding Nondominated Sorting Genetic Algorithm II*
BKS	*Best Known Solution*
EAS	*Elitist Ant System*
CFRP	*Conflict Free Routing Problem*
CGA	*Controlled Genetic Algorithm*
FJSP	*Flexible Job Shop Scheduling Problem*
GA	*Genetic Algorithm*
HSFLA	*Hybrid Shuffled Frog Leaping Algorithm*
HTSA	*Hybrid Tabu Search Algorithm*
IA	Inteligência Artificial
IACO	*Improved Ant Colony Optimization*
IP	*Integer Programming*
JSP	*Job-Shop Scheduling Problem*
KBHSA	*Knowledge Based Heuristic Searching Architecture*
KBACO	*Knowledge Based Ant Colony Optimization*
MACO	*Multi-pheromoneant Ant Colony Optimization*
MLJSP	*Multi-objective Lot-splitting Job-Shop Scheduling Problem*
MMAS	*MAX–MIN Ant System*
MOGA	*Multi-objective Genetic Algorithm*
MOPSO	*Multi-Objective Particle Swarm Optimization*
PCV	*Problema do Caixeiro Viajante*
PSO	*Particle Swarm Optimization*
PCP	Planejamento e Controle da Produção

QIA	*Quantum Immune Algorithm*
SA	*Simulated Annealing*
S-ACO	*Simple-ACO*
SPT	*Shortest Processing Time*
TS	*Tabu Search*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

A programação da produção, também chamada de *scheduling* é considerada uma atividade muito importante no sistema produtivo pelo fato de influenciar diretamente a taxa de produção e os custos. O *scheduling* é considerado um problema complexo devido a uma série de fatores, tais como: alocar um número de recursos limitados ao longo do tempo para executar um conjunto de tarefas, e determinar a sequência de operações para que o volume de produção seja maximizado e os custos operacionais sejam minimizados (AKYOL e BAYHAN, 2007).

Os sistemas produtivos se diferem em seus níveis de flexibilidade e desempenho econômico. No sistema *Job-Shop* não há um padrão comum para o fluxo de trabalho pelo fato da fábrica permitir que peças diferentes sigam caminhos diferentes através das máquinas. No entanto, a flexibilidade conduz a fabricação de uma grande variedade de ordens de produção de pequeno volume (DESROCHERS e AL-JAAR, 1995; SLACK et al., 2002). Nesta categoria, encontra-se o problema de programação *Job-Shop* Flexível (do inglês *Flexible Job-shop Scheduling Problem* - FJSP) que pode ser considerado uma ampliação do *Job-Shop* clássico (JSP). O FJSP incorpora todos os atributos do JSP e se caracteriza quando no mínimo uma operação de todo conjunto deve ser processada por mais de uma máquina, considerando que o tempo de início e término de cada operação é pré-definido, e todos os recursos do conjunto são compatíveis. Desta maneira, determina-se a atribuição e a sequência das operações nas máquinas para satisfazer os critérios de desempenho estabelecidos (ZHANG et al., 2009).

No chão de fábrica, os critérios de desempenho estabelecem índices para minimizar os custos de produção, tais como: tempo de conclusão das operações, carga de trabalho das máquinas, atraso máximo, produtividade de mão-de-obra e custo total. Os critérios de desempenho podem variar de acordo com os objetivos do sistema produtivo. Assim, o FJSP pode ser classificado como monoobjetivo ou multiobjetivo. Segundo Xue et. al. (2014) o FJSP multiobjetivo é mais realista e mais difícil de ser resolvido, tendo em vista que possa haver objetivos conflitantes ou incompatíveis.

Vários métodos foram propostos como alternativas de solução para problemas de programação *Job-Shop* e têm sido tratados sobre duas perspectivas: (1) métodos exatos, onde o objetivo é encontrar a solução ótima, os quais exigem um elevado esforço computacional; (2) métodos de aproximação, que não garantem a solução ideal, mas são capazes de encontrar boas soluções em tempo computacional aceitável (BLUM e ROLI, 2003). Muitos estudos têm focado no uso da Inteligência Artificial (IA) como alternativa de explorar diversas técnicas para obter soluções de qualidade com o mínimo de esforço computacional. Neste sentido, pesquisadores como Brandimarte (1993), Kacem et. al (2002a, 2002b), Xia e Wu (2005), Zhang et al. (2009), Xue et al. (2014), Huang et al. (2016), Deng et al. (2017) e Wang et al. (2017) propuseram abordagens heurísticas e metaheurísticas como alternativas de apresentar soluções aceitáveis para o FJSP. Uma abordagem de solução a este problema é o uso da metaheurística Otimização por Colônia de Formigas (do inglês *Ant Colony Optimization* - ACO).

Segundo Dorigo e Stützle (2004) o ACO é probabilístico e usa a experiência adquirida durante o processo de busca de forma adaptativa. Inspirado no comportamento de colônia de formigas principalmente na maneira como elas caminham em busca de alimentos, e no que diz respeito à organização do trabalho, o algoritmo constrói soluções por meio de inteligência coletiva. Uma colônia de formigas é composta por grupos de insetos sociais que apesar da simplicidade de seus indivíduos são capazes de resolver tarefas muito complexas. Conforme os autores, o ACO possui algumas variantes e pode ser explanado da seguinte forma: é uma metaheurística na qual uma colônia de formigas artificiais coopera na busca de boas soluções para diferentes problemas de otimização. A técnica consiste em alocar os recursos computacionais para um conjunto de agentes simples, denominado formigas artificiais que se comunicam indiretamente através da transformação do ambiente pelo qual caminham em busca de soluções.

Neste trabalho é apresentada uma metaheurística ACO conhecida como Sistema de Formigas (do inglês *Ant System* - AS) em conjunto com a regra do Tempo de

Processamento mais Curto (do inglês *Shortest Processing Time* - SPT) para resolução do FJSP multiobjetivo. Segundo Brandimarte (1993) o FJSP se torna menos complexo quando é resolvido de forma hierárquica, dividindo-o em dois subproblemas: (1) roteamento (atribuição das operações nas máquinas); (2) programação (sequenciamento das operações). Desta forma, a abordagem proposta emprega a heurística SPT para resolver o subproblema de roteamento e a metaheurística ACO para resolver o subproblema de programação. No processo de busca pela solução do problema, aplica-se primeiramente a regra SPT para alocação de recursos na busca da construção um modelo de atribuição ideal, e na sequência executa-se o algoritmo AS para determinar o sequenciamento das operações atribuídas, onde cada formiga constrói um cronograma de programação viável conforme as restrições que se aplicam ao problema.

1.2 Motivação

A importância de se propor técnicas diversificadas para resolver problemas de programação da produção se dá pelo ganho de informações relevantes que possam contribuir com as organizações na melhoria do sistema produtivo. Conforme Akyol e Bayhan (2007), esta tarefa é uma das etapas mais importantes de um sistema de manufatura pelo fato de definir as características dos trabalhos de um ambiente de produção.

Problemas de programação do tipo *job-shop* vem chamando atenção da comunidade acadêmica nos últimos anos pelo fato de impactar nos lucros da organização, na qual uma boa programação pode aumentar a produção no chão de fábrica e reduzir custos. Várias pesquisas foram direcionadas a minimização de um único critério, como os trabalhos de Liouane et al. (2007), Xing et al. (2010), Flórez, Gómez e Bautista (2013), Saidi-Mehrabad et al. (2015), Huang e Yu (2017), Wang et al. (2017) e Wu, Wu e Wang (2017). No entanto, as indústrias, necessitam resolver seus problemas de programação, nos quais vários objetivos precisam ser considerados para melhorar o desempenho geral do sistema produtivo.

Em relação aos problemas de programação multiobjetivo, estes são mais complexos em comparação aos problemas de programação com um único objetivo devido ao fato de que a dificuldade em encontrar uma boa solução é ainda maior, especialmente em se tratando de objetivos inconsistentes, conflitantes ou contraditórios.

Muitos trabalhos com diversas técnicas de IA têm sido propostas para a modelagem e solução do FJSP multiobjetivo, como os trabalhos de Kacem et. al (2002a, 2002b), Xia e Wu (2005), Zhang et al. (2009), Li, Keqi e Chunnan (2010), Xue et al. (2014), Xuesong e Qiaoyun (2015), Huang et al. (2016), Deng et al. (2017). Diante desta realidade, a representação do conhecimento do problema tem sido um fator muito importante no processo de busca das melhores soluções, tendo em vista que as organizações buscam sempre maximizar a produção com menor custo possível. Neste contexto, encontra-se a motivação para este trabalho, que objetiva resolver o FJSP multiobjetivo por meio de uma abordagem hierárquica com uso de um algoritmo ACO em conjunto com a heurística SPT, visando obter soluções de qualidade em comparação com outras abordagens encontradas na literatura.

1.3 Objetivos

O objetivo principal deste trabalho é a resolução do FJSP multiobjetivo com otimização dos resultados pela minimização de três critérios (índices) de desempenho. Tais critérios denotam a formulação multiobjetivo, e cada índice pode ser considerado como objetivo específico:

1. *Makespan* - tempo total de conclusão das operações;
2. *Machine workload* - carga de trabalho da máquina mais crítica; e
3. *Total workload* - carga de trabalho de todas as máquinas.

A análise dos objetivos específicos foi realizada por meio de um estudo comparativo com outras abordagens encontradas na literatura. Neste estudo foi avaliado o desempenho da abordagem proposta em diferentes cenários.

1.4 Estrutura do Trabalho

Para contextualizar o problema e demostrar os resultados, este trabalho esta organizado da seguinte maneira:

No capítulo 02 são apresentados os conceitos fundamentais do trabalho. Neste contexto é abordado uma descrição geral sobre sistemas de produção e inteligência artificial com ênfase no Algoritmo de Otimização por Colônia de Formigas.

No capítulo 03 é apresentada a revisão da literatura, com trabalhos relacionados que contribuíram na fundamentação desta pesquisa, buscando incluir algumas considerações de como a metaheurística ACO tem sido empregada na tentativa de resolução de problemas de programação da produção e as principais questões discutidas nessas aplicações.

O capítulo 04 mostra a proposta deste trabalho a partir das considerações dos capítulos 2 e 3, visando descrever sua principal contribuição. Nesse capítulo é descrito o problema de modelagem sob uma visão geral dos procedimentos.

O capítulo 05 exibe os resultados dos testes utilizando o ACO + SPT, propostos nesta pesquisa, e o estudo comparativo com alguns trabalhos encontrados na literatura mencionados no capítulo 03.

As considerações finais e os trabalhos futuros são apresentados no capítulo 06.

Capítulo 2

CONCEITOS FUNDAMENTAIS

2.1 Considerações Iniciais

Neste capítulo são abordados os conceitos fundamentais em uma perspectiva ampla dos principais tópicos na visão do problema considerado.

Na sessão 2.2 é apresentado o conceito de sistemas de manufatura com enfoque no planejamento e controle da produção abordando o conceito de programação da produção.

Na sessão 2.3 são mostrados os conceitos de inteligência artificial com enfoque no algoritmo de otimização por colônia de formigas, abordando as principais variantes.

2.2 Sistema de Manufatura

Sistema de manufatura pode ser definido como um conjunto de etapas, através de um processo relacionado a alguma atividade de transformação para alcançar determinados objetivos. O sistema de produção ou sistema produtivo é o processo usado na manufatura do serviço que transforma a matéria prima em produtos de bens e consumo (SLACK et.al., 2002). O processo produtivo desde a entrada da matéria prima até o produto final é ilustrado na figura 2.1.

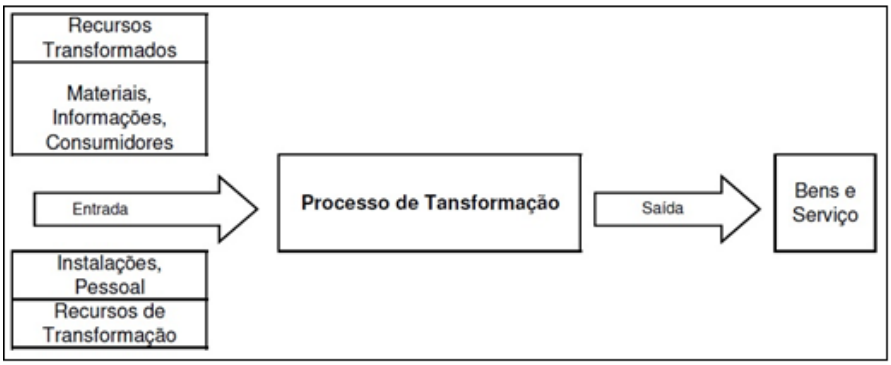

Figura 2.1 - Sistema produtivo.
Fonte: Adaptado de Slack et al. (2002)

A classificação dos sistemas produtivos tem por finalidade facilitar o entendimento das suas características e a relação com a complexidade das atividades de planejamento e controle. Os sistemas de produção são classificados pelo grau de padronização, pela natureza de operação, pelo ambiente de produção, pela natureza do produto, pelo fluxo de processo e pelo ambiente de produção dos produtos. Os sistemas de produção se diferem em seus níveis de flexibilidade e desempenho econômico, portanto apropria-se a diferentes necessidades (TUBINO, 2008; SLACK et. al., 2002).

2.2.1 Planejamento e Controle da Produção

O Planejamento e Controle da Produção (PCP) é um conjunto de atividades que envolvem uma série de decisões com o objetivo de equilibrar o fornecimento e a demanda para manter o controle do sistema produtivo (FERNANDES e FILHO, 2010).

Vollmann, Berry e Whybark (1997) destacaram algumas atividades que devem ser resolvidas através dos sistemas PCP:

- Planejar a entrega dos materiais na hora certa e na quantidade correta para produzir os produtos;
- Planejar a capacidade dos equipamentos para atender as necessidades dos clientes;
- Manter níveis de estoques satisfatórios;
- Utilizar os equipamentos que resultem em um bom retorno de investimento;

- Programar as atividades das pessoas e dos equipamentos que resultem em produtos para atender os clientes na quantidade e no tempo certo;
- Controlar os recursos;
- Comunicar com os fornecedores e clientes quaisquer problemas que afetem o seu relacionamento;
- Prover informações para outras atividades da empresa.

Desta forma, o PCP tem que ser executado através de um horizonte de tempo. Geralmente é dividido em longo, médio e curto prazo, e cada fase está relacionada a atividades de decisão estratégicas, táticas e operacionais.

O plano de longo prazo está associado às decisões estratégicas que envolvem a decisão de que produtos, equipamentos e ampliação de fábrica devem ser feitos para a sua continuidade. As decisões de médio prazo estão relacionadas a decisões táticas para operar o sistema eficientemente, planejando o uso da capacidade produtiva que deve atender às previsões de vendas e/ou pedidos em carteira acordados previamente com os clientes. Já no curto prazo, com o sistema montado, o sistema produtivo irá executar a Programação da Produção para produzir os produtos e serviços com o objetivo de entregá-los aos clientes na quantidade e no prazo previamente acordados.

O PCP é responsável pela coordenação e aplicação dos recursos produtivos de forma a atender da melhor maneira possível os planos estabelecidos nos planos de longo, médio e curto prazo. Assim, há diversas relações com outras atividades da organização para poder estabelecer o programa mestre da produção, tais como: a administração de materiais, o sequenciamento das ordens de produção, a emissão e a liberação dessas ordens (TUBINO, 2008).

2.2.2 Programação da Produção

Um sistema moderno de manufatura requer distinção e diversificação de produtos, produção em pequenos lotes, rápidas entregas e aderência à data de entrega prometida. Existe a possiblidade para receber a mudança nas ordens dos clientes, então a disponibilidade de material ou de capacidade tem tornado o tema programação da produção mais relevante

dentro do PCP. A programação da produção está inserida na terceira atividade do PCP, na qual determina em curto prazo de tempo o detalhamento das operações que serão realizadas, incluindo entre outros, onde as operações dos produtos são processadas, com quais recursos, e em qual tempo de início e término para cada ordem de produção (TUBINO, 2008; FERNANDES e FILHO, 2010).

Segundo Williem e Setiawan (2011), a programação da produção é definida por alocar alguns trabalhos em algumas máquinas de acordo com uma ordem adequada. Cada máquina só pode processar um trabalho em um determinado tempo e as matérias-primas devem ser suficientes para que a máquina trabalhe sem parar. Isso significa que a programação demanda um conjunto de máquinas capazes de processar um trabalho num certo tempo, e consiste em alocar a sequência dos trabalhos para cada máquina, sem qualquer conflito na utilização de matéria-prima em que o tempo total necessário seja realmente mínimo. Existem muitos métodos na obtenção de programação de trabalhos, mas o resultado não é livre de conflitos, ou ele não é o ideal. Embora a programação da produção se refira à alocação de tarefas a todos os recursos do sistema tais como máquinas, ferramentas, sistema de manipulação de material, entre outros, ela é mais utilizada como programação de tarefas nas máquinas. Conforme Desrochers e Al-Jaar (1995) as principais arquiteturas de chão de fábrica são:

- *Flow-Shop* - os trabalhos seguem a mesma sequência de operações e os volumes de produção são tipicamente altos, fazendo com que o sistema seja bastante eficiente;
- *Job-Shop* - peças diferentes seguem caminhos diferentes através das máquinas e não há um padrão comum para o fluxo de trabalho através da fábrica. A flexibilidade do "*Job-Shop*" permite fabricar uma grande variedade de ordens de produção de pequeno volume.

De acordo com Yahyaoui et al. (2011), a programação da produção no ambiente *job-shop* é definida da seguinte maneira: existem *n* trabalhos (do inglês *jobs*) a serem processados através *m* máquinas na ordem descrita em certas hipóteses restritivas; cada *job* deve passar por cada máquina exatamente uma vez; o processamento de um *job* numa máquina é chamado de operação e exige uma duração conhecida como o tempo de processamento. Visto que este problema é muito complexo e difícil de ser resolvido por abordagens tradicionais, se enquadra na categoria NP-completo. Insere-se nesta categoria pelo

fato de possuir uma entrada de tamanho *N*, cuja resolução leva um tempo computacional proporcional a $\geq 2N$.

Para Fnaiech et al. (2013), a programação *job-shop* tem como objetivo atribuir um número de máquinas ao longo do tempo para executar um conjunto de *jobs*. Cada *job* pode ter um número diferente de operações e sua própria ordem de processamento em máquinas. O tempo de processamento de cada operação é fixado antecipadamente e as operações não podem ser interrompidas depois de iniciadas. Este tipo de programação é considerado estático e determinista, portanto só é resolvido quando é respeitado dois tipos de restrições:

1. Restrições de sequência ou restrições de precedência - duas operações do mesmo trabalho não podem ser processadas simultaneamente;
2. Limitação de recursos - somente uma operação deve ser executada numa máquina ao mesmo tempo.

Ainda segundo os autores, a distribuição das operações segue ainda as restrições inerentes à classe do sistema produtivo, considerando as propriedades de cada máquina e de todos os demais recursos do ambiente de produção. Segundo Zhang et al. (2009), as principais categorias da programação *Job-Shop* são:

- *Job-Shop* clássico: consiste no Problema de Programação *Job-Shop* (do inglês *Job-shop Scheduling Problem* - JSP) e se caracteriza por atribuir um conjunto de recursos (máquinas, ferramentas, etc.) para processar um conjunto de operações (trabalhos) no objetivo de atender a demanda da produção;
- *Job Shop* flexível: consiste no Problema de Programação *Job-Shop* Flexível (do inglês *Flexible Job-shop Scheduling Problem* - FJSP) e pode ser considerada uma ampliação do JSP, que permite uma operação ser processada em um recurso de fora de um conjunto alternativo de recursos. Em um FJSP, no mínimo uma operação de todo conjunto deverá ser processada por mais de um recurso, sendo que todos os recursos do conjunto são compatíveis. Devido às necessidades adicionais para determinar a atribuição das operações nas máquinas, o FJSP é mais complexo e incorpora todas as dificuldades e complexidades do JSP.

Conforme Kacem et al. (2002a), o FJSP pode ser classificado em duas categorias:

- *Job-Shop* Totalmente Flexível (do inglês *Total Flexible Job-shop Scheduling Problem* - T-FJSP): todas as operações de todos os *jobs* podem ser processadas por todos os recursos disponíveis no conjunto.
- *Job-Shop* Parcialmente Flexível (do inglês *Partial Flexible Job-shop Scheduling Problem* - P-FJSP): nem todas as operações de todos *jobs* podem ser processadas por todos os recursos disponíveis no conjunto, ou seja, deve haver pelo menos uma operação de um determinado *job* que não pode ser processada por um determinado recurso.

Xia e Wu (2005) contextualizaram que o FJSP pode ser divido em dois subproblemas:

- Roteamento: consiste em atribuir cada operação a uma máquina de um conjunto de máquinas operantes;
- Programação: consiste em sequenciar as operações atribuídas em todas as máquinas para obter uma programação viável.

Xing et al. (2010) descreveram que os subproblemas do FJSP podem ser abordados de duas maneiras:

- Abordagens hierárquicas - atribuição e sequenciamento de operações nos recursos são tratados separadamente, ou seja, são considerados independentemente.
- Abordagens integradas - atribuição e sequenciamento não são diferenciados.

Os autores afirmaram que as abordagens hierárquicas baseiam-se na ideia de decompor o problema original para reduzir sua complexidade.

2.2.3 Medidas de Desempenho para Programação da Produção

As medidas de desempenho são muito importantes na tarefa de programação da produção. Uma medida é definida por um valor que mede o desempenho de uma programação em particular, e que normalmente é definida em termos das características do chão de fábrica

ou da conclusão de uma tarefa, e pode ser determinada como uma função dos tempos de conclusão das operações ou tarefas. Estes números podem ser usados para avaliar o funcionamento de uma programação (BANKS, 1996; SLACK et al., 2002).

Todo ambiente de produção segue determinados objetivos. No entanto, cada objetivo pode ser analisado por um grupo de medidas. Na tabela 2.1 são mostrado alguns objetivos com as suas medidas correspondentes.

Tabela 2.1 - Medidas de desempenho utilizadas na produção.
Fonte: Adaptado de Slack et al. (2002)

Objetivo de desempenho	Algumas medidas típicas
Qualidade	Número de defeitos por unidade Nível de reclamação de consumidor Nível de refugo Alegações de garantia Tempo médio entre falhas Escore de satisfação do consumidor
Rapidez	Tempo de cotação do consumidor Lead time de pedido Frequência de entregas Tempo de atravessamento Tempo de ciclo
Confiabilidade	Porcentagem de pedidos entregues com atraso Atraso médio de pedidos Proporção de produtos em estoque Desvio-médio de promessa de chegada Aderência à programação
Custo	Tempo mínimo e tempo médio de entrega Variação contra orçamento Utilização de recursos Produtividade da mão-de-obra Valor agregado Eficiência Custo por hora de operação

As medidas de desempenho mais utilizadas na arquitetura *Job-Shop* são (GAMILA e MOTAVALLI, 2003):

- *Makespan* - tempo máximo requerido para completar todos os trabalhos de uma produção;

- *Machine Workload* - carga de trabalho da máquina, tempo de produção gasto por qualquer recurso;
- *Total Workload* - carga de trabalho total envolvendo todas as máquinas, volume total do trabalho sobre todos os recursos;
- Atraso Máximo - diferença entre o tempo de conclusão e a data de entrega;
- Custo Total - soma de todos os gastos obtidos na produção.

Conforme os autores, a escolha apropriada de uma medida de desempenho para a programação é uma consideração importante. A análise dessas medidas pode mostrar que um procedimento de programação com um bom desempenho para uma medida não é necessariamente bom para outra.

2.3 Inteligência Artificial

Atualmente existem inúmeras definições para Inteligência Artificial (IA). Segundo Stone et al. (2016) a falta de uma definição precisa, provavelmente ajudou o campo a crescer, florescer e avançar a um ritmo cada vez mais acelerado. O campo da IA é um esforço contínuo para avançar a fronteira de inteligência da máquina.

Para Luger (2008) IA é o ramo da Ciência da Computação que se encarrega de automatizar o comportamento inteligente. A Inteligência Artificial prove princípios que incluem estruturas de dados utilizadas na representação do conhecimento de um determinado conjunto de problemas, e envolve um determinado corpo de técnicas para abordar esses problemas.

Segundo o autor, o campo da IA compõe algumas características importantes que parecem comuns a todas as técnicas. Estas características envolvem:

- O uso de computadores para fazer raciocínio, reconhecimento de padrões, aprendizado ou alguma outra forma de inferência.
- Um foco em problemas que não respondem a soluções algorítmicas. Isto está subjacente à dependência da pesquisa heurística como uma técnica de resolução de problemas de IA;

- Uma preocupação com a resolução de problemas usando informações inexatas ou mal definidas, e o uso de formalismos representativos que permitem ao pesquisador compensar esses problemas;
- Raciocínio sobre as características qualitativas significativas de uma situação;
- Uma tentativa de lidar com questões de significado semântico, bem como de forma sintática;
- Respostas que não são nem exatas nem ótimas, mas suficientes. Isto é resultado da dependência essencial dos métodos heurísticos de resolução de problemas nas situações em que os resultados ótimos ou exatos são muito caros ou não são possíveis;
- O uso de grandes quantidades de conhecimento específico do domínio na resolução de problemas;
- O uso do conhecimento de meta-nível para efetuar um controle mais sofisticado do problema com estratégias de resolução.

Conforme o contexto do autor é importante descrever que o processo de raciocínio em IA pode ser realizado por meio da lógica clássica ou lógica Fuzzy. A lógica Fuzzy também denominada lógica nebulosa ou lógica difusa, é usada para trabalhar com informações incertas, e se difere da lógica clássica por não ser restrita ao sistema binário;

De acordo com Stone et al. (2016) o campo da IA se tornou uma ferramenta de apoio a sociedade pelo fato de construir sistemas inteligentes que colaboram efetivamente com as pessoas pelas formas interativas e escaláveis para ajudar a resolver problemas do mundo real. Os autores apontam as principais subáreas da IA destacadas na literatura:

- Aprendizado de máquina - são algoritmos de aprendizado que podem ser metaheurísticas inspiradas pela natureza ou métodos probabilísticos, projetados para trabalhar com conjuntos de dados representativos. São capazes de aprender sobre um conjunto de informações de entrada por meio da experiência adquirida ao longo do processamento e gerar uma saída com uma solução aproximada.
- Robótica - treinamento de robôs para interagir com o mundo real. O avanço da robótica depende de avanços proporcionais para melhorar a confiabilidade e a

generalidade da visão computacional e outras formas de percepção de máquina.

- Visão computacional - é atualmente a forma mais proeminente de percepção da máquina. Tem sido a subárea de IA mais transformada pelo surgimento de aprendizado profundo. Neste sentido, os computadores são capazes de realizar algumas tarefas de visão melhor do que as pessoas. A pesquisa atual é focada em legendas automáticas de imagens e vídeos.
- Processamento de linguagem natural - muitas vezes associado ao reconhecimento automático de fala. Esta rapidamente se tornando em uma mercadoria para linguagens amplamente faladas com grandes conjuntos de dados. A pesquisa agora está mudando para desenvolver sistemas refinados que são capazes de interagir com as pessoas através do diálogo, não apenas reagir aos pedidos estilizados. Grandes avanços também foram feitos em tradução automática entre diferentes idiomas em tempo real.

2.3.1 Heurísticas e Metaheurísticas

As heurísticas ou algoritmos de execução única são métodos construtivos que geram melhorias iterativas (DORIGO e STÜTZLE, 2004). São algoritmos destinados à resolução de determinados problemas específicos. Assim podem ser classificados de diversas maneiras, tais como: heurísticas de construção - algoritmo guloso; heurísticas de busca - busca em vizinhança ou busca local; ou heurísticas sistemáticas - árvores binárias ou árvores de busca.

Os métodos heurísticos geralmente são aplicados na resolução de problemas complexos. Um exemplo é Brandimarte (1993), que utilizou heurísticas específicas como regras de despacho na solução do subproblema de roteamento para resolver o FJSP. Conforme o autor, as principais regras utilizadas foram:

- SPT - prioriza as operações que demandam menor tempo de processamento. Desta maneira, considerando P_{ij} como tempo de processamento da operação O_{ij} do $job\ j_i$ na máquina M_k, a prioridade da regra é dada por $\frac{1}{P_{ijk}}$. Quando um

peso *w* é dado a um *job* (como no caso do atraso ponderado) a regra ponderada (WSPT) é dada por $\frac{\mathrm{w_i}}{P_{ij}\mathrm{k}}$.

- Menor Quantidade de Trabalho Restante (do inglês *Least Work Remaining* - LWKR) - a regra é dada por $\frac{1}{\sum_{\mathrm{q}\in A_{ij}} P_{iq}}$, onde A_{ij} é o conjunto de operações restantes e P_{iq} é a média do tempo de processamento das operações.
- Maior Quantidade de Trabalho Restante (do inglês *Most Work Remaining* - MWKR) - a regra é dada por $\sum_{q\in A_{ij}} P_{iq}$.
- Data de Vencimento mais Precoce (do inglês *Earliest Due Date*) - a regra é dada por $\frac{1}{d_i}$.

Segundo Dorigo e Stützle (2004), as heurísticas geram apenas um número muito limitado de diferentes soluções, ou param na falta de qualidade ótima local. Várias abordagens, que são denominadas metaheurísticas, foram propostas para ignorar esses problemas.

Uma metaheurística é um conjunto de conceitos algorítmicos que podem ser usados para definir métodos aplicáveis a um amplo conjunto de problemas diferentes. Em outras palavras, uma metaheurística pode ser vista como um método heurístico de propósito geral projetado para orientar uma heurística específica do problema subjacente para regiões promissoras do espaço de busca contendo soluções de alta qualidade. Um método metaheurístico é, portanto, uma estrutura algorítmica geral que pode ser aplicado a problemas de otimização diferentes com relativamente poucas modificações, e se adequa a um problema específico.

Ainda conforme os autores, o uso de metaheurísticas aumentou significativamente a capacidade de encontrar soluções de alta qualidade para problemas de otimização combinatória com grau de complexidade elevado em um tempo razoável.

Um exemplo do uso de metaheurísticas são as aplicações destas abordagens na resolução do FJSP. Neste sentido, é possível destacar alguns algoritmos (KACEM et al., 2002a; XIA e WU, 2005; ZHANG et al., 2009; XUE et al., 2014; DENG et al., 2017):

- Algoritmo Genético (do inglês *Genetic Algorithm* - GA) - é um algoritmo evolutivo inspirado na genética. Possui um conjunto de indivíduos, que comparados com o sistema biológico pode ser chamado de cromossomos. Estes algoritmos funcionam com os mesmos mecanismos genéticos habituais, e o conjunto de soluções apresentadas é chamado de população.
- Otimização por Enxame de Partículas (do inglês *Particle Swarm Optimization* - PSO) - simula o comportamento dos pássaros voadores e seus meios de troca de informações. Desta forma, combina busca local por experiência própria e busca global por experiência vizinha, gerando uma alta eficiência de pesquisa. O PSO é inicializado aleatoriamente com uma população de indivíduos (soluções candidatas) conceituado como partículas.
- Busca Tabu (do inglês *Tabu Search* - TS) - é uma metaheurística aplicada a um algoritmo de busca local, cujo objetivo é auxiliar na exploração de um espaço de busca. Este método permite que a pesquisa explore soluções que não diminuam o valor da função objetivo e geralmente obtém novos resultados através do acompanhamento das últimas soluções adquiridas no processo.
- Recozimento Simulado (do inglês *Simulated Annealing* - SA) - originou na analogia entre o processo físico do resfriamento de um metal em estado de fusão. Este processo é simulado em um algoritmo para resolver problemas complexos.
- Aproximação por Localização (do inglês *Approach by Localization* - AL) - algoritmo que reduz o espaço de busca dentro de uma área onde as probabilidades de encontrar as melhores soluções são menores.
- Algoritmo Imune Artificial (do inglês *Artificial Immune Algorithm* - AIA) - simula o sistema imunológico biológico. No modelo imunológico, o operador da imunidade é composto por uma vacinação e uma seleção imune.
- Otimização por Colônia de Abelhas (do inglês *Bee Colony Optimization* - BCO) - simula o comportamento das abelhas na natureza. As técnicas que as abelhas utilizam na busca de alimentos e no processo do trabalho em grupo são capazes de resolver problemas de alta complexidade. Estas características deram origem a uma metaheurística eficiente.

2.3.2 Metaheurística ACO

A Otimização por Colônia de Formigas é uma metaheurística inspirada no comportamento forrageiro das formigas. As formigas andam aleatoriamente em busca de alimento, e quando caminham depositam no solo uma substância química denominada de feromônio. Algumas espécies utilizam o feromônio para fazer trilhas e marcar os caminhos entre o ninho e a fonte de alimento, o que aumenta a probabilidade das outras formigas seguirem o mesmo caminho. Ao encontrar alimentos, as formigas retornam para o ninho seguindo a trilha de feromônio, assim o caminho que possui maior quantidade de feromônio pode ser o melhor ou o mais curto. As formigas coordenam suas atividades de forma indireta por estimergia (do inglês *stigmergy*), ou seja, elas modificam o ambiente local e as demais respondem a esta mudança após algum tempo. O feromônio sofre evaporação ao longo do tempo, e este processo conduz algumas formigas a seguirem outros caminhos, que possivelmente não sejam os melhores (DORIGO e STÜTZLE, 2004; NETO e BECCENERI, 2009).

2.3.2.1 Origem dos Algoritmos ACO

O termo ACO define um conjunto de algoritmos inspirados no comportamento de colônias de formigas. Segundo Dorigo e Stützle (2004), as colônias de formigas são grupos de insetos sociais que apesar da simplicidade de seus indivíduos, apresentam uma organização muito bem estruturada na divisão de tarefas. O comportamento destes insetos no que diz respeito ao trabalho em grupo compõe inteligência coletiva, e o resultado destas operações conduz as formigas a realizarem tarefas complexas. Esta forma de aprendizado contribuiu com os avanços da computação bioinspirada e foi tomada como referência na construção de vários métodos inteligentes no campo da IA.

O modelo de algoritmo inspirado em colônia formigas foi proposto por Marco Dorigo em 1992, em sua tese de doutorado. Esta metaheurística foi referenciada nos modelos derivados da observação do comportamento das formigas, como fonte de inspiração ao projeto de novos algoritmos para a solução de problemas de otimização combinatória. A ideia principal é que os princípios do comportamento das formigas reais podem ser explorados para

coordenar populações de agentes que colaboram para resolver problemas computacionais. Vários aspectos diferentes do comportamento de colônias de formigas, principalmente no que diz respeito à divisão do trabalho, classificação de ninhada e transporte cooperativo, inspiraram diferentes tipos de algoritmos de formigas. O modelo natural de trilha de feromônio foi a inspiração para um sistema artificial. As formigas artificiais são denominadas de agentes, e o nível de feromônio artificial permite as formigas encontrarem as melhores soluções em um determinado caminho.

Viana et al. (2008) demonstraram a correspondência entre a Biologia e a Computação no que diz respeito às formigas reais e o algoritmo ACO. Tal correspondência é representada na Tabela 2.2.

Tabela 2.2 - Correspondência entre as formigas reais e o algoritmo ACO.
Fonte: Adaptado de Viana et al. (2008)

Formigas Reais	Algoritmo ACO
Formiga	Agente
Colônia de formigas	Conjuntos agentes conhecidos como formigas artificiais
Trilha de feromônio	Modificação do ambiente
Evaporação do feromônio	Redução da modificação do ambiente com o passar do tempo
Possíveis caminhos entre o ninho e a fonte de alimentos	Possíveis soluções
Caminho mais curto	Melhor solução
Comunicação das formigas através do feromônio	Otimização do processamento

Conforme Dorigo e Stützle (2004), o primeiro algoritmo criado com referência em colônia de formigas foi o AS, proposto em três versões: *AS-density*, *AS-quantity* e *AS-cycle*, na qual se diferenciam pelo processo de atualização do feromônio (DORIGO et al., 1991; COLORNI, et al., 1992; DORIGO, 1992). O trabalho de Dorigo (1992) foi a referência para estudos e pesquisas com enfoque na construção de outras variantes até a padronização de uma estrutura comum para esses algoritmos. Estes estudos deram origem a outros algoritmos similares que podem ser destacados como sucessores diretos do AS: Sistema de Formigas Elitista (do inglês *Elitist Ant System* - EAS), proposto por Dorigo (1992) e Dorigo et al. (1996); Sistema de Formigas Baseado em Classificação (do inglês *Rank-Based Ant System* - Asrank), proposto por Bullnheimer et al. (1999); e MAX-MIN *Ant System* (MMAS), proposto inicialmente por Stützle e Hoos (1997). As pesquisas contribuíram também para a evolução do modelo com introdução de novos mecanismos, que deram origem as extensões do AS:

Sistema de Colônia de Formigas (do inglês *Ant Colony System* - ACS), proposto inicialmente por Dorigo e Gambardella (1997a, 1997b); e Pesquisa Aproximada de Árvore Não-Determinística (do inglês *Approximate Nondeterministic Tree Search* - ANTS), proposto por Maniezzo (1999). Dentre estas variantes, surgiram outros algoritmos que formam descontinuados pelas substituições de algumas versões. A metaheurística ACO apresentada em Dorigo e Di Caro (1999) foi o resultado positivo de um estudo que padronizou todas as variantes do AS em um só modelo.

Segundo os autores o AS foi proposto inicialmente para resolver o Problema do Caixeiro Viajante (PCV) e posteriormente foi aplicado a outros problemas de otimização, tais como: problema da atribuição quadrática, coloração de grafos, roteamento de rede de telecomunicações, roteamento de veículos, despacho de carga, escalonamento, ordenação sequencial, satisfação de restrições e programação *Job-Shop*.

O PCV consiste na procura de um caminho (circuito) que possua a menor rota para percorrer uma série de cidades, visitando uma única vez cada uma delas, e retornando à cidade de origem. Assim, quanto maior o número de cidades, maior é o espaço de procura.

Na figura 2.3 é ilustrada a convergência do ACO na resolução do PCV.

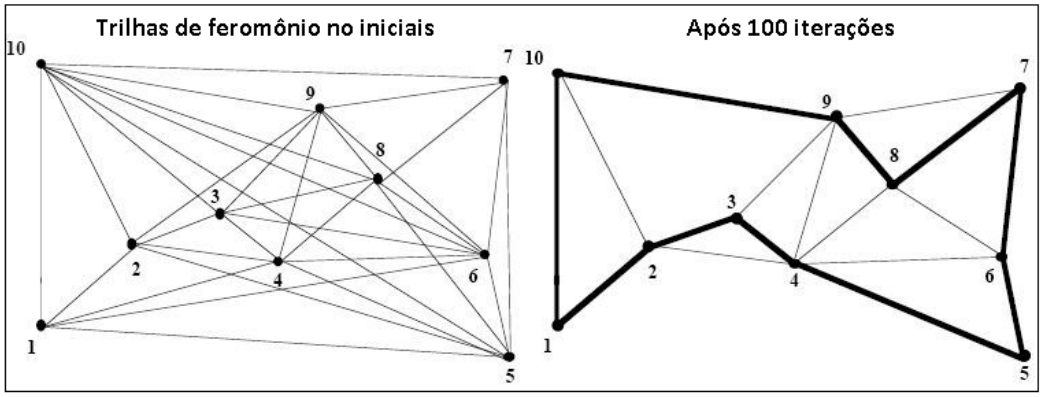

Figura 2.2 - Convergência do ACO na resolução do PCV.
Fonte: Adaptado de Dorigo (1992)

Para Neto e Becceneri (2009), os algoritmos ACO apresentam diversas soluções por meio de diferentes formigas. As melhores formigas influenciam as demais através da atualização do feromônio nos caminhos, sendo que a quantidade de feromônio reflete a convergência do algoritmo. Uma trilha de feromônio é utilizada para obter boas soluções, mas o algoritmo perde a eficiência com a evaporação. A metaheurística precisa equilibrar o processo de atualização do feromônio para que o algoritmo possa apresentar uma solução próxima da ótima.

2.3.2.2 Funcionamento e Estrutura do ACO

Dorigo e Stützle (2004) apontaram que o algoritmo ACO original para solução de problemas de otimização combinatória por meio de grafos é conhecido como Sistema de Formigas ou Otimização por Colônia de Formigas Simples (do inglês *Simple Ant Colony Optimization* - S-ACO). O modelo matemático do comportamento de colônias de formigas descreve o funcionamento do algoritmo. Dado um grafo $G(V, E)$ com arestas i, j representando um problema, o algoritmo relaciona cada aresta do grafo a uma trilha de feromônio artificial τ_{ij} e coloca cada formiga artificial em um determinado vértice (nó). Ao decorrer do processamento, as formigas selecionam o próximo nó de forma aleatória, percorrendo um caminho seguindo uma fórmula probabilística em função do feromônio, assim as informações das arestas são utilizadas para a tomada de decisão, e a quantidade inicial de feromônio é constante em todos os arcos ou arestas do grafo. Conforme os autores, o algoritmo funciona em duas etapas: (1) construção das soluções; (2) atualização da trilha de feromônio.

Na etapa (1), as formigas aplicam uma regra de transição de forma probabilística para a escolha do próximo nó a ser visitado em cada etapa de construção das soluções. Segundo os autores, a probabilidade p_{ij}^{k} de uma formiga k decidir caminhar em i, j pode ser vislumbrado na expressão 2.1.

$$p_{ij}^{k} = \begin{cases} \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{1 \in N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}, & if\ j \in N_i^k, \end{cases} \tag{2.1}$$

Onde:

- τ_{ij} define a quantidade de feromônio na trilha;
- η_{ij} define um valor heurístico conforme as características do problema;
- α é um parâmetro que determinam a influência da trilha de feromônio;
- β é um parâmetro que controla a informação heurística;
- l define o conjunto de possíveis soluções;
- N_i^k é o conjunto de nós de vizinhança viável do nó i, associado à $k - ésima$ formiga, ou seja, o conjunto de nós que a formiga k ainda não visitou. A probabilidade de uma formiga escolher um nó que não pertence a N_i^k é zero.

Desta maneira, cada formiga k mantém uma memória indicando os nós já percorridos, o que define a vizinhança N_i^k. Após a construção de todos os percursos, é aplicada a atualização do feromônio nas trilhas, conforme a equação 2.2.

$$\tau_{ij} \leftarrow \tau_{ij} + \triangle \tau^k \tag{2.2}$$

Nesta etapa (2), a k-ésima formiga retorna do nó visitado para o nó anterior depositando uma quantidade $\triangle \tau^k$ nos arcos visitados. O algoritmo define um valor constante de $\triangle \tau^k$ para todas as formigas, sendo que a intensificação do feromônio nas trilhas define os melhores percursos, ou caminhos mais curtos.

Com a evaporação do feromônio, o algoritmo tende a convergir para uma solução próxima da ótima, conforme a equação 2.3. Este processo conduz as formigas a percorrerem caminhos alternativos durante a busca, evitando uma estagnação da solução em um ótimo local.

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \qquad \forall (i,j) \in A; \qquad \rho \in [0,1] \tag{2.3}$$

Onde:

ρ é o parâmetro que define a taxa de evaporação de feromônio atribuído no AS, usado para evitar a convergência das formigas em outras regiões do espaço de busca.

O valor de τ é alterado a cada iteração do algoritmo no processo de busca. Esta alteração corresponde ao aumento e a diminuição do feromônio nos caminhos.

No mundo real, as formigas fazem o percurso do ninho até a fonte de alimentos e posteriormente o percurso inverso. Conforme modelo matemático do comportamento de colônias de formigas, a correspondência das formigas artificiais com as formigas reais pode ser descrita da seguinte maneira: no grafo, o nó de origem pode ser considerado como o ninho; o próximo nó pode ser considerado como a fonte de alimentos; as arestas podem ser consideradas como as trilhas de feromônio que levam o caminho do ninho até a fonte de alimentos, ou vice versa.

Na figura 2.2 é demostrado a construção da solução no processo de busca pelo caminho mais curto de uma fonte para o nó de destino.

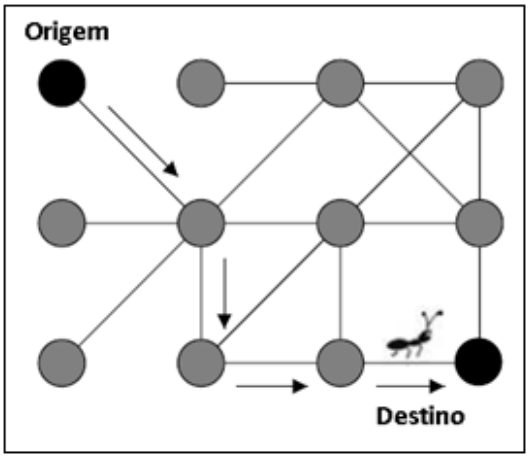

Figura 2.3 - Construção da solução no processo de busca.
Fonte: Adaptado de Dorigo e Stützle (2004)

Segundo Dorigo e Stützle (2004) a metaheurística ACO apresentado em Dorigo e Di Caro (1999) diferencia do algoritmo S-ACO pela quantidade de feromônio depositado durante o processamento por cada formiga a cada iteração. Conforme os autores, o S-ACO pode ser resumido no pseudocódigo 2.1.

Pseudocódigo 2.1 - Algoritmo de Otimização por Colônia de Formigas Simples

```
01: Inicio
02:     Inicialização dos parâmetros
03:     Repita iteração
04:         Repita Schedule Activities
05:             Cada formiga aplica uma regra de transição
06:             Atualização local da trilha de feromônio
07:             Ações daemon {opcional}
08:         Até que todas as formigas tenham apresentado uma solução
09:         Realiza o procedimento de busca local
10:         Realiza a atualização global da trilha de feromônio
11:     Até que o critério de parada seja satisfeito
12: Fim
```

A programação das atividades (do inglês *Schedule Activities*) não especifica a programação e a sincronização das três etapas no processamento do algoritmo. Em outras palavras, não diz se as atividades devem ser executadas de forma completamente paralela e independente, ou se algum tipo de sincronização é necessário entre as rotinas. O designer é

livre para especificar a maneira que os procedimentos devem interagir, levando em consideração as características do problema considerado.

Ainda segundo os autores, as ações *daemon* determinam uma rotina específica destinada a aperfeiçoar as ações de busca na tomada de decisão. São usadas para implementar ações centralizadas que não podem ser realizadas por uma única formiga. Um exemplo é a observação do caminho encontrado por cada formiga e a seleção das formigas que criaram as melhores soluções na iteração do algoritmo. Tais ações são opcionais, mas ajudam o algoritmo a convergir para as melhores soluções.

2.3.2.3 Os Avanços do ACO

Conforme apresentado na sessão 2.3.2.1, o ACO possui algumas variantes que contribuíram para a melhoria da metaheurística. Nessa seção serão discutidos os principais conceitos destas variantes, visando descrever as características de cada algoritmo citado anteriormente. Além disso, é mostrada a estrutura *Hyper* Cubo para ACO (do inglês *Hyper-Cube Framework for* ACO).

De acordo com Dorigo e Stützle (2004), a primeira melhoria no algoritmo AS foi o Elitist Ant System. O EAS é uma estratégia elitista que consiste em atualizar a trilha de feromônio apenas da melhor solução. Isto pode ser visto como feromônio adicional depositado por uma formiga chamada de melhor formiga.

Outra melhoria do AS é a versão *Rank-Based Ant System*. O *ASrank* apresenta uma característica na qual a regra de atualização do feromônio é aplicada somente pelas melhores formigas. A função de classificação leva em consideração o *rank* da formiga na construção da solução em relação às outras. Isto significa que as formigas com os melhores desempenhos têm a maior taxa de atualização da trilha de feromônio. Desta forma, cada formiga deposita uma quantidade de feromônio que pode diminuir conforme sua classificação no *rank,* além disso, a melhor formiga sempre deposita maior quantidade de feromônio a cada iteração.

O algoritmo MAX–MIN *Ant System* considera que apenas a melhor formiga de cada iteração possa atualizar as trilhas de feromônio, sendo que a quantidade do feromônio é

limitada por um valor mínimo e um valor máximo. O comportamento deste algoritmo se caracteriza diante de quatro principais modificações em relação ao AS: (1) explora fortemente os melhores caminhos, sendo que a melhor formiga da iteração atual ou a melhor formiga do grupo atualiza o feromônio; (2) limita os valores da faixa na trilha de feromônio para o intervalo $[t_{mim}, t_{max}]$ evitando que o algoritmo converge para um ótimo local; (3) as trilhas de feromônio são inicializadas para o limite superior t_{max}, aumentando a exploração dos caminhos na busca da solução; (4) as trilhas de feromônio são reiniciadas cada vez que o sistema se aproxima da estagnação ou quando não apresenta nenhuma melhoria após um determinado número de iterações. O MMAS é um dos algoritmos ACO mais estudados, e foi ampliado em várias maneiras. Em uma dessas extensões, a regra de atualização de feromônio ocasionalmente usa o melhor caminho encontrado desde a reinicialização mais recente das trilhas de feromônio durante o processamento.

Com as contribuições das pequenas mudanças na estrutura, os algoritmos ACO comentados até o momento alcançam um desempenho significativamente melhor em relação ao AS. As extensões do AS apresentaram melhorias de desempenho através da introdução de novos mecanismos baseados em ideias não incluídas no S-ACO original.

Um das principais extensões do AS é o *Ant Colony System*. Este algoritmo realiza a atualização do feromônio a cada iteração (local), além da atualização do feromônio realizada no fim do processo (global). O ACS se diferencia do S-ACO em três pontos principais: (1) explora a experiência de busca acumulada pelas formigas através do uso de uma regra de escolha mais agressiva e mais intensa em comparação com o AS; (2) o depósito e a evaporação do feromônio ocorrem apenas no melhor circuito, ou seja, somente a melhor formiga global deposita o feromônio a cada iteração do algoritmo; (3) a evaporação do feromônio ocorre nas conexões dos circuitos aumentando a exploração de outros percursos. Desta forma, sempre que é realizada a atualização global de feromônio, o algoritmo aplica também uma atualização local destes valores, assim a evaporação do feromônio ocorre imediatamente durante a etapa de construção da solução. Isto aumenta a exploração e evita que o algoritmo entre em estagnação.

Outra extensão que trouxe melhoria no desempenho do algoritmo AS foi o *Approximate Nondeterministic Tree Search*. O ANTS modificou a estrutura do cálculo da atualização da trilha de feromônio e combinou o S-ACO com técnicas de limites inferiores de programação matemática, no objetivo de melhorar o desempenho do AS original no que diz

respeito a construção de soluções e custo computacional. Este algoritmo é uma abordagem ACO que explora ideias da programação matemática. Em particular, calcula limites inferiores na conclusão de uma solução parcial para definir a informação heurística que é usada por cada formiga durante a construção da solução. O nome ANTS deriva do fato de que este algoritmo pode ser interpretado como uma pesquisa de árvores não determinística aproximada, tornando similar em estrutura aos algoritmos de busca em árvore.

Os estudos de Blum, Roli e Dorigo (2001) contribuíram nos avanços das pesquisas sobre algoritmos de colônia de formigas e introduziu o *Hyper-Cube Framework for* ACO. Este modelo redimensiona automaticamente os valores do feromônio para que eles permaneçam sempre no intervalo [0,1]. Esta escolha foi inspirada pela programação matemática da formulação de muitos problemas de otimização combinatória, nos quais as soluções podem ser representadas por vetores binários. Em tal formulação, as variáveis de decisão podem assumir os valores {0,1}, e normalmente correspondem aos componentes usados pelas formigas para construção da solução. Uma solução para o problema corresponde a n-dimensão do hipercubo, onde n é o número de variáveis de decisão. O algoritmo permite que cada variável de decisão tome valores no intervalo [0,1]. Neste caso, o conjunto de soluções viáveis consiste em todos os vetores que formam combinações convexas de vetores binários.

Capítulo 3

REVISÃO DA LITERATURA

3.1 Considerações Iniciais

É possível encontrar na literatura inúmeros trabalhos com propostas para a modelagem e solução do problema de programação da produção sobre vários aspectos. A busca por um método eficiente para obter as melhores soluções em tempo polinomial é a razão pela qual, várias técnicas de IA são aplicadas em pesquisas que envolvem o planejamento e controle da produção.

A pesquisa de Brandimarte (1993) foi pioneira e ainda é referência para os trabalhos sobre FJSP. O autor apresentou uma estratégia hierárquica para resolver o FJSP multiobjetivo que dividiu o problema em duas partes: (1) subproblema de roteamento; (2) subproblema de programação. Nesta abordagem, foi resolvido primeiro o subproblema de roteamento com aplicação de regras de despacho e posteriormente o subproblema de programação. Para o caso considerado, foi apresentada uma abordagem de dois níveis baseada na decomposição de um roteamento e uma programação *job-shop*, que foi obtido pela atribuição de cada operação de cada *job* a uma das máquinas equivalentes. Ambos os problemas foram abordados pelo algoritmo TS.

O autor propôs um benchmark denominado Brdata e apresentou um conjunto de instâncias de teste para validação da abordagem proposta. Desta forma, o pesquisador contextualizou que a solução do FJSP multiobjetivo pelo algoritmo TS agregado às regras de despacho de forma hierárquica, obteve bons resultados. Esta pesquisa motivou a comunidade acadêmica quanto uso da mesma metodologia em estudos dirigidos à resolução de problemas de programação *job-shop*.

Neste capítulo é apresentado um conjunto de trabalhos correlatos encontrados na literatura, que obtiveram resultados relevantes na resolução do problema de programação da produção em vários aspectos sobre diferentes cenários. As obras estão citadas na ordem cronológica em um texto conexo.

Na sessão 3.2 são destacados alguns trabalhos com aplicações de metaheurísticas na resolução do FJSP multiobjetivo. Os resultados publicados na literatura foram tomados como referência na validação do algoritmo proposto.

Na sessão 3.3 é apontado um grupo de trabalhos com aplicações da metaheurística ACO na programação da produção, visando especificar a maneira como esta metaheurística tem sido explorada como abordagem para tentativa de resolver este problema em diferentes cenários. Desta forma, são apresentadas pesquisas envolvendo JSP e FJSP sobre diferentes aspectos a fim de vislumbrar a eficiência e eficácia do ACO para a resolução deste problema de modo geral, e dar embasamento para esta pesquisa.

3.2 Aplicações de Metaheurísticas no FJSP Multiobjetivo

Kacem et al. (2002a) apresentou uma abordagem híbrida para resolver o FJSP multiobjetivo, visando minimizar o *makespan*, a carga de trabalho total das máquinas e a carga de trabalho da máquina mais carregada. Esta abordagem trata-se de um algoritmo genético controlado (do inglês *Controlled Genetic Algorithm* - CGA) que funciona em conjunto com o método AL em dois ambientes: totalmente flexível e parcialmente flexível.

Nesta pesquisa, os autores aplicaram o AL para resolver o problema de alocação de recursos e criar um modelo de atribuição ideal, e posteriormente aplicaram o CGA com a população inicial construída a partir do conjunto de atribuições encontradas pelo AL. Desta forma, a evolução das mutações geradas pelo CGA são controladas em duas fases: 1 - verificar se o novo indivíduo respeita o modelo imposto pelos esquemas de atribuição; 2 - aplicar a mutação para melhorar os operadores genéticos. Por fim, os autores aplicaram manipulações genéticas avançadas para melhorar a qualidade da solução.

Segundo os pesquisadores, o ambiente parcialmente flexível torna o problema mais difícil, complica o espaço de busca e aumenta o tempo de computação. Ao tratar deste

problema, eles transformaram o cenário parcialmente flexível em totalmente flexível, atribuindo um processamento fictício com tempo muito elevado, definido por 999 para cada estado proibido. O algoritmo evita atribuir uma operação a uma máquina cujo tempo de processamento é longo. Assim, estas tarefas serão rejeitadas automaticamente.

Ao realizar a validação do algoritmo proposto, os autores aplicaram o AL, o GA clássico e o método de decomposição temporal nos mesmos cenários. Os resultados dos testes mostraram que a abordagem híbrida de AL e CGA obteve melhores resultados em comparação com as demais abordagens em todos os cenários quanto ao *makespan* e a carga de trabalho total das máquinas. Os resultados da hibridização apresentaram uma pequena melhoria em comparação com os resultados do GA clássico e também em comparação com os resultados do AL, no entanto, houve uma melhoria significativa, em torno de 15% a 56% em comparação com os resultados do método de decomposição temporal.

O trabalho de Kacem et al. (2002b) foi uma extensão da pesquisa anterior. Foi proposto uma abordagem de Pareto baseada na hibridação de lógica Fuzzy e algoritmos evolutivos para resolver o FJSP multiobjetivo. Para os autores, a resolução da maioria dos problemas de otimização combinatória visam otimizar simultaneamente as funções objetivas, que muitas vezes geraram conflitos. A pesquisa investigou os possíveis ganhos e melhorias das soluções, dando prioridade à otimização das funções objetivas que apresentaram média de valores longe do valor ótimo. Desta forma, construiu um conjunto de soluções próximas ao ótimo que foram usadas para selecionar soluções de Pareto, e as demais foram rejeitadas.

Os autores apontaram que a avaliação multiobjetivo da qualidade da solução é reduzida a uma única função de fitness que mede a qualidade de acordo com os valores de limite inferior das diferentes funções objetivas. O método proposto tratou do problema em duas etapas:

- Estágio evolutivo de otimização multiobjetivo: foi proposto um método de AL e um CGA. O AL permite resolver o problema de alocação de recursos tendo em vista os tempos de processamento para minimizar a carga de trabalho total das máquinas; e produz um conjunto de atribuições para que se possa equilibrar a carga de trabalho das máquinas. O CGA funcionou a partir do conjunto de soluções iniciais fornecido pelo AL e consistiu em gerar um modelo de cromossomos que se adequa ao problema para construção de indivíduos cada vez melhor.

- Fase de avaliação multiobjetivo difusa: a aplicação da lógica Fuzzy foi usada para calcular os diferentes pesos a cada função objetivo e medir a qualidade de cada solução. Desta maneira, todas as funções objetivas foram agregadas na função fitness. A avaliação difusa foi iniciada pela determinação de um conjunto de limite inferior para o objetivo considerado. Na maioria dos casos, os valores das funções objetivas podem pertencer a diferentes intervalos, assim, para tornar a avaliação mais eficiente, devem ser homogeneizadas para evitar que algumas funções sejam sempre dominadas pelas outras.

Nesta abordagem, as probabilidades de mutação e *crossover* foram corrigidas de maneira clássica. Quanto ao mecanismo de seleção, foi dada prioridade aos melhores indivíduos de acordo com seus valores de fitness. Durante os ciclos de processamento, o CGA é parado quando o número das iterações Q for atingido.

Segundo os pesquisadores, a abordagem proposta não garantiu as melhores soluções, no entanto, mostrou-se eficiente por ter fornecido soluções de boa qualidade com custo computacional baixo.

Xia e Wu (2005) propuseram uma abordagem híbrida de PSO e SA para resolver o FJSP multiobjetivo em ambientes totalmente flexíveis e parcialmente flexíveis. Nesta abordagem, o algoritmo PSO fornece soluções iniciais para o algoritmo SA durante o processo de busca híbrido. Desta forma, o valor fitness de cada partícula é calculado pelo algoritmo SA. Assim, o SA atua no espaço de busca das soluções; e o PSO usa as soluções avaliadas pelo SA para continuar a evolução.

Neste trabalho, os pesquisadores dividiram os subproblemas de roteamento e programação, para que cada algoritmo possa ser encarregado de uma determinada tarefa, caracterizando a hibridização. Durante o processo de construção das soluções, o PSO resolve o subproblema de roteamento enquanto que o SA resolve o subproblema de programação. Neste processo, foram atribuídos níveis de precedência aos recursos, dando prioridade as operações que demandam menor tempo de processamento, e a partir deste procedimento, o algoritmo gera a população inicial aleatoriamente. Quando o tempo de processamento de uma máquina for igual a outro, a máquina que possui o número de ordem inferior tem prioridade. Após este curso, obtêm-se diferentes níveis de prioridade para todas as máquinas que processam a mesma operação. Em seguida, a posição da partícula pode ser gerada

estocásticamente de acordo com a ordem das operações, e neste procedimento o algoritmo diminui o espaço de busca para melhorar a qualidade da solução e o custo computacional. Cada partícula do enxame compartilha informações mútuas globalmente e se beneficia das descobertas e experiências anteriores de todas as outras partículas durante o processo de busca.

Ao tratar do problema FJSP no ambiente parcialmente flexível, os autores utilizaram o mesmo procedimento de Kacem et al. (2002a). Os resultados das soluções encontradas pela abordagem proposta nos cenários totalmente flexíveis e parcialmente flexíveis foram validados com três funções objetivas: $f1$ - *makespan*; $f2$ - *machine workload*; e $f3$ - *total workload*. Os valores encontrados na solução de cada objetivo são transformados em um valor de aptidão pelo método de soma ponderada, dado por: $f = w1(f1) + w2(f2) + w3(f3)$, onde os valores de $w1, w2\ e\ w3$ são pesos definidos conforme cada função objetivo.

O algoritmo híbrido foi avaliado em comparação com os resultados obtidos das abordagens de outros autores em três instâncias representativas. Segundo os pesquisadores a abordagem proposta mostrou-se eficiente por ter apresentado soluções de boa qualidade, mas não gerou soluções ótimas. Em uma instância com 56 operações, o algoritmo apresentou solução quase ótima com qualidade bem superior aos resultados de Kacem et al. (2002b) e nas demais instâncias os resultados foram equivalentes.

O trabalho de Zhang et al. (2009) ainda é uma das referências importantes na literatura quanto aos estudos do FJSP multiobjetivo. Os autores propuseram uma abordagem híbrida de PSO e TS na mesma linha de pesquisa de Xia e Wu (2005), mas com algumas particularidades. Eles usaram o TS para tratar o problema de roteamento e o PSO com algumas melhorias para tratar do problema de programação. O método de atualização da velocidade das partículas do PSO foi alterado para o modelo *crossover*, permitindo a troca das operações pré-definidas pelas operações determinadas por outra partícula.

De acordo com os autores, o PSO possui alta eficiência de pesquisa ao combinar busca local por experiência própria e busca global por experiência vizinha. O TS como um algoritmo de busca local, emprega certa probabilidade para evitar estagnação em ótimo local. O hibridismo destes dois algoritmos melhorou a eficiência da busca local e o custo computacional na resolução do FJSP multiobjetivo. Cada partícula do PSO obtém informações de atualização de suas experiências próprias e do enxame desde o início, e depois convertem as partículas atuais em uma solução do FJSP para pesquisa local usando o TS.

Assim, o PSO usa as melhores soluções para continuar a busca até que a condição de parada seja satisfeita.

Nesta abordagem, a função *Fitness* é usada para avaliar cada partícula no enxame, e cada indivíduo é avaliado para as funções objetivas pelo método de soma ponderada. A abordagem foi avaliada mediante comparação com algoritmos de outros autores em quatro instâncias representativas retiradas de Kacem et al. (2002a). Segundo os pesquisadores, o algoritmo híbrido demonstrou eficácia na resolução do problema pelos resultados obtidos e tempo computacional gasto. Em comparação como trabalho de Xia e Wu (2005) os resultados foram equivalentes, mas houve melhoria no tempo de processamento dos recursos.

Xue et al. (2014) apresentaram um algoritmo imune melhorado com os princípios da computação quântica para resolver o FJSP multiobjetivo. O método AIA proposto observa dois subproblemas: (1) atribuição de cada operação para as máquinas alternativas; (2) pedido das operações em cada máquina. Desta forma, considera três objetivos de minimização: $f1$ - *makespan*; $f2$ - *machine workload*; e $f3$ - *total workload*.

No processo de resolução do problema, o anticorpo prolifera e divide-se em um conjunto de subpopulação em grupo. Os anticorpos em um grupo de subpopulação foram representados por bits quânticos de vários estágios. Durante a atualização do anticorpo, o canal geral de rotação quântica e o mecanismo de ajuste dinâmico foram aplicados para acelerar a convergência do algoritmo. Os testes iniciais da abordagem proposta foram concluídos usando somente o algoritmo imunológico. Os princípios da computação quântica foram adicionados posteriormente no decorrer dos testes.

Os autores contextualizaram que a estrutura do esquema de codificação para resolver o FJSP usado nesta abordagem consiste em duas partes: (A) representa as máquinas que processam as operações; (B) representa a sequência das operações. O número de codificações em cada parte é igual à soma das operações em todos os *jobs*. Na parte A, cada valor mostra o número da máquina, e na parte B cada valor mostra o número do *job*. Com base em todas as descrições do problema, o modelo matemático do FJSP multiobjetivo foi definido da seguinte maneira: $f = w1(f1) + w2(f2) + w3(f3)$, onde $w1, w2\ e\ w3$ são pesos atribuídos a cada objetivo. Para a soma ponderada os pesos foram dados por: $w1 = 0{,}5$; $w2 = 0{,}3$; $w3 = 0{,}2$. O algoritmo imunológico apresentado pode ser resumido no pseudocódigo 3.1:

Pseudocódigo 3.1 - Algoritmo Imune Artificial

```
01: Inicio
02:     Criar uma população aleatória inicial A
03:     Extrair as vacinas de acordo com o conhecimento anterior
04:     Se a população atual for pior
05:         Realize a vacinação em A(k) e obtenha A'(k)
06:         Execute a seleção imune em A'(k) e obtenha o próximo A(k+1)
07:         Retorne ao passo 03
08:     Senão
09:         Finalizar a rotina
10:     Fim-Se
11: Fim
```

A abordagem foi testada e validada nos cenários de Kacem et al. (2002a). Inicialmente o algoritmo imune foi testado no problema 4x5, e os testes mostraram que esta abordagem converge para melhores soluções após certo período de tempo.

Na codificação inicial, o algoritmo apresentou os seguintes resultados: *makespan* = 13; *workload* = 12; e *total workload* = 33; então $f = 16{,}7$. De acordo com o processamento do algoritmo imunológico, o indivíduo pode ser transformado em uma permutação de operação com índices de codificação. O esquema de codificação inicial, dado por [A(3,4,2,1,2,1,4,1,5,3,3,2); B(3,1,1,2,3,4,1,3,2,2,3,4)] é alterado para um novo esquema de codificação, dado por [A(3,4,2,1,2,1,4,1,5,2,3,4); B(3,1,1,2,3,4,1,3,2,2,3,4)]. No novo esquema de codificação, o algoritmo apresentou os seguintes resultados: *makespan* = 12; *workload* = 10; e *total workload* = 34; então $f = 15{,}8$.

Os autores afirmaram que na computação quântica a unidade básica de informação é o Qubit. Um Qubit é representado por um vetor de estado no sistema de mecânica quântica de dois níveis, sendo equivalente a um vetor de espaço bidimensional definido por 2-vetor em espaço vetorial complexo, e pode estar no estado 1 ou no estado 0. Com base no algoritmo imune e no princípio quântico, o conceito quântico é usado na população inicial de forma aleatória. Os anticorpos em um grupo de subpopulação são representados por bits quânticos de vários estágios. Desta forma, os pesquisadores aplicaram o algoritmo com base nos os princípios quânticos e imunológicos, denominado Algoritmo Imune Quântico (do inglês *Quantum Immune Algorithm* – QIA) para resolver o FJSP.

O processamento do QIA obteve o melhor o esquema de codificação, dado por [A(3,4,2,1,2,1,4,1,5,3,4,2); B(3,1,1,2,3,4,1,3,2,2,3,4)]. Aplicando esta codificação, o QIA

apresentou a seguinte solução: *makespan* = 11; *workload* = 11; *total workload* = 32 e $f = 15{,}2$.

Os autores realizaram uma comparação entre o AIA e o QIA para confirmar a eficiência da abordagem proposta. Eles aplicaram os dois algoritmos no problema 10x10 e os testes mostraram que o QIA foi superior quanto aos resultados apresentados e o tempo de convergência para melhor solução. O QIA foi aplicado também nos problemas 8x8 e 15x10, e apresentou a solução ideal nas duas instâncias. Os resultados mostraram que o QIA obteve boa solução em todos os cenários testados. Os autores validaram a abordagem proposta comparando seus resultados com outras abordagens citadas na literatura e afirmaram que o QIA apresentou as melhores soluções na resolução do FJSP em comparação com os outros algoritmos.

Huang et al. (2016) propuseram um algoritmo de otimização por enxame de partículas multiobjetivo (do inglês *Multi-Objective Particle Swarm Optimization* - MOPSO) integrado a um método de busca em vizinhança variável (do inglês *Variable Neighborhood Search* - VNS) para resolver o FJSP multiobjetivo. Os autores introduziram regras de atribuição e de despacho para iniciar a população; e operadores discretos especiais para produzir novos indivíduos. Nesta abordagem o MOPSO foi projetado para minimizar três funções objetivas: (1) tempo de conclusão das operações; (2) carga da máquina mais crítica; e (3) carga total de trabalho de todas as máquinas.

Para resolver o FJSP multiobjetivo, a inicialização do procedimento inclui a atribuição das operações nas máquinas e o sequenciamento das operações. A hibridização das regras de atribuição e de despacho mostrou ser eficiente para obter indivíduos mais promissores a população inicial.

Segundo os autores, os operadores discretos especiais fazem parte de uma versão discreta do PSO incorporado no MOPSO. Estes operadores geram arquivos de melhor qualidade e são atualizados por uma estratégia de atualização de arquivo não dominada. Esta estratégia foi usada para selecionar as melhores posições, e preservar as que não foram dominadas. Memórias cognitivas e memória social podem manter a diversidade de soluções não dominadas, e o balanço entre busca local e busca global. O VNS foi aplicado para melhorar a capacidade de exploração global.

O processo de validação dos resultados e avaliação de desempenho do MOPSO foi realizado em quatro instâncias de Kacem et al. (2002a) e dez instâncias de Brandimarte (1993). Os testes foram realizaram em duas etapas: (1) instâncias de Kacem et al. (2002a) que

variam de 4 *jobs* × 5 máquinas para 15 *jobs* × 10 máquinas; (2) instâncias de Brandimarte (1993) que variam de 10 *jobs* × 6 máquinas para 20 *jobs* × 15 máquinas.

Segundo os pesquisadores, os experimentos demonstram que o algoritmo MOPSO possui melhor desempenho que algumas técnicas para resolver FJSP multiobjetivo, mas é sensível a diferentes combinações de parâmetros, e apresentou diferentes soluções em algumas instâncias. Eles afirmaram que a vantagem do MOPSO é o uso de memórias cognitivas e uma memória social. Este mecanismo de busca pode efetivamente evitar estagnação e melhorar as soluções. Assim, foi demonstrada a razão pela qual o algoritmo proposto obteve soluções de boa qualidade na maioria das instâncias testadas.

Deng et al. (2017) propuseram resolver o FJSP multiobjetivo por meio de um guia evolutivo de abelhas com algoritmo genético de classificação não dominada II (do inglês *Bee Evolutionary Guiding Nondominated Sorting Genetic Algorithm II* - BEG-NSGA-II) para minimizar o *makespan*, o *workload* e o *total workload*.

A abordagem proposta adotou um mecanismo de otimização em dois estágios durante o processo de otimização: (1) o algoritmo genético de classificação não dominada II com tempos de iteração é usado primeiramente para obter a população inicial, na qual um esquema de guia evolutivo de abelhas é apresentado para explorar extensivamente o espaço de solução; (2) o algoritmo genético de classificação não dominada II com tempos de iteração otimizado é usado para obter as soluções ótimas de Pareto, nas quais um mecanismo de atualização e alguns operadores genéticos úteis foram empregados para melhorar a capacidade de busca e evitar a convergência prematura do esquema de orientação evolutiva de abelhas. Este se concentra na exploração do espaço de solução. A população consiste em três partes, e cada uma delas muda com o tempo de iteração.

No processo de avaliação da abordagem, os pesquisadores realizaram três experimentos em algumas instâncias de benchmark publicadas: (1) quatro instâncias representativas de Kacem et al. (2002a); (2) dezoito instâncias de Dauzère-Pérèz e Paulli (1997); (3) dez instâncias de Brandimarte (1993). Os autores contextualizaram que o BEG-NSGA-II proposto obteve diferentes soluções de Pareto para a maioria dos benchmarks com tempo computacional razoável.

A pesquisa mostrou que o método de inteligência computacional (BEG-NSGA-II) poderia ser utilizado em problemas de programação de *job-shop* flexíveis, especialmente para resolver problemas de otimização multiobjetivo na programação da produção. A eficácia da abordagem foi demonstrada na comparação dos seus resultados experimentais com os resultados de alguns algoritmos já conhecidos na literatura.

3.3 Aplicações da Metaheurística ACO na Programação da Produção

Liouane et al. (2007) apresentaram uma combinação do algoritmo *Ant System* com o *Tabu Search* para resolver o FJSP. Nesta abordagem, o FJSP é representado por meio de um grafo bipartido com duas categorias de nós: O_{ij} e M_K. A operação é mapeada para o nó O_{ij}; a máquina é mapeada para o nó M_K. Assim, a operação correspondente deve ser atribuída à máquina correspondente respeitando a disponibilidade da máquina e as restrições de precedência entre as operações de diferentes *jobs*. No grafo, a distância entre O_{ij} e M_K representa o tempo de processamento de uma determinada operação em uma determinada máquina.

Os autores transformaram o FJSP em um problema de formiga viajante para o algoritmo *Ant System*. Assim, as formigas procuram viajar de um nó para o outro depositando feromônio, de modo que todas as restrições sejam satisfeitas. Cada uma das formigas constrói uma solução usando uma combinação das informações fornecidas pela trilha de feromônio e pela função heurística definida. O Sistema de Formigas determina que seja realizada a atualização de feromônio quando todas as formigas apresentarem a solução do problema. Este processo é estabelecido a cada iteração do algoritmo e permite que as formigas compartilhem informações. O fator de evaporação garante que o feromônio não é acumulado infinitamente e denota a proporção da substância que é levada para a próxima iteração do algoritmo. A abordagem híbrida de *Ant System* e *Tabu Search* funciona da seguinte maneira: o algoritmo TS age como um mecanismo que avalia a qualidade da melhor solução apresentada pelo AS.

O algoritmo permite que as formigas construam suas soluções e, em seguida, as soluções resultantes são levadas a um ótimo local pelo mecanismo de pesquisa local. Cada uma dessas soluções é usada no estágio de atualização de feromônio. A resolução do FJSP determina a escolha da máquina responsável para processar determinada tarefa e verificar se qualquer operação O_{ij} pode ser trocada entre outras máquinas, o que pode resultar em um menor valor do *makespan*.

A pesquisa local considera uma máquina problemática por vez e tenta trocar uma operação da máquina problemática com qualquer outra máquina (sem problema) na solução (operações que não sejam problema). Então, as formigas são usadas para gerar soluções promissoras de programação da produção, e o algoritmo TS é usado para tentar melhorar

essas soluções. A busca tabu é realizada em cada máquina problemática e continua até que não haja mais nenhuma melhoria da solução quanto ao valor de *makespan*.

Os autores descreveram as etapas de solução do FJSP pelo algoritmo híbrido proposto e todo procedimento pode ser resumido no pseudocódigo 3.1.

Pseudocódigo 3.2 - Abordagem híbrida de *Ant System* e *Tabu Search* para o FJSP

```
01: Inicio
02:     Iniciar os parâmetros do Ant System
03:     Criar uma solução inicial e uma lista tabu vazia de um determinado tamanho
04:     Repita
05:         Encontrar nova solução
06:         Avaliar a qualidade da nova solução
07:         Se a nova solução for melhor
08:             A nova solução assume a solução inicial
09:         Senão
10:             Aplicar a otimização de Tabu Search
11:         Fim-Se
12:         Se a lista tabu estiver cheia
13:             Excluir a entrada mais antiga da lista
14:         Fim-Se
15:         Adicionar a solução à lista de tabu
16:         Aplicar o procedimento de atualização de feromônio
17:     Até que o critério de parada seja satisfeito
18: Fim
```

A abordagem foi avaliada em seis instâncias representativas de FJSP retiradas de Brandimarte (1993) e Kacem et al. (2002a, 2002b). Os diferentes resultados obtidos foram apresentados e comparados com outras abordagens encontradas na literatura. O algoritmo com abordagem híbrida foi processado em 1000 iterações, utilizando 10 formigas e executado 10 vezes em todas as instâncias.

Segundo os pesquisadores, a eficiência da abordagem pode ser explicada pela qualidade das soluções obtidas durante o processamento da combinação do algoritmo *Ant System* com a heurística Busca Tabu. As soluções apresentadas superaram algumas técnicas encontradas na literatura em algumas instâncias. Os experimentos mostraram que as soluções obtidas foram consideradas satisfatórias no que diz respeito às instâncias do FJSP em um tempo de computação polinomial.

Xing et al. (2010) propuseram um algoritmo de otimização por colônia de formigas com base no conhecimento (do inglês *Knowledge Based Ant Colony Optimization* - KBACO) para resolver o FJSP. Esta abordagem é uma integração entre o modelo de arquitetura de pesquisa heurística com base no conhecimento (do inglês *Knowledge Based Heuristic Searching Architecture* - KBHSA) e o ACO. Assim, o modelo de conhecimento aprende algum conhecimento disponível da solução encontrada pelo ACO e, em seguida, aplica o conhecimento existente para orientar a pesquisa heurística atual.

A arquitetura KBSHA proposta é funcionalmente dividida em dois módulos:

- Modelo de Conhecimento - aprende alguns conhecimentos disponíveis a partir da otimização e, em seguida, aplica o conhecimento existente para orientar a pesquisa heurística atual.
- Modelo de Busca Heurística - se encarrega de pesquisar o vasto espaço da solução e identificar uma solução ótima.

Conforme os pesquisadores o algoritmo KBACO é uma instanciação do KBHSA e pode ser denotado no pseudocódigo 3.2:

Pseudocódigo 3.3 - Algoritmo KBACO

```
01: Inicio
02:     O conhecimento genotípico é inicializado
03:     Repita
04:         Construção de soluções usando o algoritmo ACO guiado pelo conhecimento existente
05:         O conhecimento genotípico é atualizado pela otimização da iteração atual
06:     Até que o critério de parada seja satisfeito
07: Fim
```

Neste trabalho, os autores consideraram o *makespan* como critério de otimização para resolver o FJSP. Eles afirmaram que o ciclo de processamento do algoritmo é conduzido por três regras: (1) atribuição de uma operação a uma máquina apropriada; (2) sequenciamento das operações em cada máquina; (3) um *job* pode visitar uma máquina mais de uma vez. Tais regras impõem o algoritmo as seguintes restrições:

- A configuração dos tempos das máquinas e os tempos de mudança entre as operações são insignificantes;
- As máquinas são independentes uma da outra;
- Todos os *jobs* são independentes um do outro;

- Em um determinado momento, uma máquina pode executar no máximo uma operação. O recurso fica disponível para outras operações somente se a operação que está sendo processada for concluída;
- Não há restrições de precedência entre as operações de *jobs* diferentes;
- Não é possível executar mais de uma operação do mesmo *job* de cada vez.

A estrutura do algoritmo KBACO pode ser visualizada na figura 3.1.

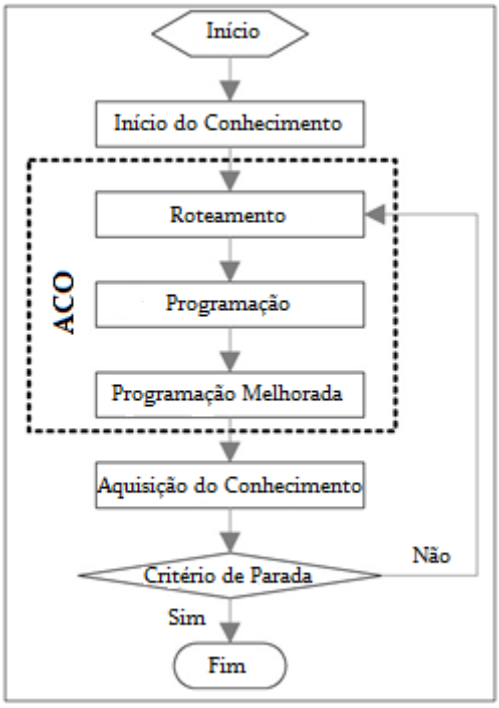

Figura 3.1 - Funcionamento do KBACO.
Fonte: Adaptado de Xing et al. (2010)

Após cada iteração, os números predefinidos da melhor solução são selecionados para inserir no conjunto de soluções de elite. Os autores chamaram este procedimento de conhecimento de solução. No conjunto, a quantidade total de soluções é fixa e a pior solução é substituída por uma boa após cada inserção. Na fase de inicialização, o conhecimento da solução elite é nulo. Na fase de aprendizagem do conhecimento, o conhecimento da solução de elite é atualizado pelos números predefinidos da melhor solução de cada iteração.

Na fase de atribuição das operações, cada formiga realiza uma tarefa viável para atribuição da operação com base no conhecimento da máquina de atribuição. Esta informação corresponde ao conhecimento acumulado de atribuir a operação dada a uma máquina mais apropriada. Na fase de sequenciamento das operações, cada formiga constrói um cronograma viável com base no conhecimento da prioridade de atribuição de operação. Esta informação

corresponde ao conhecimento acumulado da prioridade de processamento mais apropriada para a operação dada. Todo conhecimento é extraído da melhor solução global do FJSSP.

Os parâmetros no KBACO foram ajustados dinamicamente de acordo com o desempenho de otimização. Assim, muitas combinações de parâmetros foram geradas na fase de inicialização. Nesta abordagem, se a melhor solução global for melhorada após uma iteração, essa é chamada de iteração bem-sucedida. Logo, o desempenho de otimização pode ser entendido como os tempos da iteração bem-sucedida.

Para avaliar o KBACO proposto, os pesquisadores utilizaram dois conjuntos de instâncias: T-FJSP e P-FJSP. Os mecanismos gerados para essas instâncias FJSP podem ser listados da seguinte forma: (1) o número de *jobs* varia entre 50 e 200; (2) o número de máquinas varia de 10 a 20; (3) o número de operações para cada *job* varia de 10 a 30; (4) o tempo de processamento para uma operação varia de 10 a 100 unidades. As instâncias de Brandimarte (1993) e Kacem et al. (2002a, 2002b) foram utilizadas na maioria dos testes. Eles observaram que o modelo de conhecimento gera uma memória capaz de manter bons recursos da iteração anterior. Assim, este modelo foi aplicado para orientar a pesquisa heurística atual do ACO conduzindo o algoritmo a convergir para as melhores soluções.

Conforme os pesquisadores a abordagem foi avaliada por meio de testes comparativos entre a melhor solução encontrada com as soluções apresentadas por outras abordagens publicadas na literatura. Desta forma, os resultados experimentais mostraram que o algoritmo KBACO proposto supera alguns métodos na qualidade do *makespan* para o FJSP em quase todas as instâncias.

No trabalho de Li, Keqi e Chunnan (2010) foi apresentado um algoritmo ACO melhorado em combinação com um algoritmo PSO para resolver o FJSP multiobjetivo. O algoritmo PSO tem a vantagem de convergência rápida, mas o ACO melhorado tem capacidade de feedback positivo. Esta pesquisa combina os dois algoritmos para fortalecer a capacidade de pesquisa na busca de uma solução ideal com convergência rápida. Esta abordagem inclui duas partes: (1) uso da convergência rápida do PSO para procurar as melhores posições das partículas que são usadas para definir a posição inicial das formigas; (2) uso do feedback positivo e conjunto de soluções encontradas pelo ACO na pesquisa global.

Os pesquisadores afirmaram que o PSO tem escalabilidade, permite fácil integração com outras abordagens e apresenta rápida convergência, mas não é tão bom na busca local. O ACO pode encontrar melhores soluções com feedback positivo e tem uma

poderosa capacidade de transação distribuída. Neste trabalho, o ACO melhorado se difere do ACO tradicional pelos seguintes mecanismos:

- O número de subconjuntos é definido pelo número de *jobs*;
- Faz uso de um método de pesquisa local para melhorar a solução;
- A regra de evaporação da trilha de feromônio é aplicada com intensidades para evitar estado de estagnação precoce.
- Faz uso de parâmetros razoáveis para o equilíbrio da capacidade de busca global e convergência, considerando a escala do real problema.

A abordagem considera três objetivos a serem minimizados: (1) *makespan*; (2) *workload*; (3) *total workload*. O algoritmo foi avaliado com uso das instâncias de Kacem et al. (2002a) e os resultados obtidos foram comparados com outras abordagens encontradas na literatura. Assim, os pesquisadores contextualizaram que a abordagem é viável e eficaz para resolver o FJSP multiobjetivo.

Flórez, Gómez e Bautista (2013) propuseram um algoritmo ACO conhecido como *Elitist Ant System* para resolver o problema de programação *Job-Shop*. Esta abordagem consiste em encontrar soluções para minimizar o *makespan*. As soluções viáveis devem cumprir as restrições que se aplicam ao problema respeitando a precedência entre as operações que determinam a sequência de processamento sem interromper as operações até a conclusão. Cada tarefa é uma sequência de operações com tempo de processamento determinado em cada máquina.

Nesta abordagem, o JSP é representado por um grafo disjuntivo $G\ (V, C \cup D)$, onde V é o conjunto de nós representando as operações (*jobs* x máquinas) com a exceção do nó inicial (I) e dos nós finais (F) do grafo; C é um conjunto de direções ($\rightarrow$) que corresponde as operações correspondentes ao mesmo *job* (sequência de processamento); D é o conjunto de grafos não direcionados ($\leftrightarrow$) que conectam operações em execução em uma mesma máquina. Além disso, o tempo de processamento de cada operação é colocado na parte superior do nó.

Conforme os autores, o algoritmo ACO corresponde às arestas do grafo com os caminhos pelos quais as formigas produzem as soluções do problema. Neste sentido, cada formiga possui apenas informações locais que compartilham através do feromônio. A atualização da trilha de feromônio depositada nas arestas pode ser realizada de modo global ou local. Desta maneira, as formigas constroem caminhos que representam soluções viáveis, guiadas pelas trilhas de feromônio e a informação heurística de cada uma das arestas. Por esta

razão, a população de formigas realiza uma pesquisa estocástica, selecionando o próximo nó para visitar apenas com base em informações disponíveis localmente, usadas em uma abordagem probabilística, onde inicialmente as decisões das formigas são completamente aleatórias na ausência de trilhas de feromônio.

O algoritmo EAS mostrou eficiência na construção das soluções, uma vez que a trilha de feromônio do melhor caminho encontrado em cada iteração foi reforçada. Nas arestas da melhor solução gerada por uma formiga, mais feromônio é depositado através de todas as outras formigas, conduzindo o algoritmo a convergir boas soluções. Na figura 3.2 é ilustrada a representação do JSP em uma instância 3x3 por meio de um grafo disjuntivo.

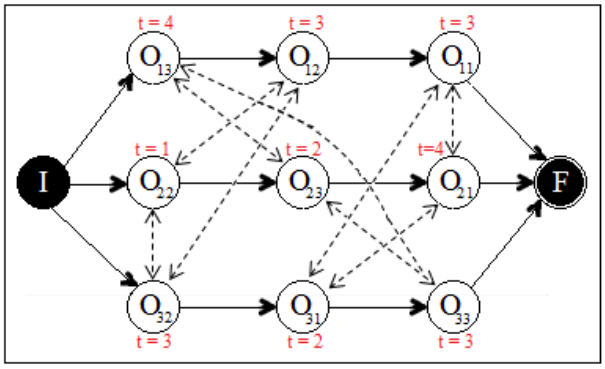

Figura 3.2 - Grafo de uma instância de 3×3 do JSP.
Fonte: Adaptado de Flórez, Gómez e Bautista (2013)

Segundo os pesquisadores, a convergência rápida deste algoritmo pode reduzir a capacidade de varredura, uma vez que as formigas logo terminam de uma única maneira, o que pode ser um ótimo local. Para compensar isso, é permitido incluir no conjunto de operações realizáveis que fazem as máquinas aguardarem em pausa algumas unidades de tempo para iniciar a execução quando o *job* correspondente ainda está ativo em outra máquina. Esta operação que atrasa ou retarda o início das máquinas só será selecionada se a aresta que atinge o nó possuir feromônio suficiente para tornar a probabilidade maior do que as operações que têm postos de trabalho disponíveis. Este método explora ainda mais o espaço de busca para encontrar soluções nas quais podem ser obtidos resultados que excedem o ótimo local encontrado nas primeiras iterações.

O algoritmo foi avaliado em 40 instâncias de diferentes tamanhos que variam de 10x5 com 50 operações até 30x10 com 300 operações que representam um total de combinações possíveis = $(30!)^{10}$. Os testes foram realizados em 30 execuções do algoritmo (1000 iterações) para cada uma das instâncias JSP. Os autores tomaram como referência a solução mais conhecida (do inglês *Best Known Solution* - BKS) que observa alguns fatores,

tais como: (1) percentual de erro relativo; (2) média de *makespan*; (3) desvio padrão; (4) número médio de avaliações da função objetivo.

Embora a eficácia do algoritmo para encontrar a melhor solução não seja alta, alcançando o BKS em 27,5% das instâncias, o erro relativo médio nas 40 instâncias é de apenas 4%, o que é uma boa aproximação ao ótimo de JSP. Em geral, 65% das instâncias executadas se aproximam de menos de 5% da BKS e 47,5% se desviam em menos de 3% da BKS.

Na figura 3.3 é mostrada a média de erro relativo pelo tamanho da instância durante o ciclo de testes através do gráfico.

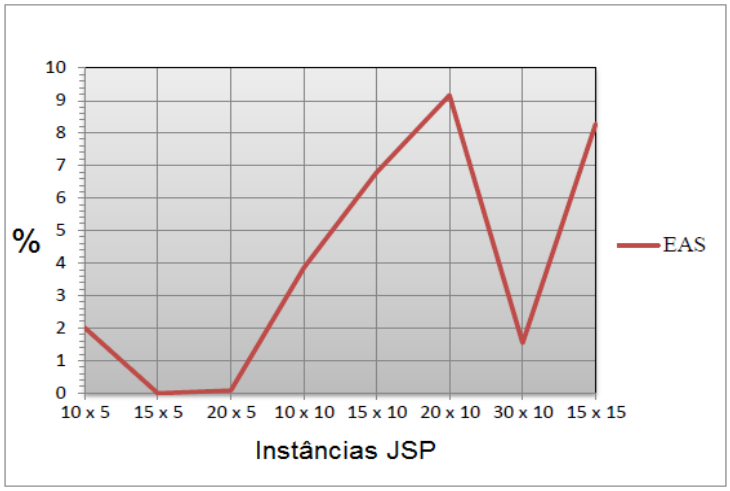

Figura 3.3 - Média de erro relativo pelo tamanho da instância JSP.
Fonte: Adaptado de Flórez, Gómez e Bautista (2013)

Os autores demostraram por meio dos resultados que o algoritmo EAS apresentou soluções de boa qualidade para o JSP em um pequeno número de avaliações da função objetivo, embora exija melhorias para obter a solução ótima em todas as instâncias. Eles contextualizam que o ACO é uma metaheurística com potencial para obter soluções eficientes de problemas de programação *Job-shop*, com custo computacional relativamente pequeno.

Saidi-Mehrabad et al. (2015) propuseram uma pesquisa para tratar do problema de programação *Job-shop* considerando os tempos de transporte dos *jobs* de uma máquina para outra. Eles utilizaram o JSP básico e o problema de roteamento livre de conflitos (do inglês *Conflict Free Routing Problem* - CFRP) para veículos guiados automatizados (do inglês *Automated Guided Vehicles* - AGV). Foi proposto um modelo matemático composto de JSP e CFRP simultaneamente, e um algoritmo ACO com ciclo de processamento em dois estágios. Tais estágios são: (1) sequência da entrega de *jobs* aos AGVs; (2) seleção das rotas pelos AGVs e a finalização das rotas de processamento dos *jobs* nas máquinas.

Nesta pesquisa, o algoritmo ACO busca soluções para resolver o modelo integrado de JSP e CFRP de forma que se possa minimizar o *makespan* considerando os seguintes fatores: (1) a rota de processamento para cada *job* difere com a rota de processamento de outros *jobs*; (2) o tempo de processamento para cada *job* em cada máquina é fornecido; (3) os *jobs* podem ser transferidos de uma máquina para outra máquina por vários AGVs.

Segundo os autores, o problema foi abordado da seguinte forma: todos os AGVs estão no ponto de partida do ambiente de produção no início do período de planejamento; na próxima unidade de tempo, apenas um *job* é atribuído a cada AGV; posteriormente, os AGVs transferem os *jobs* do ponto de partida para o primeiro ponto da máquina necessária através do caminho guia; os AGVs aguardam o termino do processamento do *job* na primeira máquina, a fim de receber e transferir esse *job* para máquina subsequente se necessário. Caso contrário eles se deslocam para o ponto final do ambiente de produção. O algoritmo determina que um AGV não deva assumir outro *job* enquanto estiver encarregado de entregar um determinado *job*; o método CFRP é aplicado para evitar colisão. Desta forma, os autores consideram os seguintes pressupostos:

- Todos os AGVs têm capacidade de unidade de *job*.
- AGVs e máquinas operam continuamente sem quebras.
- Os tempos de carga e descarga do AGV são fixos, e o tempo de viagem também é considerado.
- As máquinas não são idênticas.
- Os pontos iniciais e finais estão no ambiente de produção.
- Nenhuma máquina pode processar mais de um *job* por vez.
- O processamento das operações não pode ser interrompido.
- Todos os *jobs* e todas as máquinas estão disponíveis a partir do tempo zero.

Esta abordagem foi avaliada usando 13 problemas de testes gerados aleatoriamente em um estudo comparativo com métodos exatos processados pelo software GAMS. Para cada problema de teste foi observado os seguintes critérios: *makespan*; desvio padrão do *makespan*; média do *makespan*; tempo médio de CPU; média de gap% (diferença média%); e desvio padrão de gap%.

Conforme os autores, os resultados dos testes mostraram que em relação aos valores do desvio padrão médio e diferença%, o ACO apresentou uma boa confiabilidade.

Para problemas de grande escala, nenhuma resposta é obtida dentro de uma execução de tempo de 172800 s (48 horas) usando o software GAMS. Contudo, o ACO conseguiu criar uma solução justificável dentro de 90 s. Assim, eles afirmaram que o ACO apresentou confiabilidade e é capaz de encontrar uma boa solução para grandes instâncias, onde soluções ótimas não estão disponíveis. Desta forma, eles contextualizaram que o algoritmo proposto conseguiu resolver o problema em tempo razoável de forma eficiente.

No estudo de Xuesong e Qiaoyun (2015) foi proposto um algoritmo ACO modificado para resolver o problema de FJSP multiobjetivo. A pesquisa foi realizada em um ambiente de produção real considerando três objetivos: (1) minimizar o custo total de processamento; (2) minimizar o tempo de processamento; (3) otimizar a maior taxa de produtos qualificados. O ACO modificado consiste em salvar toda a desagregação de Pareto após iteração utilizando o mecanismo BP(t). Neste procedimento, todas as soluções não dominadas são preservadas para guiar a área de otimização e acelerar a taxa de convergência.

Segundo os pesquisadores, a indicação da localização não dominada pelas formigas conduz o algoritmo a reduzir o feromônio nestes caminhos, e guia outras formigas na experiência global para melhorar a taxa de convergência pela descoberta atual de todas as soluções não dominadas que foram salvas. O procedimento melhorou a velocidade de convergência e a atualização do feromônio com base na experiência global ideal.

Os testes foram realizados com simulação da otimização do FJSP para obter fronteira de Pareto, combinado com o processamento de equipamentos de elevação na oficina de revestimento. Desta forma, os autores utilizaram o método de coeficiente ponderado sob a condição de mesmo parâmetro para comparação dos resultados com o algoritmo ACO modificado nos três objetivos. Os pesquisadores contextualizaram por meio dos resultados da simulação que o algoritmo ACO pode efetivamente resolver o FJSP multiobjetivo por ter mostrado eficiência nos três objetivos com superioridade ao método de coeficiente ponderado.

Huang e Yu (2017) apresentaram um algoritmo de otimização por colônia de formigas com multiferomônios (do inglês *Multi-pheromoneant Ant Colony Optimization* - MACO) para resolver o problema de programação *job-shop* multiobjetivo com divisão de lotes (do inglês *Multi-objective Lot-splitting Job-Shop Scheduling Problem* - MLJSP). Os objetivos foram medidos por uma função escalar que é dada pela soma ponderada de *makespan*, atraso total e custo total de divisão de lotes. O MACO proposto compõe um grupo de algoritmos que obtém cinco métodos: (1) um novo tipo de feromônio e função heurística gananciosa; (2) três novas funções das regras de transição do estado; (3) um algoritmo de pesquisa local ágil para as melhorias da qualidade da solução; (4) mecanismo de mutação para

pesquisa divisória; (5) algoritmo PSO para ajuste adaptativo de parâmetros do ACO. Tais métodos foram implementadas para melhorar os padrões de busca e o tempo de convergência do algoritmo.

Segundo os pesquisadores, o MLJSP torna o problema em si mais relacionado aos problemas do mundo real e aumenta o grau de complexidade pelo fato de incluir divisão de lotes e medidas com multiobjetivo no JSP. Neste sentido, eles analisaram o MACO em dados simulados e extensivamente em dados do mundo real obtidos em uma empresa de impressão. Nesta pesquisa, o processo de fabricação da empresa de impressão envolve um JSP composto por 12 máquinas agrupadas em 07 *jobs*, e cada *job* tem uma ordem predeterminada de processamento nas máquinas em todas as operações. A divisão de lotes representa o processo de fabricação diária, e cada produto é tratado como um *job*.

No ambiente de produção, a etapa de divisão de lotes assume a forma de decomposição de uma impressora em sublotes iguais no objetivo de diminuir o tempo ocioso das máquinas. Nesta abordagem, os autores observaram as informações específicas do problema e aplicaram o MACO para gerar sequências viáveis de processamento dos *jobs* atribuídas as máquinas e aos trabalhadores. Nesta tarefa, o cálculo do *makespan*, o atraso total e a divisão de lotes são registrados e analisados posteriormente.

A proposta do algoritmo MACO se inspirou no comportamento de formigas com vários tipos de feromônio. Este modelo fornece mais informações e orientações de otimização para as formigas artificiais pelo fato de possuir matriz de multiferomônios. Neste algoritmo, apenas as melhores soluções locais podem atualizar a nova matriz de feromônio. Os depósitos de feromônio de cada matriz são restringidos com limites superiores e inferiores para manter a possibilidade do MACO pesquisar todo o espaço da solução e evitar a rápida convergência para o ótimo local. Se a atualização de feromônio for adicionada ou evaporada para fora dos limites inferior ou superior, ela é ajustada automaticamente ao seu valor de limite.

Nesta pesquisa, vários algoritmos são propostos. Tais algoritmos são:

- ACO - algoritmo clássico com matriz de feromônio único;
- MACO - algoritmo principal desta pesquisa;
- MACO-A - definido como MACO Ajustável. É um operador de ajuste e afinação de parâmetros no MACO por meio de um algoritmo PSO. O PSO é usado neste método devido a sua característica de convergência mais rápida em comparação ao MACO;

- MACO-AL - definido como MACO Local Adaptativo. Aplica ajuste de parâmetros adaptativos e busca local no algoritmo MACO;
- MACO-ALM - definido como MACO Local Adaptativo Mutante. Aplica ajuste de parâmetros adaptativos, busca local e formigas mutadas no algoritmo MACO. O mecanismo das formigas mutadas permite a escolha de uma formiga arbitrária entre as formigas, e altera o padrão de exploração dessa formiga. O mutante não determina a próxima base de rota na trilha de feromônio, mas conta com um mecanismo de busca tabu para melhorar a busca de padrões e trilhas de feromônio.

Todos esses algoritmos foram testados, e durante os testes foram adicionados ao algoritmo MACO um de cada vez, a fim de verificar os efeitos na resolução MLJSP. Este estudo analisa a eficiência dos algoritmos propostos e utiliza um modelo de programação inteira (do inglês *Integer Programming* - IP) para estabelecer um índice de referência absoluto na comparação dos resultados durante a avaliação. A função objetivo é definida pela soma ponderada de *makespan*, atraso total das máquinas e custo da divisão de lotes. Assim, os coeficientes de peso foram ajustados de acordo com as circunstâncias e as necessidades de fabricação.

Os testes iniciaram com instâncias de pequena escala, onde as performances de IP, ACO, MACO, MACO-A, MACO-AL, MACO-ALM foram examinadas e comparadas. Após a confirmação da eficácia dos algoritmos foi iniciado os testes com instâncias de grande escala. Os dados de simulados para os testes foram gerados em cem instâncias classificadas em dois grupos: (1) instâncias de pequena escala; (2) instância de grande escala. Todos os algoritmos foram avaliados em 30 iterações nas instâncias de pequena escala, e somente os melhores foram testados nas instâncias de grande escala. Nos testes de grande escala, os algoritmos foram comparados com outras abordagens encontradas na literatura, e o critério de parada é um número específico de iterações após o algoritmo não gerar qualquer melhoria nos valores objetivos. No total de cem instâncias, somente trinta são classificadas como pequena escala.

O conjunto de dados reais obtidos da empresa de impressão foi utilizado para avaliar o algoritmo principal. Os dados de entrada são: número de identificação do *job*; a ordem das máquinas em que o *job* requer processamento; o tempo de processamento estimado em cada uma das máquinas; o custo de divisão de lotes estimados; data de lançamento; e data de vencimento. A divisão do lote foi executada apenas nas máquinas de impressão. Antes de

avaliar o MACO, o algoritmo MACO-ALM foi aplicado para reprogramar sequências de produção históricas e ajustar os pesos ponderados de *makespan*, atraso e custo de divisão de lotes.

Os resultados obtidos durante os testes no conjunto de dados do mundo real por meio de uma empresa de impressão comprovam a eficácia da aplicação dos algoritmos propostos na resolução do MLJSP. Os padrões gerados pelo algoritmo são mais eficientes em comparação aos esquemas baseados na experiência dos trabalhadores. A empresa reconheceu o sucesso desta pesquisa pela qualidade dos resultados e expressou intenções de colaborar com novos estudos.

No trabalho de Wu, Wu e Wang (2017) foi apresentado um algoritmo de otimização por colônia de formigas híbrido com referência no modelo de gráficos disjuntivos 3D que combina os algoritmos *Elitist Ant System*, *Max-Min*, *Ant System* e mecanismo de controle dos parâmetros de estágio do ACO para resolver o FJSP considerando os seguintes objetivos: (1) tempo de conclusão mais longo; (1) atraso das máquinas; (3) tempo ocioso médio da máquina; (4) custo de produção.

Na figura 3.4 é ilustrado o modelo proposto detalhadamente.

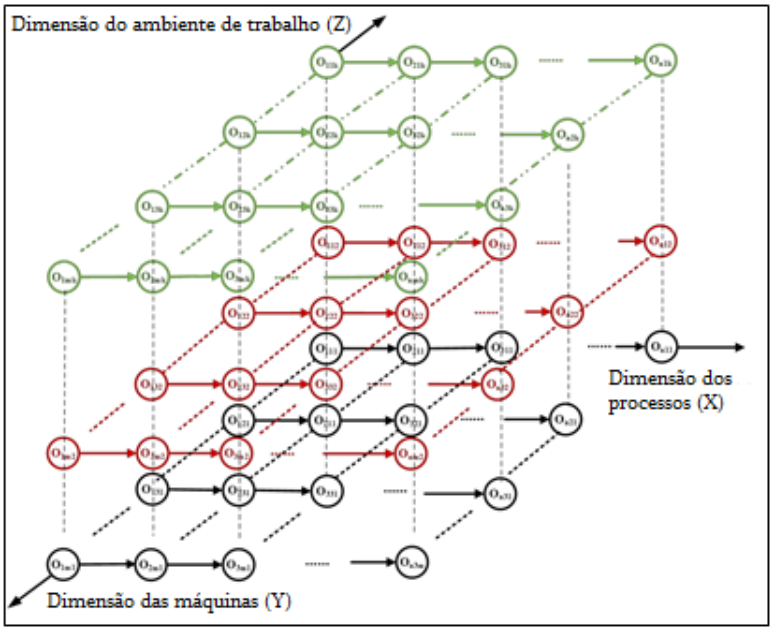

Figura 3.4 - Modelo baseado em gráfico 3D disjuntivo para FJSP.
Fonte: Adaptado de Wu, Wu e Wang (2017)

Os autores contextualizaram que o algoritmo ACO é uma boa solução para problemas de otimização, mas pode convergir para a solução ótima local durante a solução de problemas de larga escala. Nesta abordagem, foi proposto um algoritmo ACO misto para superar essa falha na esperança de obter uma solução de melhor desempenho para o problema

FJSP. Conforme os pesquisadores, a modelagem de grafos disjuntivos é essencial na solução de problemas de otimização pelo ACO. Tendo em vista a falta de informação e o fraco efeito dos modelos referenciados em grafos disjuntivos 2D ao resolver o FJSP, eles adotaram um método de modelagem baseado no gráfico disjuntivo 3D. O funcionamento básico do algoritmo proposto pode ser resumido no pseudocódigo 3.4:

Pseudocódigo 3.4 - ACO Híbrido com Referência no Modelo de Gráficos Disjuntivos 3D

```
01: Inicio
02:     Inicialização dos parâmetros
03:     Definição do número de ciclos
04:     Repita
05:         Pesquisa aleatória do ciclo 1 para ciclo N - ρ1 N
06:         Pesquisa da roleta do ciclo N - δ1 N para ciclo δ1 N - δ2 N
07:         Pesquisa da roleta do ciclo N - δ1 N para ciclo δ1 N - δ2 N
08:     Até que o critério de parada seja satisfeito
09: Fim
```

Durante os testes, o algoritmo foi executado em 100 ciclos por 10 vezes para obter a solução ótima na instância 8X8 P-FJSP de Kacem et al. (2002a). A pesquisa explica como resolver o FJSP a partir dos aspectos de descrição do problema, modelagem e projeto de algoritmo. Neste trabalho, foi analisada a efetividade da abordagem proposta por meio dos resultados que confirmou a eficiência e eficácia do algoritmo na resolução do problema.

Wang, L. et al. (2017) propuseram um algoritmo de otimização de colônia de formigas melhorado (do inglês *Improved Ant Colony Optimization* - IACO) para resolver o FJSP em vários grupos de benchmarks. Segundo os autores o ACO básico provou ser uma abordagem eficiente para lidar com o FJSP, mas possui duas desvantagens: tempo computacional e otimização local. O ACO melhorado foi proposto para superar estas desvantagens.

O IACO apresenta alguns mecanismos que se difere do ACO convencional. Este estudo mostrou que os mecanismos proporcionam melhorias na convergência do ACO quanto ao tempo computacional e otimização local. Tais mecanismos são:

- Regras de seleção da máquina - a máquina que possui o menor tempo de processamento para finalizar determinada operação de determinado *job* é selecionada usando 60% de probabilidade; a máquina que tem o menor tempo

para processar determinada operação de determinado *job* é selecionada usando 30% de probabilidade; uma máquina é selecionada aleatoriamente usando 10% de probabilidade, e a seleção aleatória pode ser alcançada usando o método de seleção de roleta;

- Inicialização do mecanismo distribuído uniforme para formigas - as formigas são distribuídas uniformemente no conjunto que contém as primeiras operações de todos os *jobs*, assim a probabilidade de encontrar a solução global se torna maior;
- Alteração do mecanismo guia de feromônio - para expandir o escopo de pesquisa das formigas e melhorar o espaço de busca do ACO básico, a inicialização do feromônio é realizada quando o valor do feromônio em um caminho é mais de 90% do valor total do feromônio em todos os caminhos;
- Método de escolha do nó e atualização do mecanismo de feromônio - quando o algoritmo estiver preso no ótimo local, o espaço de busca pela solução pode ser mais explorado com ajuste do feromônio. Desta forma, uma boa relação de equilíbrio entre "usar a informação anterior para acelerar a convergência" e "explorar novos caminhos" podem ser estabelecidos. Assim, o espaço de busca pode ser ampliado e a solução global pode ser melhorada.

A figura 3.5 mostra o gráfico de Gantt correspondente a seleção de máquinas, onde a máquina 1 é selecionada com 60% de probabilidade, maquina 2 com 30% de probabilidade e máquina 3 aleatoriamente com 10% de probabilidade.

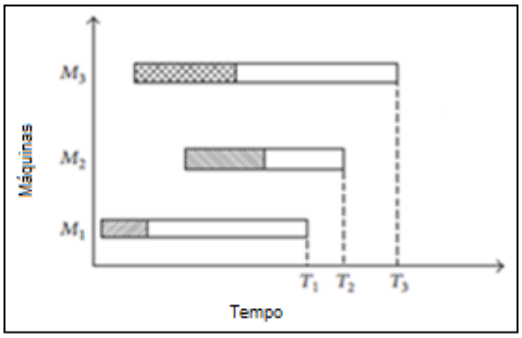

Figura 3.5 - Seleção de máquinas pelo IACO.
Fonte: Adaptado de Wang, L. et al. (2017)

Nesta abordagem, o IACO foi testado e avaliado em três grupos de simulação: (1) uma instância de produção real com um problema 8x10 P-FJSP; (2) conjunto de instâncias retiradas de Kacem et al. (2002a, 2002b); (3) conjunto de instâncias Brdata retiradas de Brandimarte (1993).

Os pesquisadores iniciaram os testes na instância de produção real por meio de uma comparação do algoritmo proposto com outros dois algoritmos de otimização por colônia de formigas. Os resultados experimentais do IACO proposto em comparação com o ACO básico e MMAS por 10 experimentos independentes avaliaram alguns critérios: a melhor solução; a solução média; a pior solução; e o tempo médio de computação. Os resultados mostraram que o IACO obteve soluções de melhor qualidade em comparação com os demais métodos comparados.

Os testes realizados com o IACO no conjunto de instâncias de Kacem et al. (2002a) mostraram bons resultados em todos os problemas. O algoritmo obteve o melhor *makespan* em poucas iterações, e foi comparado com alguns métodos propostos na literatura, incluindo outras metaheurísticas, tais como: AL + CGA proposto por Kacem et al. (2002a); PSO + TS proposto por Zhang et al. (2009); e KBACO proposto por Xing et al. (2010). Para cada instância, avaliou-se a melhor solução e o tempo computacional do IACO de forma independente em comparação com os outros algoritmos em 10 execuções.

Durante os testes realizados nas instâncias Brdata de Brandimarte (1993), foi demonstrado que os valores do *makespan* obtidos pelo IACO foram menores ou iguais em comparação aos outros algoritmos em quase todos os cenários. Segundo os autores, o algoritmo IACO é competitivo para outros algoritmos por ter demostrado eficácia e eficiência na solução do FJSP.

Capítulo 4

PROPOSTA DE TRABALHO

4.1 Considerações Iniciais

É possível observar no capítulo anterior que houve um grande crescimento do número de pesquisas com aplicações de métodos com IA na resolução do FJSP. Neste sentido, a comunidade acadêmica busca melhorar o desempenho computacional para propor soluções cada vez melhor. Conforme visto no trabalho de Huang et al. (2016), diversas técnicas de metaheurísticas podem ser aplicadas na resolução deste problema, e os resultados podem variar conforme o tamanho da instância, os objetivos e as restrições.

A literatura aponta pesquisas que tratam do problema de programação da produção sobre vários aspectos. Muitos trabalhos científicos demonstram abordagens que resolvem o problema, mas nem sempre obtém soluções ideais. Neste trabalho é apresentado um algoritmo ACO em conjunto com a heurística SPT para resolver o FJSP multiobjetivo, visando propor soluções de qualidade perante a comunidade acadêmica.

O algoritmo proposto considera alguns critérios de desempenho a serem minimizados: *makespan*; *machine workload*; e *total workload*. Estes critérios caracterizam o multiobjetivo e são avaliados mediante a comparação com outras abordagens encontradas na literatura.

4.2 Formulação do Problema

Conforme Zhang et al. (2009) o FJSP pode ser formulado da seguinte maneira: existe um conjunto de n *jobs* e um conjunto de m máquinas. O conjunto de todas as máquinas é indicado por M. Cada *job* i consiste em uma sequência n_i de operações. Todas as operações $O_{i,j}$ $(i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, n_i)$ de cada *job* deve ser processada em uma máquina M_K de um conjunto de máquinas compatíveis $M_{i,j}$ sendo que $M_K \in M_{i,j};\ M_{i,j} \subseteq M$. A execução de cada operação requer uma máquina selecionada de um conjunto de máquinas disponíveis. Além disso, o FJSP precisa definir o tempo de início e término de cada operação. O FJSP deve determinar tanto uma atribuição quanto uma sequência das operações nas máquinas para satisfazer os critérios estabelecidos.

Kacem et al. (2002a) descreveram as duas categorias do FJSP quanto à formulação matemática do problema: se houver $M_{i,j} \subset M$ para no mínimo uma operação, denota-se P-FJSP; enquanto há $M_{i,j} = M$ para todas as operações, denota-se T-FJSP.

Durante o processo de solução deste problema, os seguintes pressupostos são estabelecidos:

- Nenhuma operação pode ser interrompida durante o processamento;
- Limitação de recursos - cada máquina só pode executar uma operação ao mesmo tempo;
- Restrições de sequência (precedência) - duas operações do mesmo *job* não podem ser processadas simultaneamente;
- As restrições de sequência das operações de um trabalho podem ser definidas para qualquer par de operações;
- O tempo de configuração das máquinas e o tempo de mudança entre as operações são insignificantes;
- As máquinas são independentes uma da outra;
- Os trabalhos são independentes um do outro;
- O tempo de processamento de cada operação é fixo;
- O tempo de transporte dos trabalhos de uma máquina para outra não é considerado.

A notação utilizada neste trabalho é apresentada da seguinte maneira:

n: número total de *job;*

m: número total de máquinas;

n_i: número total de operações do *job i*;

$O_{i,j}$: a operação j_{th} do *job i*;

$M_{i,j}$: o conjunto de máquinas disponíveis para a operação $O_{i,j}$;

i: índice do *job*, *i* = 1, 2, ..., n;

k: índice da máquina, *k* = 1, 2, ..., m;

j: índice da sequência de operação, *j* = 1, 2, ..., n_i;

C_k: tempo de conclusão de M_{k};

W_k: carga de trabalho de M_{k}.

W_t: carga total de trabalho de *M*.

Os critérios a serem minimizados neste trabalho são caracterizados como funções objetivo e podem ser formulados da seguinte maneira:

- C_k: *Makespan* ou tempo máximo de conclusão das máquinas, dado por: $min\, f1 = max(C_k)\,, 1 \leq k \leq m$;
- W_k: *Workload* ou carga de trabalho da máquina crítica, que é a maior carga de trabalho entre as máquinas, dado por: $min\, f2 = max(W_k)\,, 1 \leq k \leq m$;
- W_t: *Total Workload* ou carga de trabalho total das máquinas, que é o tempo total de trabalho de todas as máquinas, dado por: $min\, f3 = \sum_{k=1}^{m} W_k$.

O multiobjetivo é caracterizado por mais de uma função objetivo e a solução do problema deve transformar todas as funções objetivas em uma única função combinada, caracterizada por função de aptidão. Este trabalho utiliza o método de soma ponderada, definido pela seguinte formulação matemática: $min\, f = w1(f1) + w2(f2) + w3(f3)$, onde $w1, w2\ e\ w3$ são pesos definidos conforme cada função objetivo. Os valores dos pesos são variados e estão explícitos no capítulo 05.

4.3 Metodologia

O algoritmo proposto neste trabalho foi construído em linguagem MATLAB com uso da ferramenta nas versões R2012b e R2017b rodando em um microcomputador pessoal Intel (R) Core (TM) i5 CPU 2.67 GHz com 04 GB de RAM e sistema operacional Windows 7 de 64 bits.

O MATLAB é uma linguagem de programação apropriada para engenharia e computação científica pelo fato de possuir um conjunto de funções pré-definidas para cálculos matemáticos. A linguagem permite a programação procedural por meio de arquivos de script, possui capacidade gráfica e possibilidade de operar com instruções simbólicas. A declaração de matrizes não requer dimensionamento, por isso permite programar e resolver problemas matemáticos com mais praticidade que muitas linguagens de programação. Possui um conjunto de softwares integrados compostos por um grupo de funções que são definidos como toolboxes. Os toolboxes são usados para resolver determinados problemas com uso de técnicas computacionais pré-definidas, tais como: redes neurais, processamento de sinais, simulação de sistemas dinâmicos, lógica Fuzzy, entre outros (HAHN e VALENTINE 2010).

Segundo MathWorks (2018), a linguagem oferece também os recursos de programação orientada a objetos, que permitem desenvolver programas de computador com mais rapidez em comparação com outras linguagens, como C ++, C # e Java. É possível definir classes e aplicar padrões de design orientados a objeto padrão no MATLAB que permitem reutilização de código, herança, encapsulamento e comportamento de referência sem se envolver nas tarefas de manutenção de baixo nível exigidas por outras linguagens. O MATLAB® é uma marca comercial da The MathWorks usada com permissão.

Os testes do algoritmo proposto foram realizados em duas etapas:

- Ajustes de parâmetros do ACO - número de formigas; influência da trilha de feromônio, dado por α; informação heurística, dado por β; taxa de evaporação do feromônio, dado por ρ; intervalo dos possíveis valores de feromônio, dado por $\geq t_{mim}$ $e \leq t_{max}$; e número de iterações.
- Avaliação da abordagem - análise de resultados realizada por meio de um estudo comparativo com outras abordagens em cinco instâncias representativas de FJSP retiradas de Brandimarte (1993) e Kacem et al. (2002a).

Neste trabalho utilizou-se quatro instâncias extraídas de Kacem et al. (2002a) e uma instância Brdata extraída de Brandimarte (1993). Tais instâncias são apresentadas abaixo:

Instância Kacem 4x5: possui 4 jobs com 12 operações que planejam ser processadas em 5 máquinas com flexibilidade total. A tabela 4.1 representa a instância 4x5.

Tabela 4.1 - Instância Kacem 4x5.
Fonte: Adaptado de Kacem et al. (2002a)

Jobs		M_1	M_2	M_3	M_4	M_5
J1	$O_{1,1}$	2	5	4	1	2
	$O_{1,2}$	5	4	5	7	5
	$O_{1,3}$	4	5	5	4	5
J2	$O_{2,1}$	2	5	4	7	8
	$O_{2,2}$	5	6	9	8	5
	$O_{2,3}$	4	5	4	54	5
J3	$O_{3,1}$	9	8	6	7	9
	$O_{3,2}$	6	1	2	5	4
	$O_{3,3}$	2	5	4	2	4
	$O_{4,3}$	4	5	2	1	5
J4	$O_{4,1}$	1	5	2	4	12
	$O_{4,2}$	5	1	2	1	2

Instância Kacem 8x8: possui 08 jobs com 27 operações que planejam ser processadas em 8 máquinas com flexibilidade parcial. A tabela 4.2 representa a instância 8x8. Por se tratar de P-FJSP, a tabela mostra o símbolo "∞" onde a atribuição é impossível. Neste caso, algumas tarefas só podem ser processadas em uma parte do conjunto de máquinas disponível.

Tabela 4.2 - Instância Kacem 8x8.
Fonte: Adaptado de Kacem et al. (2002a)

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J1	$O_{1,1}$	5	3	5	3	3	∞	10	9
	$O_{1,2}$	10	∞	5	8	3	9	9	6
	$O_{1,3}$	∞	10	∞	5	6	2	4	5
J2	$O_{2,1}$	5	7	3	9	8	∞	9	∞
	$O_{2,2}$	∞	8	5	2	6	7	10	9
	$O_{2,3}$	∞	10	∞	5	6	4	1	7
	$O_{2,4}$	10	8	9	6	4	7	∞	∞
J3	$O_{3,1}$	10	∞	∞	7	6	5	2	4
	$O_{3,2}$	∞	10	6	4	8	9	10	∞
	$O_{3,3}$	1	4	5	6	∞	10	∞	7
J4	$O_{4,1}$	3	1	6	5	9	7	8	4
	$O_{4,2}$	12	11	7	8	10	5	6	9
	$O_{4,3}$	4	6	2	10	3	9	5	7

Continuação da Tabela 4.2

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J5	$O_{5,1}$	3	6	7	8	9	∞	10	∞
	$O_{5,2}$	10	∞	7	4	9	8	6	∞
	$O_{5,3}$	∞	9	8	7	4	2	7	∞
	$O_{5,4}$	11	9	∞	6	7	5	3	6
J6	$O_{6,1}$	6	7	1	4	6	9	∞	10
	$O_{6,2}$	11	∞	9	9	9	7	6	4
	$O_{6,3}$	10	5	9	10	11	∞	10	∞
J7	$O_{7,1}$	5	4	2	6	7	∞	10	∞
	$O_{7,2}$	∞	9	∞	9	11	9	10	5
	$O_{7,3}$	∞	8	9	3	8	6	∞	10
J8	$O_{8,1}$	2	8	5	9	∞	4	∞	10
	$O_{8,2}$	7	4	7	8	9	∞	10	∞
	$O_{8,3}$	9	9	∞	8	5	6	7	1
	$O_{8,4}$	9	∞	3	7	1	5	8	∞

Instância Kacem 10x10: possui 10 jobs com 30 operações que planejam ser processadas em 10 máquinas com flexibilidade total. A tabela 4.3 representa a instância 10x10.

Tabela 4.3 - Instância Kacem 10x10.
Fonte: Adaptado de Kacem et al. (2002a)

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
	$O_{1,2}$	4	1	1	3	4	8	10	4	11	4
	$O_{1,3}$	3	2	5	1	5	6	9	5	10	3
J2	$O_{2,1}$	2	10	4	5	9	8	4	15	8	4
	$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,3}$	6	11	2	7	5	3	5	14	9	2
J3	$O_{3,1}$	8	5	8	9	4	3	5	3	8	1
	$O_{3,2}$	9	3	6	1	2	6	4	1	7	2
	$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
J4	$O_{4,1}$	5	10	6	4	9	5	1	7	1	6
	$O_{4,2}$	4	2	3	8	7	4	6	9	8	4
	$O_{4,3}$	7	3	12	1	6	5	8	3	5	2
J5	$O_{5,1}$	7	10	4	5	6	3	5	15	2	6
	$O_{5,2}$	5	6	3	9	8	2	8	6	1	7
	$O_{5,3}$	6	1	4	1	10	4	3	11	13	9
J6	$O_{6,1}$	8	9	10	8	4	2	7	8	3	10
	$O_{6,2}$	7	3	12	5	4	3	6	9	2	15
	$O_{6,3}$	4	7	3	6	3	4	1	5	1	11
J7	$O_{7,1}$	1	7	8	3	4	9	4	13	10	7
	$O_{7,2}$	3	8	1	2	3	6	11	2	13	3
	$O_{7,3}$	5	4	2	1	2	1	8	14	5	7
J8	$O_{8,1}$	5	7	11	3	2	9	8	5	12	8
	$O_{8,2}$	8	3	10	7	5	13	4	6	8	4
	$O_{8,3}$	6	2	13	5	4	3	5	7	9	5
J9	$O_{9,1}$	3	9	1	3	8	1	6	7	5	4
	$O_{9,2}$	4	6	2	5	7	3	1	9	6	7
	$O_{9,3}$	8	5	4	8	6	1	2	3	10	12

Continuação da Tabela 4.3

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J10	$O_{10,1}$	4	3	1	6	7	1	2	6	20	6
	$O_{10,2}$	3	1	8	1	9	4	1	4	17	15
	$O_{10,3}$	9	2	4	2	3	5	2	4	10	23

Instância Kacem 15x10: possui 15 jobs com 56 operações que planejam ser processadas em 10 máquinas com flexibilidade total. Considerado pela literatura uma instância de larga escala. A tabela 4.4 representa a instância 15x10.

Tabela 4.4 - Instância Kacem 15x10.
Fonte: Adaptado de Kacem et al. (2002a)

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	4
	$O_{1,2}$	1	1	3	4	8	10	4	11	4	3
	$O_{1,3}$	2	5	1	5	6	9	5	10	3	2
	$O_{1,4}$	10	4	5	9	8	4	15	8	4	4
J2	$O_{2,1}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,2}$	6	11	2	7	5	3	5	14	9	2
	$O_{2,3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2,4}$	9	3	6	1	2	6	4	1	7	2
J3	$O_{3,1}$	7	1	8	5	4	9	1	2	3	4
	$O_{3,2}$	5	10	6	4	9	5	1	7	1	6
	$O_{3,3}$	4	2	3	8	7	4	6	9	8	4
	$O_{3,4}$	7	3	12	1	6	5	8	3	5	2
J4	$O_{4,1}$	6	2	5	4	1	2	3	6	5	4
	$O_{4,2}$	8	5	7	4	1	2	36	5	8	5
	$O_{4,3}$	9	6	2	4	5	1	3	6	5	2
	$O_{4,4}$	11	4	5	6	2	7	5	4	2	1
J5	$O_{5,1}$	6	9	2	3	5	8	7	4	1	2
	$O_{5,2}$	5	4	6	3	5	2	28	7	4	5
	$O_{5,3}$	6	2	4	3	6	5	2	4	7	9
	$O_{5,4}$	6	5	4	2	3	2	5	4	7	5
J6	$O_{6,1}$	4	1	3	2	6	9	8	5	4	2
	$O_{6,2}$	1	3	6	5	4	7	5	4	6	5
J7	$O_{7,1}$	1	4	2	5	3	6	9	8	5	4
	$O_{7,2}$	2	1	4	5	2	3	5	4	2	5
J8	$O_{8,1}$	2	3	6	2	5	4	1	5	8	7
	$O_{8,2}$	4	5	6	2	3	5	4	1	2	5
	$O_{8,3}$	3	5	4	2	5	49	8	5	4	5
	$O_{8,4}$	1	2	36	5	2	3	6	4	11	2
J9	$O_{9,1}$	6	3	2	22	44	11	10	23	5	1
	$O_{9,2}$	2	3	2	12	15	10	12	14	18	16
	$O_{9,3}$	20	17	12	5	9	6	4	7	5	6
	$O_{9,4}$	9	8	7	4	5	8	7	4	56	2
J10	$O_{10,1}$	5	8	7	4	56	3	2	5	4	1
	$O_{10,2}$	2	5	6	9	8	5	4	2	5	4
	$O_{10,3}$	6	3	2	5	4	7	4	5	2	1
	$O_{10,4}$	3	2	5	6	5	8	7	4	5	2

Continuação da Tabela 4.4

Jobs		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J11	$O_{11,1}$	1	2	3	6	5	2	1	4	2	1
	$O_{11,2}$	2	3	6	3	2	1	4	10	12	1
	$O_{11,3}$	3	6	2	5	8	4	6	3	2	5
	$O_{11,4}$	4	1	45	6	2	4	1	25	2	4
J12	$O_{12,1}$	9	8	5	6	3	6	5	2	4	2
	$O_{12,2}$	5	8	9	5	4	75	63	6	5	21
	$O_{12,3}$	12	5	4	6	3	2	5	4	2	5
	$O_{12,4}$	8	7	9	5	6	3	2	5	8	4
J13	$O_{13,1}$	4	2	5	6	8	5	6	4	6	2
	$O_{13,2}$	3	5	4	7	5	8	6	6	3	2
	$O_{13,3}$	5	4	5	8	5	4	6	5	4	2
	$O_{13,4}$	3	2	5	6	5	4	8	5	6	4
J14	$O_{14,1}$	2	3	5	4	6	5	4	85	4	5
	$O_{14,2}$	6	2	4	5	8	6	5	4	2	6
	$O_{14,3}$	3	25	4	8	5	6	3	2	5	4
	$O_{14,4}$	8	5	6	4	2	3	6	8	5	4
J15	$O_{15,1}$	2	5	6	8	5	6	3	2	5	4
	$O_{15,2}$	5	6	2	5	4	2	5	3	2	5
	$O_{15,3}$	4	5	2	3	5	2	8	4	7	5
	$O_{15,4}$	6	2	11	14	2	3	6	5	4	8

Instância Brdata MK01: possui 10 jobs com 55 operações que planejam ser processadas em 6 máquinas. Nesta instância, todos os estágios impossíveis, estão denotados na tabela com o símbolo "∞". Considerado pela literatura uma instância de larga escala. A tabela 4.5 representa a instância MK01.

Tabela 4.5 - Instância Brdata MK01.
Fonte: Adaptado de Brandimarte (1993)

Jobs		*M1*	*M2*	*M3*	*M4*	*M5*	*M6*
J1	*O1,1*	*5*	∞	*4*	∞	∞	∞
	O1,2	∞	*1*	*5*	∞	*3*	∞
	O1,3	∞	∞	*4*	∞	∞	*2*
	O1,4	*1*	*6*	∞	∞	∞	*5*
	O1,5	∞	∞	*1*	∞	∞	∞
	O1,6	∞	∞	*6*	*3*	∞	*6*
J2	*O2,1*	∞	*6*	∞	∞	∞	∞
	O2,2	∞	∞	*1*	∞	∞	∞
	O2,3	*2*	∞	∞	∞	∞	∞
	O2,4	∞	*6*	∞	*6*	∞	∞
	O2,5	*1*	*6*	∞	∞	∞	*5*
J3	*O3,1*	∞	*6*	∞	∞	∞	∞
	O3,2	∞	∞	*4*	∞	∞	*2*
	O3,3	*1*	*6*	∞	∞	∞	*5*
	O3,4	∞	*6*	*4*	∞	∞	*6*
	O3,5	*1*	∞	∞	∞	*5*	∞

Continuação da Tabela 4.5.

Jobs		*M1*	*M2*	*M3*	*M4*	*M5*	*M6*
J4	*O4,1*	*1*	*6*	∞	∞	∞	*5*
	O4,2	∞	*6*	∞	∞	∞	∞
	O4,3	∞	∞	*1*	∞	∞	∞
	O4,4	∞	*1*	*5*	∞	*3*	∞
	O4,5	∞	∞	*4*	∞	∞	*2*
J5	*O5,1*	∞	*1*	*5*	∞	*3*	∞
	O5,2	*1*	*6*	∞	∞	∞	*5*
	O5,3	∞	*6*	∞	∞	∞	∞
	O5,4	*5*	∞	*4*	∞	∞	∞
	O5,5	∞	*6*	∞	*6*	∞	∞
	O5,6	∞	*6*	*4*	∞	∞	*6*
J6	*O6,1*	∞	∞	*4*	∞	∞	*2*
	O6,2	*2*	∞	∞	∞	∞	∞
	O6,3	∞	*6*	*4*	∞	∞	*6*
	O6,4	∞	*6*	∞	∞	∞	∞
	O6,5	*1*	*6*	∞	∞	∞	*5*
	O6,6	*3*	∞	∞	*2*	∞	∞
J7	*O7,1*	∞	∞	∞	∞	∞	*1*
	O7,2	*3*	∞	∞	*2*	∞	∞
	O7,3	∞	*6*	*4*	∞	∞	*6*
	O7,4	*6*	*6*	∞	∞	*1*	∞
	O7,5	∞	∞	*1*	∞	∞	∞
J8	*O8,1*	∞	∞	*4*	∞	∞	*2*
	O8,2	∞	*6*	*4*	∞	∞	*6*
	O8,3	*1*	*6*	∞	∞	∞	*5*
	O8,4	∞	*6*	∞	∞	∞	∞
	O8,5	∞	*6*	∞	*6*	∞	∞
J9	*O9,1*	∞	∞	∞	∞	∞	*1*
	O9,2	*1*	∞	∞	∞	*5*	∞
	O9,3	∞	∞	*6*	*3*	∞	*6*
	O9,4	*2*	∞	∞	∞	∞	∞
	O9,5	∞	*6*	*4*	∞	∞	*6*
	O9,6	∞	*6*	∞	*6*	∞	∞
J10	*O10,1*	∞	∞	*4*	∞	∞	*2*
	O10,2	∞	*6*	*4*	∞	∞	*6*
	O10,3	∞	*1*	*5*	∞	*3*	∞
	O10,4	∞	∞	∞	∞	∞	*1*
	O10,5	∞	*6*	∞	*6*	∞	∞
	O10,6	*3*	∞	∞	*2*	∞	∞

O processo de validação desta abordagem e os resultados dos testes experimentais estão apresentados no próximo capítulo.

4.4 Descrição da Abordagem Proposta

A abordagem proposta neste trabalho emprega um algoritmo ACO para resolver o FJSP em instâncias T-FJSP e P-FJSP de forma hierárquica em dois estágios: no primeiro estágio é resolvido o subproblema de roteamento usando a heurística SPT para alocação de recursos na busca da construção um modelo de atribuição ideal; no segundo estágio é resolvido o subproblema de programação pelo ACO, que determina o sequenciamento das operações atribuídas, onde cada formiga constrói um cronograma de programação viável conforme as restrições que se aplicam ao problema, e por fim, apresenta os resultados.

Com referência no trabalho de Brandimarte (1993), a heurística SPT foi integrada nesta abordagem para atribuir níveis de precedência aos recursos, dando preferência as operações que demandam menor tempo de processamento. Quando o tempo de processamento de uma máquina for igual a outro, é dado preferência à máquina que possui o menor índice. A heurística SPT define a atribuição da operação que demanda menor tempo de processamento em uma das máquinas do conjunto de recurso disponíveis. Ao tratar do P-FJSP adota-se a metodologia usada no trabalho de Kacem et al. (2002a), que consiste em atribuir um processamento fictício com tempo muito elevado, definido por 999 para cada estado proibido. Desta forma, o algoritmo rejeitara estas operações automaticamente.

O algoritmo ACO proposto é representado por um grafo disjuntivo da mesma forma que a abordagem de Wang, L. et al. (2017). No grafo, as arestas representam as trilhas de feromônio, que são os caminhos percorridos pelas formigas, e os vértices representam o processamento das operações nas máquinas. O tempo de processamento pode ser visto como o tempo de percurso da formiga no caminho de um nó para o outro. Neste processo o algoritmo é iniciado da seguinte maneira: todas as formigas estão inicialmente no vértice 0, em seguida percorrem o grafo aleatoriamente e escolhem os vértices de acordo com a quantidade de feromônio depositada nas arestas. A probabilidade de escolher um caminho em um determinado momento depende da quantidade total de feromônio no caminho, que por sua vez é proporcional ao número de formigas que usaram o caminho até esse momento. A atualização do feromônio é realizada a cada iteração, e as soluções são construídas por meio da combinação entre as trilhas de feromônio e a informação heurística.

Este procedimento pode ser exemplificado em um problema 3x3 com nove operações, onde cada *job* possui três operações: J1-1, J1-2 e J1-3 representam as operações do *job* 1; J2-1, J2-2 e J2-3 representam as operações do *job* 2; J3-1, J3-2 e J3-3 representam as operações do *job* 3. De acordo com as restrições da sequência do processo e a ocupação da máquina, as formigas viajam pelas nove operações dos três trabalhos formando o sequenciamento das operações em cada máquina, e em seguida a apresentam a solução. Este processo pode ser visualizado na figura 4.1.

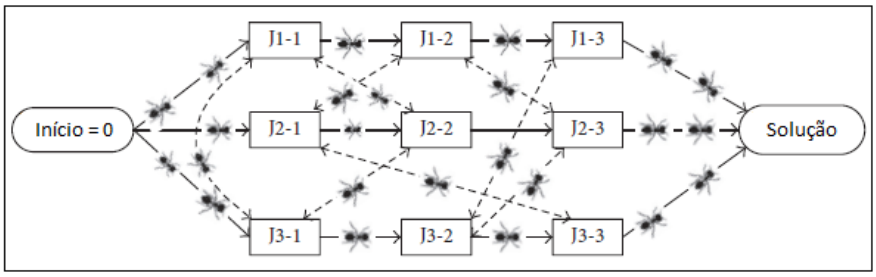

Figura 4.1 - Grafo disjuntivo representando o movimento das formigas.
Fonte: Adaptado de Wang, L. et al. (2017)

Neste trabalho o ACO é encarregado de resolver o subproblema de programação, no entanto, a metodologia empregada no algoritmo é mesma de um algoritmo ACO básico que poderia ser aplicado como uma abordagem integrada na resolução do FJSP. As particularidades desta pesquisa se caracterizam pela integração do algoritmo *Ant System* com a heurística SPT para resolver o FJSP como uma abordagem hierárquica.

A representação matemática do algoritmo ACO proposto pode ser visto da seguinte maneira: a probabilidade P_{ijk}^{f} de uma formiga f escolher as operações O_{ij} do *job* j_i para ser processadas nas máquinas $M_{\mathrm{i,j}}$ é dado pela trilha de feromônio τ_{ij} e pela função heurística definida por $\eta_{ij=}\, p_{ijk}$. A construção de soluções para uma programação viável pode ser formalizada pelas expressões 2.1, 2.2 e 2.3 descritas no capítulo 2.

O funcionamento básico da abordagem proposta pode ser representado pelo fluxograma mostrado na figura 4.2.

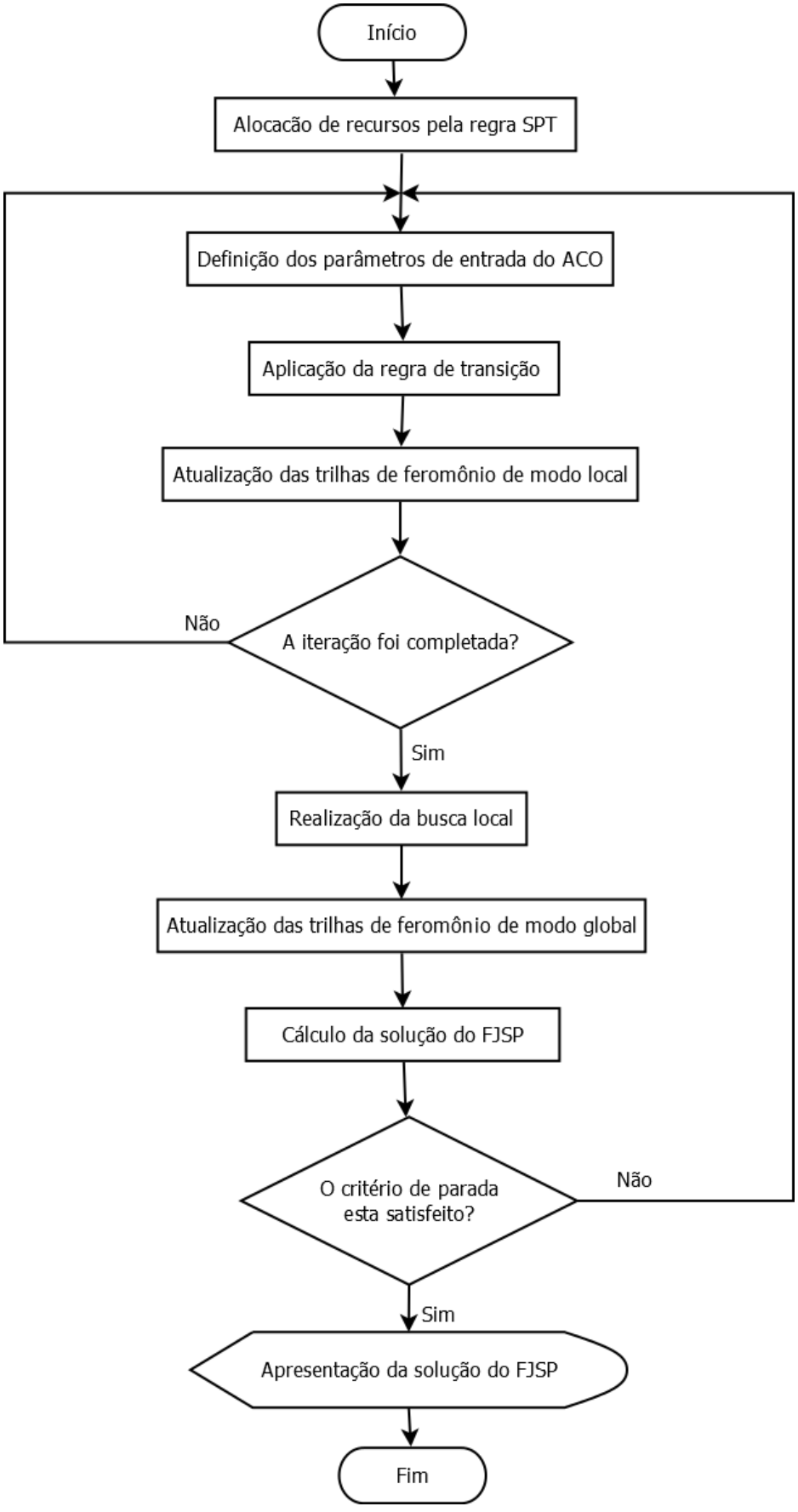

Figura 4.2 - Fluxograma do algoritmo proposto.

Capítulo 5

APLICAÇÃO DA ABORDAGEM E ANÁLISE DE RESULTADOS

5.1 Considerações Iniciais

Neste capítulo é apresentada a validação da abordagem proposta por meio da análise dos resultados obtidos em cada ciclo de testes em todos os cenários. A validação do algoritmo é realizada por meio de um estudo comparativo entre outras abordagens encontradas na literatura. Tais abordagens foram citadas no capítulo 03 e seus resultados foram demonstrados na sessão 5.3.

O algoritmo foi executado 200 vezes com 100 iterações em cada instância dos benchmarks utilizados. Definiu-se este número de execuções durante os ajustes de parâmetros do ACO, quando foi constatada a convergência do algoritmo proposto para as melhores soluções nas instâncias de larga escala no intervalo [1, 187] do número total de execuções consecutivas, com variações dos resultados durante o período de testes.

5.2 Ajustes de Parâmetros do ACO

É possível observar na literatura que a configuração dos parâmetros adotada para abordagens ACO pode ser determinada por meio de procedimentos experimentais com valores alternativos (i.e., tentativas), como demonstra o trabalho de Xing et al. (2010). Além disso, observou-se que vários trabalhos correlatos como as obras de Liouane et al. (2007) e

Wang, L. et al. (2017) fazem referência aos experimentos realizados e adotam o mesmo conjunto de parâmetros para todas as instâncias de testes.

De acordo Dorigo e Stützle (2004) os parâmetros do ACO podem ser ajustados conforme o tamanho e a dificuldade do problema a ser resolvido. A definição dos parâmetros pode ser realizada observando algumas afirmações: (1) o intervalo dos possíveis valores de feromônio, embora importante, não é o suficiente para permitir a solução efetiva de grandes problemas de otimização; (2) atualização de feromônio referenciada na qualidade da solução é importante para convergência rápida; (3) valores elevados para influência da trilha de feromônio pode causar flutuações iniciais aleatórias, acarretando comportamento inadequado do algoritmo; (4) quanto maior o número de formigas, melhor o comportamento de convergência do algoritmo, embora demande maior tempo de processamento; (5) a evaporação do feromônio é importante quando se tenta resolver problemas complexos. Partindo desses pressupostos, os parâmetros definidos para o algoritmo proposto foram calibrados empiricamente durante os experimentos em todas as instâncias de modo a obter os melhores resultados.

A configuração adotada neste trabalho para os parâmetros do ACO correspondem aos valores abaixo:

- Número de formigas = 15;
- $\alpha = 2$;
- $\beta = 4$;
- $\rho = 0.001$;
- Possíveis valores de feromônio $[t_{mim} = 0.001,\ t_{max} = 10]$;
- Número de iterações = 100.

Os experimentos comprovaram a teoria de Dorigo e Stützle (2004) e mostraram que valores mais altos de β e valores mais baixos de α e ρ melhoram a qualidade da solução. Notou-se que o intervalo dos possíveis valores de feromônio interfere nos resultados quando a distância entre t_{mim} e t_{max} é muito curta; e o número de formigas pode influenciar nos resultados conforme o número de iterações, ou vice versa. Quando o número de formigas é pequeno, a solução é melhorada se o número de iterações for grande. O algoritmo proposto utilizou a mesma configuração de parâmetros para todas as instâncias dos benchmarks utilizados.

5.3 Avaliação da Abordagem

Nas seções a seguir são apresentados os resultados da abordagem proposta e a comparação de desempenho em cada uma das instâncias utilizadas com outras abordagens de pesquisas correlatas encontradas na literatura. Após a apresentação dos resultados de cada objetivo é mostrada uma tabela com os valores de média e desvio padrão das soluções obtidas em todas as execuções consecutivas. O tempo de execução também é mostrado, mas não é considerado como critério de desempenho na avaliação do algoritmo proposto.

Os pesos atribuídos para a soma ponderada no cálculo da função de aptidão foram definidos com referência no trabalho de Xue et al. (2014), que definiu os respectivos valores: $w1 = 0{,}5;\ \ w2 = 0{,}3;\ \ w3 = 0{,}2$. Desta forma, foi ponderado um peso maior na soma para o valor de C_k, seguido pelo valor de W_k e W_t respectivamente. Neste trabalho os pesos são aplicados conforme a formulação matemática do problema apresentado no capítulo anterior.

Os gráficos de Gantt apresentados nas sessões abaixo correspondem à programação de cada solução encontrada durante o conjunto de testes em todas as execuções consecutivas.

O estudo comparativo é realizado por meio da comparação dos resultados apresentados com as seguintes abordagens: AL + CGA proposto por Kacem et al. (2002a); PSO + SA proposto por Xia e Wu (2005); PSO + TS proposto por Zhang et al. (2009); QIA proposto por Xue et al. (2014); MOPSO proposto por Huang et al. (2016); BEG-NSGA-II proposto por Deng et al. (2017). Todas as abordagens apontadas neste estudo foram citadas no capítulo 03. Os dados foram apresentados em uma tabela com os melhores resultados destacados em negrito e ilustrados por um gráfico em cada instância de teste.

5.3.1 Solução do FJSP Multiobjetivo na Instância Kacem 4x5

O algoritmo proposto se manteve instável durante os testes e apresentou a seguinte solução: C_k = 11, W_k = 10 e W_t = 32. Aplicando a soma ponderada, o valor da função de aptidão é 14.9.

O tempo computacional gasto durante as 200 execuções consecutivas foi de aproximadamente 01 minuto e 25 segundos. As informações estatísticas de todos os testes para cada objetivo são mostradas na tabela 5.1.

Tabela 5.1 - Informações estatísticas dos testes na instância Kacem 4x5.

Objetivo	Média	Desvio Padrão
C_k	11	0
W_k	10	0
W_t	32	0

O gráfico de Gantt é apresentado abaixo na figura 5.1.

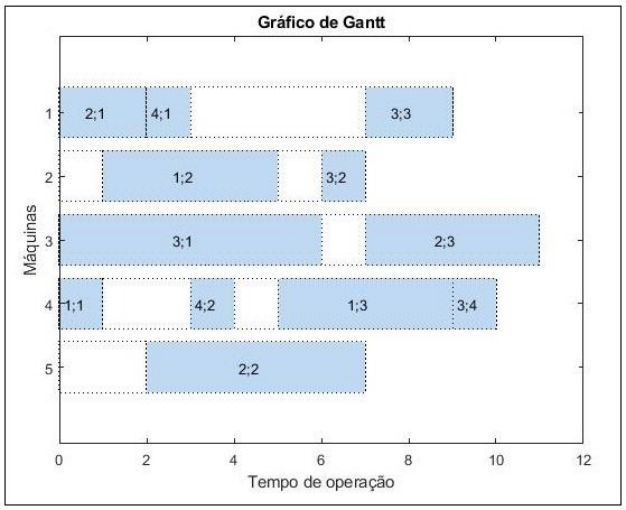

Figura 5.1 - Gráfico de Gantt do problema 4x5.

A tabela 5.2 mostra a comparação dos resultados da abordagem proposta com outras abordagens encontradas na literatura.

Tabela 5.2 - Comparação dos resultados do problema 4x5.

Abordagens	C_k	W_k	W_t
AL + CGA	16	**10**	34
PSO + TS	**11**	**10**	**32**
MOPSO	**11**	**10**	**32**
Abordagem proposta	**11**	**10**	**32**

O gráfico apresentado na figura 5.2 ilustra o desempenho da abordagem proposta em comparação com as outras abordagens.

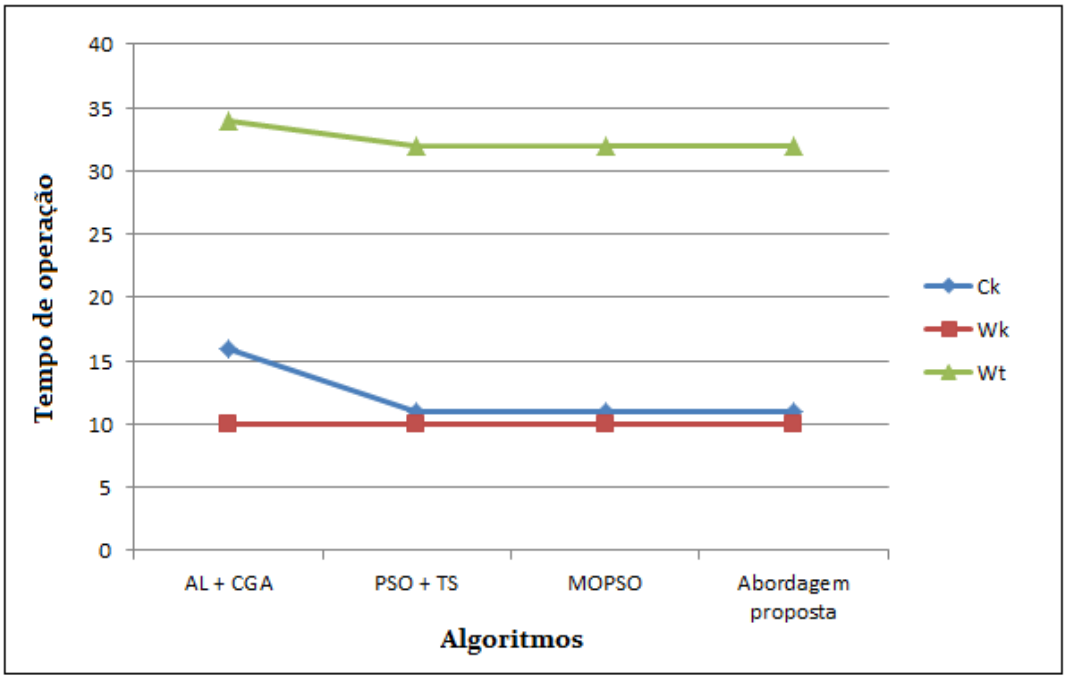

Figura 5.2 - Gráfico comparativo do problema 4x5.

O algoritmo proposto convergiu para as melhores soluções nos três objetivos, foi compatível com os algoritmos PSO + TS e AL + CGA e obteve melhores resultados de C_k e W_t em comparação ao algoritmo AL + CGA.

5.3.2 Solução do FJSP Multiobjetivo na Instância Kacem 8x8

O algoritmo proposto não apresentou variação dos resultados durante os testes e obteve a seguinte solução: C_k = 14, W_k = 12 e W_t = 78. Aplicando a soma ponderada, o valor da função de aptidão é 26.2.

O tempo computacional gasto durante as 200 execuções consecutivas foi de aproximadamente 05 minutos e 45 segundos. As informações estatísticas de todos os testes para cada objetivo são mostradas na tabela 5.3.

Tabela 5.3 - Informações estatísticas dos testes na instância Kacem 8x8.

Objetivo	Média	Desvio Padrão
C_k	14	0
W_k	12	0
W_t	78	0

O gráfico de Gantt é apresentado abaixo na figura 5.3.

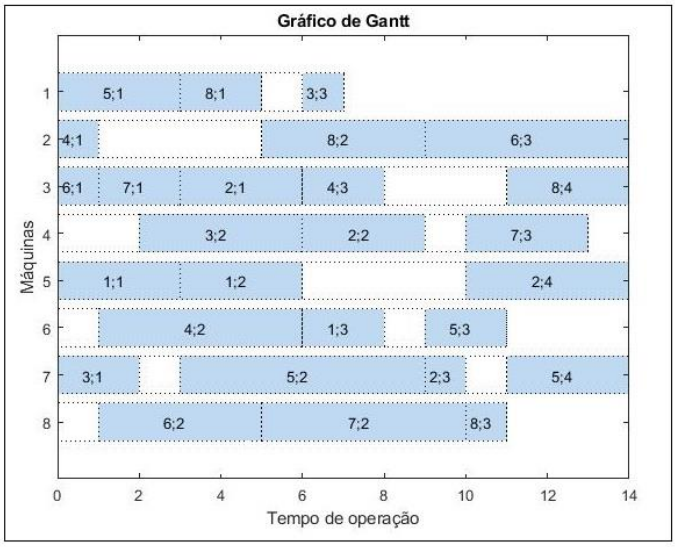

Figura 5.3 - Gráfico de Gantt do problema 8x8.

A tabela 5.4 mostra a comparação dos resultados da abordagem proposta com outras abordagens encontradas na literatura, considerando a melhor solução apresentada por cada algoritmo.

Tabela 5.4 - Comparação dos resultados do problema 8x8.

Abordagens	C_k	W_k	W_t
AL + CGA	15	13	79
	16	13	75
PSO + SA	15	12	75
	16	13	**73**
PSO + TS	**14**	12	77
	15	12	75
QIA	**14**	12	77
MOPSO	16	13	73
	14	12	77
	16	**11**	77
	15	12	75
Abordagem proposta	**14**	12	78

O gráfico apresentado na figura 5.4 ilustra o desempenho da abordagem proposta em comparação com as outras abordagens.

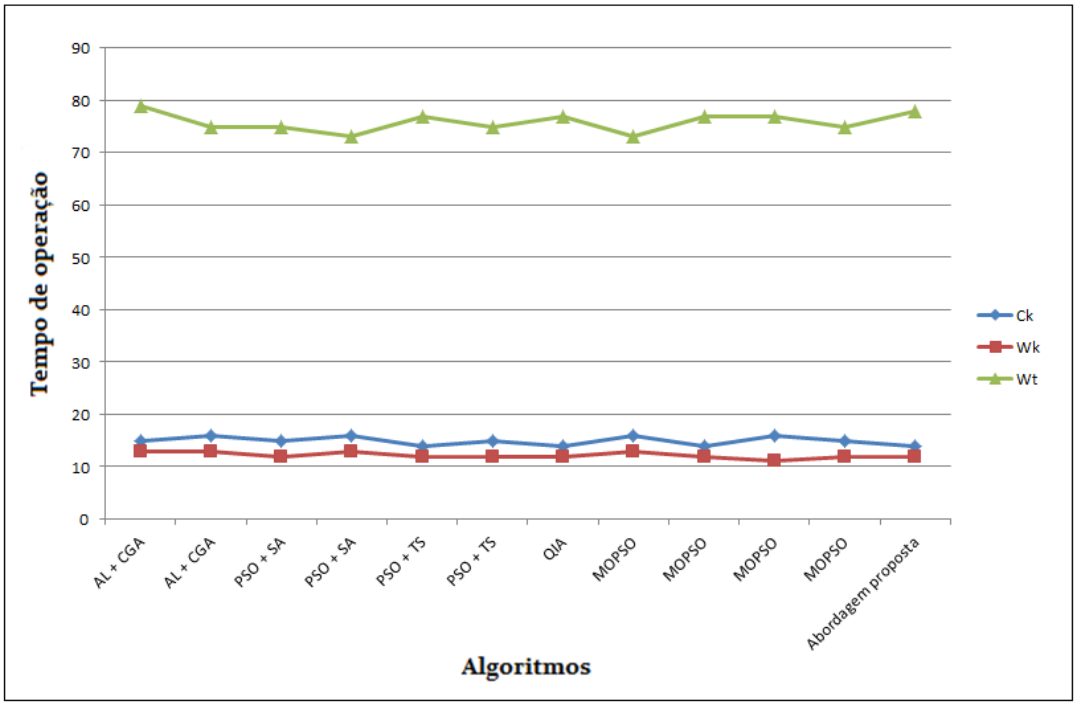

Figura 5.4 - Gráfico comparativo do problema 8x8.

Considerando os menores valores de resultados para todos os objetivos, observou-se que a maioria das abordagens mostraram soluções aceitáveis nesta instância, mas a análise individual dos objetivos mostrou que a abordagem proposta apresentou boas soluções nos objetivos C_k e W_k em comparação aos demais algoritmos. Analisando os resultados de W_k, percebe-se que uma das soluções apresentadas pelo algoritmo MOPSO supera os demais algoritmos, que apontaram variações nos resultados, mas foram compatíveis, com exceção do algoritmo AL + CGA que foi inferior neste critério de desempenho. Quanto aos resultados de W_t, o algoritmo PSO + SA apresentou a melhor solução e mostrou superioridade aos algoritmos PSO + TS, QIA e MOPSO que foram compatíveis. Neste objetivo o algoritmo proposto superou apenas o algoritmo AL + CGA e foi inferior aos outros algoritmos. Conforme o estudo realizado, nota-se que a solução do FJSP multiobjetivo obtida pelo algoritmo proposto é compatível em comparação aos demais algoritmos.

5.3.3 Solução do FJSP Multiobjetivo na Instância Kacem 10x10

O algoritmo proposto se manteve instável durante os testes e apresentou a seguinte solução: C_k = 7, W_k = 6 e W_t = 42. Aplicando a soma ponderada, o valor da função de aptidão é 13.7.

O tempo computacional gasto durante as 200 execuções consecutivas foi de aproximadamente 09 minutos e 13 segundos. As informações estatísticas de todos os testes para cada objetivo são mostradas na tabela 5.5.

Tabela 5.5 - Informações estatísticas dos testes na instância Kacem 10x10.

Objetivo	Média	Desvio Padrão
C_k	7	0
W_k	6	0
W_t	42	0

O gráfico de Gantt é apresentado abaixo na figura 5.5.

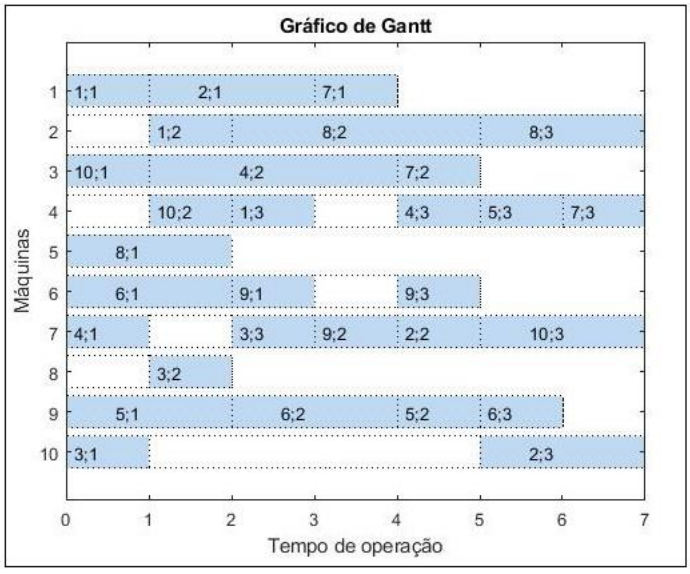

Figura 5.5 - Gráfico de Gantt do problema 10x10.

A tabela 5.6 mostra a comparação dos resultados da abordagem proposta com outras abordagens encontradas na literatura, considerando a melhor solução apresentada por cada algoritmo.

Tabela 5.6 - Comparação dos resultados do problema 10x10.

Abordagens	C_k	W_k	W_t
AL + CGA	**7**	**5**	45
PSO + SA	**7**	6	44
PSO + TS	**7**	6	43
QIA	**7**	6	**42**
MOPSO	**7**	6	**42**
Abordagem proposta	**7**	6	**42**

O gráfico apresentado na figura 5.6 ilustra o desempenho da abordagem proposta em comparação com as outras abordagens.

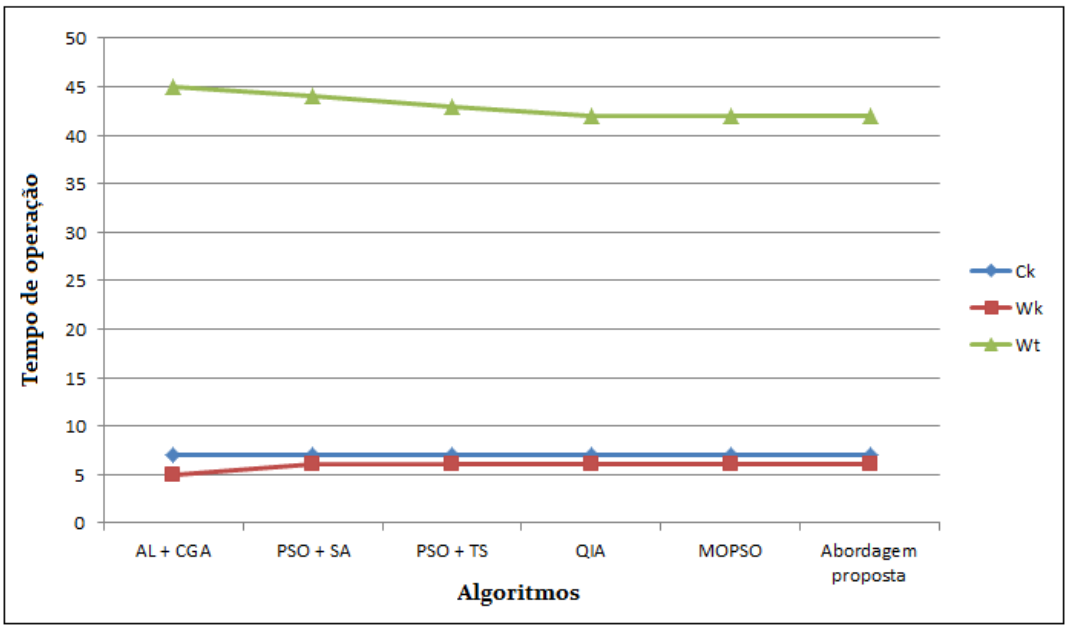

Figura 5.6 - Gráfico comparativo do problema 10x10.

Os resultados mostraram que o algoritmo proposto convergiu para as melhores soluções e foi compatível com os algoritmos QIA e MOPSO. Em relação aos critérios de desempenho, observou-se que todos os algoritmos convergiram para a melhor solução de C_k, mas apresentaram resultados diferentes na solução dos demais objetivos. O algoritmo AL + CGA foi superior aos outros algoritmos na solução de W_k, enquanto que as demais abordagens foram compatíveis entre si. Quanto à solução de W_t, a abordagem proposta apresentou os melhores resultados e foi compatível com os algoritmos QIA e MOPSO, mostrando superioridade aos algoritmos PSO + TS, seguido pelo PSO + SA e AL + CGA. Conforme o estudo realizado, nota-se que o algoritmo proposto e os algoritmos QIA e MOPSO são dominantes, seguido pelos algoritmos PSO + TS, AL + CGA e PSO + SA. O

estudo comparativo mostrou que o algoritmo proposto e as demais abordagens dominantes demonstraram as melhores soluções em quase todos os objetivos, mas é possível destacar que no geral todas as abordagens apresentaram boas soluções para o FJSP multiobjetivo nesta instância.

5.3.4 Solução do FJSP Multiobjetivo na Instância Kacem 15x10

O algoritmo proposto apresentou variação dos resultados durante os testes e obteve duas soluções:

- Solução 1 - C_k = 12, W_k = 12 e W_t = 94. Aplicando a soma ponderada, o valor da função de aptidão é 28.4;
- Solução 2 - C_k = 11, W_k = 11 e W_t = 91. Aplicando a soma ponderada, o valor da função de aptidão é 27.

O tempo computacional gasto durante as 200 execuções consecutivas foi de aproximadamente 27 minutos e 12 segundos. As informações estatísticas de todos os testes para cada objetivo são mostradas na tabela 5.7.

Tabela 5.7 - Informações estatísticas dos testes na instância Kacem 15x10.

Objetivo	Média	Desvio Padrão
C_k	11,98	0,14
W_k	11,98	0,14
W_t	93,94	0,42

O gráfico de Gantt da solução 1 é apresentado abaixo na figura 5.7.

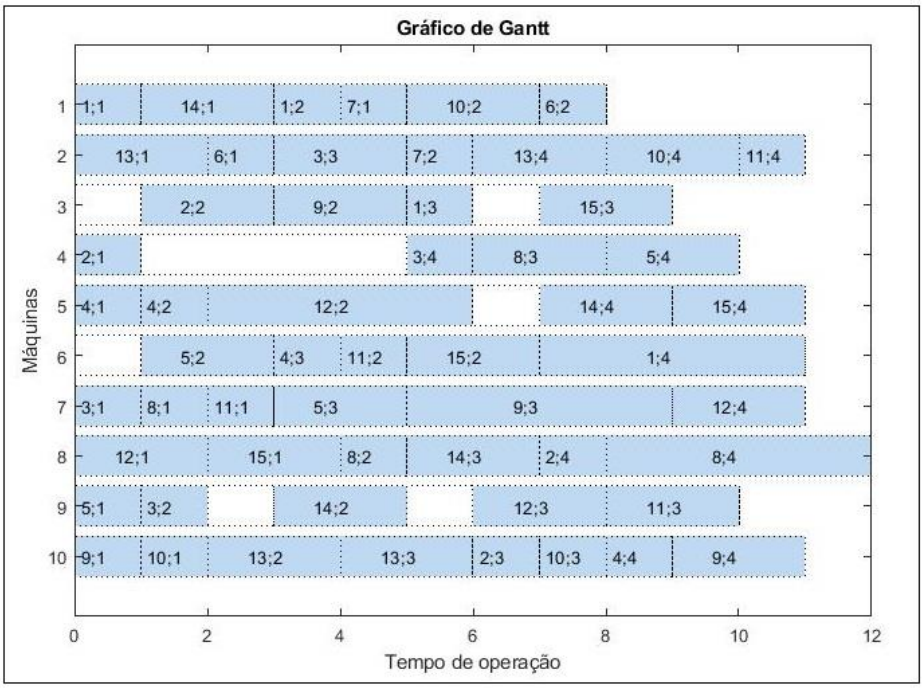

Figura 5.7 - Gráfico de Gantt da solução 1 do problema 15x10.

O gráfico de Gantt da solução 2 é apresentado abaixo na figura 5.8.

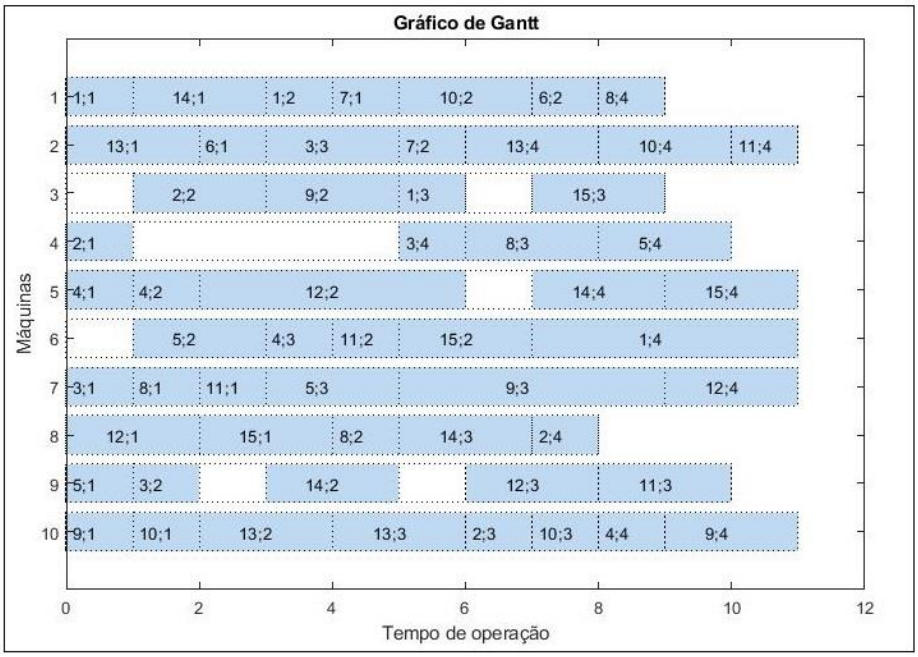

Figura 5.8 - Gráfico de Gantt da solução 2 do problema 15x10.

A tabela 5.8 mostra a comparação dos resultados da abordagem proposta com outras abordagens encontradas na literatura, considerando a melhor solução apresentada por cada algoritmo.

Tabela 5.8 - Comparação dos resultados do problema 15x10.

Abordagens	C_k	W_k	W_t
AL + CGA	23	11	95
PSO + SA	12	11	**91**
PSO + TS	**11**	11	93
QIA	**11**	11	**91**
MOPSO	**11**	11	**91**
	11	**10**	93
Abordagem proposta	12	12	94
	11	11	**91**

O gráfico apresentado na figura 5.9 ilustra o desempenho da abordagem proposta em comparação com as outras abordagens.

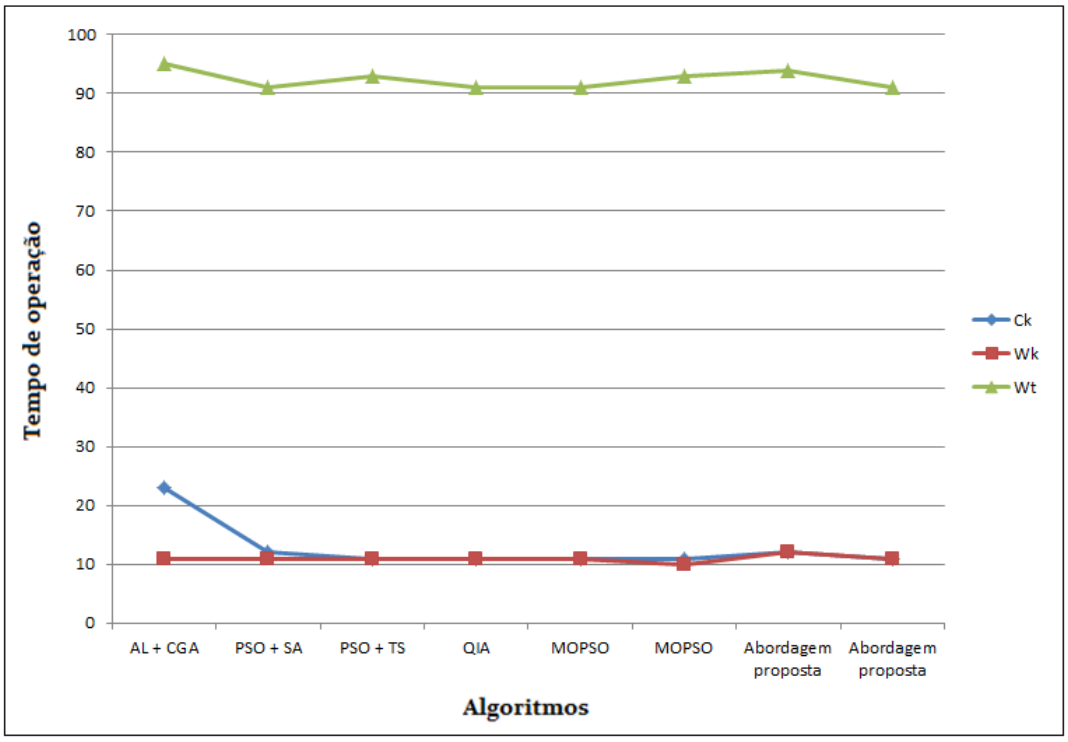

Figura 5.9 - Gráfico comparativo do problema 15x10.

Considerando os menores valores de resultados obtidos para solução de todos os objetivos, é possível destacar que o algoritmo proposto convergiu para as melhores soluções e foi compatível com os algoritmos QIA e MOPSO. Em relação aos critérios de desempenho, observou-se que o algoritmo proposto e os algoritmos PSO + TS, QIA e MOPSO apresentaram as melhores soluções de C_k, seguido pelo algoritmo PSO + SA e por fim o algoritmo AL + CGA. Uma das soluções apresentadas pelo algoritmo MOPSO corresponde

ao menor valor de W_k, no entanto, os demais algoritmos também demonstraram bons resultados neste critério de desempenho. Quanto à solução de W_t, a abordagem proposta exibiu um dos melhores resultados e foi compatível com os algoritmos PSO + SA, QIA e MOPSO, mostrando superioridade ao algoritmo PSO + TS, seguido pelo algoritmo AL + CGA. No geral, o estudo comparativo revela que o algoritmo proposto apresentou um dos melhores resultados e foi compatível com os algoritmos QIA e MOPSO, que superaram o algoritmo PSO + TS, seguido pelos algoritmos PSO + SA e AL + CGA. É possível notar que as melhores soluções obtidas pelo algoritmo proposto em todos os objetivos são consideradas ótimas pela literatura. As demais abordagens também revelaram boas soluções para o FJSP multiobjetivo nesta instância, porém a abordagem AL + CGA apresentou um resultado muito distante dos outros algoritmos com relação à solução apresentada para C_k.

5.3.5 Solução do FJSP Multiobjetivo na Instância Brdata MK01

O algoritmo proposto mostrou variação dos resultados durante os testes e obteve três soluções:

- Solução 1 - C_k = 41, W_k = 37 e W_t = 165. Aplicando a soma ponderada, o valor da função de aptidão é 64,6;
- Solução 2 - C_k = 40, W_k = 36 e W_t = 167. Aplicando a soma ponderada, o valor da função de aptidão é 64,2;
- Solução 3 - C_k = 42, W_k = 42 e W_t = 167. Aplicando a soma ponderada, o valor da função de aptidão é 67.

O tempo computacional gasto durante as 200 execuções consecutivas foi de aproximadamente 11 minutos e 10 segundos. As informações estatísticas de todos os testes para cada objetivo são mostradas na tabela 5.9.

Tabela 5.9 - Informações estatísticas dos testes na instância Brdata MK01.

Objetivo	Média	Desvio Padrão
C_k	40,99	0,26
W_k	37,11	0,88
W_t	165,14	0,51

O gráfico de Gantt da solução 1 é apresentado abaixo na figura 5.10.

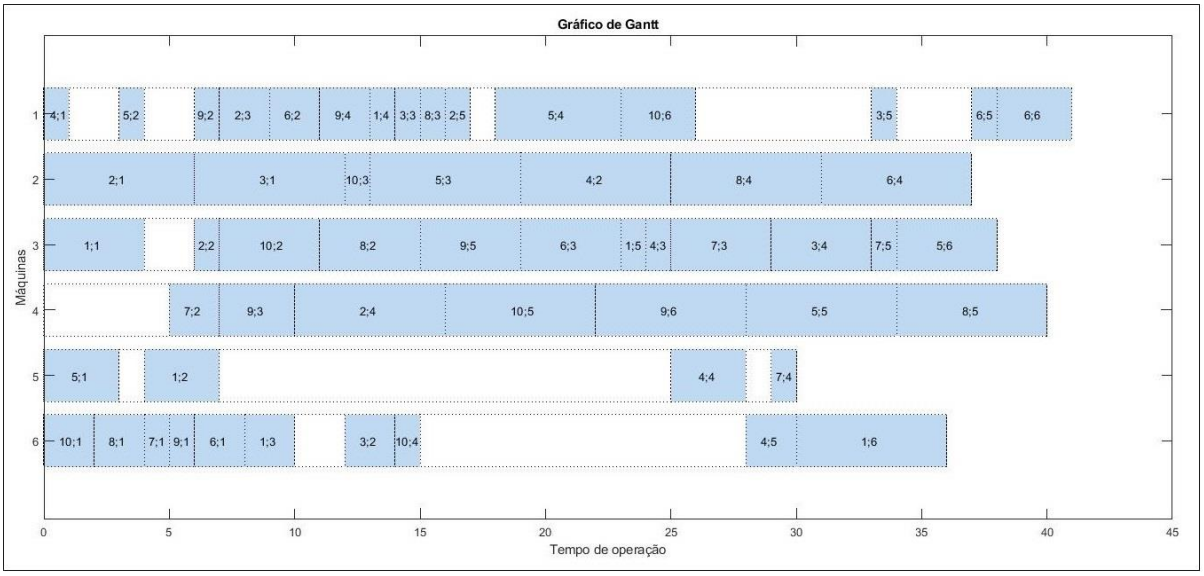

Figura 5.10 - Gráfico de Gantt da solução 1 do problema MK01.

O gráfico de Gantt da solução 2 é apresentado abaixo na figura 5.11.

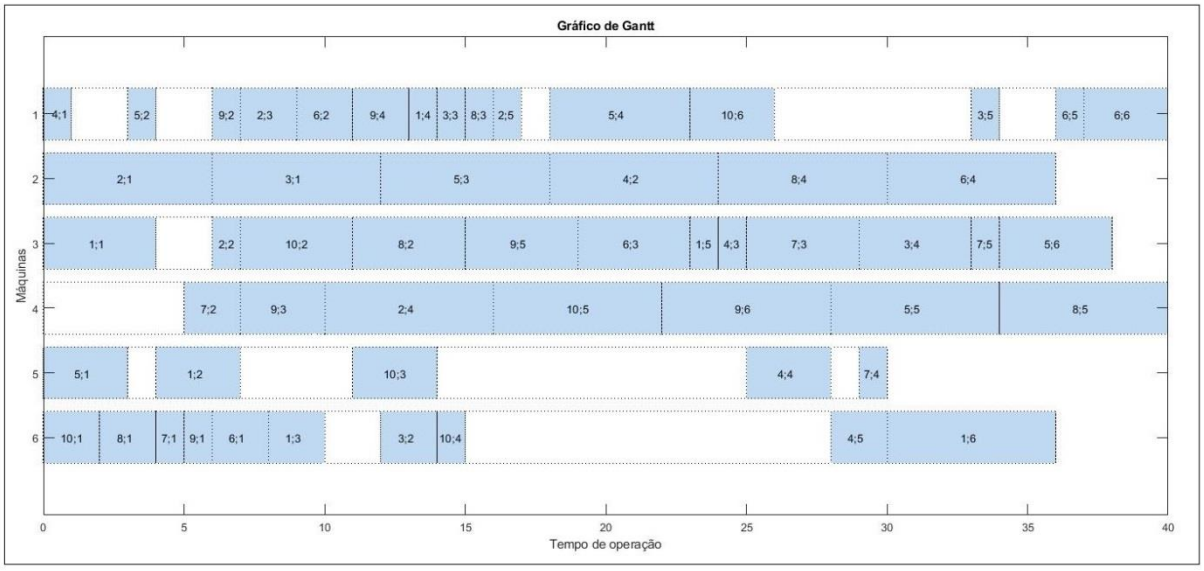

Figura 5.11 - Gráfico de Gantt da solução 2 do problema MK01.

O gráfico de Gantt da solução 3 é apresentado abaixo na figura 5.12.

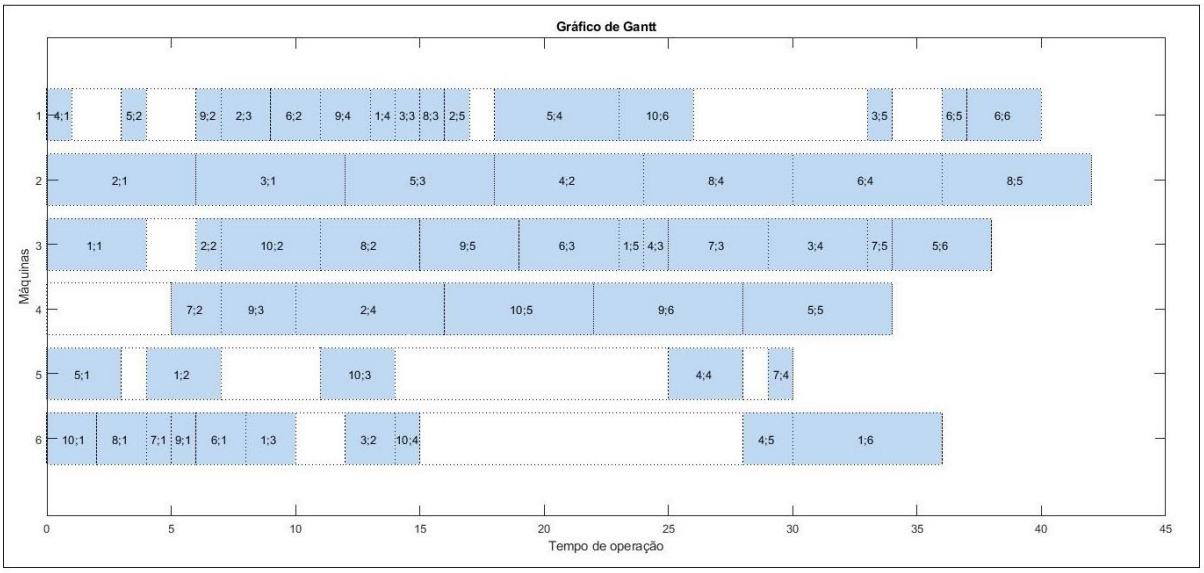

Figura 5.12 - Gráfico de Gantt da solução 3 do problema MK01.

Os dados dos algoritmos utilizados neste estudo foram retirados de Huang et al. (2016) e Deng et al. (2017).

Huang et al. (2016) apresentaram um estudo comparativo com as seguintes abordagens: algoritmo Xing proposto por Xing et al. (2009); algoritmo genético multiobjetivo (do inglês *Multi-objective Genetic Algorithm* - MOGA) proposto por Wang, X. et al. (2010); algoritmo híbrido de busca tabu (do inglês *Hybrid Tabu Search Algorithm* - HTSA) proposto por Li et al. (2010); algoritmo híbrido de salto de sapo embaralhado (do inglês *Hybrid Shuffled Frog Leaping Algorithm* - HSFLA) proposto por Li et al. (2012); e algoritmo AIA proposto por Bagheri et al. (2010). Estes dados também foram utilizados neste trabalho.

A tabela 5.10 mostra a comparação dos resultados da abordagem proposta com outras abordagens encontradas na literatura, considerando a melhor solução apresentada por cada algoritmo.

Tabela 5.10 - Comparação dos resultados do problema MK01.

Abordagens	C_k	W_k	W_t
Xing	42	42	162
MOGA	**40**	**36**	169
HTSA	**40**	**36**	167
HSFLA	**40**	37	165
AIA	**40**	**36**	171

Continuação da tabela 5.10.

Abordagens	C_k	W_k	W_t
MOPSO	**40**	**36**	167
BEG-NSGA-II	**40**	39	158
	44	40	**154**
	43	40	155
	40	**36**	169
	47	**36**	167
	47	37	165
	49	37	164
	50	38	162
Abordagem proposta	41	37	165
	40	**36**	167
	42	42	167

O gráfico apresentado na figura 5.13 ilustra o desempenho da abordagem proposta em comparação com as outras abordagens.

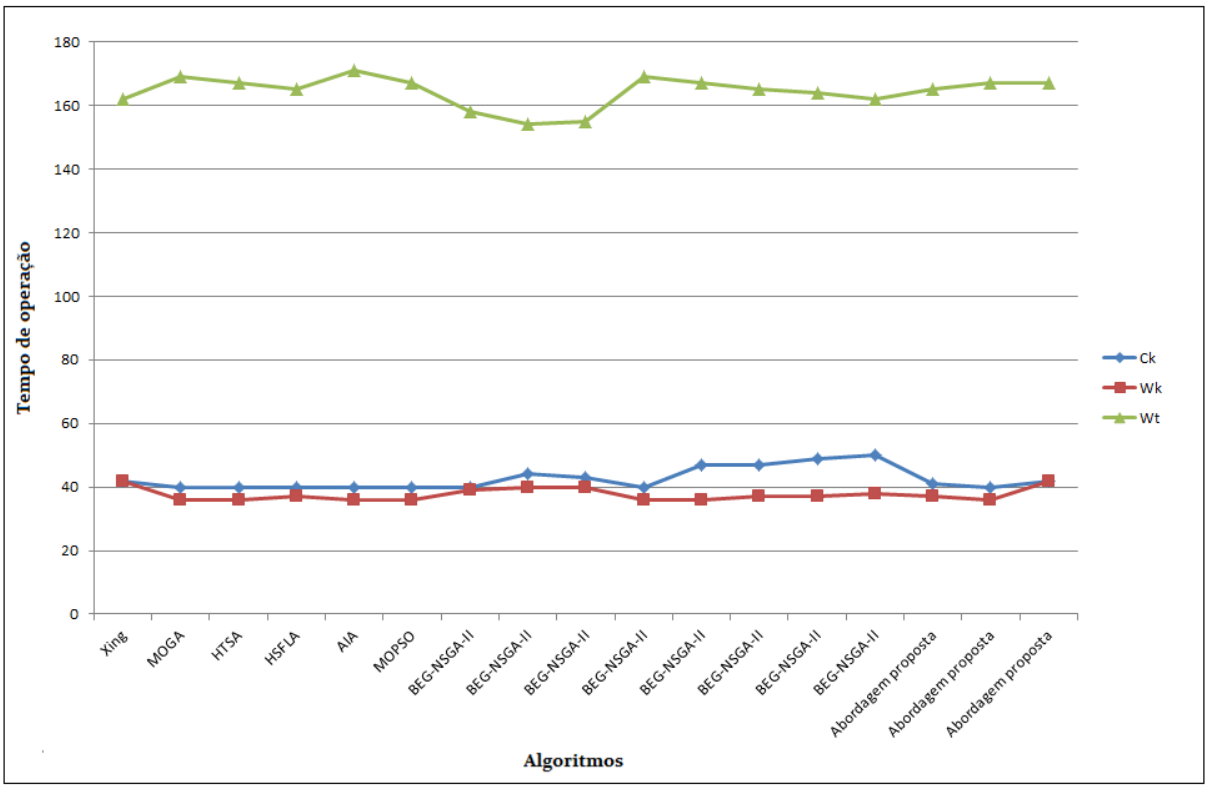

Figura 5.13 - Gráfico comparativo do problema MK01.

Considerando os menores valores dos resultados apresentados em cada objetivo por todas as abordagens, é possível contextualizar que a abordagem proposta apresentou uma das melhores soluções em comparação com as demais abordagens destacadas neste trabalho, e foi compatível com o algoritmo MOPSO. Em relação aos critérios de desempenho, observou-se que todos os algoritmos mostraram soluções aceitáveis para C_k, no entanto, um dos resultados obtidos pelo algoritmo proposto corresponde a melhor solução. Quanto às soluções de W_k, nota-se que o algoritmo proposto apresentou os melhores resultados com compatibilidade aos algoritmos MOGA, HTSA, AIA, MOPSO e BEG-NSGA-II, e foi superior ao algoritmo HSFLA, seguido pelo algoritmo Xing. O menor valor de W_t foi revelado em uma das soluções obtidas pelo algoritmo BEG-NSGA-II. Os resultados da solução deste objetivo foram mais dispersos em relação aos outros, mas é possível destacar que a abordagem proposta obteve soluções aceitáveis neste critério de desempenho, porém foi inferior aos algoritmos BEG-NSGA-II e Xing, mostrou compatibilidade ao algoritmo HSFLA e superioridade aos demais algoritmos. Neste critério de desempenho vislumbra-se que o algoritmo HSFLA superou os algoritmos HTSA e MOPSO, que foram compatíveis, os quais apresentaram melhores resultados que o algoritmo MOGA, seguido pelo algoritmo AIA. Conforme o estudo realizado é possível destacar que o algoritmo proposto obteve resultados compatíveis ao conjunto de soluções apresentadas pelos demais algoritmos.

Capítulo 6

CONSIDERAÇÕES FINAIS

Nos últimos anos, houve uma preocupação crescente com o problema de programação do tipo FJSP, que é uma extensão do JSP clássico. Considerado pela literatura como um problema NP-difícil, na qual via minimizar os objetivos predefinidos. Devido a sua complexidade, pode ser divido em dois subproblemas: um é o roteamento, que consiste em alocar as operações nas máquinas e o outro é a programação que consiste em sequenciar a ordem das operações em cada máquina. Vale ressaltar que a maior parte das pesquisas busca apenas um único alvo de otimização, e os modelos existentes carecem de versatilidade devido à falta de restrições sobre a produção real. Portanto, neste trabalho foi proposto um algoritmo ACO para resolver o FJSP multiobjetivo.

Nesta abordagem o algoritmo ACO é aplicado de forma hierárquica em dois estágios. No primeiro estágio é resolvido o subproblema de roteamento usando a regra SPT e no segundo estágio é resolvido o subproblema de programação, onde cada formiga constrói um cronograma de programação viável conforme as restrições que se aplicam ao problema, e em seguida, as soluções resultantes são levadas a um ótimo local pelo mecanismo de pesquisa local. Cada uma dessas soluções é usada no processo de atualização de feromônio. Por meio deste procedimento a abordagem proposta convergiu para soluções ótimas com custo computacional relativamente baixo, considerando o tempo total de produção (C_k), a carga de trabalho da máquina mais crítica (W_k) e a carga de trabalho de todas as máquinas (W_t) como critérios de desempenho.

O algoritmo proposto foi avaliado em quatro instâncias Kacem e uma instância Brdata, e comparado com outras abordagens encontradas na literatura mediante os resultados obtidos nos três critérios de desempenho. Os experimentos demonstraram que o algoritmo proposto é eficaz para resolver o FJSP multiobjetivo e mostrou eficiência nas instâncias de

grande escala apresentando melhores resultados em comparação com a maioria das abordagens observadas. Além disso, pode-se inferir que a abordagem proposta foi capaz de resolver o problema em quantidade razoável de tempo, de forma eficiente em todos os cenários.

6.1 Trabalhos Futuros

Visto que a indústria e os sistemas de manufatura são caracterizados por rupturas, eventos não planejados e incidentes imprevistos que podem acontecer a qualquer momento, como quebra da máquina, manutenção, entre outros, observa-se a possiblidade de estender esta pesquisa para tratar do FJSP multiobjetivo, considerando a programação reativa da produção.

Percebe-se que os valores de parâmetros do ACO impactam muito na qualidade das soluções apresentadas. Outra possibilidade de trabalho futuro com extensão desta pesquisa seria a inclusão de um mecanismo inteligente para ajuste de parâmetros automáticos. Nesta abordagem seria criado um banco de dados para guardar os valores de parâmetros, onde cada registro desta base de dados corresponderia a um conjunto de parâmetros do ACO. Esta base de informações poderia ser os dados de entrada para uma rede neural artificial que seria treinada para apresentar uma solução de parâmetros ao algoritmo ACO.

REFERÊNCIAS

AKYOL, D. E.; BAYHAN, G. M. **A review on evolution of production scheduling with neural networks**. Computers & Industrial Engineering, v. 53, n. 1, p. 95-122, 2007.

BAGHERI, A.; ZANDIEH, M.; MAHDAVI, I.; YAZDANI, M. **An artificial immune algorithm for the flexible job-shop scheduling problem**. Future Generation Computer Systems, v. 26, n. 4, p. 533-541, 2010.

BANKS, J.; CARSON, J. S.; NELSON, B. L. **Disrete-Event System Simulation**. Prendice-Hall, Englewood Cliffs, NJ, 1996.

BLUM, C.; ROLI, A. **Metaheuristics in combinatorial optimization: Overview and conceptual comparison**. ACM Computing Surveys (CSUR), v. 35, n. 3, p. 268-308, 2003.

BLUM, C.; ROLI, A.; DORIGO, M. HC–ACO: **The hyper-cube framework for Ant Colony Optimization.** In Proceedings of MIC'2001-Metaheuristics International Conference, vol. 2, 399–403, 2001.

BRANDIMARTE, P. **Routing and scheduling in a flexible job shop by tabu search**. Annals of Operations research, v. 41, n. 3, p. 157-183, 1993.

BULLNHEIMER, B.; HARTL, R. F.; STRAUSS, C. **A new rank-based version of the Ant System: A computational study.** Central European Journal for Operations Research and Economics, 7(1), 25–38, 1999.

COLORNI, A.; DORIGO, M.; MANIEZZO, V. **An investigation of some properties of an ant algorithm**. In R. Männer & B. Manderick (Eds.), Proceedings of PPSN-II, Second INTERNATIONAL CONFERENCE ON PARALLEL PROBLEM SOLVING FROM NATURE, p. 509–520, Amsterdam, Elsevier, 1992.

DAUZÈRE-PÉRÈS, S.; PAULLI, J. **An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search**. Annals of Operations Research, v. 70, p. 281-306, 1997.

DENG, Q.; GONG, G.; GONG, X.; ZHANG, L.; LIU, W.; REN, Q. **A Bee Evolutionary Guiding Nondominated Sorting Genetic Algorithm II for Multiobjective Flexible Job-Shop Scheduling**. Computational intelligence and neuroscience, v. 2017, 2017.

DESROCHERS, A. A.; AL-JAAR, R. Y. **Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis**. IEEE, 1995.

DORIGO, M. **Optimization, learning and natural algorithms**. PhD thesis, Dipartiment di Elettronica, Politecnico di Milano, Italy, 1992.

DORIGO, M.; DI CARO, G. **Ant Colony Optimisation: A new metaheuristic.** Proceedings of the Congress on Evolutionary Computation, p. 1470 - 1477, IEEE Press, 1999.

DORIGO, M.; GAMBARDELLA, L. M. **Ant colonies for the traveling salesman problem**. BioSystems, 43(2), 73–81, 1997a.

DORIGO, M.; GAMBARDELLA, L. M. **Ant Colony System:** A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1), 53–66, 1997b.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. **Positive feedback as a search strategy**. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. **Ant System: Optimization by a colony of cooperating agents.** IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26(1), 29–41, 1996.

DORIGO, M.; STUTZLE, T. **Ant Colony Optimization**. Massachusetts Institute of Technology, 2004.

FERNANDES, F. C. F; FILHO, M. G. **Planejamento e Controle da Produção**: dos fundamentos ao essencial. 1. ed. - São Paulo: Atlas, 2010.

FLÓREZ, E.; GÓMEZ, W.; BAUTISTA, L. **An ant colony optimization algorithm for job shop scheduling problem**. International Journal of Artificial Intelligence & Applications (IJAIA), v. 4, n. 4, p. 53-66, 2013.

FNAIECH, N.; HAMMAMI, H.; YAHYAOUI, A.; CHRISTOPHE, V.; FNAIECH, F.; NOUREDDINE, Z. **New Hopfield Neural Network for joint Job Shop Scheduling of production and maintenance.** In: IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society. IEEE, p. 5535-5541, 2013.

GAMILA, M. A.; MOTAVALLI, S. **A modeling technique for loading and scheduling problems in FMS**. Robotics and Computer-Integrated Manufacturing, v. 19, n. 1, p. 45-54, 2003.

HAHN, B.; VALENTINE, D. **Essential MATLAB for engineers and scientists**. 4. ed. - Academic Press, 2010.

HUANG, S.; TIAN, N.; WANG, Y.; JI, Z. **Multi-objective flexible job-shop scheduling problem using modified discrete particle swarm optimization**. SpringerPlus, v. 5, n. 1, p. 1432, 2016.

HUANG, R.; YU, T. **An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting**. Applied Soft Computing, v. 57, p. 642-656, 2017.

KACEM, I.; HAMMADI, S.; BORNE, P. **Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems**. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, v. 32, n. 1, p. 1-13, 2002a.

KACEM, I.; HAMMADI, S.; BORNE, P. **Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic**. Mathematics and Computers in Simulation, v. 60, p. 245-276, 2002b.

LI, J.; PAN, Q.; LIANG, Y. **An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems**. Computers & Industrial Engineering, v. 59, n. 4, p. 647-662, 2010.

LI, J.; PAN, Q.; XIE, S. **An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems**. Applied Mathematics and Computation, v. 218, n. 18, p. 9353-9371, 2012.

LI, L.; KEQI, W.; CHUNNAN, Z. **An improved ant colony algorithm combined with particle swarm optimization algorithm for multi-objective flexible job shop scheduling problem**. In: Machine Vision and Human-Machine Interface (MVHI), International Conference on. IEEE, p. 88-91, 2010.

LIOUANE, N.; SAAD, I.; HAMMADI, S.; BORNE, P. **Ant systems & local search optimization for flexible job shop scheduling production**. International Journal of Computers Communications & Control, v. 2, n. 2, p. 174-184, 2007.

LUGER, G. F. **Artificial Intelligence:** Structures and Strategies for Complex Problem Solving. 6. ed. - United States of America: Pearson Education, 2008.

MANIEZZO, V. **Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem.** INFORMS Journal on Computing, 11(4), p. 358-369, 1999.

MATHWORKS. **Object-Oriented Programming in MATLAB**. Disponível em: <https://www.mathworks.com/discovery/object-oriented-programming.html> Acesso em: 13 abr. 2018.

NETO, A. J. S.; BECCENERI, J. C. **Técnicas de Inteligência Computacional Inspiradas na Natureza – Aplicação em Problemas Inversos em Transferência Radiativa.** Sociedade Brasileira de Matemática aplicada e Computacional (SBMAC), v. 41, 2009.

SAIDI-MEHRABAD, M.; DEHNAVI-ARANI, S.; EVAZABADIAN, F.; MAHMOODIAN, V. **An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs**. Computers & Industrial Engineering, v. 86, p. 2-13, 2015.

SLACK, N.; CHAMBERS, S.; JOHNSTON, R. **Administração da produção**. 1. ed. - São Paulo: Editora Atlas, 2002.

STONE, P.; BROOKS, R.; BRYNJOLFSSON, E.; CALO, R.; ETZIONI, O.; HAGER, G.; LEYTON-BROWN, K. **Artificial intelligence and life in 2030**. One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel, 2016.

STUTZLE, T.; Hoos, H. H. **The MAX-MIN Ant System and local search for the traveling salesman problem.** In T. Back, Z. Michalewicz, X. Yao (Eds.), Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), p. 309–314. Piscataway, NJ, IEEE Press, 1997.

TUBINO, D. F. **Planejamento e Controle da Produção**: teoria e prática**.** 1. ed. - São Paulo: Atlas, 2008.

VIANA, F. A. C.; KOTINDA, G. I.; RADE D. A.; STEFFEN JR, V., **Tuning dynamic vibration absorbers by using ant colony optimization.** Computer and Structures, vol. 86, p. 1539-1549, 2008.

VOLLMANN, T. E.; BERRY, W. L.; WHYBARK, D. C.; **Manufacturing Planning and Control Systems.** 4 ed. - Boston: Irwin McGraw-Hill, 1997.

WANG, L.; CAI, J.; LI, M; LIU, Z. **Flexible job shop scheduling problem using an improved ant colony optimization**. Scientific Programming, v. 2017, 2017.

WANG, X.; GAO L.; ZHANG C.; SHAO, X. **A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem**. The International Journal of Advanced Manufacturing Technology, v. 51, n. 5-8, p. 757-767, 2010.

WILLIEM, R. S.; SETIAWAN, K. **Reinforcement learning combined with radial basis function neural network to solve Job-Shop scheduling problem**. In: Business Innovation and Technology Management (APBITM), 2011 IEEE International Summer Conference of Asia Pacific. IEEE, p. 29-32, 2011.

WU, J.; WU, G. D.; WANG, J. J. **Flexible Job-Shop Scheduling Problem Based On Hybrid ACO Algorithm**. International Journal of Simulation Modelling (IJSIMM), v. 16, n. 3, 2017.

XIA, W.; WU, Z. **An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems**. Computers & Industrial Engineering, v. 48, n. 2, p. 409-425, 2005.

XING, L.; CHEN, Y.; WANG, P.; ZHAO, Q.; XIONG, J. **A knowledge-based ant colony optimization for flexible job shop scheduling problems**. Applied Soft Computing, v. 10, n. 3, p. 888-896, 2010.

XING, L.; CHEN, Y.; YANG, K. **An efficient search method for multi-objective flexible job shop scheduling problems**. Journal of Intelligent Manufacturing, v. 20, n. 3, p. 283-293, 2009.

XUE, H.; ZHANG, P.; WEI, S.; YANG, L. **An Improved Immune Algorithm for Multi-objective Flexible Job-shop Scheduling**. JNW, v. 9, n. 10, p. 2843-2850, 2014.

XUESONG, J.; QIAOYUN, T. **Multi-objective flexible job shop schedule based on ant colony algorithm**. In: Distributed Computing and Applications for Business Engineering and Science (DCABES), 14th International Symposium on. IEEE, p. 70-73, 2015.

YAHYAOUI, A.; FNAIECH, N.; FNAIECH, F. **A suitable initialization procedure for speeding a neural network job-shop scheduling**. Industrial Electronics, IEEE Transactions on, v. 58, n. 3, p. 1052-1060, 2011.

ZHANG, G.; SHAO, X.; LI, P.; GAO, L. **An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem**. Computers & Industrial Engineering, v. 56, n. 4, p. 1309-1318, 2009.