Universidade Estadual de Campinas

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Departamento de Telemática

Algoritmos Bio-inspirados para Minimização do Makespan do Problema de Escalonamento de Produção

Márcia Braga de Carvalho

Orientador: Prof.Dr. Akebo Yamakami
FEEC/UNICAMP

Co-Orientadora: Profa.Dra. Tatiane Regina Bonfim
IES2/Campinas

Tese apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para obtenção do título de **Doutor em Engenharia Elétrica** - Área de Concentração: Automação.

Banca Examinadora:

Profa. Dra. Berenice Camargo Damasceno	.....	UNESP-Ilha Solteira
Profa. Dra. Isamara Carvalho Alves	.....	IM/UFBA
Prof. Dr. Romis Ribeiro de Faissol Attux	.....	DCA/FEEC/UNICAMP
Prof. Dr. Takaaki Ohishi	.....	DENSIS/FEEC/UNICAMP
Prof. Dr. Akebo Yamakami	.....	DT/FEEC/UNICAMP (Presidente)

Campinas, SP

Setembro de 2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

C253a

Carvalho, Márcia Braga de
Algoritmos bio-inspirados para minimização do makespan do problema de escalonamento de produção / Márcia Braga de Carvalho. --Campinas, SP: [s.n.], 2011.

Orientadores: Akebo Yamakami, Tatiane Regina Bonfim.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Algoritmo da formiga. 2. Métodos bio-inspirados. 3. Escalonamento de produção. 4. Algoritmo genético. 5. Números fuzzy. I. Yamakami, Akebo. II. Bonfim, Tatiane Regina. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Bio-inspired algorithms for minimizing the makespan of the production scheduling problem
Palavras-chave em Inglês: Ant Algorithm, Bio-inspired methods, Scheduling production, Genetic algorithm, Fuzzy numbers
Área de concentração: Automação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Berenice Camargo Damasceno, Isamara Carvalho Alves, Romis Ribeiro de Faissol Attux, Takaaki Ohishi
Data da defesa: 30-09-2011
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidata: Márcia Braga de Carvalho

Data da Defesa: 30 de setembro de 2011

Título da Tese: "Algoritmos Bio-Inspirados para Minimização do Makespan do Problema de Escalonamento de Produção"

Prof. Dr. Akebo Yamakami (Presidente): _____

Profa. Dra. Berenice Camargo Damasceno: _____

Profa. Dra. Isamara Carvalho Alves: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

Prof. Dr. Takaaki Ohishi: _____

Resumo

Este trabalho propõe novas abordagens híbridas baseadas em técnicas da computação bio-inspirada para o problema de escalonamento do tipo *Job Shop*. Como o problema do tipo *job shop* pertence a classe *NP*-difícil e não existe algoritmo exato capaz de solucionar todos os tipos deste problema. Normalmente é necessária a elaboração de métodos de resolução mais sofisticados para contornar essa alta complexidade. Desta forma, nesta tese propomos abordagens híbridas baseadas em algoritmo memético e algoritmo de otimização por colônia de formigas a fim de contornar essa complexidade e ser capaz de explorar eficientemente o espaço de busca obtendo resultados de alta qualidade. Os algoritmos híbridos propostos são aplicados tanto no problema de *job shop* com tempo de processamento preciso, como nos problemas de *job shop* com tempo de processamento incerto. No caso de problema com tempo de processamento incerto, os algoritmos visam encontrar um conjunto diversificado de escalonamentos com alto grau de possibilidade de serem ótimos.

Palavras-chave: Escalonamento *Job Shop*, Algoritmos Bio-inspirados, Sistema de Colônia de Formigas, Algoritmo Genético, Números *Fuzzy*.

Abstract

This work proposes new hybrid approaches based on techniques of bio-inspired computing for the Job Shop scheduling problem. As the job shop scheduling problem is *NP*-hard and there is no exact algorithm capable of solving all kinds of this problem. Usually it is necessary to elaborate more sophisticated methods of resolution to overcome this high complexity. Thus, in this work we propose hybrid approaches based on memetic algorithm and ant colony optimization algorithm in order to explore the search space in an efficient manner and obtain high quality results. The proposed hybrid algorithms are applied in both the job shop scheduling problem with precise processing time, as in job shop scheduling problems with uncertain processing time. In the case of problem with uncertain processing time, the algorithms obtain a diversified set of schedules with high possibility of being optimal.

Keywords: Job Shop Scheduling, Bio-inspired Algorithms, Ant Colony System, Genetic Algorithm, Fuzzy Numbers.

Dedicatória

Aos meus amores Maurício e Maurício Benjamin, marido e filho que embora ainda dentro do meu ventre me faz muito feliz.

À minha avó Margarida Neres Braga (in memorian) e à minha tia Joelma Souza Braga (in memorian), que embora não estejam mais presentes, ficariam muito orgulhosas por esta conquista.

Agradecimentos

Agradeço à Deus em primeiro lugar, por todas as conquistas e oportunidades.

Ao meu orientador, o professor Akebo Yamakami, por toda confiança, pela orientação, oportunidade e motivação nos momentos de dificuldades. Para mim foi um imenso prazer trabalhar com você.

À Tatiane pela valiosa co-orientação, pelas correções e motivação.

Ao meu esposo Maurício, meu grande amor. Sem você certamente não chegaria até aqui. Sua presença em minha vida é mais um presente de Deus.

À Deus mais uma vez pela benção que me concedeu ao final desta etapa, me concedendo o tão sonhado e já amado filho Maurício Benjamin.

À minha querida mãe, Jussara, por todo seu apoio e companheirismo. A sua criação e educação me fizeram chegar até aqui.

À toda minha família Braga, por sempre me apoiarem e torcerem muito por mim.

À todos os amigos da FEEC, especialmente as amigas que participaram diretamente comigo desta jornada: Juliana Verga, Priscila e Jussara.

Às minhas amigas da Bahia que sempre torceram por mim: Gleice, Jucy, Juliana e Carla.

Aos professores da banca: Berenice Camargo Damasceno, Isamara Carvalho Alves, Romis Ribeiro de Faissol Attux e Takaaki Ohishi, pelas correções e sugestões que levaram ao amadurecimento da tese.

À todos os colegas que prestigiaram minha defesa, especialmente Benito, Diego e Juliana.

À todos os professores e funcionários da Faculdade de Engenharia Elétrica - UNICAMP.

À todos os colegas do DQE/UESB, Campus Jequié.

À CAPES e à FAPESP, pelo apoio financeiro para realização deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Um problema de escalonamento trata da alocação de recursos, levando-se em consideração a execução de um conjunto de operações, com o intuito de alcançar um determinado objetivo, sujeito a um conjunto de restrições (Bonfim, 2006). O problema de escalonamento de tarefas do tipo *job shop* consiste de um conjunto de n tarefas que precisam ser processadas em um conjunto de m máquinas. Neste problema, cada tarefa consiste de uma sequência de operações que especificam a ordem das máquinas pelas quais as tarefas deverão ser processadas. Cada tarefa é composta por uma lista ordenada de operações contendo a máquina que irá processá-la e o tempo de processamento. Cada operação precisa ser processada por uma determinada máquina durante um período de tempo, sem interrupção. O problema tem como objetivo encontrar uma forma de ordenar as tarefas nas máquinas de tal forma que minimize o tempo total de atraso (*makespan*) entre o início da primeira tarefa e o fim da última tarefa.

O problema de escalonamento do tipo *job shop* pertence à classe *NP*-difícil (Lenstra and Rinooy Kan, 1977 e Gary and Johnson, 1979), ou seja, não é conhecido nenhum algoritmo polinomial capaz de resolver todos os tipos deste problema. Como alternativa, métodos heurísticos e metaheurísticos são estudados para uma resolução aproximada deste problema, visto que o tempo para encontrar uma solução ótima por meio de um método exato pode ser inviável.

Problemas de otimização têm como objetivo minimizar ou maximizar uma função em um domínio restrito. Para que a resolução de problemas deste tipo seja possível, eles devem ser modelados matematicamente, isto é, deve ser construída uma representação matemática do problema captando toda sua essência. Diversos problemas reais podem ser considerados como problemas de otimização, tais como o famoso problema do caixeiro viajante, que consiste do descobrimento de rotas entre cidades, a melhor forma de empacotar objetos, o escalonamento de tarefas, entre outros. Resolver tais problemas pode ser considerado como uma busca pela melhor solução em um espaço de soluções potenciais.

Para espaços de busca pequenos, os métodos clássicos exaustivos são usualmente suficientes. No entanto, para espaços de busca extensos, técnicas da computação bio-inspirada são recomendadas, já que tais problemas são de difícil resolução por possuírem uma quantidade alta de soluções, o que torna inviável a sua simples enumeração. Dentre os algoritmos pertencentes a esta classe, podemos citar os algoritmos genéticos e algoritmos de

otimização por colônia de formigas.

Os algoritmos genéticos são algoritmos populacionais baseados na evolução natural das espécies. Neste contexto, os indivíduos (soluções de um problema) são submetidos a processos de reprodução, mutação, competição e seleção. A reprodução permite a transmissão de informações genéticas dos pais para os filhos, enquanto a mutação modifica parte genética dos indivíduos com o objetivo de aumentar a diversidade da população. Esse mecanismo é repetido por várias gerações, fazendo com que, a cada geração, indivíduos mais aptos tendam a ser selecionados para fazerem parte da próxima geração.

O algoritmo de otimização por colônia de formigas foi proposto no início da década de 90 por Marco Dorigo (Dorigo, 1991), e baseou-se nas atividades cotidianas das colônias de formigas reais. Foi observado que as formigas, ao realizarem suas atividades rotineiras (buscar comida para o formigueiro), são capazes de encontrar um caminho mínimo entre o formigueiro e a comida. Isso se deve ao fato de que, apesar de as formigas realizarem a busca pela comida de forma independente, elas contam com um nível de comunicação direta ou indireta, de forma que elas possuem informações globais para melhorar seus próprios caminhos. Essa forma de comunicação se dá através de uma substância química conhecida como feromônio. À medida que as formigas se movimentam, elas liberam o feromônio. Quanto maior a quantidade de formigas utilizando o mesmo caminho, maior a quantidade de feromônio neste caminho, consequentemente, mais atrativo este caminho se torna. Desta forma, o método de otimização por colônia de formigas foi inicialmente modelado no âmbito do conhecido Problema do Caixeiro Viajante (Dorigo and Gambardella, 1997), em que possíveis caminhos são discretizados em forma de grafo e o objetivo é encontrar o menor caminho, de ida e volta, entre dois pontos.

Neste trabalho, abordamos o problema de escalonamento de tarefas do tipo *job shop* em dois contextos, o primeiro considerando parâmetros precisos e o segundo considerando parâmetros *fuzzy*. Em ambos os contextos, foi utilizada uma representação por meio de grafos. Esta representação fornece uma modelagem prática e consistente do problema, facilitando a implementação dos algoritmos que auxiliam na obtenção de soluções.

Vários estudos podem ser encontrados com relação ao problema de escalonamento *job shop* com parâmetros precisos (Blazewicz et al. 1996, Fisher and Thompson 1963, Yamada and Nakano 1997, Udomsakdigool and Kachitvichyanukul 2008, etc). Na resolução do problema de escalonamento *job shop* com parâmetros precisos, foram implementadas quatro abordagens híbridas, sendo que a população inicial é gerada pelo algoritmo de colônia de formigas e, para evoluir a população, foi utilizado o algoritmo genético com busca local, que é conhecido como algoritmo memético. Para calcular o *makespan* do problema, utilizamos o algoritmo clássico de Ford-Moore-Bellman adaptado.

Em situações reais, é comum encontrarmos incertezas nas informações associadas. Por exemplo, se falarmos sobre a temperatura de um copo para uma pessoa, a temperatura poderá estar extremamente agradável para uma pessoa e, para a outra, muito quente ou fria. Isso vai depender de ponto de vista do observador. Desta forma, as tomadas de decisões baseiam-se no conhecimento individual (Gomide and Pedrycz 1998).

Ao lidar com dados incertos, uma informação deixa de ser representada por um único valor e passa a ser representada por um conjunto. Assim, o uso da teoria clássica dos

conjuntos torna-se inviável devido à sua ineficiência no tratamento de informações imprecisas. Entretanto, essas incertezas podem ser modeladas e tratadas de forma mais robusta utilizando a teoria dos conjuntos *fuzzy* (Zadeh 1965, Gomide and Pedrycz 1998 e Dubois and Prade 1980).

O marco inicial da teoria *fuzzy* foi o artigo *Fuzzy Sets*, publicado em 1965 pelo matemático Lotfi Asker Zadeh (Zadeh, 1965), com a principal intenção de dar tratamento matemático a certos termos linguísticos subjetivos. Esse foi o primeiro passo no sentido de se programar e armazenar conceitos vagos em computadores, tornando possível cálculos com informações imprecisas.

Na resolução do problema de escalonamento do tipo *job shop* com parâmetros *fuzzy*, foi considerado que os tempos de processamento das operações não são precisamente conhecidos e são representados por números triangulares *fuzzy*. Foram implementadas quatro abordagens híbridas para resolver o problema *fuzzy* com o objetivo de encontrar um conjunto de escalonamentos com alto grau de possibilidade de serem ótimos. Por se tratar de um problema no qual os escalonamentos possuem tempo de processamento *fuzzy*, foi utilizado o método de rankeamento de números *fuzzy* para definir qual o escalonamento ótimo, e para avaliar o grau de otimalidade do conjunto de escalonamentos, foi utilizada a teoria da possibilidade. O método de rankeamento é importante e pode ser utilizado em situações nas quais se tem vários escalonamentos e se quer definir qual é o melhor entre eles, segundo o critério do decisor. Como os tempos de processamento das tarefas são números *fuzzy*, calcular o *makespan* deste problema torna-se ainda mais complicado. Desta forma, nesta tese, adaptamos o algoritmo proposto por Hernandes (2007), que é baseado no clássico de Ford-Moore-Bellman, para calcular o *makespan* do problema *fuzzy*.

1.1 Objetivos

Esta tese tem como objetivo principal elaborar novas abordagens híbridas para a resolução do problema de sequenciamento de tarefas *job shop*, utilizando as metaheurísticas de otimização por colônia de formigas e algoritmos genéticos com busca local, mais conhecidos como algoritmos meméticos.

Estudamos o problema do tipo *job shop* em dois contextos. No primeiro o tempo de processamento das tarefas é representado por números *crisp* e no segundo, o tempo de processamento das tarefas é representado por números *fuzzy*. No primeiro caso, desenvolvemos quatro algoritmos híbridos cujo objetivo é encontrar bons escalonamentos para o problema. Para calcular o *makespan* do problema, utilizamos o algoritmo clássico de Ford-Moore-Bellman adaptado. Seu funcionamento será descrito no Algoritmo 4.

No segundo caso, desenvolvemos quatro algoritmos híbridos, porém como o problema é incerto, o objetivo principal é encontrar não apenas um escalonamento ótimo, mas um conjunto de escalonamentos com alto grau de possibilidade de serem ótimos, visando contornar a questão da complexidade. Em Hernandes (2007), é proposto um algoritmo baseado no método clássico de Ford-Moore-Bellman para calcular o caminho mínimo em grafos *crisp* com parâmetros *fuzzy*. Como o objetivo do nosso problema é encontrar o caminho crítico em grafos com parâmetros *fuzzy*, e não existe algoritmo exato para

resolver este tipo de problema, fizemos algumas modificações no algoritmo proposto por Hernandes (2007) para encontrar o *makespan* do problema *job shop fuzzy* (ver Subseção 6.4.1 do Capítulo 6).

1.2 Organização da Tese

Esta tese está dividida em oito capítulos, cujo são estudados e apresentados os resultados do problema de escalonamento do tipo *job shop* tanto considerando o tempo de processamento das tarefas como números precisos, quanto considerando como números *fuzzy*.

Nos Capítulos 2 e 3, são apresentados respectivamente, os conceitos básicos da teoria dos conjuntos *fuzzy* e dos algoritmos bio-inspirados utilizados nesta tese.

O problema de escalonamento do tipo *job shop* com tempo de processamento preciso é apresentado no Capítulo 4. Neste capítulo, são apresentados também as abordagens híbridas propostas para resolver este problema. Os resultados obtidos por cada uma das abordagens são apresentados no Capítulo 5.

No Capítulo 6, é apresentado o problema de *job shop* com tempo de processamento incerto. Nele são descritas as abordagens híbridas desenvolvidas para resolver o problema *fuzzy*, os operadores utilizados, bem como o funcionamento do algoritmo para calcular o *makespan*. Os resultados obtidos com a aplicação das abordagens propostas são apresentados no Capítulo 7.

Finalmente, as conclusões e propostas de trabalhos futuros são apresentadas no Capítulo 8.

Capítulo 2

Conceitos da teoria dos conjuntos *fuzzy*

A teoria dos conjuntos *fuzzy* foi introduzida pelo matemático Lotfi A. Zadeh em 1965 quando ele trabalhava com problemas de classificação de conjuntos que não possuíam fronteiras bem definidas. Sua principal intenção era dar um tratamento matemático a certos termos linguísticos subjetivos, como "aproximadamente", "em torno de", dentre outros.

O termo *fuzzy* significa nebuloso, difuso, e se refere ao fato de, em muitos casos, não conhecermos completamente os sistemas que estamos analisando. No cotidiano, isso ocorre na descrição de situações rotineiras, como: o dia está "parcialmente" nublado, preciso perder "alguns" quilos.

Em muitos problemas de física e matemática, não temos dificuldade em classificar elementos como pertencentes ou não a um dado conjunto clássico. Por exemplo, afirmamos com certeza que o número 7 pertence ao conjunto dos números naturais e que o número -7 não pertence a este mesmo conjunto. No entanto, podemos discordar quanto ao fato de o número $6,5$ pertencer ou não ao conjunto dos números aproximadamente iguais a 7. Neste caso, a resposta não é única e objetiva: pertencer ou não poderá depender do tipo de problema que se está analisando.

Neste capítulo, são introduzidos alguns conceitos básicos sobre a teoria *fuzzy*. Para maior aprofundamento, recomendamos ao leitor consultar Pedrycz e Gomide (1998) e Dubois e Prade (1995).

Este capítulo está organizado da seguinte forma: a Seção 2.1 apresenta a definição de conjunto *fuzzy* e algumas operações. Na Seção 2.2, é definido o modelo de número *fuzzy* que foi utilizado nesta tese e também algumas operações entre esses números. Na Seção 2.3, é definida a relação de ordem utilizada nesta tese. Na Seção 2.4, é apresentado o método de comparação entre números *fuzzy*. A teoria de possibilidade é apresentada na Seção 2.5. Finalmemte, na Seção 2.6, é apresentado o método de *defuzzificação* utilizado.

2.1 Conjuntos *fuzzy*

A teoria dos conjuntos *fuzzy*, introduzida por Zadeh em 1965, pode ser vista como uma generalização da teoria de conjuntos clássicos, possibilitando atribuir um grau de

pertinência real para cada elemento. Portanto, a função de pertinência do conjunto *fuzzy* mapeia cada elemento do universo de discurso a seu espaço de escala.

A extensão sugerida por Zadeh (1965) possibilita não simplesmente que um elemento pertença ou não a um conjunto, como acontece na teoria clássica, mas que um elemento possa pertencer a um conjunto com um grau de pertinência que varia no intervalo $[0, 1]$, onde o valor 0 indica uma completa exclusão, o valor 1 representa completa pertinência e os valores deste intervalo representam graus intermediários de pertinência do elemento com relação ao conjunto. A função que define os graus de pertinência dos elementos, denominada função de pertinência, é uma generalização da função característica da teoria clássica, uma vez que esta associa, para todo elemento do universo de discurso, um valor do intervalo $[0, 1]$, ao invés do conjunto de apenas dois elementos $\{0, 1\}$. Desta forma, temos a definição abaixo.

Definição 2.1 *Um conjunto fuzzy $\tilde{A}$ é caracterizado por uma função de pertinência $\mu_{\tilde{A}}(x)$, representada pela Equação 2.1, que mapeia os elementos x em um domínio X no intervalo [0,1].*

$$\mu_{\tilde{A}}(x) : X \rightarrow [0, 1] \tag{2.1}$$

De acordo com a Equação 2.1, sendo $\tilde{A}$ um conjunto *fuzzy* definido no universo de discurso X, cada elemento $x \in X$ receberá um grau de pertinência com relação a ele através da função de pertinência $\mu_{\tilde{A}}(x)$. Quanto mais perto de 1 for o valor retornado por $\mu_{\tilde{A}}(x)$, com maior certeza o elemento x pertencerá ao subconjunto $\tilde{A}$. Sendo assim, a função de pertinência vai indicar o quanto é possível um elemento $x \in X$ pertencer ao subconjunto $\tilde{A}$.

Por exemplo, considere o conjunto das mulheres altas $M = \{x \in \Re | x \geq 1,75\}$. De acordo com a teoria clássica uma mulher com 1,74m não pode ser considerada alta, ao passo que uma mulher com 1,76m é seguramente alta (Figura 2.1).

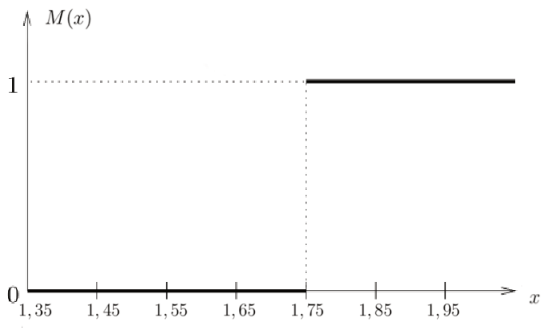

Figura 2.1: Exemplo de uma função característica clássica

Já um subconjunto *fuzzy* $\tilde{A}$ em X é definido por uma função de pertinência $\mu_{\tilde{A}}$ que associa a cada ponto de X um número real no intervalo $[0, 1]$, com o valor de $\mu_{\tilde{A}}$ em x representando o grau de pertinência de x em $\tilde{A}$. Então, quanto mais próximo o valor da função característica estiver da unidade, maior será o grau de pertinência de x relativamente a $\tilde{A}$ (Figura 2.2).

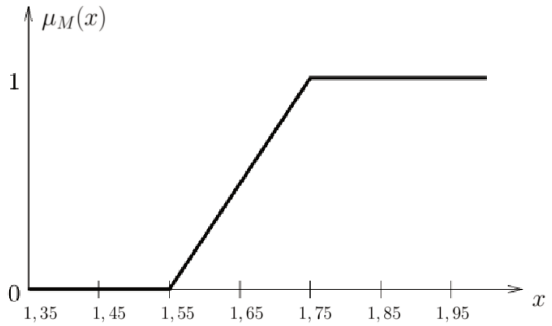

Figura 2.2: Exemplo de uma função de pertinência

Definição 2.2 *O conjunto de α-cortes de um conjunto fuzzy $\tilde{A}$, denotado por $\tilde{A}_\alpha$, é definido por $\tilde{A}_\alpha = \{x \in X | \mu_{\tilde{A}}(x) \geq \alpha\}$, tal que $\alpha \in [0, 1]$ (Figura 2.3).*

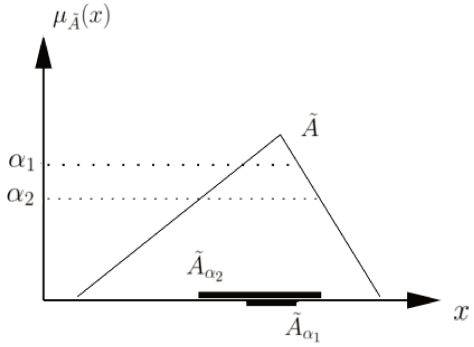

Figura 2.3: Exemplo de α-cortes

Definição 2.3 *Um conjunto fuzzy $\tilde{A}$ é convexo se sua função de pertinência é tal que:*

$$\mu_{\tilde{A}}[\lambda x_1 + (1 - \lambda)x_2] \geq min[\mu_{\tilde{A}}(x_1), \mu_{\tilde{A}}(x_2)]$$

para quaisquer $x_1, x_2 \in X e \lambda \in [0, 1]$.

Definição 2.4 *Seja X o conjunto universo. Um conjunto fuzzy $\tilde{A}$ é normal se e somente se $\exists x \in X$ tal que $\mu_{\tilde{A}}(x) = 1$.*

Caso não exista um valor x tal que o valor supremo (altura) da função de pertinência seja igual a um ($sup(\mu_{\tilde{A}}(x)) \neq 1, \forall x \in A$), então $\tilde{A}$ é um conjunto subnormal.

Definição 2.5 *O suporte de um conjunto fuzzy $\tilde{A}$ é definido por:*

$$supp(\tilde{A}) = \{x \in X | \mu_{\tilde{A}}(x) > 0\}$$

ou seja, o suporte é formado pelos elementos que possuem graus de pertinência não-nulos.

Definição 2.6 *O núcleo de um conjunto fuzzy $\tilde{A}$ é definido por:*

$$nucleo(\tilde{A}) = \{x \in X | \mu_{\tilde{A}}(x) = 1\}$$

Definição 2.7 *Dados dois números fuzzy $\tilde{A}^1$ e $\tilde{A}^2$, a altura de sua intersecção é dado por $hgt(\tilde{A}^1 \cap \tilde{A}^2)$.*

2.1.1 Operações com conjuntos *fuzzy*

A união, a intersecção e o complemento são operações essenciais realizadas em conjuntos clássicos. Com base nisto, Zadeh (1965) definiu, a partir da função de pertinência, estas operações para conjuntos *fuzzy*. A Figura 2.4 ilustra estas operações nos conjuntos *fuzzy*.

Definição 2.8 *A união de dois conjuntos fuzzy $\tilde{A}$ e $\tilde{B}$ é dada da seguinte forma:*

$$\mu_{(\tilde{A} \cup \tilde{B})}(x) = max[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)] = \mu_{\tilde{A}}(x) \vee \mu_{\tilde{B}}(x)$$

Definição 2.9 *A intersecção de dois conjuntos fuzzy $\tilde{A}$ e $\tilde{B}$ é dada da seguinte forma:*

$$\mu_{(\tilde{A} \cap \tilde{B})}(x) = min[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)] = \mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(x)$$

Definição 2.10 *O complemento de um conjunto fuzzy $\tilde{A}$ é dado da seguinte forma:*

$$\mu_{\tilde{\bar{A}}}(x) = 1 - \mu_{\tilde{A}}(x)$$

A intersecção e a união podem ser identificadas pela conjunção (E) e pela disjunção (OU), respectivamente. Assim, estas operações podem ser representadas pelos operadores $\wedge$ e $\vee$ (Gomide e Pedrycz, 1998).

2.2 Números *fuzzy*

Um número *fuzzy* expressa a incerteza relacionada ao parâmetro modelado por este número.

Definição 2.11 *Um número fuzzy $\tilde{A}$ é um conjunto fuzzy no espaço dos números reais R com função de pertinência $\mu_{\tilde{A}} : \Re \rightarrow [0, 1]$.*

Definição 2.12 *Um número fuzzy $\tilde{A}$ é positivo se $\mu_{\tilde{A}}(x) = 0$ para $x \leqslant 0$.*

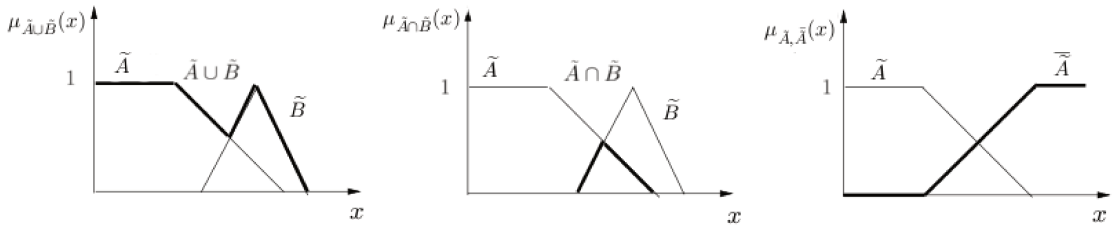

Figura 2.4: Operações com conjuntos *fuzzy*

2.2.1 Número triangular *fuzzy*

A lógica *fuzzy* é um meio de aproximar a precisão matemática clássica e a *imprecisão* do mundo real. A teoria *fuzzy* consegue manipular e operar quantidades exatas e inexatas (quantificadas por meio de valores linguísticos). Existem vários tipos de números *fuzzy* (Gomide e Pedrycz 1998, Dubois et al. 1995), mas nesta tese, vamos tratar apenas do número triangular *fuzzy*.

Definição 2.13 *Um número triangular fuzzy é um conjunto fuzzy convexo e normal cuja função de pertinência é representada por uma função contínua, onde um único elemento possui grau de pertinência igual a um ($\mu_{\tilde{A}}(x) = 1$ para exatamente um único $x \in X$).*

Definição 2.14 *Um número fuzzy denotado por $\tilde{A} = (a_i, a_m, a_s)$ é um número triangular fuzzy se sua função de pertinência, $\mu_{\tilde{A}}(x)$, for definida da seguinte forma.*

$$\mu_{\tilde{A}}(x) = \begin{cases} 0, & \text{se } x < a_i \\ \frac{x-a_i}{a_m-a_i}, & \text{se } a_i \leqslant x \leqslant a_m \\ \frac{x-a_s}{a_m-a_s}, & \text{se } a_m < x \leqslant a_s \\ 0, & \text{se } a_s < x \end{cases} \tag{2.2}$$

tal que:

a_m: valor modal;
a_i: valor inferior;
a_s: valor superior.

A representação gráfica de um número triangular *fuzzy* é apresentada na Figura 2.5.

Definição 2.15 *Valor modal é o valor $x \in [a_i, a_s]$ para o qual a função de pertinência tem valor máximo.*

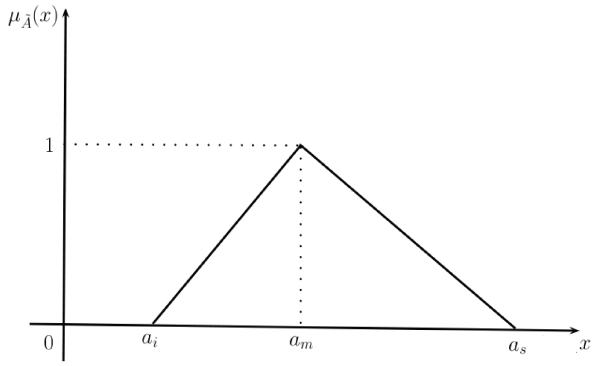

Figura 2.5: Número triangular *fuzzy*

2.2.2 Operações com números triangulares *fuzzy*

Nesta seção, são definidas algumas operações com números triangulares *fuzzy* que foram utilizadas neste trabalho.

Definição 2.16 *Sejam $\tilde{a}$ e $\tilde{b}$ dois números triangulares fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e $\tilde{b} = (b_i, b_m, b_s)$. Então, a soma fuzzy é denotada por:*

$$\tilde{a} + \tilde{b} = (a_i, a_m, a_s) \oplus (b_i, b_m, b_s) = (a_i + b_i, a_m + b_m, a_s + b_s)$$

Definição 2.17 *Sejam $\tilde{a}$ e $\tilde{b}$ dois números triangulares fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e $\tilde{b} = (b_i, b_m, b_s)$. Então, a subtração fuzzy é denotada por:*

$$\tilde{a} - \tilde{b} = (a_i, a_m, a_s) \ominus (b_i, b_m, b_s) = (a_i - b_s, a_m - b_m, a_s - b_i)$$

Definição 2.18 *Sejam $\tilde{a}$ e $\tilde{b}$ dois números triangulares fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e $\tilde{b} = (b_i, b_m, b_s)$. Então, a multiplicação fuzzy é denotada por:*

$$\tilde{a} \otimes \tilde{b} = [Min\{a_i b_i, a_i b_s, a_s b_i, a_s b_s\}, a_m b_m, Max\{a_i b_i, a_i b_s, a_s b_i, a_s b_s\}]$$

Definição 2.19 *Sejam $\tilde{a}$ um número triangular fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e k um número ordinário. Então, a operação de multiplicação é denotada por:*

$$\tilde{a} \otimes k = [Min\{a_i k, a_s k\}, a_m k, Max\{a_i k, a_s k\}]$$

Definição 2.20 *Sejam $\tilde{a}$ e $\tilde{b}$ dois números triangulares fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e $\tilde{b} = (b_i, b_m, b_s)$. Então, divisão fuzzy é denotada por:*

$$\frac{\tilde{a}}{\tilde{b}} = \left[Min\left\{\frac{a_i}{b_i}, \frac{a_i}{b_s}, \frac{a_s}{b_i}, \frac{a_s}{b_s}\right\}, \frac{a_m}{b_m}, Max\left\{\frac{a_i}{b_i}, \frac{a_i}{b_s}, \frac{a_s}{b_i}, \frac{a_s}{b_s}\right\}\right]$$

Definição 2.21 *Sejam $\tilde{a}$ e $\tilde{b}$ dois números triangulares fuzzy, $\tilde{a} = (a_i, a_m, a_s)$ e $\tilde{b} = (b_i, b_m, b_s)$. Então, a operação inversa fuzzy é denotada por:*

$$\tilde{a}^{-1} = \left[Min\left\{\frac{1}{a_i}, \frac{1}{a_s}\right\}, \frac{1}{a_m}, Max\left\{\frac{1}{a_i}, \frac{1}{a_s}\right\}\right], (note\ que \quad \frac{\tilde{a}}{\tilde{b}} = \tilde{a}.\tilde{b}^{-1})$$

2.3 Relação de ordem

No Capítulo 6 é abordado o problema de escalonamento tipo *job shop* com tempo de processamento incerto e, para determinar o *makespan* deste problema é utilizado o algoritmo proposto por Hernandes (2007) com algumas modificações. Este algoritmo tem como finalidade encontrar todos os caminhos não-dominados entre o nó origem e o nó destino, e utiliza a relação de ordem proposta por Okada e Soper (2000) (necessária para fazer a comparação entre números *fuzzy*). No algoritmo proposto, são utilizados números triangulares *fuzzy*, e é determinado o seguinte critério de dominância parcial *fuzzy* (Okada e Soper 2000):

Definição 2.22 *(Critério de dominância fuzzy) Sejam* $\tilde{a} = (a_i, a_m, a_s)$ *e* $\tilde{b} = (b_i, b_m, b_s)$ *dois números triangulares fuzzy. Então,* $\tilde{a} \prec \tilde{b}$ *(*$\tilde{a}$ *domina* $\tilde{b}$*) se e somente se* $a_m \leqslant b_m$, $a_i \leqslant b_i$, $a_s \leqslant b_s$ *e* $\tilde{a} \neq \tilde{b}$.

Definição 2.23 *Sejam um número triangular fuzzy* $\tilde{a} = (a_i, a_m, a_s)$ *e um escalar* $\varepsilon \in [0, 1]$. *O conjunto* $\{x \in \Re | \mu_{\tilde{a}}(x) \geq \varepsilon\}$ *é um intervalo fechado, denotado por* $[(a_i)_\varepsilon, (a_s)_\varepsilon]$, *e chamado de* ε*-corte de um número triangular fuzzy* $\tilde{a}$ *para* $\varepsilon > 0$. *O zero-corte é definido como* $[a_i, a_s]$.

Definição 2.24 *(Dominância parcial fuzzy) Sejam* $\tilde{a} = (a_i, a_m, a_s)$ *e* $\tilde{b} = (b_i, b_m, b_s)$ *dois números triangulares fuzzy e* $\varepsilon \in [0, 1]$. *Então,* $\tilde{a}$ *domina* $\tilde{b}$ *com grau* ε, *denotado por* $\tilde{a} \prec_\varepsilon \tilde{b}$, *se e somente se* $a_m \leqslant b_m$, $(a_i)_\varepsilon \leqslant (b_i)_\varepsilon$, $(a_s)_\varepsilon \leqslant (b_s)_\varepsilon$ *e* $\tilde{a} \neq \tilde{b}$.

A Figura 2.6 exemplifica a Definição 2.24. Para valores maiores ou iguais a ε, $\tilde{b}$ domina $\tilde{a}$ e para valores menores que ε nenhum dos dois números domina o outro.

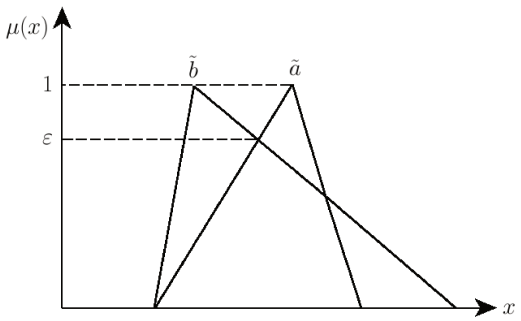

Figura 2.6: Exemplo de dominância parcial

2.4 Comparação de números triangulares *fuzzy*

A comparação entre números triangulares *fuzzy* é um recurso utilizado no desenvolvimento dos algoritmos propostos nesta tese, pois é necessário, entre vários números *fuzzy*

definir qual é o maior ou menor. O método de comparação utilizado (Bortolan e Degani 1993) consiste em definir um número real que represente o número triangular *fuzzy* (NTF). A seguir, são apresentados os 3 critérios de comparação de números *fuzzy* utilizados para ordenar os números triangulares *fuzzy*.

Primeiramente, tenta-se ordenar os números de acordo com o primeiro critério de ordenação. Se este não propiciar uma única ordem linear, o segundo critério é utilizado. Se este também não for suficiente, utiliza-se então o terceiro critério a fim de se obter uma sequência ordenada.

1. Primeiro critério de ordenação

   Seja $\tilde{a} = (a_i, a_m, a_s)$ um número triangular *fuzzy*. Este primeiro critério consiste em definir um número real que represente o número triangular *fuzzy* correspondente. Este número é calculado conforme a Equação 2.3 e ilustrado na Figura 2.7.

$$Cr_1(\tilde{a}) = \frac{a_i + 2a_m + a_s}{4} \tag{2.3}$$

   O NTF que possuir o maior Cr_1 será considerado maior.

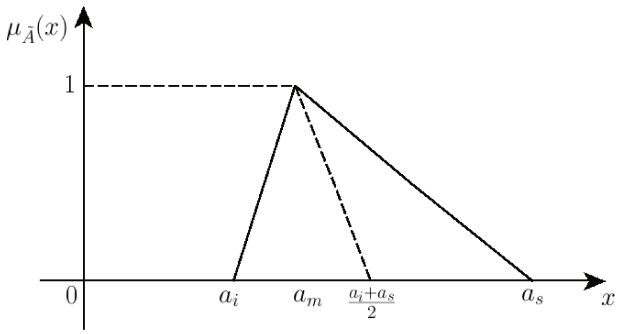

Figura 2.7: Maior número real associado à $\tilde{a}$

2. Segundo critério de ordenação

   Este segundo critério utiliza como parâmetro o valor modal do número triangular *fuzzy* $\tilde{a}$. Este número é calculado conforme a Equação 2.4.

$$Cr_2(\tilde{a}) = a_m \tag{2.4}$$

   O NTF que possuir maior valor de Cr_2 será considerado maior, desde que possuam o mesmo valor para Cr_1 .

3. Terceiro critério de ordenação

   Se dois números triangulares *fuzzy* possuírem o mesmo valor de Cr_1 e Cr_2 então, utiliza-se o terceiro critério de ordenação, dado pela Equação 2.5.

$$Cr_3(\tilde{a}) = a_s - a_i \tag{2.5}$$

Os números triangulares *fuzzy* são ordenados em ordem decrescente ou crescente, de acordo com Cr_3.

2.5 Teoria da Possibilidade

Seja $G = (N, A)$ um grafo com custo associado $\tilde{c} \in \Re^n$. Sejam dois números *fuzzy* $\tilde{T}^1$ e $\tilde{T}^2$, $\tilde{T}^1 \neq \tilde{T}^2$. Podemos dizer que $\tilde{T}^1$ tem um grau de possibilidade de ser menor que $\tilde{T}^2$ dado por (Okada 2001).

$$\tilde{w} = Poss(\sum_{ij \in \tilde{T}^1} \tilde{c}_{ij} \leqslant \sum_{ij \in \tilde{T}^2} \tilde{c}_{ij}) = sup \min_{u \leqslant v} \{\mu_{\tilde{T}^1}(u), \mu_{\tilde{T}^1}(v)\}$$

onde:

- Poss é a medida de Possibilidade;
- $\mu_{\tilde{T}^1}$ e $\mu_{\tilde{T}^2}$ são as funções de pertinência dos custos de $\tilde{T}^1$ e $\tilde{T}^2$, respectivamente;
- e *sup* min é o valor supremo do mínimo, ou seja, o maior grau de pertinência que pode ser obtido do conjunto resultante da intersecção das funções de pertinência $\mu_{\tilde{T}^1}$ e $\mu_{\tilde{T}^2}$.

A Figura 2.8 representa as funções de possibilidade de $\tilde{T}^1$ e $\tilde{T}^2$.

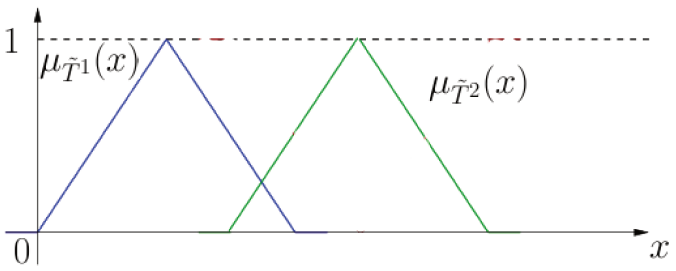

Figura 2.8: Funções de pertinência $\mu_{\tilde{T}^1}(x)$ e $\mu_{\tilde{T}^2}(x)$

Para encontrar uma solução *fuzzy* utilizando a teoria de possibilidade, é preciso encontrar todas as soluções que possuem algum grau de possibilidade de ser a solução ótima e comparar estas soluções para obter o grau de possibilidade de cada uma (Okada, 2001). O grau de possibilidade é dado pela Equação 2.6.

$$D_T = \min_{T^k \in \mathcal{T}} \{Poss(\sum_{ij \in T^k} c_{ij} \leqslant \sum_{ij \in \mathcal{T}} c_{ij})\} \tag{2.6}$$

onde $\mathcal{T}$ é o conjunto de todas as soluções.

Isso torna o problema de difícil solução, pois, além de ser preciso enumerar todas as soluções, a comparação entre elas torna o problema *NP*-Completo (Takahashi, 2004).

Pode-se ainda verificar se $\tilde{T}^2 \leq \tilde{T}^1$ de acordo com a Equação 2.7. Sejam $\tilde{T}^1 = (T_i^1, T_m^1, T_s^1)$ e $\tilde{T}^2 = (T_i^2, T_m^2, T_s^2)$, então:

$$\begin{cases} Poss(\tilde{T}^2 \leq \tilde{T}^1) = 1, & \text{se e somente se } \tilde{T}_m^1 \geq \tilde{T}_m^2 \\ Poss(\tilde{T}^2 \leq \tilde{T}^1) = 0, & \text{se } \tilde{T}_s^1 < \tilde{T}_i^2 \\ Poss(\tilde{T}^1 \leq \tilde{T}^2) = \text{hgt}(\tilde{T}^1 \cap \tilde{T}^2) \end{cases} \tag{2.7}$$

onde $\text{hgt}(\tilde{T}^1 \cap \tilde{T}^2)$ representa a altura da intersecção dos números *fuzzy* $\tilde{T}^1$ e $\tilde{T}^2$. Para compararmos os números $\tilde{T}^1$ e $\tilde{T}^2$, é preciso calcular a $Poss(\tilde{T}^2 \leq \tilde{T}^1)$ e a $Poss(\tilde{T}^1 \leq \tilde{T}^2)$. A Figura 2.9 representa o grau de possibilidade de $\tilde{T}^2$ ser menor que $\tilde{T}^1$.

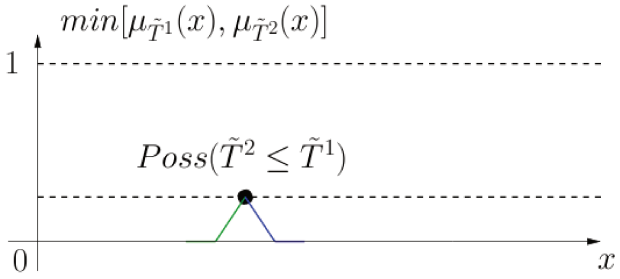

Figura 2.9: Possibilidade $Poss(\tilde{T}^2 \leq \tilde{T}^1)$

2.6 *Defuzzificação*

A *defuzzificação* é o processo inverso da *fuzzificação*, e é utilizado quando se pretende converter um resultado *fuzzy* em um valor *crisp*. Existem vários métodos de *defuzzificação* (média dos máximos, centro das somas, centróide, etc). O importante é escolher o método que melhor se adequa ao problema. Nesta tese, a técnica de *defuzzificação* utilizada foi o método de centróide (Sugeno (1985)), que também é conhecida como método do centro de gravidade. Este método é um dos mais utilizados no processo de *defuzzificação* devido à sua precisão. Ele fornece a abscissa do centro de massa associada ao gráfico da função de pertinência e a saída é o valor no universo que divide a área sob a curva da função de pertinência em duas partes iguais. O centro da área pode ser obtido pela Equação 2.8 abaixo.

$$z^* = \frac{\int_a^b \mu_{C(\tilde{z})} z dz}{\int_a^b \mu_{C(\tilde{z})} dz} \tag{2.8}$$

onde z^* representa o centro de massa da função de pertinência $\mu_{C(\tilde{z})}$ do número *fuzzy* $C(\tilde{z})$, e a e b são os intervalos de $\tilde{C}$. A Figura 2.10 ilustra a aplicação do método de centróide.

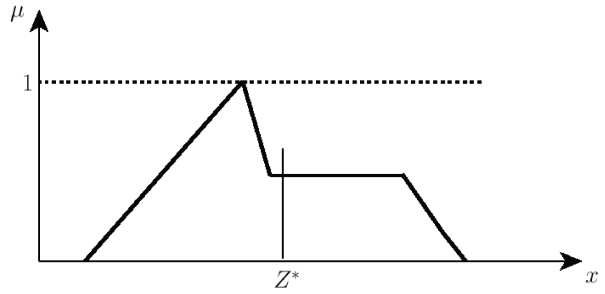

Figura 2.10: Representação gráfica de *defuzzificação* pelo método da centróide

Capítulo 3

Algoritmos bio-inspirados

Neste capítulo, são introduzidos alguns conceitos básicos dos algoritmos bio-inspirados utilizados nesta tese. Inicialmente, é apresentada a idéia básica dos algoritmos inspirados em colônia de formigas, assim como a metodologia adotada no âmbito desta tese, o Sistema de Colônia de Formigas (*Ant Colony System - ACS*). Adicionalmente, é apresentado o conceito de Algoritmo Genético e algumas definições básicas acerca deste assunto. Para um estudo mais aprofundado sobre os algoritmos genéticos, consultar Holland (1992) ou Michalewicz (1996) e para os algoritmos de otimização por colônia de formigas consultar Dorigo (1992)ou Bonabeau et al. (1999).

Este capítulo está organizado da seguinte forma: Na Seção 3.1, é apresentada um pouco da história da computação bio-inspirada. Na Seção 3.2, é apresentada a metaheurística de otimização por colônia de formigas e algumas semelhanças entre as formigas reais e as artificiais. Finalmente, na Seção 3.3, é apresentada um pouco dos algoritmos genéticos e de seus operadores.

3.1 Computação bio-inspirada

A Computação bio-inspirada estuda técnicas de computação inspiradas na biologia e tem por objetivo construir sistemas computacionais (algoritmos e ferramentas) semelhantes a seres vivos (baseado em processos naturais) para solucionar problemas complexos de otimização (Castro, 2001). Algumas de suas sub-áreas são: redes neurais artificiais, computação evolutiva por exemplo (Algoritmos Genéticos), robótica, colônias de formigas e inteligência de enxame.

Nos últimos anos, as metaheurísticas baseadas nas técnicas de computação bio-inspirada têm sido bastante utilizadas. Um dos primeiros algoritmos bio-inspirados originou-se da tentativa de entender o funcionamento do sistema nervoso (McCulloch and Pitts, 1943). Este trabalho resultou no primeiro modelo matemático do funcionamento de um neurônio biológico.

Um outro algoritmo bio-inspirado de grande importância é o algoritmo genético. Este algoritmo foi desenvolvido por Holland em 1975 com a finalidade de formalizar matematicamente e explicar os processos de adaptação em sistemas naturais para, posteriormente,

desenvolver sistemas artificiais simulados em computador que retenham os mesmos mecanismos originais encontrados em sistemas naturais. Neste trabalho, Holland (1975) utilizou o conceito de evolução das espécies de Darwin (1859), onde cada espécie evolui por meio da competição entre indivíduos pela sobrevivência e reprodução. Os indivíduos mais adaptados sobreviverão e tendem a propagar seu material genético para as próximas gerações. Durante a reprodução, ocorre um processo denominado *crossover* por meio do qual parte do material genético do pai combina-se com parte do material genético da mãe e como resultado é gerado o novo indivíduo filho que terá material genético de ambos os indivíduos pais. Este novo indivíduo filho terá maiores chances de ser melhor que seus pais ou pelo menos, melhor que a média da população devido a *pressão seletiva*.

O Sistema baseado em colônia de formigas foi proposto por Dorigo (1992), surgindo da observação de que as formigas são capazes de encontrar um menor caminho entre a colônia e a fonte de alimento. O primeiro algoritmo bio-inspirado em colônias de formigas para problemas de otimização combinatória foi o denominado *Ant System - AS* e foi aplicado inicialmente no famoso problema do caixeiro viajante. Posteriormente Dorigo e Gambardella (1997) propuseram o algoritmo denominado Sistema de Colônia de Formigas (*Ant Colony Systems - ACS*) para melhorar o desempenho do *AS*.

Os dois tipos de algoritmos bio-inspirados que são tratados nesta tese são os algoritmos genéticos (AG) e o de sistema de colônia de formigas (ACS). Enquanto o AG é motivado pela teoria da evolução natural em que indivíduos mais adaptados sobrevivem em detrimentos dos outros, o sistema de colônia de formigas é baseado no comportamento de colônias de formigas reais.

3.2 Otimização por colônia de formigas

As idéias utilizadas na metaheurística otimização por colônia de formigas provém da observação das colônias de formigas reais. Foi verificado que as formigas são capazes de encontrar um menor caminho entre o ninho e a fonte de alimento por meio de uma comunicação indireta, utilizando uma substância conhecida como feromônio.

Durante o processo de coleta de comida para abastecer o formigueiro, as formigas tendem a seguir um mesmo caminho com apenas algumas variações, depositando o feromônio no caminho por onde passa. Dependendo da extensão do percurso, essas formigas poderão passar rapidamente por ele e, a cada passagem, reforçar a quantidade de feromônio. Se o caminho for longo, o feromônio evaporará mais rapidamente, pois o intervalo entre as passagens das formigas aumenta. Isso faz com que esse caminho seja cada vez menos escolhido pelas formigas e assim elas tendem a explorar novos caminhos. Nesta busca por caminhos mais curtos entre o ninho e a fonte de alimento as formigas tendem a fazer sempre um caminho ótimo ou quase ótimo se comparado a todos os caminhos possíveis da rota.

A seguir serão apresentados dois experimentos com formigas reais realizados por Deneubourg et al. (1990) e Goss et al. (1989) que serviram de inspiração à criação do método de Otimização de Colônia de Formigas. Esta experiência consistiu na submissão de uma colônia de formigas *Iridomyrmex humilis* a uma fonte de alimento por meio de

dois caminhos distintos. No primeiro caso, o caminho entre o ninho e a fonte de alimento é dividido em duas partes iguais como pode ser visto na Figura 3.1 abaixo.

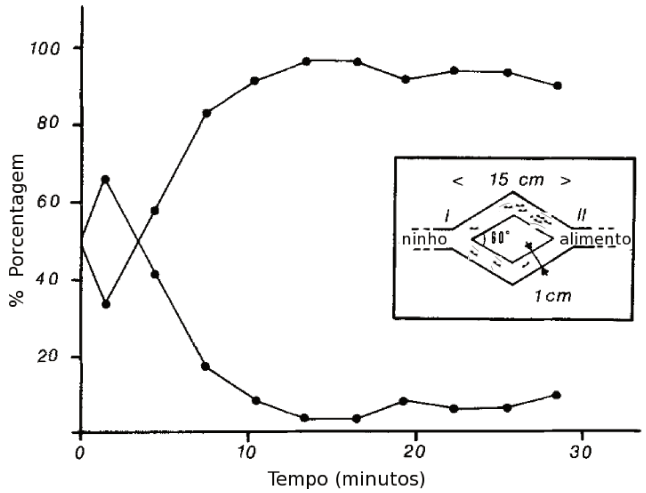

Figura 3.1: Comportamento das formigas durante alguns instantes.

Deneubourg et al. (1990) constatou que, inicialmente, as formigas procuram a fonte de alimento por meio de um caminho aleatório, mas rapidamente elas convergem para uma das pontes. Na Figura 3.1 é apresentado o percentual de formigas seguindo cada um dos caminhos em um espaço de tempo definido.

Já o segundo experimento tinha como objetivo verificar como as formigas se comportariam sujeitas a mudanças no ambiente global visto que isto normalmente ocorre entre o ninho e o formigueiro. Para isto foi desenvolvido um novo caminho entre o ninho e a fonte de alimento. Neste caminho existem duas partes mais longas e duas partes mais curtas conforme Figura 3.2. Inicialmente, quando apresentados os dois caminhos ao mesmo tempo, as formigas só podem percorrer os caminhos mais longos. Após a liberação de feromônio neste pior caminho e consequentemente fixação da trilha são liberados os caminhos mais curtos para serem percorridos.

Neste experimento Goss et al. (1989) concluiu que as formigas não trocam de caminho imediatamente quando apresentado o novo (menor) caminho, porém, quando este é encontrado ele tende a ser cada vez mais utilizado e, com o passar do tempo todas as formigas tendem a percorrê-lo. A explicação para tal comportamento está relacionada ao feromônio depositado pelas formigas durante sua locomoção. Isto ocorre porque cada formiga tende a escolher um caminho proporcionalmente à concentração de feromônio percebida. Como o menor caminho será percorrido mais rapidamente, a concentração de feromônio tende a crescer mais rapidamente que sua evaporação, atraindo as formigas para este caminho. Já em caminhos mais longos, a taxa de reposição do feromônio não é suficientemente rápida para evitar uma evaporação significativa, causando um efeito de esquecimento.

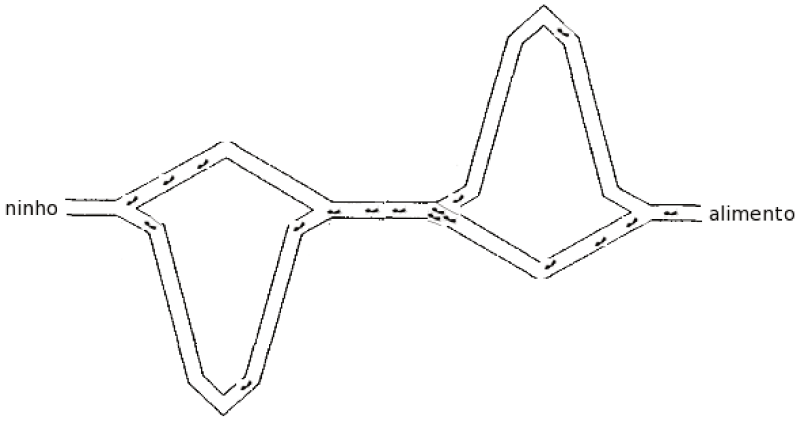

Figura 3.2: Caminho no 2^o experimento com trilhas longas e curtas.

3.2.1 Inteligência por colônia: Formigas reais e artificiais

Existem características que conferem semelhanças entre as formigas reais e as artificiais. Algumas dessas semelhanças são abordadas ao longo deste texto. Podemos citar três idéias do comportamento natural das formigas que foram transferidas para as colônias de formigas artificiais.

1. O deposito do feromônio media uma comunicação entre as formigas.
2. A preferência por trajetos com nível elevado de feromônio.
3. A taxa de crescimento mais elevado da quantidade de feromônio em alguns trajetos mais curtos.

A estas formigas artificiais foram dadas algumas potencialidades que não têm as contrapartes naturais, mas que foram observadas para serem bem sucedidas nas aplicações desta metaheurística.

As formigas artificiais são dotadas de uma memória denominada lista tabu para memorizar as rotas já visitadas. Essa memória é esvaziada no começo de cada nova viagem e é atualizada adicionando a rota inicial corrente ao conjunto das rotas já visitadas. Essa memória é muito importante no algoritmo, pois evita que uma rota seja visitada duas vezes, define o conjunto de rotas que uma formiga ainda têm para visitar e permite que a formiga calcule o comprimento do tour e retroceda o caminho para depositar o feromônio.

As formigas são insetos sociais que possuem um sistema complexo de organização e divisão de tarefas, cuja principal função é garantir a sobrevivência do formigueiro. Elas são capazes de encontrar o menor percurso entre a fonte de alimento e o ninho, sem usar sugestões visuais ou qualquer outro tipo de controle centralizado (Dorigo and Gambardella 1997).

Essas funcionalidades são obtidas por um processo de estigmergia e auto-organização em que agentes simples são capazes de emergir comportamentos complexos e direcionados

a um objetivo. Individualmente uma formiga pode não produzir boas soluções para o problema em questão, mas por meio da cooperação são capazes de encontrar um caminho ótimo ou quase ótimo.

3.2.1.1 Estigmergia

O processo de estigmergia é um processo que explica a coordenação de tarefas na reconstrução por exemplo de ninhos de cupins (para maiores detalhes consultar Bonabeau et al. 1999). Um exemplo bastante evidente da estigmergia é a busca por alimentos realizada pelas formigas. A interação dos insetos para produzir um sistema auto-organizado pode ocorrer de duas formas distintas:

1. Direta: ocorre por meio do contato táctil entre os animais por suas antenas ou mandíbulas, do contato visual ou mesmo quimicamente pelos odores de outros insetos, como no caso das formigas com o feromônio.

2. Indireta: esta forma de comunicação ocorre quando um indivíduo de uma população altera seu meio próximo, fazendo com que o ambiente local onde ele está seja modificado e que outros indivíduos que estejam no mesmo meio em um momento posterior tenham suas decisões afetadas por esta modificação individual.

3.2.1.2 Auto-organização

A auto-organização é um mecanismo dinâmico no qual uma estrutura global bem definida surge a partir de interações de componentes de baixo nível, no caso as formigas. Para que esse processo de auto-organização ocorra é necessário que algumas regras especifiquem as formas de interações dos componentes com o meio local onde estão localizados, sem referência a todo o sistema.

Este conjunto de regras pode ser resumido em quatro grupos básicos:

1. Auto-alimentação positiva (amplificação): a amplificação, que é preferência no caso das formigas dos melhores caminhos, faz com que estes se tornem caminhos preferenciais na busca.

2. Auto-alimentação negativa: utilizado para contrabalançar o efeito da alimentação positiva fazendo com que caminhos pouco utilizados sejam esquecidos com o passar do tempo.

3. Aleatoriedade: a auto-organização surge da amplificação das flutuações que permitem a localização de novas soluções superando mínimos locais.

4. Um único indivíduo pode produzir uma estrutura organizada por meio de uma trilha estável de feromônio. Entretanto esta estrutura não será otimizada. Para que um sistema auto-organizado encontre boas soluções é necessário que múltiplos indivíduos interajam entre si a partir dos seus resultados e por meio do ambiente comum.

3.2.2 ACS: Ant Colony System

A primeira metaheurística de otimização baseada em colônia de formigas foi proposta por Dorigo em 1991 e denominada Ant System - AS. Entretanto os resultados apresentados por este algoritmo não foram suficientes para que este pudesse competir com os melhores algoritmos existentes. Desta forma, Dorigo e Gambardella (1997) propuseram o algoritmo Ant Colony System - ACS como uma melhoria do algoritmo AS. O algoritmo básico de ACS está ilustrado no Algoritmo 1 e seus procedimentos internos são explicados a seguir.

Algoritmo 1 Algoritmo básico de Sistema de Colônia de Formigas.

```
enquanto não_convergir faça
  para cada formiga faça
    construir_solução (fórmula determinística)
    ou construir_solução (fórmula probabilística)
    atualização_local_feromônio()
  fim para
  atualização_global_feromônio()
  se melhor solução encontrada
  fim se
fim enquanto
```

Como pode ser visto no Algoritmo 1, ACS é uma metaheurística bem simples que consiste na execução de cinco procedimentos: um critério de convergência do algoritmo; duas funções para construção da solução, levando em conta as informações inerentes do problema e aquelas adquiridas pelos agentes (formigas); e duas funções (uma local e outra global) para atualizar o feromônio nas arestas e cumprir o papel de comunicação indireta. Duas importantes alterações com relação ao algoritmo AS devem ser consideradas:

1. Em ACS o feromônio é adicionado somente aos arcos que pertencem a melhor solução global. Já em AS esse feromônio é adicionado em todos os arcos por onde as formigas passam.

2. Em ACS cada vez que uma formiga utiliza o arco (i, j) para mover-se da cidade i para a cidade j, remove-se uma quantidade de feromônio desse arco. Já em AS esse processo de evaporação do feromônio não existe.

Este novo algoritmo ACS foi proposto na tentativa de competir com outros algoritmos existentes, sejam estes populacionais ou não, bem como sua aplicação à classes de problemas mais complexos, como por exemplo problemas envolvendo mais de um objetivo. No geral os algoritmos de otimização por colônia de formigas foram introduzidos para resolver problemas de otimização discreta. Por isso, a maioria deles emprega uma representação do problema em forma de um grafo $G = (N, A)$, no qual V é o conjunto dos nós ou vértices do grafo e A é o conjunto de arcos ou arestas do grafo. No caso do problema de escalonamento *job shop*, o custo de cada arco corresponde ao tempo de processamento de cada tarefa.

Baseado nesta representação em grafo do problema de escalonamento *job shop* (JSSP), o algoritmo ACS constrói uma solução fazendo com que um conjunto de agentes móveis, denominados *formigas*, caminhe do nó inicial do grafo até o nó final do grafo satisfazendo as restrições do problema. Antes que cada formiga inicie seu percurso, cada arco (i, j) do grafo é definido por uma variável τ_{ij}, conhecida como trilha artificial de feromônio. Inicialmente este τ_{ij} é igual para todos os arcos do grafo. Assim, para construir uma solução do JSSP colocamos uma formiga k em um nó inicial i, escolhido aleatoriamente dentre os disponíveis, e sucessivamente escolhemos o próximo nó j a ser visitado, seguindo umas das regras descritas abaixo.

O algoritmo ACS usa um critério de seleção mais agressivo, denominado por *pseudo-randômico-proporcional*. Neste critério de seleção é introduzido um parâmetro q_0, tal que $q_0 \in [0, 1]$, seja responsável por controlar a construção das soluções, de forma probabilística ou de forma determinística.

Para o critério de seleção da geração de soluções, um número aleatório q é gerado e comparado a um parâmetro q_0, sempre que a formiga necessitar escolher o próximo nó para se mover. Se o valor de q for menor ou igual ao parâmetro q_0, o algoritmo funcionará de forma determinística, ou seja, de acordo com o tempo de processamento (Equação 3.1).

$$u = argmax\{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}\}, \quad \text{se} \quad q \leqslant q_0 \tag{3.1}$$

onde η_{ij} representa o valor heurístico relacionado à natureza do problema que permite uma maior exploração. Neste caso, o tempo de processamento de cada tarefa. Desta forma, os parâmetros α e β controlam a intensidade do feromônio τ_{ij} e a qualidade da aresta η_{ij}, respectivamente.

A Equação 3.1 foi normalizada dividindo-se o produto dos termos desta equação pelo somatório de todas as probabilidades possíveis. Assim obteve-se uma fórmula probabilística na qual cada formiga k constrói o seu caminho (*scheduling*) movendo-se por meio de uma sequência de locais vizinhos, onde os movimentos são selecionados seguindo a distribuição de probabilidades (Equação 3.2). Esta equação probabilística é utilizada sempre que o valor de q for maior que q_0.

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k}[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}, \quad q > q_0 \tag{3.2}$$

onde J_i^k representa a lista de nós não visitados pela formiga k que se encontra atualmente no nó i. Podemos observar que, se $\alpha = 0$, as formigas seguirão a heurística do vizinho mais próximo para dar o próximo passo, enquanto que se $\beta = 0$, as formigas selecionarão um caminho com maior nível de feromônio e pode ocorrer uma estagnação com o passar das gerações, levando o algoritmo a pontos sub-ótimos. De acordo com a Equação 3.2, a preferência da formiga por um determinado caminho é maior para caminhos com maior nível de feromônio e com menor tempo de processamento.

O processo de depósito e evaporação do feromônio em ACS é realizado de forma local e global. Toda vez que uma formiga se move de um arco para outro, a quantidade de feromônio associada ao arco é reduzida, tornando-o menos desejável para as próximas

formigas. Esse processo de atualização e evaporação local aumenta o poder de exploração das formigas. A forma de atualização local do feromônio é dada pela Equação 3.3.

$$\tau_{ij} = (1-\rho).\tau_{ij} + \rho.\tau_0, \quad \tau_0 = (n.L_{nn})^{-1} \tag{3.3}$$

onde n é o número de tarefas, L_{nn} é um valor heurístico de *makespan* encontrado por uma formiga e ρ é um parâmetro que determina a velocidade de evaporação do feromônio. No início, τ_0 é igual em todos os arcos do grafo.

Uma vez que todas as formigas construíram um caminho (*scheduling*) e tiveram seus desempenhos avaliados, o caminho da formiga com menor valor de *makespan* terá um depósito de feromônio em cada aresta do grafo por onde ela passou, ou seja, o feromônio é adicionado apenas nos arcos associados à melhor solução global. Este processo de atualização global pretende recompensar as arestas que pertencem a rotas (*scheduling*) mais curtas. A atualização global do feromônio é feita de acordo com a Equação 3.4. Assim, a cada arco (i, j) do grafo, adiciona-se uma quantidade de feromônio proporcional à qualidade da solução, ou seja, quanto menor o valor do caminho (*makespan*), maior será a quantidade de feromônio depositada.

$$\tau_{ij} = (1-\rho).\tau_{ij} + \rho.\Delta\tau_{ij}, \quad \rho \in [0,1] \tag{3.4}$$

onde,

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{L^k}, & \text{se } (i,j) \text{ usado} \\ 0, & \text{caso contrário} \end{cases} \tag{3.5}$$

O primeiro termo das equações (Eq. 3.3 e Eq. 3.4) é responsável pela evaporação do feromônio. O parâmetro ρ é utilizado para que os caminhos que são menos utilizados sejam esquecidos com o passar do tempo. O segundo termo da Equação 3.4 é responsável por aumentar a concentração de feromônio apenas nos arcos visitados pela melhor formiga, onde L^k representa o *makespan* da melhor formiga da iteração.

Desta forma, o ACS tornou-se um excelente candidato para resolução de problemas dinâmicos, caracterizados pela possibilidade de alteração do seu espaço de busca a qualquer instante. Este método possibilita um processo de busca contínuo no espaço de busca, a partir da alteração de boas soluções já encontradas. Neste tipo de situação, é crucial que o algoritmo seja capaz de ajustar sua direção de busca, acompanhando as mudanças do problema a ser resolvido.

3.3 Algoritmos genéticos

Nesta seção são fornecidos alguns conceitos básicos sobre os algoritmos genéticos, bem como seus operadores, alguns mecanismos de seleção e sua estrutura básica. A intenção aqui não é de fornecer um texto completo sobre os algoritmos genéticos, mas apenas disponibilizar algumas informações sobre os principais conceitos utilizados nesta tese, de modo a facilitar a compreensão do método proposto. Para um estudo mais aprofundado sobre os algoritmos genéticos consultar (Holland, 1992 e Michalewicz, 1996).

3.3.1 Conceitos básicos

Os algoritmos genéticos foram introduzidos nos anos 60 por John Holland e consolidado em 1975 com a finalidade de formalizar matematicamente e explicar os processos de adaptação em sistemas naturais, para posteriormente, desenvolver sistemas artificiais simulados em computador que retenham os mesmos mecanismos originais encontrados em sistemas naturais.

A terminologia utilizada no desenvolvimento dos algoritmos genéticos foi baseada na teoria da evolução natural e da genética proposta por Darwin em 1859.

Uma das principais características dos AG's é de não encontrarem uma única solução (indivíduo), mas uma população deles, caracterizando-se como método estocástico de busca e seleção. Segundo Michalewicz (1996), para percorrer este espaço de busca é necessário manter equilíbrio entre dois pontos conflitantes:

- **Explotação:** consiste no aproveitamento das melhores soluções;
- **Exploração:** consiste na exploração do espaço de busca.

Diversos estudos comprovaram que os algoritmos genéticos mantém um equilíbrio notável nos pontos de contraste apresentados anteriormente e, portanto, conseguem fazer o aproveitamento de melhores soluções e exploração do espaço de busca. Embora sejam identificadas etapas não-determinísticas em seu desenvolvimento, os algoritmos genéticos não são métodos de busca totalmente aleatórios, e sim métodos que relacionam variações aleatórias com seleção polarizada pelos valores de adequação (*fitness*) atribuídos a cada indivíduo.

Os AG's são capazes de desenvolver soluções para problemas do mundo real, tais como problemas de busca, de otimização, entre outros problemas complexos de engenharia. Embora eles não garantam eficiência total na obtenção da solução, geralmente garantem uma boa aproximação. Eles utilizam os conceitos da evolução biológica, tais como: genes, cromossomos, cruzamento, mutação e seleção. Desta forma, as espécies evoluem aleatoriamente via operadores, estando sujeitas à seleção natural. Assim os indivíduos mais adaptados sobrevivem e se reproduzem, propagando o seu material genético para as próximas gerações.

A seguir, é apresentada uma descrição mais detalhada das partes constituintes de um algoritmo genético.

Codificação: A escolha da representação (codificação) dos cromossomos é uma etapa bastante importante na construção do algoritmo genético, pois deve permitir uma representação da solução completa do problema, sem dificultar a medida da função de *fitness* ou a aplicação dos operadores genéticos. Os algoritmos genéticos clássicos adotam a codificação binária, isto é, cada gene (bit) pode assumir valores 0 ou 1.

A motivação para o uso dessa codificação vem da teoria dos esquemas (Holand, 1992). Holland argumenta que seria benéfico para o desempenho do algoritmo maximizar o paralelismo implícito inerente ao algoritmo genético. Contudo outros tipos de codificação

podem ser utilizadas, como por exemplo, codificação inteira, real e mista.

Cromossomo: é a estrutura nucleoprotéica dentro da célula que armazena o DNA dos seres vivos. Dentro de cada cromossomo, o DNA fica como uma fita enrolada em espiral. Em algoritmos genéticos, um cromossomo haplóide (apenas uma cadeia de DNA representa o cromossomo) geralmente corresponde a uma cadeia de *bits* que representa um candidato à solução do problema (Figura 3.3).

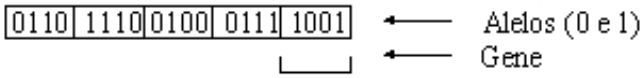

Figura 3.3: Exemplo de cromossomo com codificação binária.

Gene: blocos funcionais de DNA, os quais codificam uma proteína específica. No caso do algoritmo genético, um gene corresponde a um único *bit*, ou então a um pequeno bloco de *bits* adjacentes que codificam um elemento particular da solução candidata.

Alelo: representa uma das formas alternativas de um gene. Nos AG's representam os valores que os genes podem assumir. Por exemplo, um gene que representa o parâmetro cor de um objeto poderia ter o alelo de cor azul, preto, verde, etc.

Indivíduo: formado pelo cromossomo e sua aptidão.

População: soluções que representam os cromossomos ou conjunto de indivíduos que evoluem por meio de operadores genéticos.

Fenótipo: é a manifestação do genótipo no comportamento, fisiologia e morfologia do indivíduo, como um produto de sua interação com o ambiente.

Genótipo: Estrutura de dados que representam uma solução candidata à um determinado problema.

Função de Avaliação (*fitness*): esta função associa a cada indivíduo da população uma medida de aptidão. Ela deve ser escolhida de forma a medir o desempenho de cada indivíduo como solução do problema.

3.3.2 Operadores genéticos

Os operadores genéticos frequentemente utilizados nos AG's são o *crossover* e a mutação. Estes operadores representam o núcleo de um AG. O objetivo deles é produzir novos cromossomos que possuam propriedades genéticas superiores às encontradas nos pais. A

seguir apresentamos alguns desses operadores e seus principais aspectos.

***Crossover*:** consiste na troca aleatória do material genético entre dois cromossomos (strings - séries) selecionados. Entretanto existem alguns tipos de crossover, tais como:

- *Crossover de um ponto:* escolhe-se dois indivíduos da população (pais), seleciona-se aleatoriamente o ponto de corte e efetua-se a troca de modo a gerar dois novos indivíduos filhos (Figura 3.4).

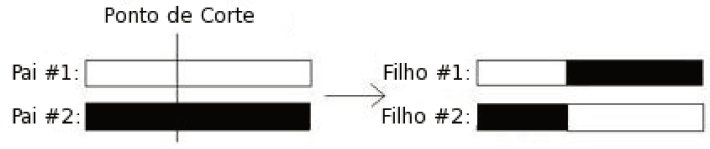

Figura 3.4: Exemplo de *crossover* de um ponto

- *Crossover de dois pontos:* seleciona-se dois indivíduos da população (pais), escolhe-se aleatoriamente dois pontos de cortes e efetua-se a troca de modo a gerar dois novos indivíduos filhos (Figura 3.5).

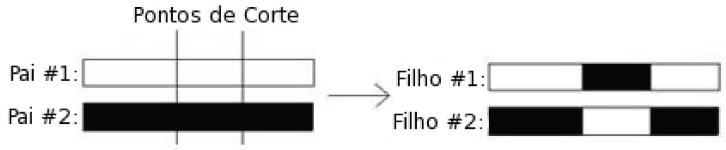

Figura 3.5: Exemplo de *crossover* de dois pontos

- *Crossover Uniforme:* para cada gene no primeiro filho é decidido (com alguma probabilidade fixa p) qual pai vai contribuir com seu valor para aquela posição (Figura 3.6).

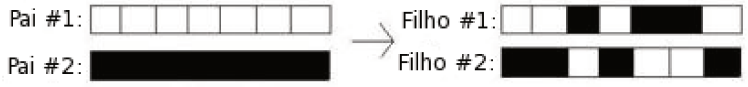

Figura 3.6: Exemplo de *crossover* Uniforme

- *Crossover OX:* escolhe-se dois indivíduos pais da população para gerar dois filhos. Primeiro copia-se um trecho de um dos indivíduos pais para um filho e o restante do cromossomo do novo indivíduo filho é preenchido com as informações do outro indivíduo pai na mesma sequência para evitar valores repetidos nos genes (Figura 3.7).

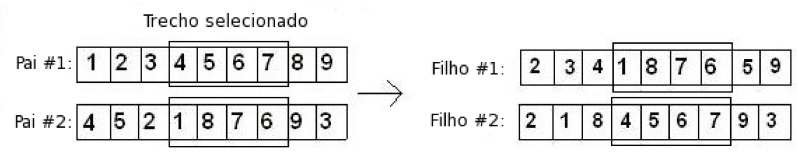

Figura 3.7: Exemplo de *crossover* OX

Mutação: é a geração de um novo indivíduo (filho) partindo de um único pai. O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene normalmente é pequena. A idéia é apenas de criar uma variabilidade extra na população, mas sem destruir o processo já obtido com a busca. Existem alguns operadores de mutação, entretanto nos restringiremos apenas aos operadores abaixo.

- *Mutação Inversiva:* é bastante utilizada e pode ser feita por meio de uma simples troca de posição entre genes de um mesmo cromossomo (Figura 3.8).

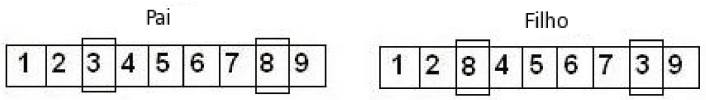

Figura 3.8: Exemplo de mutação inversiva

- *Mutação Uniforme:* é bastante utilizada em problemas com codificação em ponto flutuante assim como a mutação gaussiana. Este operador seleciona aleatoriamente um componente $k \in \{1, 2, ..., n\}$ do cromossomo $\mathbf{x} = [x_1, ..., x_n]$, e altera o gene na posição k de tal forma que o indivíduo gerado após mutação seja $\mathbf{x'} = [x_1, ..., x'_k, ...x_n]$, onde x'_k é um número aleatório (com distribuição de probabilidade uniforme) no intervalo $[x_{kmin}, x_{kmax}]$, onde x_{kmin} e x_{kmax} são, respectivamente, os limites inferior e superior do gene na posição k.

3.3.3 Mecanismos de Seleção

O processo de seleção num AG pode ser definido como sendo uma escolha probabilística de cromossomos de uma população tendo como base as suas aptidões (*fitness*), ou seja, quanto maior for a aptidão de um indivíduo, maior a chance dele ser selecionado para a próxima geração. Existem diferentes estratégias de seleção. Abaixo citaremos algumas dessas.

- *Roleta:* muito utilizado no AG clássico, esse método de seleção atribui a cada indivíduo da população uma probabilidade de passar para a próxima geração proporcional a seu valor de *fitness*, ou seja, quanto maior o *fitness* de um indivíduo, maior a probabilidade deste passar para a próxima geração. A grande desvantagem desse método é a probabilidade da perda do melhor indivíduo da população atual, ou seja, a possibilidade dele não passar para a próxima geração. A Figura 3.9 ilustra a idéia desse mecanismo de seleção.

Indivíduo	Cromossomo	Fitness	% do total
a	11001	1	10
b	10101	3	30
c	10001	2	20
d	11110	4	40

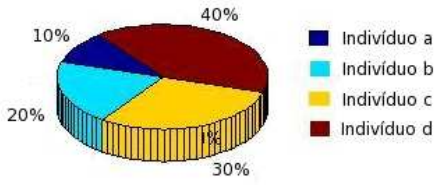

Figura 3.9: Exemplo de seleção por roleta

- *Elitista:* é selecionada uma porcentagem dos melhores indivíduos da população atual.
- *Bi-classista:* são selecionados os $P\%$ melhores indivíduos e os $(1 - P)\%$ piores indivíduos.
- *Rank:* esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com o *fitness* para determinar a probabilidade de seleção por meio de mapeamentos lineares ou não-lineares.
- *Torneio Binário:* duas soluções são indicadas aleatoriamente e a melhor é selecionada se um número aleatório $r \in [0, 1]$ for menor que o parâmetro $p \in [0, 1]$ previamente determinado; caso contrário, a outra solução é escolhida.

A estrutura de um AG é baseada nos conceitos de reprodução celular e evolução natural, podendo ser descrito em quatro elementos básicos: uma população de soluções, uma função de desempenho, operadores genéticos e processo de seleção. A estrutura de um AG básico é facilmente encontrada na literatura. No Algoritmo 2 é apresentado um pseudocódigo de um AG clássico.

Algoritmo 2 Algoritmo genético clássico

Início:
  $t = 0$;
  Inicializa_População (P, t);
  Avalia (P, t);
  enquanto $t \neq d$; % critério de parada não atingido;
    faça
      $t = t + 1$;
      seleção_dos_pais (P, t);
      reprodução (crossover) (P, t);
      mutação (P, t);
      avaliação $(P, t)'$;
  fim enquanto
Fim

Capítulo 4

O problema de escalonamento *job shop*

Problemas de escalonamento consistem em alocar uma determinada quantidade de recursos, levando em consideração um conjunto de operações sujeito a restrições e com intuito de alcançar um determinado objetivo. Estes problemas possuem inúmeras aplicações práticas, incluindo engenharia industrial, linha de produção, transporte, agendamento de pessoas, construção, entre outros. Em geral, todos estes problemas são complexos e de grande dificuldade de resolução.

O problema de escalonamento do tipo *job shop* é de considerável importância industrial, mas sua dificuldade está no elevado número de restrições, complexidade e incerteza (Lenstra et al., 1977). A complexidade é evidenciada na quantidade de produtos com diferentes gamas de processamentos e na quantidade de máquinas para processá-los. As restrições podem ser temporais e físicas. As temporais são datas de entrega, precedências entre as operações, tempos de preparação das máquinas e tempos de transporte. As físicas podem ser capacidade de recursos e capacidade do nível do estoque. A incerteza acontece quando as decisões têm que ser tomadas, muitas vezes, com informações incompletas e incertas sobre o estado futuro do ambiente de produção.

Desta forma, o problema de escalonamento do tipo *job shop* é combinatório e classificado como *NP*-completo (Blazewicz et al., 1993), sendo considerado um dos mais complexos desta classe (Yamada and Nakano, 1997). Uma evidência da dificuldade em resolver esse problema é o *benchmark* proposto por Fisher e Thompson em 1963 (Fisher and Thompson, 1963), composto por 10 máquinas e 10 tarefas, que permaneceu sem resposta por mais de 20 anos.

Neste capítulo, será abordado o problema de escalonamento *job shop* (JSSP) com parâmetros precisos. Serão apresentadas algumas modelagens matemáticas das abordagens híbridas propostas.

Este capítulo está organizado da seguinte forma: na Seção 4.1 é apresentada a definição do problema. A modelagem matemática do problema está na Seção 4.2. Na Seção 4.3, é apresentada a modelagem do problema através de grafo disjuntivo. Na Seção 4.4, são apresentados alguns métodos de resolução para o problema. A abordagem de resolução do problema e os algoritmos propostos são apresentados na Seção 4.5.

4.1 Definição do problema

O problema de escalonamento *job shop* consiste em um conjunto de J tarefas que precisam ser processadas em um conjunto de M máquinas. Cada tarefa consiste de uma sequência de operações pré-definida, que especifica a ordem das máquinas pelas quais as tarefas devem ser processadas. Os tempos de processamento das operações também são pré-definidos, ou seja, o número de unidades de tempo que uma operação leva para ser processada é conhecido (Tabela 4.1).

Tarefas	Ordem das operações (máquina, tempo de processamento)		
1	$O_{11}(1,3)$	$O_{12}(2,3)$	$O_{13}(3,3)$
2	$O_{21}(1,2)$	$O_{23}(3,3)$	$O_{22}(2,4)$
3	$O_{32}(2,3)$	$O_{31}(1,2)$	$O_{33}(3,1)$

Tabela 4.1: Instância do problema de *job shop* (3×3).

Na resolução do problema de escalonamento *job shop*, são estabelecidas as seguintes restrições:

- No início, todas as tarefas estão prontas para o processamento e todas as máquinas estão prontas para executá-las.
- As operações não podem ser interrompidas, ou seja, a operação tem de ser completada antes de se iniciar uma outra operação na mesma máquina.
- Não existe restrição de precedência entre operações de tarefas diferentes.
- Existe restrição de precedência entre operações dentro de uma mesma tarefa, sendo determinado por qual máquina a tarefa passará em cada janela de tempo.

O objetivo do problema é encontrar uma ordem de escalonar as tarefas nas máquinas, de tal maneira que se minimize o tempo total de execução (*makespan*). O termo *makespan* está relacionado com o tempo total entre o início da primeira operação, e o fim da última operação e é ele quem determina quão eficiente é a utilização dos recursos.

4.2 Modelagem matemática do *job shop*

Seja $\{J_j\}_{1 \leqslant j \leqslant n}$ um conjunto de n tarefas que devem ser processadas em um conjunto de m máquinas $\{M_k\}_{1 \leqslant k \leqslant m}$. Seja $O = \{o_{jk}\}_{1 \leqslant j \leqslant n, 1 \leqslant k \leqslant m}$ o conjunto de operações, onde o_{11} é considerada a operação inicial da tarefa J_1 na máquina M_1, e o_{nm} a operação final da tarefa J_n na máquina M_m. Neste problema, todas as tarefas possuem a mesma quantidade de operações, que é igual ao número de máquinas.

As operações estão interligadas por dois tipos de restrições. A primeira restrição é a de precedência, que significa que cada operação i só pode ser escalonada após todas as operações antecessores a ela, P_i, terem sido concluídas. A segunda restrição impõe que uma operação i só pode ser escalonada se a máquina vigente estiver livre.

Nesta modelagem, F_i representa o tempo final da operação i e um escalonamento pode ser representado por um vetor de tempos finais $(F_1, F_m, \ldots, F_n)$. Seja $A(t)$ um conjunto de operações a serem processadas em um tempo t, e seja $r_{i,k} = 1$ se a operação i requerer a máquina k para ser processada e $r_{i,k} = 0$ caso contrário.

Para cada operação i, o tempo de processamento p_i é fixado. O objetivo do problema é encontrar um escalonamento que minimize o tempo total gasto entre o início da primeira operação e o término da última operação, que representa o tempo de processamento total (*makespan*) denominado por C_{max}. A modelagem matemática para este problema é caracterizada como:

Função objetivo: Min $F_{n\times m} = C_{max}$

Sujeito a:

$$\begin{aligned} & F_k \leqslant F_i - p_i \qquad i = 1, \ldots, n; \quad k \in P_i \\ & \sum_{i \in A(t)} r_{i,k} \leqslant 1 \qquad k = 1, \ldots, m; \quad t \geq 0 \\ & F_i \geq 0 \qquad i = 1, \ldots, n. \end{aligned} \tag{4.1}$$

Nesta modelagem matemática, a primeira restrição assegura que a sequência de processamento das operações em cada tarefa corresponde a uma ordem pré-determinada. A segunda restrição assegura que existe somente uma tarefa sendo atendida por uma máquina em um determinado momento. A terceira restrição finalmente, assegura a execução de todas as operações e que o tempo final não seja negativo. Qualquer solução que atenda essas três restrições é chamada de escalonamento.

4.3 Modelagem por grafo disjuntivo

Uma abordagem gráfica de uma situação permite que ela seja mais facilmente compreendida, pois é possível prever os impactos de pequenas alterações na situação original e, além disso, as relações entre os componentes podem ser visualizados em um único contexto.

Podemos descrever o problema de *job shop* por meio de uma modelagem em grafos, conforme descrito em Balas (Balas 1969). Seja um grafo disjuntivo $G = (N, C \cup D)$ onde:

- N é o conjunto de nós que representam as operações O_{jk} de cada tarefa J_j na máquina M_k. A esse conjunto são adicionados dois nós artificiais, um representando o nó inicial e o outro o nó final. O número máximo de nós no grafo é $|N| = n.m + 2$ (onde n é o número de tarefas e m o número de máquinas);

- C é o conjunto de arcos conjuntivos (arcos que possuem mesma direção) que representam a sequência tecnológica das operações O_{jk} dentro de uma mesma tarefa;
- D é o conjunto de arcos disjuntivos (arcos que não têm direções definidas) que representam os pares de operações que devem ser realizadas na mesma máquina M_k.

Encontrar uma solução para o problema de *job shop* por meio do grafo disjuntivo é simples. Para isso basta definir as direções dos arcos disjuntivos, de tal maneira, que o grafo resultante seja acíclico.

Na Figura 4.1 apresentamos o grafo para o problema da Tabela 4.1. Nele, é possível ver o conjunto de arcos conjuntivos C representados pelas flechas contínuas e o conjunto de arcos disjuntivos D representados pelas flechas pontilhadas. Cada operação é representada por um nó e os arcos que partem de cada nó são ponderados pelo tempo de processamento da operação correspondente. Neste exemplo, todos as tarefas iniciam no instante 0.

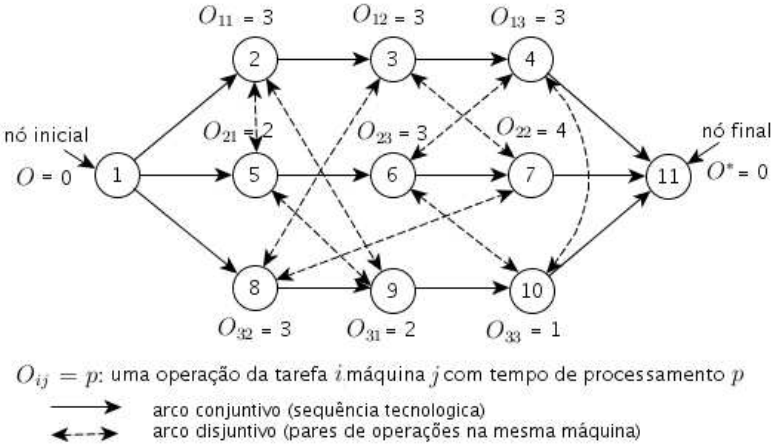

Figura 4.1: Grafo disjuntivo do JSSP ($n = 3$ e $m = 3$).

Inicialmente, o grafo disjuntivo não representa uma solução para o problema de *job shop*. Quando todas as direções dos arcos disjuntivos são definidas, obtém-se um grafo direcionado. Se o grafo resultante for acíclico, tem-se uma solução (escalonamento) factível para o problema de *job shop*. Uma solução com a definição de todas as direções nos arcos disjuntivos é representada na Figura 4.2.

Para determinar o *makespan*, tempo total de processamento, basta calcular o caminho crítico do grafo acíclico (Figura 4.2). Este caminho é determinado pela maior distância entre o nó inicial e o nó final do grafo.

4.3.1 Representação de soluções pelo diagrama de Gantt

Outra maneira de visualizar uma solução encontrada para o problema de escalonamento *job shop* é o diagrama de Gantt. Este diagrama representa a ocupação das máquinas em função do tempo. A construção deste diagrama é feita associando a cada máquina

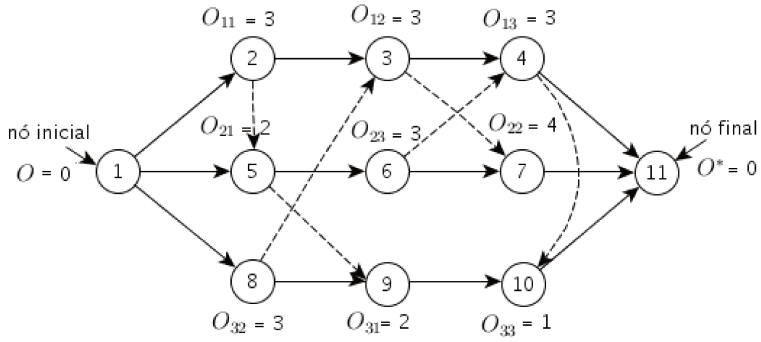

Figura 4.2: Representação de uma solução acíclica para o problema de *job shop*.

M_k uma barra horizontal, sendo as operações de cada tarefa J_j representadas por retângulos localizados nesta barra. Esses retângulos possuem um comprimento proporcional ao tempo de processamento de cada tarefa, e a posição que eles ocupam na barra é definida pelo instante de início da respectiva tarefa.

Um escalonamento para o problema da Tabela 4.1 é representado pelo diagrama de Gantt da Figura 4.3: o caminho crítico é representado pelos blocos sombreados e seu valor de *makespan* é de 12 unidades.

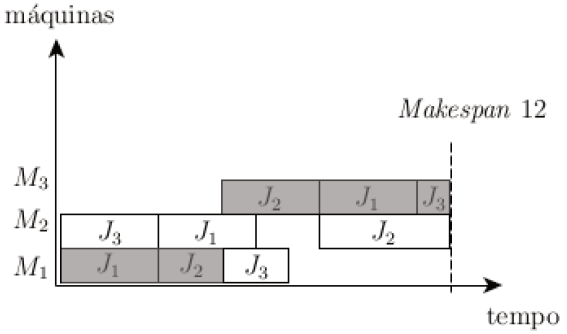

Figura 4.3: Diagrama de Gantt de um escalonamento para o problema job shop 3×3.

4.4 Métodos de resolução para o problema *job shop*

O problema de sequenciamento do tipo *job shop* é um problema prático, de considerável importância industrial e bastante estudado na literatura. Tipicamente, resolver este problema por meio de métodos exatos pode ser difícil ou custoso, sendo necessária a utilização de métodos que buscam soluções aproximadas para a obtenção de boas soluções em tempo real.

4.4.1 Métodos exatos

Métodos de resolução exatos são aqueles que pretendem obter uma solução ótima para um determinado problema. Entretanto, nem sempre é possível resolver um problema por meio destes métodos em tempo viável, principalmente se o problema for combinatório.

4.4.2 Métodos heurísticos

Os métodos de resolução heurística (aproximada) permitem obter soluções aceitáveis em tempo viável, mas não garantem a obtenção da solução ótima. Estes são os mais requisitados em problemas de otimização combinatória.

Dentre os diversos métodos aplicáveis no problema de *job shop*, os mais utilizados são os métodos de listas devido à sua simplicidade. Estes métodos são baseados em regras de despacho prioritário ou algoritmos de lista (Blazewicz et al., 2006), nos quais a solução é construída por meio de uma sequência de decisões baseadas no que localmente parece ser ótimo. Alguns algoritmos conhecidos são:

- SPT (*Shortest Processing Time*) - Seleciona a operação com menor tempo de processamento.
- LPT (*Longest Processing Time*) - Seleciona a operação com maior tempo de processamento.
- EDD (*Earliest Due-Date*) - Seleciona a operação que pertence à tarefa cuja data de entrega é a mais próxima.
- MWKR (*Most WorK Remaining*) - Seleciona a operação que pertence ao produto que tem o tempo de processamento restante maior.
- MOPNR (*Most Operations Remaining*) - Seleciona a operação que pertence ao produto que possui o maior número de operações ainda por processar.

Outro método heurístico bastante utilizado é o *Shifting Bottleneck* (Adams et al., 1988). Este é um método iterativo baseado no grafo disjuntivo. Em cada iteração, é resolvido um problema relaxado de planejamento em uma única máquina com datas de disponibilidade e de entrega, de modo a sequenciar as máquinas que formam o problema de *job shop*.

Além dos métodos mencionados acima, os algoritmos genéticos e os algoritmos de otimização por colônia de formigas têm sido aplicados com sucesso em diversos problemas combinatórios, inclusive na resolução do problema de *job shop*. Na literatura, a primeira utilização dos algoritmos genéticos na resolução do problema tipo *job shop* é reportado no trabalho de Davis em 1985. Dentre os diversos trabalhos nesta área, podemos também citar os trabalhos de (González et al., 2007; Yamada and Nakano, 2007; Udomsakdigool and Kachitvichyanukul, 2008; Lei, 2010).

4.5 Abordagens propostas

Encontrar uma solução ótima ou próxima da ótima em tempo computacional razoável para o problema de escalonamento do tipo *job shop* por meio de métodos convencionais não é nada simples, já que este problema pertence a classe *NP*-completo. Com isso, a busca por métodos que possam resolver o problema em tempo computacional viável tem sido crescente. Os métodos heurísticos e metaheurísticos, embora não garantam encontrar à solução ótima, conseguem uma boa aproximação da solução em um tempo computacional bem menor.

Os algoritmos genéticos são métodos heurísticos muito eficientes e largamente aplicados em problemas reais de otimização. Contudo, existem muitas situações onde o algoritmo genético puro não oferece bons resultados.

Algoritmos de otimização por colônia de formigas são heurísticas cuja fonte de inspiração é o comportamento cotidiano de colônias de formigas reais. Baseado na observação de que as formigas são capazes de encontrar um menor caminho entre o ninho e a fonte de alimento, esses métodos de otimização têm se mostrado eficiente na resolução de diversos problemas combinatórios (Carvalho et al., 2010; Udomsakdigool and Kachitvichyanukul, 2008).

Na literatura existem diversos métodos heurísticos que são utilizados para resolver o problema de *job shop*, contudo, devido à complexidade deste problema, nenhum único algoritmo tem se mostrado suficiente para resolvê-lo. Sendo assim, existe uma margem para buscar novos métodos híbridos que sejam capazes de resolver este problema com eficiência e em tempo computacional real.

Nesta tese foram desenvolvidos quatro algoritmos híbridos que trabalham conjuntamente com as metaheurísticas de otimização por colônia de formigas (ACS) e o algoritmo genético com busca local. Nas abordagens propostas foram desenvolvidos inicialmente um algoritmo genético diferente do AG tradicional, no qual a população inicial é gerada pelo algoritmo de otimização por colônia de formigas, o Ant Colony System (ACS), denominado AG-ACS. Foram realizados testes com o AG-ACS e propostas três técnicas de buscas locais denominadas, Troca de Tarefas Adjacentes no Caminho Crítico (CC), Troca de Tarefas na Máquina mais Ociosa (MO) e a junção destas duas técnicas (CC-MO). Um AG com busca local é mais conhecido como algoritmo memético (MA). Desta forma, por conveniência, neste trabalho os algoritmos híbridos implementados são denominados MA-ACS (CC), MA-ACS (MO), MA-ACS (CC-MO).

Nas abordagens híbridas desenvolvidas nesta tese, existe um mecanismo de cooperação e compartilhamento de soluções entre os dois algoritmos, de forma que a população inicial do algoritmo memético é gerada por meio do algoritmo de colônia de formigas, garantindo uma população inicial diversificada e com qualidade. A seguir são destacados os passos dos métodos híbridos propostos aplicados ao problema de escalonamento *job shop* com parâmetros precisos.

4.5.1 Algoritmo híbrido de sistema de colônia de formigas e genético para o problema de *job shop*

A representação da solução de um problema de otimização é uma etapa muito importante, pois está diretamente relacionada com a definição dos operadores de movimento, vizinhança, factibilidade e outros elementos fundamentais para a eficiência da busca. Tratando-se de algoritmos populacionais como o algoritmo genético e o algoritmo de sistema de colônia de formigas, é muito importante que essa representação permita uma boa exploração do espaço de busca e, principalmente, permita um bom controle de diversidade suportado por métricas bem definidas e eliminação de soluções redundantes.

Para o problema de *job shop* com parâmetros precisos, a representação foi feita por meio do grafo disjuntivo como ilustrado na Figura 4.1. Uma solução do grafo disjuntivo é dada por um vetor V, de $n.m + 2$ colunas, onde n é o número de tarefas e m o número de máquinas. Cada campo do vetor V, v_k com $1 \leqslant k \leqslant n.m + 2$, é preenchido por um número k que indica o número do nó no grafo disjuntivo. Este vetor possui uma sequência que corresponde à ordem que as tarefas são atendidas pelas máquinas. A representação do problema foi dada na forma de grafo disjuntivo devido à facilidade de manipulação do grafo, tornando a resolução do problema mais simples.

No vetor abaixo, podemos verificar um exemplo de uma solução para o problema 3×3 ($n = 3$ e $m = 3$) da Tabela 4.1. Esta solução corresponde ao escalonamento ótimo representado pelo diagrama de Gantt, apresentado na Figura 4.3.

$$V = (1 \rightarrow 2 \rightarrow 8 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 9 \rightarrow 4 \rightarrow 7 \rightarrow 10 \rightarrow 11)$$

De acordo com o vetor solução acima, o primeiro número (1) e o último número ($n.m + 2 = 11$) representam os nós artificiais do grafo disjuntivo. O segundo número representado pelo número 2, representa a primeira operação do escalonamento, que é da tarefa 1 na máquina 1. O terceiro número representado pelo número 8, representa a segunda operação do escalonamento, que é da tarefa 3 na máquina 2, e assim por diante.

4.5.1.1 Geração da população inicial

O primeiro passo na execução do algoritmo híbrido é a geração da população inicial. Esta população é gerada por meio do algoritmo de colônia de formigas, e sua representação e solução é dada da maneira descrita acima. A seguir, é explicado o funcionamento do algoritmo de sistema de colônia de formigas para o problema de *job shop* e seu pseudocódigo é apresentado no Algoritmo 3.

Algoritmo de sistema de colônia de formigas para o problema de *job shop*

Seja um grafo disjuntivo como ilustrado na Figura 4.1. Os nós artificias identificados como (1) e (11) representam o ponto de partida e chegada respectivamente de cada formiga.

O conjunto S_f das próximas operações de uma formiga f que se encontra no nó $r_{(1 \leqslant r \leqslant n.m+2)}$ não é formado por todos os nós não-visitados como acontece no ACS em

geral (Bonabeau et al. 1999), mas pelos nós que obedecem às restrições de precedência do problema. No início de uma iteração do algoritmo, S_f é inicializado para ser a primeira operação de todas as tarefas. Quando uma formiga f se move do nó r para o nó s, o nó r é deletado, mas o nó s é adicionado a S_f se a máquina estiver livre.

A regra de transição de estado da formiga f em ACS pode ser descrita como:

$$s = \begin{cases} argmax_{u \in S_f}\{[\tau(r,u)] \cdot [\eta(u)^\beta]\}, & \text{se } q \in q_0 \\ S, & \text{caso contrário} \end{cases}$$

onde S é determinado segundo a expressão:

$$P_f(r,s) = \begin{cases} \frac{[\tau(r,s)] \cdot [\eta(s)^\beta]}{\sum_{u \in S_f}[\tau(r,u)] \cdot [\eta(u)^\beta]}, & \text{se } s \in S_f \\ 0, & \text{caso contrário} \end{cases}$$

onde:

$P_f(r,s)$: probabilidade da formiga f, que se encontra no nó r, escolher o nó s como próximo a ser visitado;

$\tau(r,s)$: quantidade de feromônio existente na aresta (r,s). Inicialmente adota-se o mesmo valor τ_0 para todos os arcos;

$\eta(s) = 1/p(s)$: inverso do tempo de processamento da operação s;

β: valor heurístico que pondera a influência relativa da distância $\eta(s)$;

q: número aleatório distribuído uniformemente no intervalo $[0,1]$;

q_0: parâmetro definido pelo usuário com $0 \leqslant q_0 \leqslant 1$;

S_f: conjunto das operações que a formiga f pode visitar no próximo passo;

A atualização do feromônio é realizada de forma local e global. Enquanto realiza seu percurso, uma formiga modifica o valor do feromônio nos arcos por onde ela passa aplicando a regra de atualização local.

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \tau_0, \qquad \rho \in [0,1]$$

Onde ρ é um parâmetro que determina a evaporação do feromônio. Após todas as formigas terminarem seu percurso, o valor do feromônio é atualizado nos arcos por onde a formiga de melhor valor de *makespan* passar. Essa atualização é realizada de forma global de acordo com a regra abaixo:

$$\tau(r,s) = (1-\rho_1)\cdot\tau(r,s) + \rho_1\cdot\Delta\tau(r,s), \qquad \rho_1 \in [0,1]$$

onde:

$$\Delta\tau(r,s) = \begin{cases} (L_{max})^{-1}, & \text{se } (r,s) \in \text{melhor percurso} \\ 0, & \text{caso contrário} \end{cases}$$

L_{max} é o *makespan* do melhor escalonamento encontrado por uma formiga f ao final de uma iteração.

O pseudocódigo do algoritmo de sistema de colônia de formigas aplicado ao problema de *job shop* com parâmetros precisos é apresentado no Algoritmo 3.

4.5.1.2 Avaliação da função de fitness

O passo seguinte na execução do algoritmo é a avaliação da função de *fitness*. O critério utilizado para avaliar cada solução foi o *makespan*. Para calcular o *makespan* no problema de *job shop* com parâmetros precisos foi utilizado o algoritmo de Ford-Moore-Bellman (FBM) adaptado.

No problema estudado o objetivo é minimizar o *makespan*, que é o caminho de custo máximo entre os nó inicial e o final do grafo disjuntivo. O algoritmo clássico de Ford-Moore-Bellman consiste em encontrar um caminho de custo mínimo entre dois nós específicos do grafo e possui a vantagem de ser aplicado em grafos com arcos negativos.

Como $min\{f(x)\} \equiv -max\{-f(x)\}$ desta forma, nesta tese o algoritmo clássico de FBM foi adaptado para o problema estudado. A estrutura do algoritmo de Ford-Moore-Bellman é mantida.

Foi considerado como valor de *fitness* o valor do *makespan*. Sendo assim, a cada iteração do algoritmo de colônia de formigas a melhor solução é aquela que apresenta o menor valor de *makespan*. Toda a população é avaliada e aquele indivíduo que possuir o menor valor de *fitness* é considerado o melhor da população. O algoritmo de colônia de formigas foi utilizado para formar uma população de indivíduos. Esses indivíduos formam a população inicial do algoritmo genético. O pseudocódigo do algoritmo de Ford-Moore-Bellman (Bellman, 1958) é descrito no Algoritmo 4.

4.5.1.3 Critério de parada

Os passos seguintes, em que são executados os operadores genéticos e os métodos de busca local, são realizados enquanto o critério de parada não for satisfeito. O critério de parada utilizado no algoritmo híbrido é encontrar a solução conhecida como ótima ou o número de gerações do algoritmo memético.

4.5.1.4 Operadores genéticos

Os operadores genéticos utilizados nos algoritmos híbridos propostos nesta tese foram o *crossover* e mutação. Os operadores genéticos são aplicados no grafo disjuntivo.

Algoritmo 3 Pseudocódigo do algoritmo de colônia de formigas aplicado ao problema de escalonamento *job shop* com parâmetros precisos

Entrada:

m: número de máquinas;
n: número de tarefas;
$O_{m \times n}$: matriz com sequência de operações para cada tarefa;
$o_{ij} = (m_{ij}, p_{ij})_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m}$: par ordenado de cada operação composto pela máquina que irá processa-lá e o tempo de processamento da operação;
N_{it}: número de iterações;
N_{form}: número de formigas;
ρ: parâmetro que determina a evaporação do feromônio;
β: valor heurístico que pondera a influência relativa da distância $\eta(s)$;
ρ_1: parâmetro da atualização global do feromônio;
q_0: parâmetro definido pelo usuário com $0 \leqslant q_0 \leqslant 1$;
T_{pop}: tamanho da população.

Início do algoritmo:

Para cada arco (r, s) do grafo disjuntivo, estabeleça um nível inicial de feromônio $\tau(r, s) = \tau_0$

para $it = 1$ até $it = N_{it}$, **faça**

para $k = 1$ até $k = N_{form}$, **faça**

a primeira operação de uma formiga f é aleatória;
construa uma solução aplicando a regra de transição de estado;
calcule o valor de *makespan* L_f do caminho encontrado por cada formiga;
atualize o feromônio nas arestas por onde a formiga passou, de acordo com a regra de atualização local;

fim para

encontre o menor $L_f(t)$;

se $L_f(t) \leqslant L_{max}$ **então** $S^{max} \leftarrow S_f(t)$;

atualize o nível de feromônio no melhor escalonamento S^{max};

fim para

Saída: Escalonamento com menor valor de *makespan* encontrado por uma formiga.

Algoritmo 4 Pseudocódigo do algoritmo de Ford-Moore-Bellman

Entrada:

O: conjunto dos nós;
N_{it}: contador de iterações;
N_o: número de nós;
$C_{r,s}$: custo do arco $(-r, -s)$;
$L_s^{N_{it}}$: custo do caminho entre os nós 1 e s na iteração N_{it};
Γ_s^{-1}: conjunto dos nós predecessores de s.

Passo 1: Inicialização das variáveis

$L_1^0 = 0$;
$L_s^0 = \infty, \quad s = 2, 3 \ldots, N_o$;
$N_{it} \leftarrow 1$;

Passo 2: Determinação dos caminhos das próximas iterações

$L_1^{N_{it}} = 0$;
$\forall s \in O, \quad s = 2, 3, \ldots, N_o$, **faça**
$\quad L_s^{N_{it}} = \min(L_s^{N_{it}-1}, min_{r \in \Gamma_s^{-1}}(L_r^{N_{it}-1} + C_{rs}))$;

Passo 3: Critério de parada

se $L_s^{N_{it}} = L_s^{N_{it}-1}, \forall s \in O$
fim do se

senão, se $N_{it} \leqslant N_o - 1 \Longrightarrow$ **Passo 2**
fim do senão

Saída: Caminho crítico do escalonamento encontrado por uma formiga.

O *crossover* é realizado entre dois indivíduos pais da população gerando dois novos indivíduos filhos. Nesta operação genética é sorteada uma tarefa dos indivíduos pais e a ordem de localização que estas tarefas aparecem nos indivíduos filhos é preservada. O restante do cromossomo do filho é preenchido de acordo com a ordem que as outras tarefas aparecem no outro pai. A Figura 4.4 ilustra um exemplo do funcionamento do operador de *crossover*.

A mutação é realizada em um indivíduo pai gerando um indivíduo filho. Neste trabalho a mutação utilizada foi a denominada mutação inversiva. Nela escolhe-se aleatoriamente duas sequências de tarefas e realiza a troca entre elas. As demais tarefas do cromossomo permanecem na mesma ordem. Os operadores de *crossover* e mutação são aplicados com

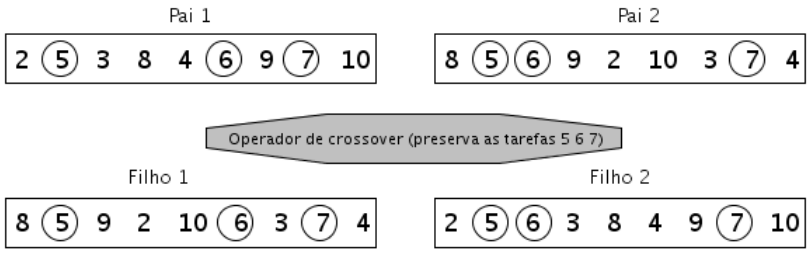

Figura 4.4: Operador de *crossover*

probabilidades P_c e P_m respectivamente, nos algoritmos híbridos. A Figura 4.5 ilustra um exemplo do funcionamento do operador de mutação.

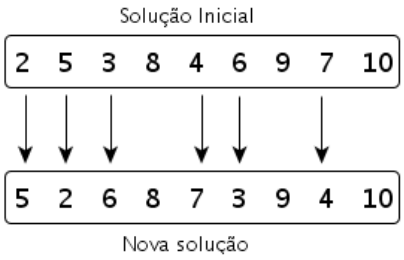

Figura 4.5: Operador de mutação: troca a sequência de tarefas [2 3 4] por [5 6 7]

4.5.1.5 Seleção

Após aplicar os operadores genéticos no algoritmo híbrido, é preciso realizar uma seleção dos melhores indivíduos. O método de seleção utilizado foi o elitista (Michalewicz 1996). Este método privilegia os indivíduos com menor valor de *fitness*. Os indivíduos mais adaptados são selecionados e preservados na população para a próxima geração.

4.5.1.6 Memória

A memória é importante quando uma carga de informações encontra-se disponível mas não é interessante levá-las ao longo de todas as iterações. No entanto, a informação não deve ser perdida pois seu conteúdo pode ser importante no futuro, sendo assim ela deve ser armazenada.

Quando a população inicial é gerada, os melhores indivíduos são guardados na memória (elitismo). Ao final de cada geração, a memória que já contém X indivíduos, recebe mais um grupo de indivíduos. Um mecanismo de atualização de memória é aplicado na população da memória, reduzindo-a novamente para X indivíduos distintos (supressão).

4.5.1.7 Métodos de busca local

Na literatura, existem diferentes regras de busca local que são adicionadas ao algoritmo genético para melhorar a qualidade das soluções do problema de *job shop*. Nesta tese os algoritmos híbridos propostos inicialmente criam uma população de soluções factíveis, ordena-a de acordo com seu valor de *fitness* e então, aplica-se os operadores de *crossover* e mutação para gerar a população da próxima geração.

O objetivo da busca local é melhorar uma solução obtida na população inicial ou pelos operadores de *crossover* e mutação. Por isso, a busca local é realizada apenas no melhor indivíduo da população sendo que, este novo indivíduo só é aceito para a próxima geração se melhorar a qualidade da solução corrente, caso contrário ele é descartado. As técnicas de busca local são discutidas a seguir.

1. *Troca de Tarefas Adjacentes no Caminho Crítico (CC)*

Nesta técnica de busca local calculamos o caminho crítico em cada solução por meio do algoritmo FBM. Em seguida, dentro deste caminho crítico, são identificados os pares de tarefas adjacentes pertencentes à mesma máquina e então é realizada a mudança de direção entre os arcos que ligam estas tarefas adjacentes.

Observe que esta mudança de direção dos arcos pertencentes ao caminho crítico de cada solução representa a mudança de direção dos arcos disjuntivos (arcos pertencentes à mesma máquina) no gráfico disjuntivo (Figura 4.1). É importante lembrar que esta mudança de direção dos arcos pode resultar em um escalonamento infactível. Por esta razão, antes de aceitar a troca dos arcos é necessário verificar a factibilidade do escalonamento. Na Figura 4.6 ilustramos um exemplo de solução para o problema da Tabela 4.1.

Seja V_1 o cromossomo que representa a solução do grafo disjuntivo (Figura 4.6 (a))

$$V_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 10 \rightarrow 11)$$

e seja cc_1 seu caminho crítico (Figura 4.6 (a)).

$$cc_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 11)$$

O *makespan* de V_1 é 15 unidades como ilustrado nos blocos críticos (blocos sombreados, Figura 4.6(b)). A busca local CC é realizada nas tarefas adjacentes pertencentes à mesma máquina, no caso os nós 5 e 2. Uma mudança de direção entre os arcos que ligam estes nós (Figura 4.6(c)) é realizada dando origem a uma nova solução V_2.

$$V_2 = (1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 10 \rightarrow 11)$$

Esta nova solução (escalonamento) tem *makespan* de 13 unidades e caminho crítico cc_2 representado pelo grafo disjuntivo (Figura 4.6(c)) e pelos blocos sombreados (Figura 4.6(d)).

$$cc_2 = (1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 11)$$

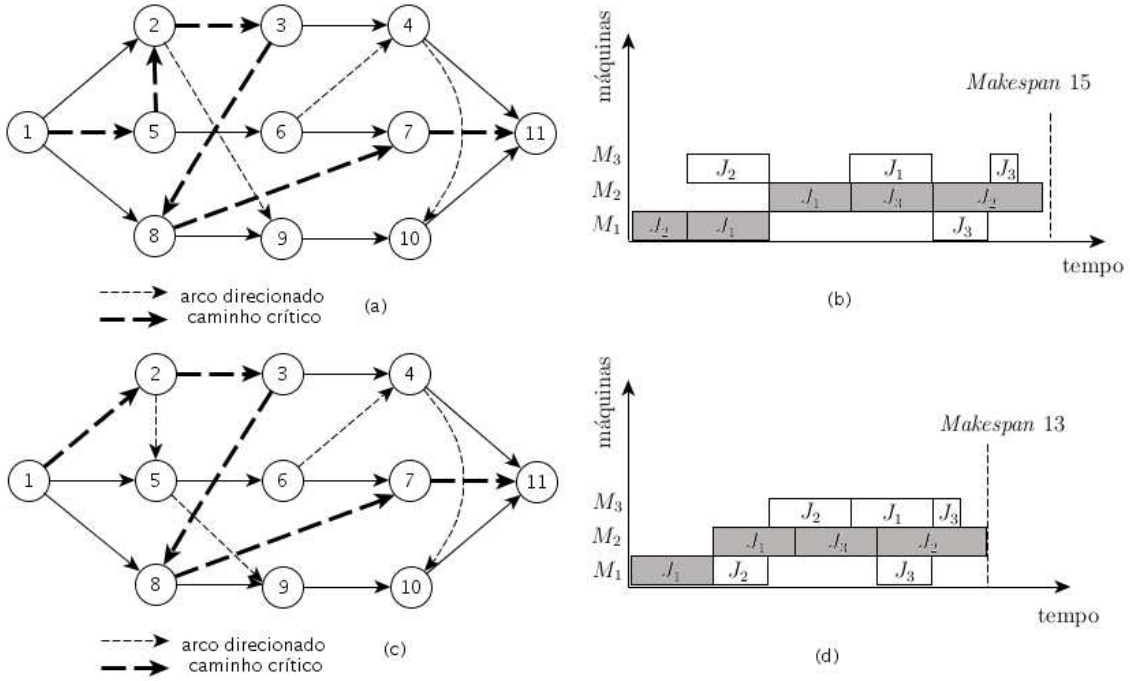

Figura 4.6: Gráfico de Gantt e grafo disjuntivo: (a) e (b) respectivamente, antes de aplicar a busca local, (c) e (d) depois de aplicar a busca local CC.

Como pode ser visto nas Figuras (Fig. 4.6(a) e Fig. 4.6(c)) a mudança de direção do arco que liga o nó 5 ao nó 2 resultou na inversão da direção entre os arcos que pertencem a máquina 1. Isto fez com que a solução resultante V_2 obtivesse um valor de *makespan* menor.

2. *Troca de Tarefa na Máquina mais Ociosa (MO)*

Resolver o problema de *job shop* geralmente gera soluções nos quais o escalonamento tem *gaps* entre as tarefas nas máquinas. Estes *gaps* surgem no escalonamento devido às restrições de precedências impostas às tarefas. Uma grande quantidade de *gaps* numa máquina faz com que ela fique ociosa, o que aumenta o valor de *makespan* do escalonamento.

A busca local MO identifica a máquina ociosa no escalonamento e a tarefa candidata que será remanejada para dentro de um *gap*, com a finaldade de obter um melhor escalonamento que reduza o valor do *makespan*. Caso a troca de tarefas adjacentes pertencentes a máquina mais ociosa não apresente nenhuma melhora no valor de *makespan*, a próxima máquina será a candidada para realizar a busca. Esta busca local de troca de tarefas adjacentes na máquina somente será permitida se o escalonamento resultante for factível. A Figura 4.7 ilustra a busca local MO.

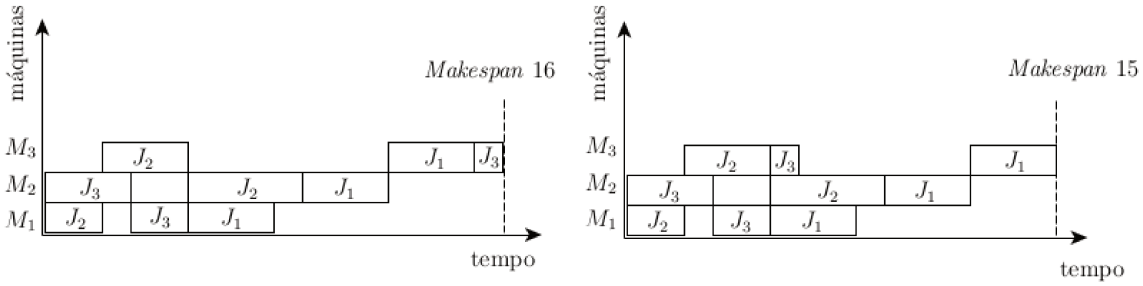

Figura 4.7: Exemplo para MO: (a) antes da busca local, (b) depois da busca MO.

No escalonamento representado pelo gráfico de Gantt(Figura 4.7(a)), a máquina ociosa é a M_3. Desta forma, a busca local MO será aplicada nesta máquina. Como J_3 é processado após J_1 nesta máquina, podemos alocar o J_3 antes do J_1 como mostra a Figura 4.7(b). Como pode ser visto na Figura 4.7 (a), a troca das tarefas na máquina mais ociosa levou a uma redução no valor do *makespan* do novo escalonamento (Figura 4.7 (b)).

3. *Junção das duas técnicas de buscas locais CC-MO*

A terceira técnica de busca local denominada CC-MO é a junção das duas técnicas de busca local descritas anteriormente. Esta técnica aplica primeiramente a busca local de troca de tarefas adjacentes no caminho critíco e, posteriormente, no cromossomo resultante aplica a busca local de troca de tarefas adjacentes na máquina mais ociosa (com a finalidade de reduzir *gaps*). A Figura 4.8 ilustra um exemplo do funcionamento da busca local CC-MO para o problema com 3 máquinas e 3 tarefas.

A Figura 4.8 (a) ilustra o exemplo de uma solução antes de aplicar a busca local. A Figura 4.8 (c) ilustra o exemplo de uma solução depois de aplicar a busca local CC no grafo da Figura 4.8 (a). E a Figura 4.8 (e) ilustra o exemplo de uma solução depois de aplicar a busca local MO no cromossomo da Figura 4.8 (c).

De acordo com a Figura 4.8 (d), depois de aplicar a busca local CC, o cromossomo resultante obteve como máquina ociosa a M_3, porém, todas as trocas de tarefas adjacentes nesta máquina geram escalonamentos infactíveis ou escalonamentos que não reduzem o valor de *makespan*. Desta forma, uma alternativa é realizar as trocas de tarefas adjacentes na próxima máquina, no caso a máquina 2 (M_2). Como pode ser visto no grafo de Gantt (Figura 4.8 (d)) a tarefa J_3 pode ser escalonada sem problemas de factibilidade antes da tarefa J_1, resultando em um novo escalonamento com valor de *makespan* menor (Figura 4.8 (f)). Para este problema de 3 máquinas e 3 tarefas, o melhor escalonamento possível (ótimo) tem *makespan* de 12 unidades.

Um pseudocódigo dos algoritmos híbridos aplicados ao problema de escalonamento *job shop* é apresentado no Algoritmo 5.

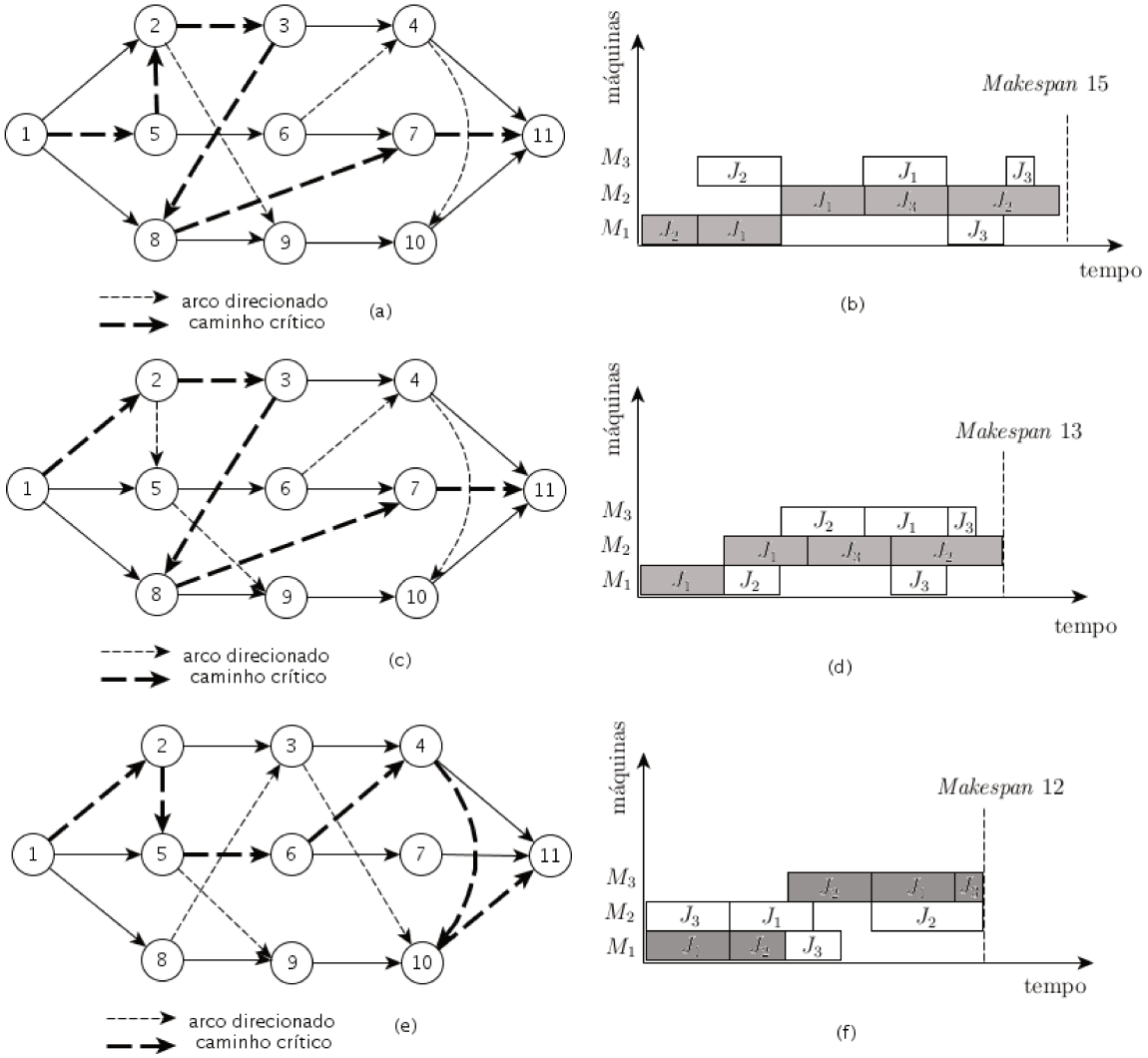

Figura 4.8: Exemplo para CC-MO: (a) antes da busca local, (c) depois da busca CC e (e) depois da busca MO.

Algoritmo 5 Pseudocódigo dos algoritmos híbridos

Entrada:

m: número de máquinas;
n: número de tarefas;
$O_{m \times n}$: matriz com sequência de operações para cada tarefa;
$o_{ij} = (m_{ij}, p_{ij})_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m}$: par ordenado de cada operação composto pela máquina que irá processa-lá e o tempo de processamento da operação;
T_{pop}: tamanho da população;
N_{it}: número de iterações;
N_{form}: número de formigas;
ρ: nível de feromônio;
β: informação heurística;
ρ_1: informação de feromônio;
q_0: informação heurística;
N_{gera}: número de gerações;
P_c: probabilidade de *crossover*;
P_m: probabilidade de mutação.

Início do algoritmo:

Gerar a população inicial por meio do algoritmo de colônia de formigas - ACS;
Calcular o valor de *makespan* de cada indivíduo;
para $z = 0$ até $z = N_{gera}$, **faça**

- Selecionar 2 indivíduos da população inicial;
- Aplicar *crossover* gerando 2 novos indivíduos para a memória;
- Selecionar 1 indivíduo da população inicial;
- Aplicar mutação gerando 1 novo indivíduo para a memória;
- calcular o valor de *makespan* da memória;
- Aplicar supressão na memória e verificar diversidade;
- Selecionar os $x\%$ melhores indivíduos da memória para a próxima geração;
- Atualizar a memória da próxima geração completando com novos indivíduos, se necessário;
- Se MA-ACS(MO), MA-ACS(CC) ou MA-ACS(CC-MO) aplicar busca local no melhor indivíduo da população e atualizar seu valor de *fitness*;
- Se AG-ACS, passo anterior não existe.

fim para

Saída: Melhor indivíduo da população.

Capítulo 5

Resultados numéricos das abordagens com parâmetros precisos

Neste capítulo, são apresentados e discutidos os resultados obtidos com os quatro algoritmos híbridos propostos, AG-ACS, MA-ACS(MO), AG-ACS(CC) e AG-ACS(CC-MO) para resolver o problema de escalonamento *job shop*. Para avaliar as abordagens propostas, foram utilizadas 13 instâncias clássicas da literatura, de diferentes autores, e com diferentes números de tarefas e máquinas, extraídas do OR-Library (Tamura, 2007). As instâncias foram assim escolhidas com a finalidade de verificar o comportamento das abordagens em diferentes tipos de problemas. Uma observação importante é que instâncias de mesmo número de máquinas e tarefas podem apresentar características diferentes.

Na literatura, o valor conhecido como *makespan* ótimo de cada instância é conhecido. Para verificar a eficiência dos algoritmos implementados, cada experimento foi executado 10 vezes por instância, sendo apresentado aqui o melhor resultado das 10 rodadas, o valor conhecido na literatura como ótimo (C_{max}), a média e o desvio padrão de cada experimento.

A Tabela 5.1 apresenta algumas informações destas instâncias, como o nome dado a cada problema, o tamanho e o nome do autor que a criou. As instâncias são caracterizadas pelas seguintes informações: número de tarefas, número de máquinas e uma tabela contendo a sequência de operações de cada tarefa, incluindo o número da máquina que irá processá-la e o tempo de processamento da operação.

5.1 Plataforma

Os quatro algoritmos foram implementados em Matlab 8. Os testes foram realizados na plataforma Linux, Intel Core 2Duo com 2.0GHz e 4Gb de memória.

5.2 Parâmetros

Os parâmetros adotados na execução dos algoritmos são apresentados a seguir.

Instância	n^o **de tarefas** $\times$ n^o **máquinas**	**Descrição do problema**
ft06	6×6	Fisher e Thompson
abz6	10×10	Adams, Balas e Zawack
orb02	10×10	Applegate e Cook
la01	10×5	Lawrence
la02	10×5	Lawrence
la06	15×5	Lawrence
la08	15×5	Lawrence
la11	20×5	Lawrence
la12	20×5	Lawrence
la16	10×10	Lawrence
la17	10×10	Lawrence
la20	10×10	Lawrence
la23	15×10	Lawrence

Tabela 5.1: Instâncias do *job shop* utilizadas neste trabalho

- Número de iterações = 100;
- Número de formigas = 15;
- $\rho_1 = 0,1$;
- $\beta = 2$;
- $\rho = 0,01$;
- $q_0 = 0,7$;
- População inicial: gerada pelo algoritmo ACS;
- Número de gerações: 1000;
- Tamanho da população: 60 indivíduos;
- Probabilidade de crossover: $0,8$;
- Probabilidade de mutação: $0,6$;
- Supressão: elimina indivíduos de mesmo *makespan*;
- Diversidade: insere novos indivíduos criados pelo ACS;
- Taxa miníma de diversidade populacional: 50%;
- Busca local: apenas no melhor indivíduo da população;
- Critério de parada: encontrar a melhor solução conhecida ou número de gerações.

5.3 Apresentação dos resultados

Nesta seção, são apresentados os resultados obtidos pelas quatro abordagens propostas quando aplicadas às 13 instâncias extraídas da literatura. Com o intuito de obter uma boa avaliação, foram utilizados diversos tipos de instâncias, tais como: instâncias de diferentes números de tarefas e máquinas e, instâncias de diferentes autores para os testes.

Os resultados são apresentados por ordem de autor. Em cada bloco de autor, é apresentada uma tabela contendo as instâncias testadas, o melhor resultado conhecido na literatura (C_{max}), o melhor resultado obtido por cada algoritmo, a média e o desvio padrão de todas as rodadas. Em negrito são apresentados os resultados das abordagens que obtiveram valor igual ao da literatura.

A análise dos resultados é apresentada na Seção 5.4. É apresentada também nesta seção, gráficos com a comparação entre os resultados obtidos pelos algoritmos e o da literatura, bem como o erro médio dos algoritmos em relação a C_{max}.

5.3.1 Instância de Adams et al. (1988)

Adams et al. (1988) propõem um método de aproximação baseado em máquina gargalo (*Shifting Bottleneeck Procedure*) e reotimização local com o objetivo de minimizar o *makespan* do problema do *job shop*. Neste trabalho os autores propõem também um conjunto de instâncias para testes.

Na Tabela 5.2, são apresentados os resultados das abordagens quando aplicadas na instância *abz6* de Adams et al. (1988). Esta instância possui 10 tarefas e 10 máquinas.

Instância	Algoritmo	Melhor Resultado	Média	Désvio Padrão
abz6 (10x10)	C_{max}	**943**		
	AG-ACS	948	962,6	8,461
	MA-ACS(MO)	948	960,5	10,977
	MA-ACS(CC)	948	959,9	9,243
	MA-ACS(CC-MO)	945	949,1	4,532

Tabela 5.2: Resultados obtidos pelos algoritmos híbridos para a instância *abz6*.

5.3.2 Instância de Applegate e Cook (1991)

No trabalho, *A computational study of the job-shop scheduling instance* (Applegate e Coock, 1991), os autores propõem uma heurística baseada em método de plano de corte e algoritmo de *Branch and Bound* com a finalidade de obter bons escalonamentos. Na literatura encontram-se disponíveis algumas de suas instâncias que são utilizadas para testes.

Na Tabela 5.3, são apresentados os resultados dos quatro algoritmos propostos quando aplicados na instância *orb02*, com 10 máquinas e 10 tarefas de Applegate e Cook (1991).

Instância	Algoritmo	Melhor Resultado	Média	Désvio Padrão
orb02 (10x10)	C_{max}	**888**		
	AG-ACS	894	904,5	7,50
	MA-ACS(MO)	893	917,2	9,91
	MA-ACS(CC)	890	910	10,72
	MA-ACS(CC-MO)	889	905,8	7,56

Tabela 5.3: Resultados obtidos pelos algoritmos híbridos para a instância *orb02*.

5.3.3 Instância de Fisher e Thompson (1963)

Um dos trabalhos pioneiros em aprendizagem probabilística na resolução do *job shop* foi o de Fisher e Thompson (1963). Nesse trabalho, eles comprovam que a combinação de regras de programação (conhecida como regra de despacho ou de prioridades) é superior a qualquer uma destas regras, se tomadas separadamente.

Na Tabela 5.4, são apresentados os resultados das quatro abordagens propostas quando aplicadas na instância *ft06*, com 6 tarefas e 6 máquinas de Fisher e Thompson (1963), juntamente com o melhor resultado conhecido da literatura (C_{max}).

Instância	Algoritmo	Melhor Resultado	Média	Désvio Padrão
ft06 (6x6)	C_{max}	**55**		
	AG-ACS	**55**	55	0,0
	MA-ACS(MO)	**55**	55	0,0
	MA-ACS(CC)	**55**	55	0,0
	MA-ACS(CC-MO)	**55**	55	0,0

Tabela 5.4: Resultados obtidos pelos algoritmos híbridos para a instância *ft06*.

5.3.4 Instâncias de Lawrence (1984)

Na literatura, podemos encontrar um conjunto com diversas instâncias para testes geradas por Lawrence (1984). Estas instâncias pertencem à classe *la*, e são amplamente utilizadas na literatura. Dentre as diversas instâncias de Lawrence (1984) disponíveis na literatura, foram escolhidas 10 instâncias de diferentes tamanhos e complexidade para testes.

A Tabela 5.5 apresenta os resultados das quatro abordagens propostas nesta tese quando aplicadas nas instâncias de Lawrence (1984), o melhor resultado conhecido na literatura (C_{max}), a média e o desvio padrão de cada experimento.

5.4 Análise dos resultados

De acordo com as Tabelas 5.2, 5.3 e 5.4 5.5 é visível a qualidade das soluções encontradas pelos algoritmos, especialmente da abordagem MA-ACS(CC-MO), já que em praticamente todos os testes esta abordagem conseguiu encontrar a mesma solução que C_{max}, ou então obteve um *makespan* muito próximo de C_{max}. A média e o desvio padrão do algoritmo MA-ACS(CC-MO) também são dados que mostram a superioridade desta abordagem.

Os algoritmos genéticos são algoritmos bio-inspirados muito utilizados na literatura para resolver uma grande variedade de problemas de otimização. Contudo, existem situações onde o algoritmo genético sozinho não oferece bons resultados. Nestes casos, as regras de busca local CC e MO melhoram a qualidade das soluções devido À reordenação das tarefas e prioridade na máquina ociosa. A combinação das buscas locais CC e MO melhoram ainda mais o algoritmo, pois além de reordenar as tarefas, ela reduz os *gaps* existentes na máquina mais ociosa.

Também podemos ver no gráfico (Figura 5.1) uma comparação dos *makespans* encontrados por cada um dos algoritmos em relação ao melhor valor disponível na literatura (C_{max}), para cada uma das instâncias testadas.

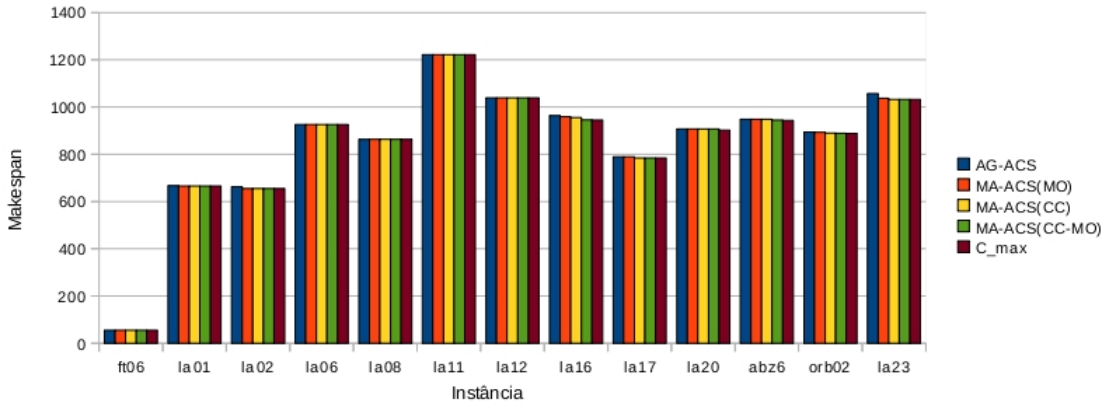

Figura 5.1: Comparação entre os melhores resultados encontrados por cada algoritmo e o da literatura.

Ainda é possível visualizar no gráfico (Figura 5.2), uma comparação entre a média dos resultados encontrados por cada algoritmo e o melhor valor disponível na literatura. Fazendo uma análise destes resultados, pode-se perceber que, para instâncias menores, as abordagens propostas obtêm uma média próxima de C_{max}, porém quando o tamanho

Instância	Algoritmo	Melhor Resultado	Média	Désvio Padrão
	C_{max}	**666**		
la01	AG-ACS	667	670,3	4,056
(10x5)	**MA-ACS(MO)**	**666**	666,6	1,075
	MA-ACS(CC)	**666**	666	0,0
	MA-ACS(CC-MO)	**666**	666	0,0
	C_{max}	**655**		
la02	AG-ACS	662	667,8	6,303
(10x5)	**MA-ACS(MO)**	**655**	661,6	5,680
	MA-ACS(CC)	**655**	660,9	6,789
	MA-ACS(CC-MO)	**655**	660,2	5,553
	C_{max}	**926**		
la06	**AG-ACS**	**926**	926	0,0
(15x5)	**MA-ACS(MO)**	**926**	926	0,0
	MA-ACS(CC)	**926**	926	0,0
	MA-ACS(CC-MO)	**926**	926	0,0
	C_{max}	**863**		
la08	**AG-ACS**	**863**	863	0,0
(15x5)	**MA-ACS(MO)**	**863**	863	0,0
	MA-ACS(CC)	**863**	863	0,0
	MA-ACS(CC-MO)	**863**	863	0,0
	C_{max}	**1222**		
la11	**AG-ACS**	**1222**	1222	0,0
(20x5)	**MA-ACS(MO)**	**1222**	1222	0,0
	MA-ACS(CC)	**1222**	1222	0,0
	MA-ACS(CC-MO)	**1222**	1222	0,0
	C_{max}	**1039**		
la12	**AG-ACS**	**1039**	1039	0,0
(20x5)	**MA-ACS(MO)**	**1039**	1039	0,0
	MA-ACS(CC)	**1039**	1039	0,0
	MA-ACS(CC-MO)	**1039**	1039	0,0
	C_{max}	**945**		
la16	AG-ACS	964	978,7	5,907
(10x10)	MA-ACS(MO)	959	979,1	7,171
	MA-ACS(CC)	956	980,6	10,243
	MA-ACS(CC-MO)	946	975,1	12,395
	C_{max}	**784**		
la17	AG-ACS	789	793,3	2,110
(10x10)	MA-ACS(MO)	789	792,2	1,686
	MA-ACS(CC)	**784**	791	3,231
	MA-ACS(CC-MO)	**784**	792	2,828
	C_{max}	**902**		
la20	AG-ACS	907	914,4	8,316
(10x10)	MA-ACS(MO)	907	913	10,677
	MA-ACS(CC)	907	911,6	9,347
	MA-ACS(CC-MO)	907	909,3	3,888
	C_{max}	**1032**		
la23	AG-ACS	1057	1080,7	16,090
(15x10)	MA-ACS(MO)	1037	1076	26,654
	MA-ACS(CC)	**1032**	1086,4	42,429
	MA-ACS(CC-MO)	**1032**	1053,7	12,238

Tabela 5.5: Resultados obtidos pelos algoritmos híbridos para as instâncias *la*.

do problema cresce como é o caso das 4 últimas instâncias do gráfico, esta média vai se distanciando deste valor. Dentre as quatro abordagens, a que obtem a melhor média dos resultados em comparação ao melhor valor da literatura é a abordagem MA-ACS(CC-MO), confirmando o bom desempenho do algoritmo.

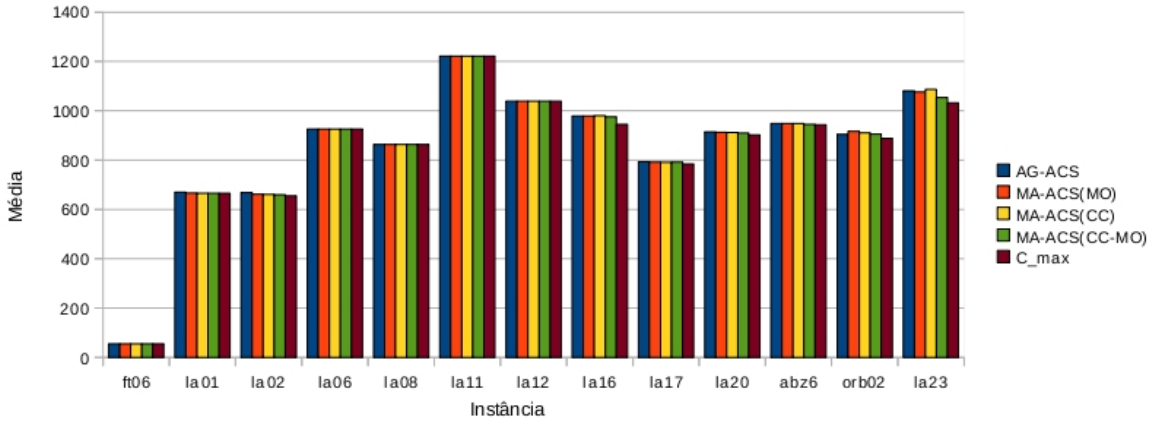

Figura 5.2: Comparação da média entre os melhores resultados encontrados por cada algoritmo e o da literatura.

Outra medida utilizada para verificar a eficiência das abordagens propostas nesta tese é o *erro médio* E_m (Equação 5.1) dos valores encontrados por cada um dos algoritmos híbridos. Este erro médio é calculado por meio da comparação do melhor valor obtido por cada algoritmo, denominado aqui (L^*), com o melhor valor conhecido na literatura (C_{max}).

$$E_m = \frac{(L^* - C_{max})}{C_{max}} \times 100 \tag{5.1}$$

Na Tabela 5.6 são exibidos os valores encontrados por cada algoritmo. Nesta tabela, também pode-se verificar a *média do erro relativo ARE* (Equação 5.2) obtida por cada algoritmo. Esta média é importante para medir o desempenho da abordagem, pois, quanto mais próximo este valor estiver de zero, melhor a sua performance.

$$ARE = \frac{\sum E_m}{N_i} \tag{5.2}$$

onde N_i representa o número total de instâncias utilizadas para testes.

Analisando os resultados da Tabela 5.6, pode-se perceber que os algoritmos propostos obtêm resultados satisfatórios, com erros médios bem pequenos. Além disso, pode-se observar que a abordagem MA-ACS(CC-MO) obteve uma média de erro relativo bem próxima de zero, comprovando a qualidade desta abordagem.

Instâncias	AG-ACS	MA-ACS(MO)	MA-ACS(CC)	MA-ACS(CC-MO)
abz6	0,53%	0,53%	0,53%	0,20%
orb02	0,67%	0,56%	0,22%	0,11%
ft06	0,0%	0,0%	0,0%	0,0%
la01	0,15%	0,0%	0,0%	0,0%
la02	1,07%	0,0%	0,0%	0,0%
la06	0,0%	0,0%	0,0%	0,0%
la08	0,0%	0,0%	0,0%	0,0%
la11	0,0%	0,0%	0,0%	0,0%
la12	0,0%	0,0%	0,0%	0,0%
la16	2,01%	1,48%	1,16%	0,10%
la17	0,64%	0,64%	0,0%	0,0%
la20	0,55%	0,55%	0,55%	0,55%
la23	2,42%	0,48%	0,0%	0,0%
ARE	0,62%	0,33%	0,19%	0,07%

Tabela 5.6: Erro médio em relação a C_{max}.

5.5 Conclusões

Embora o problema do *job shop* seja muito antigo e popular, ainda não existe um único algoritmo que possa garantir a melhor solução para todos os problemas, especialmente para problemas maiores que aparecem na literatura. A hibridização com algoritmos bio-inspirados como ACS, AG e MA está ganhando popularidade devido à sua eficiência em resolver problemas de otimização. Os resultados mostram que as abordagens propostas encontraram a solução ótima ou soluções próximas da ótima em quase todas as instâncias testadas. Desta forma, pode-se concluir que os algoritmos mostraram-se competitivos e, ao mesmo tempo promissores na resolução do problema.

Capítulo 6

O problema de escalonamento *job shop* com parâmetros *fuzzy*

Muitos métodos propostos na literatura baseiam-se na condição de que todos os dados do problema são conhecidos. Entretanto, esta suposição pode trazer dificuldades na prática, pois em problemas reais, pode haver uma incerteza em um número de fatores, como disponibilidade de equipamentos, tempo de processamento, tempo de transporte e custos. Assim, nestes casos, as soluções geradas usando modelos com incertezas certamente deixarão o problema mais próximo da realidade.

A teoria dos conjuntos *fuzzy* introduzida por Zadeh em 1965, é um meio de aproximar a precisão da matemática clássica à *inexatidão* do mundo real. Esta teoria consegue manipular e operar quantidades exatas e inexatas quantificadas por meio de valores linguísticos.

O problema de escalonamento *fuzzy* foi formulado por Dubois et al. em 1995 como um problema de otimização com satisfação de restrições *fuzzy*, no qual as restrições associadas ao problema são representadas por um conjunto *fuzzy*. No problema de escalonamento *job shop fuzzy*, os tempos de processamento das operações são imprecisos, e desta forma, a noção de escalonamento ótimo é considerada também imprecisa já que um escalonamento é avaliado através de um número *fuzzy*. Neste caso, a noção de escalonamento ótimo é avaliado seguindo um critério de ordenação entre os valores dos escalonamentos. Esse critério é importante e pode ser utilizado em situações nos quais se tem vários escalonamentos e se quer saber qual é o melhor entre eles.

Neste capítulo será abordado o problema de escalonamento *job shop* com tempo de processamento incerto.

Este capítulo está organizado da seguinte forma: na Seção 6.1 são comentados os principais trabalhos da literatura. A definição do problema está na Seção 6.2. A Seção 6.3 apresenta a modelagem matemática do problema. A modelagem do problema por grafo disjuntivo e o algoritmo para calcular o *makespan* estão na Seção 6.4. As abordagens propostas são apresentadas na Seção 6.5.

6.1 Revisão bibliográfica

O problema de escalonamento *job shop* com parâmetros *fuzzy* é um problema de alta complexidade computacional, visto que os tempos de processamentos das operações são representados por números *fuzzy* e a comparação entre estes é dificíl, sendo às vezes impossível determinar qual o melhor escalonamento. Devido a isso, muitos trabalhos utilizam índices de *defuzzificação* existentes na literatura, para assim resolver um problema clássico (*crisp*). Outros utilizam algum método de ordenação (rankeamento) de números *fuzzy* que associa a cada número *fuzzy* um valor *crisp* e com este valor resolvem um problema clássico associado. Ou ainda, utilizam métodos de ordenação lexicográfica ou comparação baseada em α-cortes com o objetivo de otimizar um escalonamento em termos do *makespan fuzzy*.

Dentre os principais trabalhos da literatura, alguns que consideramos importantes são: Fortemps (1997), Lin (2000), Bonfim (2006), González et al. (2007) e Niu et al. (2008).

Uma das primeiras aplicações significativas que considera a incerteza nos parâmetros de tempo foi o trabalho de Fortemps (1997). Ele resolve o problema de *job shop* modelando o tempo de duração das tarefas como números de seis-pontos *fuzzy* com o objetivo de minimizar o *makespan*. O *makespan* é obtido através da comparação de números *fuzzy* utilizando o algoritmo simulated annealing.

Em Lin (2000) as incertezas nos tempos de processamento são modeladas usando tanto números triangulares *fuzzy* como λ-cortes. Os autores minimizam o *makespan* resolvendo o problema de *job shop crisp* que resulta da *defuzzificação* dos tempos de processamento.

Bonfim (2006) resolve o problema utilizando um algoritmo memético com população estruturada e utiliza o conceito de possibilidade para avaliar o *makespan* de cada indivíduo. Neste trabalho o *makespan* é calculado através de uma janela de tempo onde cada valor é representado por um número *crisp* associado.

No trabalho de González et al. (2007) os autores propõem um algoritmo memético para resolver o problema de *job shop fuzzy* e introduz um modelo de escalonamento baseado no valor esperado do *makespan*. Este valor associa a cada variável *fuzzy* um valor esperado e assim resolve o problema com o objetivo de minimizar o *makespan* esperado $E[C_{max}(x, \xi, \nu)]$.

Niu et al. (2008) propõem um algoritmo de *particle swarm* combinado com operadores genéticos denominado GPSO, para resolver o problema de *job shop* com tempo de processamento *fuzzy*. Neste trabalho o tempo de processamento das operações é descrito por números triangulares *fuzzy* e o objetivo do problema é encontrar um escalonamento que minimize o *makespan* e sua incerteza. Os autores utilizam um método de classificação de números *fuzzy* para estimar o valor do *makespan fuzzy* e sua incerteza.

Os trabalhos citados acima possuem a desvantagem de lidarem somente com um problema associado, ou seja, em todos os trabalhos é utilizado algum método que associa um número real ao número *fuzzy* para então encontrar o *makespan* do problema. Por isso, nesta tese propomos alguns algoritmos que contornam estas dificuldades, mantendo as incertezas em todo o processo de resolução do problema e encontrando não apenas uma única solução mas um conjunto de soluções (escalonamentos) com alto grau de possibilidade de serem ótimos.

6.2 Definição do problema

O problema de escalonamento *job shop* aqui considerado é o mesmo descrito na Seção 4.1. Entretanto, aqui os tempos de processamento das operações não são necessariamente conhecidos e são representados por números triangulares *fuzzy* da forma $\tilde{p} = (p_i, p_m, p_s)$. A Tabela 6.1 apresenta uma instância gerada para o problema com 3 tarefas e 3 máquinas com tempo de processamento incerto.

Tarefas	Ordem das operações (máquina, tempo de processamento)		
1	O_{11}(1, [2,40 3 3,23])	O_{12}(2, [2,45 3 3,35])	O_{13}(3, [2,00 3 3,36])
2	O_{21}(1, [1,18 2 2,79])	O_{23}(3, [2,22 3 3,44])	O_{22}(2, [3,37 4 4,31])
3	O_{32}(2, [2,41 3 3,44])	O_{31}(1, [1,63 2 2,31])	O_{33}(3, [0,96 1 1,61])

Tabela 6.1: Instância do problema *job shop fuzzy* (3×3).

Para a geração dos números triangulares *fuzzy* apresentados na Tabela 6.1 foi utilizado o *benckmark* original (Tamura 2007), onde o valor modal p_m de cada número triangular *fuzzy* corresponde ao valor *crisp* do *benckmark* e os espalhamentos a esquerda $(p_m - p_i)$ e a direita $(p_s - p_m)$ foram gerados segundo uma distribuição uniforme no intervalo $[0, 1]$.

Na instância exemplificada acima, a ordem de processamento das tarefas nas máquinas é pré-estabelecida. De acordo com a Tabela 6.1, a tarefa 1 precisa ser processada inicialmente na máquina 1 com um tempo *fuzzy* de [2,40 3 3,23], em seguida pela máquina 2 com tempo *fuzzy* de [2,45 3 3,35] e assim por diante. Um escalonamento é factível quando a ordem das operações é preservada e cada máquina processa apenas uma operação de cada vez.

6.3 Modelagem matemática do *job shop fuzzy*

Seja $\{\tilde{J}_j\}_{1 \leqslant j \leqslant n}$ um conjunto de n tarefas que devem ser processadas em um conjunto de m máquinas $\{\tilde{M}_k\}_{1 \leqslant k \leqslant m}$. Seja $\tilde{O} = \{\tilde{o}_{jk}\}$ o conjunto de operações, onde $\tilde{o}_{11}$ é considerada a operação inicial da tarefa $\tilde{J}_1$ na máquina $\tilde{M}_1$ e $\tilde{o}_{nm}$ a operação final da tarefa $\tilde{J}_n$ na máquina $\tilde{M}_m$. Neste problema todas as tarefas possuem a mesma quantidade de operações que é igual ao número de máquinas.

As operações estão interligadas por dois tipos de restrições. A primeira restrição é a de precedência, onde cada operação i só pode ser escalonada após todas as operações antecessores a ela, $\tilde{P}_i$, terem sido concluídas. Em segundo lugar é a restrição de que uma operação i só pode ser escalonada se a máquina vigente estiver livre.

Nesta modelagem, $\tilde{F}_i$ representa o tempo final da operação i e um escalonamento pode ser representado por um vetor de tempos finais $(\tilde{F}_1, \tilde{F}_m, \ldots, \tilde{F}_n)$. Seja $\tilde{A}(t)$ um conjunto de operações a serem processadas em um tempo t e seja $r_{i,k} = 1$ se a operação i requerer a máquina k para ser processada e $r_{i,k} = 0$ caso contrário.

Para cada operação i o tempo de processamento $\tilde{p}_i$ é fixado. O objetivo do problema é encontrar um escalonamento que minimize o tempo total gasto entre o início da primeira operação e o término da última operação, que representa o tempo de processamento total (*makespan*) denominado por $\tilde{C}_{max}$. Nesta modelagem *fuzzy* cada tempo de processamento do problema é um número triangular *fuzzy* e seu valor de *makespan* também é representado por um número triangular *fuzzy*. A modelagem matemática para este problema é caracterizado como a seguir:

Função objetivo: Min $\tilde{F}_n = \tilde{C}_{max}$

Sujeito à:

$$\begin{aligned} &\tilde{F}_k \leqslant \tilde{F}_i - \tilde{p}_i \qquad i = 1, \ldots, n; \quad k \in \tilde{P}_i \\ &\sum_{i \in \tilde{A}(t)} r_{i,k} \leqslant 1 \qquad k \in \tilde{M}; t \geq 0 \\ &\tilde{F}_i \geq 0 \qquad i = 1, \ldots, n. \end{aligned} \tag{6.1}$$

As restrições desta modelagem são:

- No início, todas as tarefas estão prontas para o processamento e todas as máquinas estão prontas para executá-las.
- As operações não podem ser interrompidas, ou seja, a operação tem de ser completada antes de se iniciar uma outra operação na mesma máquina.
- Não existe restrição de precedência entre operações de tarefas diferentes.
- Existe restrição de precedência entre operações dentro de uma tarefa, indicando em qual máquina a tarefa passará em cada tempo.

O objetivo do problema é encontrar uma ordem de escalonar as tarefas nas máquinas, de tal maneira que minimize o tempo total de execução (*makespan fuzzy*). O termo *makespan* está relacionado com o tempo total entre o início da primeira operação e o fim da última operação e é ele quem determina quão eficiente está a utilização dos recursos.

6.4 Modelagem por grafo disjuntivo com parâmetros *fuzzy*

A característica *fuzzy* de um problema pode ser encontrada em diversos níveis: da estrutura do grafo (nós e arestas) aos parâmetros associados ao grafo (custo, tempo, etc). Problemas com estrutura de grafo *crisp* e parâmetros *fuzzy* é um dos mais estudados da literatura. Estes são problemas em que a estrutura do grafo é bem conhecida e rígida e os parâmetros associados são representados por números *fuzzy*.

Um dos problemas de grafos com parâmetros *fuzzy* mais estudados na literatura é o de caminho mínimo com parâmetros incertos. Um outro problema de otimização que pode ser modelado na forma de grafo com parâmetros *fuzzy* bastante complexo e estudado na literatura é o problema de escalonamento *job shop*.

Inicialmente, estes dois problemas não apresentam nenhuma relação, já que o objetivo do problema de caminho mínimo é encontrar um menor caminho entre dois nós do grafo, ou seja, minimizar a função objetivo ($min\{f(x)\}$), e o objetivo do problema de *job shop* é minimizar o *makespan* (caminho máximo) do grafo, ou seja, $min(max\{f(x)\})$. Entretando, pode-se estabelecer uma relação entre estes dois problemas já que,

$$min\{f(x)\} \equiv -max\{-f(x)\},$$

o que é bastante interessante pois não existe algoritmo exato para resolver o problema de caminho máximo.

No trabalho de Hernandes (2007) é proposto um algoritmo de caminho mínimo baseado no algoritmo clássico de Ford-Moore-Bellman que pode ser aplicado em grafos com parâmetros *fuzzy* negativos. Desta forma, fizemos as modificações necessárias e utilizamos este algoritmo para calcular o caminho crítico (*makespan*) para o problema do *job shop fuzzy*.

A modelagem do problema de *job shop fuzzy* (JSSPF) por meio do grafo disjuntivo com tempo de processamento incerto é a mesma descrita na Seção 4.3, porém, aqui os tempos de processamento das tarefas nas máquinas são representados por números *fuzzy* (Figura 6.1).

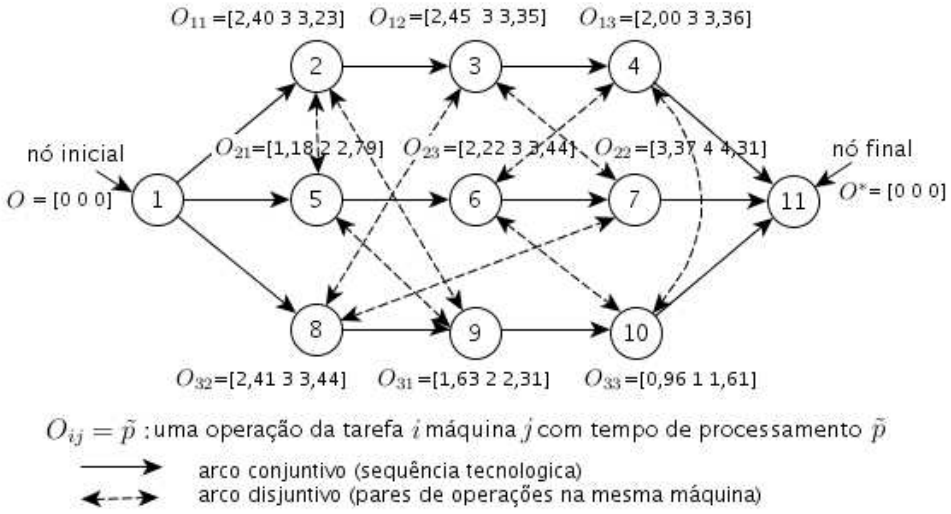

Figura 6.1: Grafo disjuntivo com parâmetros *fuzzy* do JSSPF ($n = 3$ e $m = 3$).

Na literatura não foi encontrado nenhum trabalho que resolva o problema de *job shop fuzzy* modelado por meio do grafo disjuntivo com tempo de processamento incerto, sendo esta mais uma justificativa para os algoritmos propostos nesta tese.

Inicialmente, o grafo disjuntivo *fuzzy* não representa uma solução para o problema de *job shop fuzzy*. Quando todas as direções dos arcos disjuntivos são definidas obtém-se um grafo direcionado e, se o grafo resultante for acíclico temos uma solução (escalonamento) factível para o problema. Uma solução com a definição de todas as direções nos arcos disjuntivos é representada na Figura 6.2.

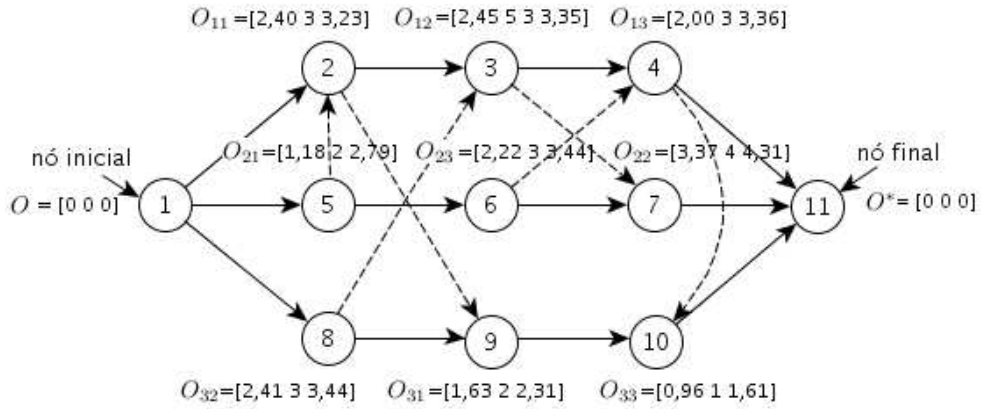

Figura 6.2: Representação de uma solução acíclica para o problema *job shop fuzzy*.

Para calcular o *makespan* do problema basta calcular o caminho crítico do grafo acíclico *fuzzy* (Figura 6.2). Este caminho é determinado pela maior distância entre o nó inicial e o nó final do grafo. O algoritmo que utilizamos para calcular o *makespan* será descrito na Seção 6.4.1.

6.4.1 Algoritmo para calcular o *makespan* do problema *job shop fuzzy*

Hernandes (2007) propõe um algoritmo para encontrar o caminho mínimo em grafos com parâmetros *fuzzy* e este algoritmo pode ser utilizado em grafos com parâmetros negativos. Tal algoritmo é baseado no algoritmo clássico de Ford-Moore-Bellman e utiliza a relação de ordem descrita na Seção 2.3 para determinar o conjunto de caminhos não-dominados. Desta forma, adaptou-se o algoritmo proposto por Hernandes (2007) para calcular o *makespan* do problema *job shop fuzzy*. O pseudocódigo do algoritmo é apresentado no Algoritmo 6.

6.4.1.1 Apresentação do algoritmo

Este é um algoritmo iterativo possuindo como critério de parada o número de iterações ou a não alteração dos custos (tempos de processamento das operações) de todos os caminhos (escalonamentos) encontrados na iteração anterior com relação à iteração atual. Como a relação de Okada e Soper (2000) pode apresentar entre dois nós mais de um caminho não-dominado, cada caminho recebe um rótulo (etiqueta) para que este seja

construído no passo final do algoritmo. O algoritmo é baseado nos seguintes passos:

a) No Passo 1 é feita a inicialização das variáveis, onde são atribuídos os valores (rótulos e tempos de processamento) iniciais dos caminhos (iteração "zero"), para assim após a primeira iteração (Passo 2) verificar o critério de parada.

b) O Passo 2 é composto de duas importantes etapas. Na primeira (Item 2 do Passo 2) são determinados todos os caminhos entre os nós 1 (inicial) e i (final) (são determinados por meio das distâncias associadas no Passo 1 e com os tempos de processamento dos arcos (i, j)) $(\widetilde{l}_{ij})$. Na segunda etapa (Item 3 do Passo 2) é aplicada a relação de dominância de Okada e Soper (2000) para descartar os caminhos dominados.

c) No Passo 3 é verificado o critério de parada. Se este é satisfeito, os caminhos não-dominados são construídos no Passo 4 e o algoritmo é finalizado no Passo 5. Caso contrário, o algoritmo retorna ao Passo 2 (como o algoritmo analisa todos os nós a cada iteração, similar ao de Ford-Moore-Bellman, as distâncias (tempos de processamento das operações) entre alguns caminhos podem alterar seus valores) ou finaliza caso exista circuito negativo.

Uma observação importante é que este algoritmo foi proposto para números triangulares *fuzzy*, porém sua extensão para números trapezoidais é simples e imediata.

Informações sobre o algoritmo

V: conjunto dos nós;

it: contador de iterações;

$(a_s)^i$: limitante superior do tempo de processamento do nó i;

$\widetilde{l}_{ji}$: tempo de processamento do arco (j, i);

$\widetilde{c}^{it}_{(i,k)}$: tempo de processamento do caminho entre os nós 1 e i, com a etiqueta k, na iteração it;

M: um número com valor grande, substitui o ∞ do algoritmo clássico de FBM;

Γ_i^{-1}: conjunto dos nós predecessores de i.

Algoritmo 6 Pseudocódigo do algoritmo para calcular o *makespan* em grafos com parâmetros *fuzzy*

Passo 1: Inicialização das variáveis

1. $\widetilde{c}^{0}_{(1,1)} = (0, 0, 0)$
2. $\widetilde{c}^{0}_{(j,1)} = (M + 2, 1, 1)$, $j = 2, 3, \ldots, r$

   talque:

   - r: número de nós;
   - $M = \sum_{i=1}^{na} |(a_s)^i|$;
   - na: número de arcos.
3. $it \leftarrow 1$.

Passo 2: Determinação dos caminhos e verificação da dominância

1. $\widetilde{c}^{it}_{(1,1)} = (0, 0, 0)$
2. $\forall j \in \Gamma_i^{-1}$, $i = 2, 3, \ldots, r$, faça:

   - $\widetilde{c}^{it}_{(i,k1)} = \widetilde{c}^{it-1}_{(j,k2)} \oplus \widetilde{l}_{ij}$ (construção dos custos dos caminhos)
3. Verificação da dominância entre as etiquetas do nó i (relação de Okada):

   - Se $\widetilde{c}^{it}_{(i,m)} > \widetilde{c}^{it}_{(i,n)} \Rightarrow$ elimine a *m-ésima* etiqueta
   - Se $\widetilde{c}^{it}_{(i,m)} < \widetilde{c}^{it}_{(i,n)} \Rightarrow$ elimine a *n-ésima* etiqueta

Passo 3: Critério de parada

1. Se $(\widetilde{c}^{it}_{(i,k1)} = \widetilde{c}^{it-1}_{(i,k1)}, \forall i \in V)$ ou $(it = r)$ faça:

   - $it = r$ e $\widetilde{c}^{it}_{(i,k1)} \neq \widetilde{c}^{it-1}_{(i,k1)} \Rightarrow$ **Passo 5**
2. Senão $it \leftarrow it + 1 \Rightarrow$ volte ao **Passo 2**

Passo 4: Composição dos caminhos

- Encontra todos os caminhos não-dominados entre os nós 1 e i

Passo 5: Saída: Todos os caminhos não-dominados e seus tempos de processamento.

6.5 Abordagens propostas

A complexidade de problemas de escalonamento como o de *job shop* normalmente envolvem estratégias heurísticas para resolvê-los. Em se tratando do problema onde o tempo de processamento das operações são representados por números *fuzzy*, essa complexidade aumenta e é necessária uma reformulação significativa do problema e métodos eficientes para resolvê-lo.

Desta forma, nesta tese propomos três algoritmos híbridos que englobam o algoritmo de colônia de formigas (ACS) e o algoritmo genético (AG) com busca local para resolver o problema de escalonamento *job shop* com tempo de processamento incerto. As técnicas de buscas locais são as descritas na Seção 4.5.1. Elas são denominadas Troca de Tarefas Adjacentes no Caminho Crítico (CC), Troca de Tarefa na Máquina mais Ociosa (MO) e a junção destas duas buscas locais (CC-MO). O algoritmo de colônia de formigas é utilizado para criar uma população inicial de escalonamentos factíveis e o algoritmo genético com busca local é utilizado para evoluir esses escalonamentos.

Os algoritmos híbridos são utilizados para encontrar boas soluções para o problema e para avaliar a melhor solução é utilizado o método de comparação descrito na Seção 2.4. Este método tem a finalidade de definir, entre vários números *fuzzy*, qual é o maior ou menor.

6.5.1 Algoritmo híbrido de sistema de colônia de formigas e genético para o problema de *job shop fuzzy*

Um conceito importante e crítico na execução de um algoritmo é a escolha da representação de possíveis soluções para o problema. Nesta tese o problema de *job shop* com parâmetros *fuzzy* (Tabela 6.1) é representado por um grafo disjuntivo (Figura 6.1) e cada solução é representada por meio de um vetor V de $n.m + 2$ colunas, onde n é o número de tarefas e m o número de máquinas. Cada campo do vetor v_k com $1 \leqslant k \leqslant n.m + 2$ é preenchido por um número k que indica o número do nó no grafo disjuntivo. Este vetor possui uma sequência que corresponde à ordem que as tarefas são atendidas pelas máquinas. Como dito anteriormente, na literatura não foi encontrado nenhum trabalho que resolvesse este problema por meio desta representação, sendo essa mais uma razão para esta abordagem.

6.5.1.1 Geração da população inicial

O primeiro passo do algoritmo híbrido é a geração da população inicial e esta foi gerada utilizando o algoritmo de colônia de formigas que será descrito a seguir:

Algoritmo de sistema de colônia de formigas para o problema do job shop fuzzy

O algoritmo de colônia de formigas utilizado para resolver o problema do *job shop fuzzy* é semelhante ao descrito na Seção 4.5.1, porém aqui os tempos de processamento das operações são representados por números triangulares *fuzzy*. Desta forma, a matriz

de feromônio é dada por uma matriz triangular como descrita na Figura 6.3. O funcionamento do algoritmo será descrito a seguir:

Dado um grafo disjuntivo como ilustrado na Figura 6.1, os nós artificiais identificados como (1) e (11) representam o ponto de partida e chegada respectivamente de cada formiga. Na matriz da trilha do feromônio ilustrada na Figura 6.3, as linhas representam os nós origem e as colunas os nós destinos (máquina), o elemento $\tau(r, s, c)$ é a quantidade de feromônio correspondente à aresta (r, s) na célula c. Observe que a matriz de feromônio é tridimensional ($(n * m + 2, n * m + 2, 3)$, onde n é o número de tarefas e m o número de máquinas), pois estamos trabalhando com número triangular *fuzzy*.

$$\left[\begin{array}{ccc|ccc|ccc} \overbrace{\tau(1,1,1)} & \dots & \tau(1,S,1) & \overbrace{\tau(1,1,2)} & \dots & \tau(1,S,2) & \overbrace{\tau(1,1,3)} & \dots & \tau(1,S,3) \\ \tau(2,1,1) & \dots & \tau(2,S,1) & \tau(2,1,2) & \dots & \tau(2,S,2) & \tau(2,1,3) & \dots & \tau(2,S,3) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \tau(R,1,1) & \dots & \tau(R,S,1) & \tau(R,1,2) & \dots & \tau(R,S,2) & \tau(R,1,3) & \dots & \tau(R,S,3) \end{array}\right]$$

Figura 6.3: Matriz tridimensional da trilha de feromônio

O conjunto S_f das operações de uma formiga f é formado por todos os nós que obedecem as restrições do problema. Inicialmente $S_f = \{u_{i1c}/i \in [1, n], c = 1 : 3\}$ é formado pela primeira operação de cada tarefa. A regra de construção de uma solução pode ser descrita como a seguir (Equação 6.2):

$$s = \begin{cases} argmax_{u \in S_f}\{[\tau(r,u,c)] \cdot [\eta(u,c)^\beta]\}, & \text{se } q \in q_0 \\ P, & \text{caso contrário} \end{cases} \tag{6.2}$$

Para evitar uma convergência prematura, P é determinado segundo a expressão (Equação 6.3):

$$P_f(r,s) = \begin{cases} \frac{[\tau(r,s,c)] \cdot [\eta(s,c)^\beta]}{\sum_{u \in S_f}[\tau(r,u,c)] \cdot [\eta(u,c)^\beta]}, & \text{se } s \in S_f \\ 0, & \text{caso contrário} \end{cases} \tag{6.3}$$

onde, o parâmetro $\eta(s, c) = 1/p(s, c)$ indica o inverso do tempo de processamento incerto da operação s e β é um valor heurístico que pondera a influência relativa da distância $\eta(s, c)$.

A atualização do feromônio é feita de forma local e global. Ela é necessária para evitar a convergência prematura do algoritmo. A atualização local se dá por meio da evaporação do feromônio nas arestas, de forma a reduzir a quantidade de feromônio e forçar a próxima

formiga a procurar uma melhor solução. Tradicionalmente esta atualização é realizada por meio da regra de atualização local (Equação 6.4).

$$\tau(r,s,c) = (1-\rho)\cdot\tau(r,s,c) + \rho\cdot\tau_0, \qquad \rho \in [0,1] \tag{6.4}$$

A cada iteração, após todas as formigas terminarem seus percursos, é aplicada a regra de atualização de feromônio global. De acordo com esta regra de atualização, a trilha de feromônio é adicionada somente ao melhor caminho encontrado pela formiga. Esta atualização é realizada de acordo com a Equação 6.5.

$$\tau(r,s,c) = (1-\rho_1)\cdot\tau(r,s,c) + \rho_1\cdot\Delta\tau(r,s,c), \qquad \rho_1 \in [0,1] \tag{6.5}$$

onde $\Delta\tau(r,s,c) = 1/L_{max}$ e L_{max} é o *makespan* do melhor escalonamento encontrado pelas formigas ao final das iterações.

6.5.1.2 Cálculo do makespan

Para calcular o *makespan* do problema do *job shop fuzzy* é utilizado o algoritmo descrito na Seção 6.4.1.1. Este algoritmo encontra todos os caminhos não-dominados entre o nó inicial e o nó final do grafo, sendo necessária a utilização de um método capaz de decidir qual é o maior entre os valores encontrados, pois este representa o *makespan* do problema de *job shop fuzzy*. Para entender melhor segue um exemplo.

Exemplo 6.1 *Considere o problema do job shop fuzzy (Tabela 6.1) representado pelo grafo disjuntivo (Figura 6.1).*

Executando o algoritmo híbrido neste problema obtem-se uma solução (escalonamento) factível dada pelo vetor V abaixo:

$$V = (5 \rightarrow 8 \rightarrow 2 \rightarrow 9 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 10)$$

Este escalonamento corresponde ao grafo acíclico (Figura 6.2). Cada número do vetor V representa o número do nó no grafo acíclico e a sequência dos números representa a ordem de processamento das tarefas nas máquinas.

Dada uma solução V é preciso encontrar o maior caminho (*makespan*) entre o nó inicial e o nó final do grafo e este caminho será encontrado pelo algoritmo descrito na Seção 6.4.1.1. Executando o Algoritmo 4 para a solução V acima obtem-se dois caminhos não-dominados entre o nó 1 e o nó 11.

- $c_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 10 \rightarrow 11)$ com custo [8.99 12 14.35]
- $c_2 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 11)$ com custo [9.40 12 13.69]

O objetivo do problema é encontrar o *makespan* e este é dado pelo maior caminho entre o nó inicial e o nó final, utiliza-se o critério de comparação de números *fuzzy*, descrito na Seção 2.4, para escolher entre os 2 caminhos acima qual deles representa o *makespan*. De acordo com o critério de comparação, o caminho crítico é:

- $c_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 10 \rightarrow 11)$ com custo [8.99 12 14.35].

6.5.1.3 Passos seguintes dos algoritmos

Nos passos seguintes dos algoritmos híbridos são executados os operadores genéticos enquanto o critério de parada não for alcançado. O critério de parada utilizado foi o número de gerações. Os operadores genéticos executados nestes algoritmos foram o *crossover* e a mutação. Os operadores utilizados e os métodos de busca local foram os mesmos descritos na Seção 4.5 no qual o problema tratado possui parâmetros precisos.

Como saída dos algoritmos é apresentado um conjunto solução com alto grau de otimalidade. Cada indivíduo tem seu valor de *fitness* representado por um número triangular *fuzzy*. Para apresentar o valor *crisp* foi utilizado um método de *defuzzificação* que consiste de um mapeamento de informações *fuzzy* em valores *crisp*. O método de *defuzzificação* utilizado foi o da centróide, descrito na Seção 2.6.

Um pseudocódigo dos algoritmos híbridos aplicados ao problema de escalonamento *job shop* com parâmetros *fuzzy* é apresentado no Algoritmo 7.

Algoritmo 7 Pseudocódigo dos algoritmos híbridos aplicados ao problema de escalonamento *job shop fuzzy*

Entrada:

m: número de máquinas;
n: número de tarefas;
$O_{m \times n}$: matriz com sequência de operações para cada tarefa;
$o_{ij} = (m_{ij}, \widetilde{p}_{ij})_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m}$: par ordenado de cada operação composto pela máquina que irá processa-lá e o tempo de processamento da operação;
T_{pop}: tamanho da população;
N_{it}: número de iterações;
N_{form}: número de formigas;
ρ: nível de feromônio;
β: informação heurística;
ρ_1: informação de feromônio;
q_0: informação heurística;
N_{gera}: número de gerações;
P_c: probabilidade de *crossover*;
P_m: probabilidade de mutação.

Início do algoritmo:

- Gerar a população inicial por meio do algoritmo de colônia de formigas;
- Calcula o valor de *makespan* de cada indivíduo;
- Ordenar a população utilizando o valor de *makespan* (método descrito na Seção 2.4);

para $z = 0$ até $z = N_{gera}$, **faça**

- Selecionar 2 indivíduos da população inicial;
- Aplicar *crossover* gerando 2 novos indivíduos para a memória;
- Selecionar 1 indivíduo da população inicial;
- Aplicar mutação gerando 1 novo indivíduo para a memória;
- calcular o valor de *makespan* da memória (algoritmo descrito na Seção 6.4.1.1);
- Aplicar supressão na memória e verificar diversidade;
- Ordenar a população utilizando o valor de *makespan* através do método descrito na Seção 2.4
- Selecionar os $x\%$ melhores indivíduos da memória para a próxima geração;
- Atualizar a memória da próxima geração completando com novos indivíduos, se necessário;
- Se MA-ACS(MO) ou MA-ACS(CC) aplicar busca local no melhor indivíduo da população e atualizar seu valor de *fitness*;
- Se AG-ACS, passo anterior não existe.

fim para

Saída: Melhor escalonamento da população e seu valor de *makespan*.

Capítulo 7

Resultados numéricos das abordagens com parâmetros *fuzzy*

Neste capítulo são apresentados e discutidos os resultados obtidos com a aplicação das quatro abordagens híbridas (AG-ACS, MA-ACS(MO), MA-ACS(CC) e MA-ACS(CC-MO)) propostas nesta tese, para resolver o problema de escalonamento *job shop* com parâmetros *fuzzy*. Na literatura não existem instâncias *fuzzy* para testes. Sendo assim, utilizamos 13 instâncias clássicas conhecidas na literatura (Tamura, 2007) onde o tempo de processamento das operações são representados por números *crisp*. Estas instâncias foram fuzzificadas utilizando a teoria *fuzzy* e os tempos de processamento das operações foram representados por números triangulares *fuzzy*, de acordo com a definição descrita na Seção 6.2.

Para avaliar as abordagens foram utilizadas instâncias de diferentes tamanhos e diferentes autores. As instâncias foram assim escolhidas com a finalidade de verificar o comportamento das abordagens em diferentes tipos de problemas.

Para medir a eficiência dos quatro algoritmos implementados, cada experimento foi executado 10 vezes por instância, sendo que, em cada uma delas foi obtido como resultado um conjunto solução composto por escalonamentos com alto grau de possibilidade de serem ótimos.

Como estamos lidando com um problema com incertezas, a noção de "escalonamento ótimo" também é incerta. Desta forma, é considerado como escalonamento ótimo o melhor resultado encontrado pelos algoritmos.

Um ponto forte e importante das abordagens *fuzzy* propostas nesta tese é que os algoritmos encontram não apenas uma única solução para o problema, mas um conjunto de soluções com alto grau de possibilidade de serem ótimas.

A Tabela 7.1 apresenta algumas informações destas instâncias, como o nome dado a cada problema, o tamanho e o nome do autor que a criou. As instâncias são caracterizadas pelas seguintes informações: número de tarefas, número de máquinas e uma tabela contendo a sequência de operações de cada tarefa, incluindo o número da máquina que irá processá-la e o tempo de processamento da operação.

Instância	n^o de tarefas $\times$ n^o máquinas	Descrição do problema
ft06	6×6	Fisher e Thompson
abz6	10×10	Adams, Balas e Zawack
orb02	10×10	Applegate e Cook
la01	10×5	Lawrence
la02	10×5	Lawrence
la06	15×5	Lawrence
la08	15×5	Lawrence
la11	20×5	Lawrence
la12	20×5	Lawrence
la16	10×10	Lawrence
la17	10×10	Lawrence
la20	10×10	Lawrence
la23	15×10	Lawrence

Tabela 7.1: Problemas do *job shop* utilizados nesta tese

7.1 Plataforma

Os quatro algoritmos foram implementados em Matlab 8. Todos os testes foram realizados na plataforma Linux, Intel Core 2Duo com 2.0GHz e 4Gb de memória.

7.2 Parâmetros

Os parâmetros adotados na execução dos algoritmos são apresentados a seguir.

- Número de iterações = 100;
- Número de formigas = 15;
- $\rho_1 = 0,1$; $\beta = 2$; $\rho = 0,01$; $q_0 = 0,7$;
- População inicial: gerada pelo algoritmo ACS;
- Número de gerações: 500;
- Tamanho da população: 40 indivíduos;
- Probabilidade de crossover: $0,8$;
- Probabilidade de mutação: $0,6$;
- Supressão: elimina indivíduos de mesmo *makespan*;
- Diversidade: insere novos indivíduos criados pelo ACS;

- Taxa miníma de diversidade populacional: 50%;
- Busca local: apenas no melhor indivíduo da população;
- Critério de parada: número de gerações.

7.3 Apresentação dos resultados

Nesta seção são apresentados os resultados obtidos pelas quatro abordagens propostas quando aplicadas nas instâncias extraídas da literatura (Tamura, 2007). Com o intuito de obter uma boa avaliação, foram utilizados diversos tipos de instâncias, tais como: instâncias de diferentes números de tarefas e máquinas e, diferentes autores para os testes. Instâncias com mesmo número de tarefas e máquinas podem apresentar características diferentes e diferentes níveis de dificuldades em encontrar boas soluções.

Os resultados dos experimentos realizados são apresentados por ordem de autor. Em cada bloco de autor é apresentada uma tabela contendo as instâncias testadas, os algoritmos implementados, o melhor resultado obtido por cada algoritmo denominado *makespan fuzzy*, a *defuzzificação* (método da centróide) do melhor resultado encontrado e o *makespan crisp* extraído da literatura.

A análise dos resultados será apresentada na Seção 7.4.

7.3.1 Instância de Adams et al. (1988)

Em *The shifting bottleneck procedure for job shop scheduling* (Adams et al., 1988), os autores propõem um conjunto de instâncias para testes. O tempo de processamento das operações são representados por números triangulares *fuzzy* do tipo (p_i, p_m, p_s), no qual, p_m corresponde ao valor modal, p_i o limitante inferior e p_s o limitante superior.

Na Tabela 7.2, são apresentados os resultados das abordagens quando aplicadas na instância *abz6* de Adams et al. (1988). Esta instância possui 10 tarefas e 10 máquinas. O tempo médio para realizar cada uma das quatro abordagens foi de 948 seg.

Instância	Algoritmo	Makespan fuzzy	Defuzzificação	Makespan crisp
abz6 (10x10)	AG-ACS	[894,24 972 1020,6]	962,28	943
	MA-ACS(MO)	[871,24 947 994,35]	937,50	
	MA-ACS(CC)	[871,24 947 994,35]	937,50	
	MA-ACS(CC-MO)	[870,03 946 993,3]	936,44	

Tabela 7.2: Comparação dos resultados para minimizar o *makespan fuzzy*.

7.3.2 Instância de Applegate e Cook (1991)

No trabalho, *A computational study of the job-shop scheduling instance* (Applegate e Coock, 1991), os autores propõem uma heurística baseada em método de plano de corte

e algoritmo de Branch and Bound com a finalidade de obter bons escalonamentos. Na literatura encontra-se disponível algumas de suas instâncias que são utilizadas para testes.

Na Tabela 7.3 são apresentados os resultados das abordagens quando aplicadas na instância *orb02*, com 10 tarefas e 10 máquinas de Applegate e Cook (1991). O tempo médio para realizar cada uma das quatro abordagens para esta instância foi de 1024 seg.

Instância	**Algoritmo**	**Makespan fuzzy**	**Defuzzificação**	**Makespan crisp**
orb02 (10x10)	AG-ACS	[859,28 934 980,7]	924,66	888
	MA-ACS(MO)	[834,4 907 952,35]	897,93	
	MA-ACS(CC)	[839,04 912 957,6]	902,88	
	MA-ACS(CC-MO)	[828 900 945]	891	

Tabela 7.3: Comparação dos resultados para minimizar o *makespan fuzzy*.

7.3.3 Instância de Fisher e Thompson (1963)

Um dos trabalhos pioneiros em aprendizagem probabilística na resolução do *job shop* foi o de Fisher e Thompson (1963), cujo título é *Probabilistic learning combination of local job shop scheduling rules*. Neste trabalho eles comprovam que a combinação de regras de programação (conhecida como regra de despacho ou de prioridades) é superior a qualquer uma destas regras, se tomadas separadamente.

Na Tabela 7.4 são apresentados os resultados das quatro abordagens propostas quando aplicadas na instância *ft06*, com 6 tarefas e 6 máquinas de Fisher e Thompson (1963). O tempo médio para realizar cada uma das quatro abordagens para esta instância foi de 179 seg.

Instância	**Algoritmo**	**Makespan fuzzy**	**Defuzzificação**	**Makespan crisp**
ft06 (6x6)	AG-ACS	[50,6 55 57,75]	54,4	55
	MA-ACS(MO)	[50,6 55 57,75]	54,4	
	MA-ACS(CC)	[50,6 55 57,75]	54,4	
	MA-ACS(CC-MO)	[50,6 55 57,75]	54,4	

Tabela 7.4: Comparação dos resultados para minimizar o *makespan fuzzy*.

7.3.4 Instâncias de Lawrence (1984)

Na literatura, pode-se encontrar um conjunto com diversas instâncias para testes geradas por Lawrence (1984). Estas instâncias pertencem a classe *la*. Dentre as diversas instâncias de Lawrence (1984), disponíveis na literatura, foram escolhidas 10 instâncias com diferentes números de tarefas e máquinas para testes.

Instância	Algoritmo	Makespan fuzzy	Defuzzificação	Makespan crisp
la01 (10x5)	AG-ACS	[612,72 666 699,3]	659,3	666
	MA-ACS(MO)	[612,72 666 699,3]	659,3	
	MA-ACS(CC)	[612,72 666 699,3]	659,3	
	MA-ACS(CC-MO)	[612,72 666 699,3]	659,3	
la02 (10x5)	AG-ACS	[602,6 655 687,75]	648,45	655
	MA-ACS(MO)	[602,6 655 687,75]	648,45	
	MA-ACS(CC)	[602,6 655 687,75]	648,45	
	MA-ACS(CC-MO)	[602,6 655 687,75]	648,45	
la06 (15x5)	AG-ACS	[851,92 926 972,3]	916,74	926
	MA-ACS(MO)	[851,92 926 972,3]	916,74	
	MA-ACS(CC)	[851,92 926 972,3]	916,74	
	MA-ACS(CC-MO)	[851,92 926 972,3]	916,74	
la08 (15x5)	AG-ACS	[793,9 863 906,1]	854,3	863
	MA-ACS(MO)	[793,9 863 906,1]	854,3	
	MA-ACS(CC)	[793,9 863 906,1]	854,3	
	MA-ACS(CC-MO)	[793,9 863 906,1]	854,3	
la11 (20x5)	AG-ACS	[1124,24 1222 1283,1]	1209,8	1222
	MA-ACS(MO)	[1124,24 1222 1283,1]	1209,8	
	MA-ACS(CC)	[1124,24 1222 1283,1]	1209,8	
	MA-ACS(CC-MO)	[1124,24 1222 1283,1]	1209,8	
la12 (20x5)	AG-ACS	[955,88 1039 1090,95]	1028,6	1039
	MA-ACS(MO)	[955,88 1039 1090,95]	1028,6	
	MA-ACS(CC)	[955,88 1039 1090,95]	1028,6	
	MA-ACS(CC-MO)	[955,88 1039 1090,95]	1028,6	
la16 (10x10)	AG-ACS	[879,52 956 1003,8]	946,44	945
	MA-ACS(MO)	[879,52 956 1003,8]	946,44	
	MA-ACS(CC)	[882,28 959 1006,95]	949,41	
	MA-ACS(CC-MO)	[870,32 946 993,3]	936,54	
la17 (10x10)	AG-ACS	[729,56 793 832,65]	785,07	784
	MA-ACS(MO)	[724,05 787 826,35]	779,13	
	MA-ACS(CC)	[729,56 793 832,65]	785,07	
	MA-ACS(CC-MO)	[722,2 785 824,2]	777,13	
la20 (10x10)	AG-ACS	[834,44 907 952,35]	897,93	902
	MA-ACS(MO)	[834,44 907 952,35]	897,93	
	MA-ACS(CC)	[834,44 907 952,35]	897,93	
	MA-ACS(CC-MO)	[834,44 907 952,35]	897,93	
la23 (15x10)	AG-ACS	[1004,64 1092 1146,6]	1081,1	1032
	MA-ACS(MO)	[969,68 1054 1106,7]	1043,5	
	MA-ACS(CC)	[966 1050 1102,5]	1039,3	
	MA-ACS(CC-MO)	[966 1050 1102,5]	1039,3	

Tabela 7.5: Comparação dos resultados para minimizar o *makespan fuzzy*.

A Tabela 7.5 apresenta os resultados das quatro abordagens propostas, quando aplicadas nas instâncias de Lawrence (1984).

O tempo médio para executar cada uma das quatro abordagens para as instâncias de Lawrence está ilustrado na Tabela 7.6.

Instância	*la01*	*la02*	*la06*	*la08*	*la11*	*la12*	*la16*	*la17*	*la20*	*la23*
Tempo médio (seg)	440	415	930	898	1720	1615	1077	973	1165	2523

Tabela 7.6: Tempo médio de processamento.

7.4 Análise dos resultados

Na execução de cada instância foram verificados também quantos indivíduos da população obtiveram seus valores de *makespan fuzzy* até 80% do melhor indivíduo encontrado pelos algoritmos. A Tabela 7.7 apresenta a quantidade de indivíduos com 80% de possibilidade de serem ótimos para cada instância testada.

Instância	AG-ACS	MA-ACS(MO)	MA-ACS(CC)	MA-ACS(CC-MO)
abz6	0	15	15	15
orb02	0	12	16	15
ft06	2	2	2	4
la01	16	16	16	16
la02	13	14	15	15
la06	16	19	20	16
la08	16	13	16	15
la11	18	20	17	18
la12	13	14	16	22
la16	12	15	10	21
la17	12	15	13	15
la20	18	17	17	18
la23	0	19	16	16

Tabela 7.7: Quantidade de indivíduos com +80% de possibilidade de serem ótimos.

Analisando os resultados da Tabela 7.2, percebe-se que os algoritmos híbridos com buscas locais conseguiram obter melhores soluções (escalonamentos) em comparação com o algoritmo ACS-AG. Além disso, eles conseguiram encontrar um conjunto solução com alto grau de possibilidade de serem ótimas. Enquanto o algoritmo AG-ACS não conseguiu encontrar nenhum escalonamento com mais de 80% de possibilidade de ser ótimo, os algoritmos MA-ACS(MO), MA-ACS(CC) e MA-ACS(CC-MO) encontraram 15 escalonamentos (Tabela 7.7). Esta diferença pode ser explicada pelo fato dos algoritmos trabalharem com técnicas de buscas locais, o que faz com que estes consigam encontrar melhores soluções que o AG-ACS.

De acordo com os resultados da Tabela 7.3 ambas abordagens encontraram diferentes resultados, isso pode demonstrar a dificuldade em solucionar a instância. Porém, o algoritmo MA-ACS(CC-MO) obteve um melhor rendimento, pois conseguiu encontrar o melhor *makespan fuzzy* entre as abordagens. Além disso, ele conseguiu encontrar 15 es-

calonamentos (Tabela 7.7) com alto grau de possibilidade de serem ótimos, enquanto a ACS-AG não conseguiu encontrar nenhum escalonamento que satisfizesse esse grau de possibilidade.

Analisando os resultados da Tabela 7.4 observa-se que apesar de ambas as abordagens conseguirem obter resultados parecidos, o algoritmo MA-ACS(CC-MO) (Tabela 7.7) obteve um melhor rendimento, conseguindo encontrar 4 escalonamentos com alto grau de possibilidade de serem ótimos, enquanto que as outras abordagens encontraram 2 escalonamentos.

Na Tabela 7.5 são apresentados os resultados das instâncias pertencentes à classe *la*. Nesta classe existem instâncias com diferentes números de máquinas e tarefas e, instâncias de mesmo tamanho, já que instâncias com mesmo número de máquinas e tarefas podem apresentar características diferentes. De acordo com a Tabela 7.5 todas as abordagens conseguem encontrar bons valores de *makespan fuzzy*. Porém, apesar de algumas instâncias apresentarem resultados semelhantes, podemos perceber que, na maior parte das instâncias testadas, a abordagem que trabalha com a junção das duas técnicas de buscas locais MA-ACS(CC-MO) apresenta os melhores resultados quando comparados com as outras três abordagens. Essa superioridade da abordagem é evidente principalmente, nos casos onde as instâncias são mais complicadas de se obter uma boa solução, como é o caso das instâncias *la16*, *la17* e *la23*.

Ainda é possível visualizar, por meio do gráfico de barras (Figura 7.1), uma comparação dos valores *defuzzificados* encontrados por cada um dos 4 algoritmos em relação ao *makespan crisp* disponível na literatura.

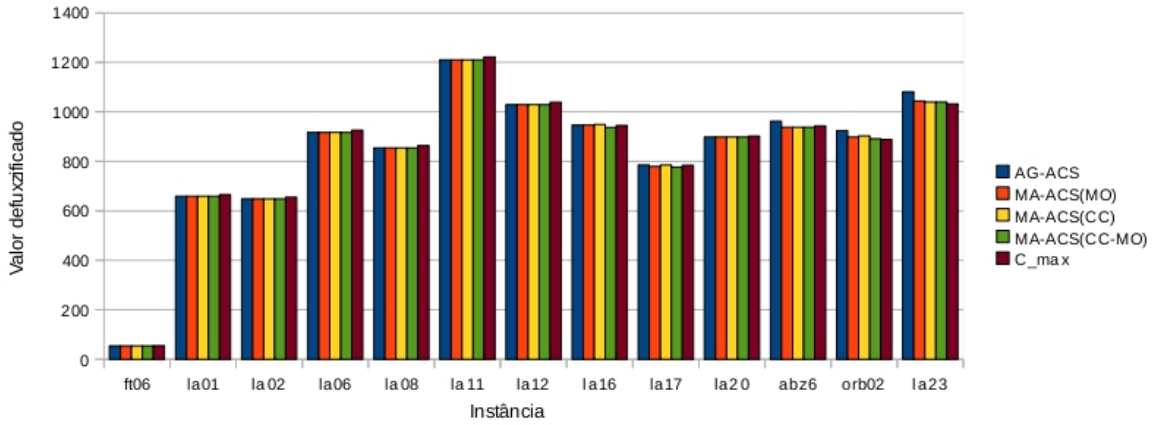

Figura 7.1: Comparação do melhor valor *defuzzificado* encontrado por cada algoritmo com relação ao *makespan crisp*.

No gráfico (Figura 7.1) percebe-se que os valores *defuzzificados* encontrados pelas abordagens estão próximos do *makespan crisp* (C_{max}) e, em alguns casos, estes valores ficam abaixo deste valor, como é o caso por exemplo da instância *la16*. Isso acontece quando o valor modal é bem próximo do valor C_{max} e é caracterizado como uma solução

ótima.

No gráfico da Figura 7.2, é ilustrada a convergência dos indivíduos da população que obtiveram valor de *makespan fuzzy* com mais de 80% de possibilidade de otimalidade para a instância *la02*. Como pode ser visto, para esta instância, todos os algoritmos encontraram indivíduos com grau de possibilidade 1. De uma população com 40 indivíduos, a abordagem ACS-AG obteve 13 indivíduos com mais de 80% de possibilidade de serem ótimos, a abordagem MA-ACS(MO) obteve 14 indivíduos e as abordagens MA-ACS(CC) e MA-ACS(CC-MO) obtiveram 15 indivíduos cada. Apesar de ambas MA-ACS(CC) e MA-ACS(CC-MO) obterem 15 indivíduos, podemos perceber que MA-ACS(CC-MO) consegue obter indivíduos com maior grau de possibilidade que MA-ACS(CC).

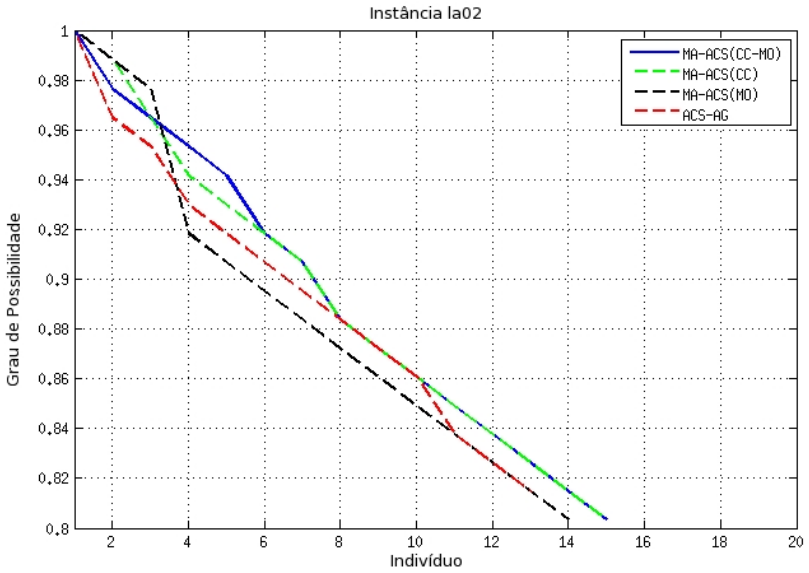

Figura 7.2: Relação entre indivíduo e grau de possibilidade para a instância *la02*

Na Figura 7.3 é ilustrada a convergência dos indivíduos da população que obtiveram valor de *makespan fuzzy* com mais de 80% de possibilidade de serem ótimos para a instância *la16*. Esta instância possui 10 máquinas e 10 tarefas e é uma instância mais difícil de se obter bons resultados. Como pode ser visto no gráfico (Figura 7.3) para esta instância (*la16*), apenas a abordagem MA-ACS(CC-MO) encontrou indivíduo com grau de possibilidade 1. De uma população com 40 indivíduos, a abordagem ACS-AG obteve 12 indivíduos com mais de 80% de possibilidade de serem ótimos, a abordagem MA-ACS(MO) obteve 15 indivíduos, a abordagem MA-ACS(CC) obteve 10 indivíduos e MA-ACS(CC-MO) obteve 21 indivíduos.

De acordo com a Figura 7.3, o algoritmo MA-ACS(CC-MO) não apenas conseguiu encontrar uma maior quantidade de indivíduos com mais de 80% de possibilidade de serem ótimos, mas também conseguiu encontrar indivíduos com maior grau de possibilidade.

Na Tabela 7.8 é possível verificar a porcentagem de instâncias que obtiveram erro entre $0\% - 1\%$, ou entre $1\% - 10\%$. Este erro é calculado por meio da comparação do *makespan*

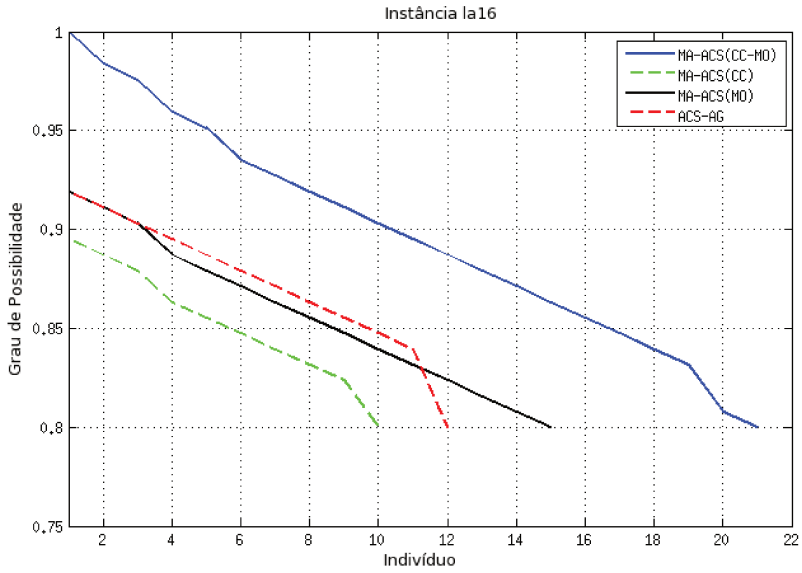

Figura 7.3: Relação entre indivíduo e grau de possibilidade para a instância *la16*

fuzzy defuzzificado encontrado por cada algoritmo híbrido, com relação ao *makespan crisp* disponível na literatura.

Erro	AG-ACS	MA-ACS(MO)	MA-ACS(CC)	MA-ACS(CC-MO)
$0\% - 1\%$	76,92%	84,62%	92,31%	100%
$1\% - 10\%$	23,08%	15,38%	7,69%	0%

Tabela 7.8: Porcentagem de instâncias em relação ao erro.

Pela Tabela 7.8 é possível verificar que o algoritmo híbrido que trabalha com a junção das duas técnicas de buscas locais obteve os melhores resultados. Por exemplo das 13 instâncias testadas, ele obteve 13 instâncias com um erro entre $0\% - 1\%$ do *makespan* ótimo.

A comparação dos resultados com outras abordagens da literatura muitas vezes é complicado, pois o banco de dados do OR-Library é grande e nem sempre se tem as mesmas instâncias utilizadas nos testes. Além disso, a maioria dos trabalhos encontrados na literatura apresentam seus resultados *defuzzificados*, sendo difícil estabelecer uma comparação dos resultados.

Desta forma, dentre os trabalhos citados na revisão bibliográfica, o trabalho de Fortemps 1997 utiliza 3 instâncias iguais as que utilizamos nos testes, porém os resultados apresentados são *defuzzificados*. Além disso, o autor apresenta uma única solução para cada uma das instâncias testadas, enquanto que as abordagens propostas nesta tese apresentam um conjunto solução com alto grau de otimalidade. Para a instância *ft06*, de 6 máquinas e 6 tarefas, o trabalho de Fortemps apresenta como melhor resultado encontrado o valor *defuzzificado* de 55,02. Neste trabalho, as abordagens encontram *makespan fuzzy*

de [50,6 55 57,75], o que defuzzificando resulta em 54,4. Além disso, a abordagem MA-ACS(CC-MO) encontra 4 soluções com mais de 80% de possibilidade de serem ótimas. Para a instância *la11*, Fortemps encontra como melhor resultado o *makespan* de 1222, enquanto que as abordagens desta tese encontram o *makespan fuzzy* de [1124,24 1222 1283,1], o que defuzzificando resulta em 1209,8. Além disso, as abordagens encontram até 20 indivíduos com mais de 80% de possibilidade de serem ótimos. Para a instância *la12*, Fortemps encontra o *makespan* de 1041, enquanto que as abordagens propostas encontram *makespan fuzzy* de [955,88 1039 1090,95], o que defuzzificando resulta em 1028,6. Além disso, a abordagem MA-ACS(CC-MO) encontra 22 indivíduos com mais de 80% de possibilidade de serem ótimos, isso para uma população com 40 indivíduos.

No trabalho de Niu et al. 2008 são utilizadas também 3 instâncias iguais as que foram utilizadas nos testes. Eles utilizam um algoritmo de enxames de partículas combinado com operadores genéticos denominado GPSO para encontrar o *makespan* do problema *fuzzy*. Seus resultados também são apresentados defuzzificados. Para a instância *ft06*, o melhor *makespan* encontrado por Niu foi de 56,08, enquanto que a abordagem MA-ACS(CC-MO) encontra um *makespan fuzzy* de [50,6 55 57,75], o que defuzzificando resulta em 54,4 e 4 indivíduos com mais de 80% de possibilidade de serem ótimas. Para a instância *la01*, os autores encontram *makespan* defuzzificado de 684,4, enquanto as abordagens desta tese encontram um *makespan* de [612,72 666 699,3], o que defuzzificando resulta em 659,3 e 16 indivíduos com alto de qualidade. Para a instância *abz6*, com 10 máquinas e 10 tarefas, os autores encontram um *makespan* de 969,8, enquanto que a abordagem MA-ACS(CC-MO) encontra um *makespan fuzzy* de [870,03 946 993,3], o que defuzzificado resulta em 936,44 e ainda encontra 15 escalonamentos com alto grau de possibilidade de serem ótimos.

Em Lin 2000, os autores utilizam duas formas para resolver o problema, uma denominada F^*_{max} e a outra F^o_{max}, em ambas os números *fuzzy* são associados a um número *crisp* para encontrar o *makespan* associado do problema. São utilizadas 3 instâncias iguais as que utilizamos nos testes. Assim como nas abordagens citadas anteriormente, os resultados são apresentados defuzzificados. Para a instância *ft06*, Lin apresenta como *makespan* o valor de $F^*_{max} = 52,261$ e $F^o_{max} = 54,217$. A abordagem MA-ACS(CC-MO) encontra não apenas uma única solução, mas um conjunto de 4 soluções com alto grau de qualidade, sendo que o melhor *makespan fuzzy* encontrado é de [50,6 55 57,75], o que defuzzificando resulta em 54,4. Para a instância *la01*, Lin apresenta como melhor *makespan* encontrado o valor de $F^*_{max} = 660,162$ e $F^o_{max} = 664,254$. Já as abordagens propostas nesta tese encontram um *makespan fuzzy* de [612,72 666 699,3] o que defuzzificando resulta em 659,3 e 16 indivíduos com mais de 80% de possibilidade de serem ótimos. Para a instância *la06* Lin encontra $F^*_{max} = 919,783$ e $F^o_{max} = 923,632$. As abordagens desta tese encontram *makespan* de [851,92 926 972,3], o que defuzzificando resulta em 916,74, sendo que a abordagem MA-ACS(CC) encontra 20 indivíduos com alto grau de otimalidade.

Não foi possível estabelecer uma comparação entre os resultados obtidos pelas instâncias testadas neste trabalho e as instâncias de González et al. 2007, pois o conjunto de instâncias utilizadas nos testes são diferentes. Mas pode-se estabelecer uma comparação em relação as abordagens. Em González, os autores consideram o problema *fuzzy* como uma família de N problemas *crisp*. Desta forma, os autores encontram N *makespan's*

crisp e resolvem o problema minimizando o *makespan* esperado ($E[C_{max}(\sigma, p, \nu)]$).

Uma vantagem das abordagens propostas nesta tese é que encontramos não apenas uma única solução para o problema, mas um conjunto-solução com alto grau de otimalidade, mantendo a incerteza em todo processo de resolução do problema.

7.5 Conclusões

Um ponto forte e interessante das abordagens propostas nesta tese é que os algoritmos híbridos implementados possuem a vantagem de resolver o problema do *job shop* com parâmetros *fuzzy* na sua íntegra, sem a necessidade de métodos de *defuzzificação* ou comparação de números *fuzzy* para encontrar o *makespan* (atingir seu objetivo). Uma outra vantagem bastante interessante e útil da abordagem proposta nesta tese, é que os algoritmos são capazes de encontrar um conjunto de soluções com alto grau de possibilidade de serem ótimas.

Capítulo 8

Conclusões e Trabalhos Futuros

Neste trabalho, foi abordado o problema de escalonamento de tarefas do tipo *job shop* em dois contextos. O primeiro no qual o tempo de processamento das tarefas são representados por números *crisp*, ou seja, números conhecidos, e o segundo no qual o tempo de processamento das tarefas é representado por números incertos (*fuzzy*). Em ambos os contextos o objetivo do problema é encontrar um escalonamento ou um conjunto-de-escalonamentos com a finalidade de minimizar o tempo total de atraso (*makespan*) do problema.

Uma grande quantidade de problemas reais podem ser representados na forma de grafo. Esta abordagem permite obter uma melhor manipulação das entidades envolvidas, de forma que a implementação de métodos para solucionar os problemas por meio desta modelagem torna-se mais simples e intuitiva. Desta forma, neste trabalho foi utilizado como representação do problema de *job shop* a modelagem por meio de grafo disjuntivo. Uma vantagem desta representação é que ela dá subsídios para o desenvolvimento de procedimentos como movimento, vizinhança e diversidade que promovem alta exploração e explotação da superfície de busca.

Muitos métodos propostos na literatura baseiam-se na condição de que todos os dados do problema são conhecidos. Entretanto, esta suposição pode trazer dificuldades na prática, pois, em problemas reais, podem haver incertezas em um número de fatores, como disponibilidade de equipamentos, tempo de processamento, tempo de transporte e custos. Desta forma, o uso da teoria clássica torna-se incapaz de resolver estes tipos de problemas. Uma maneira de contornar estas dificuldades é fazer uso de modelos que utilizam a teoria dos conjuntos *fuzzy*, pois estes deixarão o problema mais próximo da realidade.

Baseando-se em técnicas da computação bio-inspirada, um algoritmo híbrido que trabalha com a junção das metaheurísticas de otimização por colônia de formigas e algoritmo genético foi proposto para resolver o problema de escalonamento *job shop* com parâmetros *crisp* e com parâmetros *fuzzy*, cujo objetivo é explorar o espaço de busca de soluções de forma eficiente visando encontrar soluções de qualidade e de forma a contornar a questão da complexidade.

Na resolução do problema de escalonamento do tipo *job shop* com parâmetros precisos, foi utilizado o algoritmo de otimização por colônia de formigas para gerar a população inicial do algoritmo memético e foram implementadas três técnicas de buscas locais.

Para calcular o *makespan* do problema, foi utilizado o algoritmo clássico de Ford-Moore-Bellman com modificações.

No problema *job shop* com parâmetros *fuzzy*, foram utilizados números triangulares *fuzzy*. Para gerar a população inicial do algoritmo híbrido foi utilizado o algoritmo de otimização por colônia de formigas e para calcular o *makespan* foi utilizado o algoritmo proposto por Hernandes (2007) com algumas modificações. Este algoritmo é baseado no clássico de Ford-Moore-Bellman e é proposto para números triangulares *fuzzy*, porém sua extensão para números trapezoidais é simples e imediata (Hernandes 2007). Uma vantagem do algoritmo para calcular o *makespan* é que este pode ser implementado utilizando diferentes relações de ordem. A relação de ordem utilizada neste trabalho encontra um conjunto-solução de escalonamentos com alto grau de qualidade. A noção de escalonamento ótimo é avaliado seguindo um critério de ordenação entre os valores dos escalonamentos. Esse critério é importante e pode ser utilizado em situações nas quais se têm vários escalonamentos e se quer saber qual é o melhor entre eles. Muitos trabalhos da literatura utilizam índices de *defuzzificação* para assim resolver um problema clássico (*crisp*). Outros utilizam algum método de ordenação (rankeamento) de números *fuzzy* que associa a cada número *fuzzy* um valor *crisp* e com este valor resolvem um problema clássico associado. Ou ainda, utilizam métodos de ordenação lexicográfica ou comparação baseada em α-cortes com o objetivo de otimizar um escalonamento em termos do *makespan fuzzy*. Uma outra vantagem do algoritmo proposto é que ele trata o problema *fuzzy* na sua íntegra, mantendo a incerteza em todas as etapas do processo de resolução.

Os resultados obtidos pelas abordagens propostas nesta tese estão dentro dos objetivos esperados. Estes objetivos não se resumem apenas a elaboração de abordagens de resolução do problema de *job shop* com o objetivo de minimizar o *makespan*, mas uma abordagem capaz de demonstrar eficiência em diferentes especificidades de teste, encontrando soluções diversificadas e de alta qualidade.

Portanto, a principal contribuição desta tese foi a construção de uma nova metodologia de resolução do problema de *job shop*, aliando-se às importantes características de robustez e eficiência dos algoritmos de otimização por colônia de formigas e genéticos de forma a obter soluções de qualidade.

8.1 Trabalhos Futuros

Algumas sugestões que podem ser utéis em trabalhos futuros são apresentadas a seguir:

- A utilização de outras técnicas da computação evolutiva como algoritmos co-evolutivos podem ser eficientes para resolver os problemas estudados.
- Novas técnicas de seleção e manutenção de diversidade podem ser adaptadas ao algoritmo memético de forma a melhorar a qualidade dos resultados.
- As técnicas de busca local são realizadas apenas no melhor indivíduo da população sendo que, este novo indivíduo só é aceito para a próxima geração se melhorar a qualidade da solução corrente, caso contrário ele é descartado. é possível deixar esse

novo indivíduo na população, pois esse processo pode deixar o algoritmo com maior variabilidade, possibilitando obter resultados melhores.

- A utilização de outras técnicas de relação de ordem para o problema *fuzzy* é um outro caso que pode ser estudado, assim como a utilização de novos índices de comparação entre números *fuzzy* para efeito de comparação com o adotado.

- Devido aos bons resultados obtidos pela aplicação da metaheurística híbrida para resolver o problema de *job shop* com parâmetros *fuzzy*, pode-se propor a implementação da mesma para resolver outros problemas reais, tais como problemas de transporte e similares.

Referências Bibliográficas

Adams, J., Balas E., Zawack D. (1988). *The shifting bottleneck procedure for job shop scheduling*, Management Science 34, 391-401.

Almeida Applegate, D., Cook W., (1991). *A computational study of the job-shop scheduling instance*, ORSA Journal on Computing 3, 149-156.

Balas, E. (1969). *Machine scheduling via disjunctive graphs*. Operations Research, vol. 17, 941-957.

Bellman, R. E. (1958). *On a routing problem*, Quarterly Applied Mathematics (16): 87–90.

Blazewicz, J., Domschke, W., Pesch, E. (1993). *The job shop scheduling problem: Conventional and new solution techniques*. European Journal of Operational Research, 93(3), 1-33.

Blazewicz, J. et al.(1996). *Scheduling Computer Manufactoring Process*. Springer-Vertag, Berlin.

Bonabeau, E., Dorigo, M., Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York.

Bonfim, T. R. (2006). *Escalonamento Memético e Neuro-Memético de Tarefas*. Tese de Doutorado, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

Bortolan, G., Degani, R. (1993). *A review of some methods for ranking fuzzy subsets, in: Dubois, D., Prade, H., Yager, R. (eds.)*, Readings in Fuzzy Sets for Intelligent Systems, Morgan Kaufmann, San Francisco, CA, pp. 149-158.

Carvalho, M. B., Yamakami, A., Bonfim, T. R. (2010). *Ant Colony Optimization Combined with Genetic for the Job Shop Scheduling Problem*. ALIO/INFORMS International Meeting, Buenos Aires, Argentina.

Castro, L. N. (2001). *Engenharia Imunológica: Desenvolvimento e Aplicação de*

Ferramentas Computacionais Inspiradas em Sistemas Imunológicos Artificiais. Tese de Doutorado, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas. Deneubourg, J. L., S. Aron, S. Goss, and J. M. Pasteels. (1990). *The self-organizing exploratory pattern of the argentine ant*. Journal of Insect Behavior, vol. 3, No. 2, 159- 168.

Darwin, C. (1859). *On The Origin of Species*. 1 st edition, Harward University Press. MA.

Dorigo, M. (1991). *The ant system: an autocatalytic optimization process*. Technical Report Revised. Dipartimento di Elettronica, Politecnico di Milano, Itália.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. Ph.D.Thesis, Politecnico di Milano, Italy.

Dorigo, M., and Gambardella, L. M. (1997). *Ant colonies for the traveling salesman problem*. BioSystems vol. 43, n.2, pp. 73-81.

Dubois, D., Fargier, H., Prade, H. (1995). *Fuzzy constraints in job-shop scheduling*, Journal of Intelligent Manufacturing, 6(4): 215-234.

Dubois, D. and Prade, H. (1980). *Fuzzy Sets and Systems: Theory and Applications*, Academic Press.

Fisher, H., Thompson, G. L. (1963). *Probabilistic learning combination of local job shop scheduling rules*. Industrial Scheduling, pp. 225-251.

Fortemps, P.(1997). *Jobshop scheduling with imprecise durations: A fuzzy aproach*. IEEE Trans. Fuzzy Syst., vol.(4), pp. 557-569.

Gary, M.R., Johnson, D.S. (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman.

Gomide, F., Pedrycz, W. (1998). *An Introduction to Fuzzy Sets: Analysis and Design*. A Bradford Book.

González Rodrígues, I., Vela, C. R., Puente, J. (2007). *A memetic approach to fuzzy job shop based on expectation model*. In: Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE, London, pp. 692-697.

Goss, S., Aron, S., Deneubourg, J., and Pasteels, J. (1989). *Self-organized shortcuts in the Argentine ant*. Naturwissenschaften, 76(12), pp. 579-581.

Hernandes, F. (2007). *Algoritmos para Problemas de Grafos com Incertezas*. Tese de doutorado da Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. 2 edn, MIT Press, MA, USA.

Lawrence, S. (1984). *Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Lei D. (2010). *A genetic algorithm for flexible job shop scheduling with fuzzy processing time*. International Journal of Production Research, 48(10), pp. 2995-3013.

Lenstra, J. K., Rinooy Kan, A. H. G. (1977). *Complexity of machine scheduling problem*, Annals of Discrete Mathematics 1, 343-363.

Lin, F.-T. (2000). *Fuzzy job-shop scheduling based on ranking level* $(\lambda, 1)$ *interval-valued fuzzy numbers*, IEEE Trans. Fuzzy Syst., vol.10(4), pp. 510-522.

McCulloch, W. S., and Pitts, W. H. (1943). *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, 5, pp. 115-133.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer- Verlage, Berlim.

Moscato, P., Normam, M. (1992).*A memetic approach for the travelling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems*. Proc. Intl. Conf. on Parallel Computing and Transportation Applications.

Niu Q., Jiao B., Gu X. (2008). *Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time*. Applied Mathematics and Computation, 205 (1), pp. 148-158.

Okada, S. e Soper, T. (2000). *A shortest path problem on a network with fuzzy arc lengths*. Fuzzy Sets and Systems (109), pp. 129-140.

Okada, S. (2001). *Interactions among paths in fuzzy shortest path problems*. 9th International Fuzzy Systems Association World Congress. pp. 41–46.

Sugeno, M. (1985). *An introductory survey of fuzzy control*. Inf. Science, (36), pp. 59-83.

Tamura, N.(2007). OR-library. Available online at: http://bach.istc.kobe-u.ac.jp/csp2sat/jss/. Acessado em 15 julho de 2007.

Takahashi, M. T. (2004). *Contribuições ao Estudo de Grafos Fuzzy: Teoria e Algoritmos*. Tese de Doutorado, Faculdade de Engenharia Elétrica e de Computação/UNICAMP.

Udomsakdigool, A., Kachitvichyanukul, V. (2008). *Multiple colony ant algorithm for job-shop scheduling problem*. International Journal of Production Research, 46(15), (pp. 4155-4175).

Zadeh, L. (1965). *Fuzzy Sets*. Information and Control, (8), pp. 338-353.

Yamada, T., Nakano, R.(1997). *Genetic Algorithms for Job-Shop Scheduling Problems*. Proc. of Modern Heuristics for Decision Suport, 67-81, UNICOM seminar, London.