UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

PROPOSTA DE SOLUÇÃO DE PROBLEMAS DE *SCHEDULING* CONSIDERANDO POSSIBILIDADE DE TERCEIRIZAÇÃO USANDO A TÉCNICA DE OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

ROBERTO FERNANDES TAVARES NETO

SÃO CARLOS
2010

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

PROPOSTA DE SOLUÇÃO DE PROBLEMAS DE *SCHEDULING* CONSIDERANDO POSSIBILIDADE DE TERCEIRIZAÇÃO USANDO A TÉCNICA DE OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

ROBERTO FERNANDES TAVARES NETO

Tese apresentada ao programa de Pós-Graduação em Engenharia de Produção para obtenção do título de Doutor em Engenharia de Produção
Orientação: Prof. Dr. Moacir Godinho Filho

SÃO CARLOS
2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária/UFSCar

T231ps

Tavares Neto, Roberto Fernandes.
Proposta de solução de problemas de scheduling considerando possibilidade de terceirização usando a técnica de otimização por colônia de formigas / Roberto Fernandes Tavares Neto. -- São Carlos : UFSCar, 2010.
129 f.

Tese (Doutorado) -- Universidade Federal de São Carlos, 2010.

1. Programação da produção. 2. Sequenciamento da produção. 3. Terceirização. 4. Formiga - comportamento - modelos matemáticos. 5. Inteligência artificial. I. Título.

CDD: 658.53 (20ª)

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO
UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO
Rod. Washington Luís, Km. 235 - CEP. 13565-905 - São Carlos - SP - Brasil
Fone/Fax: (016) 3351-8236 / 3351-8237 / 3351-8238 (ramal: 232)
Email : ppgep@dep.ufscar.br

FOLHA DE APROVAÇÃO

Aluno(a): Roberto Fernandes Tavares Neto

TESE DE DOUTORADO DEFENDIDA E APROVADA EM 25/08/2010 PELA COMISSÃO JULGADORA:

Prof. Dr. Moacir Godinho Filho
Orientador(a) PPGEP/UFSCar

Prof. Dr. Reinaldo Morabito Neto
PPGEP/UFSCar

Prof. Dr. Paulo Rogério Politano
DC/PPGEP/UFSCar

Prof. Dr. Miguel Cezar Santoro
POLI/USP

Profª Drª Roseli Aparecida Francelin Romero
ICMC/USP

Prof. Dr. Roberto Antonio Martins
Coordenador do PPGEP

Agradecimentos

Ao meu orientador, prof. Dr. Moacir Godinho Filho cuja orientação e incentivo foram essenciais para o andamento desse trabalho.

Aos professores Dra. Roseli Aparecida Francelin Romero, Dr. Paulo Rogério Politano, Dr. Reinaldo Morabito Neto, Dr. Miguel Cezar Santoro e Dr. André Ponce de Leon pelas críticas de extrema valhia, sem as quais esse trabalho não teria convergido para o formato atual.

Ao meu pai, Roberto Fernandes Tavares Filho, por me mostrar a importância social do engenheiro-pesquisador, onde a teoria e a prática caminham juntos, se complementando.

À minha mãe, Maria da Consolação G. Cunha F. Tavares, por me mostrar, através de exemplos, como uma grande pesquisadora pode também fazer a diferença além dos limites da própria pesquisa.

À minha irmã, Ana Flávia Tavares, por ser um exemplo de dedicação e amor pela profissão. Exemplo seguido pelo irmão mais velho.

Ao meu irmão, Tiago Fernandes Tavares, por ser um exemplo de como a pesquisa em engenharia é ampla e, com a base teórica correta, se obtém resultados fantásticos. Exemplo constantemente lembrado pelo irmão mais velho.

Aos colegas e amigos prof. Fábio Molina da Silva e prof. Flávio Molina da Silva, pela ajuda e pelas trocas de idéias - referentes à tese ou não - que em muito ajudaram direta ou indiretamente na finalização desse trabalho.

A todos os docentes do DEP/UFSCar que apoiaram, direta ou indiretamente, o dia-a-dia da execução dessa pesquisa.

Aos secretários do DEP/UFSCar, que sempre estiveram presentes para me ajudar em cada etapa desse trabalho.

Vai ter com a formiga, ó preguiçoso,
olha para os seus caminhos, sê sábio
(Provérbios, 6:6)

Resumo

Embora a literatura de *scheduling* seja vasta, poucas pesquisas até o momento levam em consideração a possibilidade de terceirização de tarefas. Dentre a literatura pesquisada, apenas dois trabalhos trataram deste problema multicritério, ambos para ambientes de máquina única. Juntamente com isso, uma análise preliminar da literatura pôde estabelecer o algoritmo ACO (do inglês *Ant Colony Optimization* - Otimização por Colônia de Formigas) como uma estratégia promissora para a solução de problemas combinatórios, incluindo problemas de *scheduling*. Neste cenário, a presente tese de doutoramento trata de dois problemas de *scheduling* com possibilidade de terceirização: (i) um problema de *scheduling* em ambiente de máquina única, já proposto na literatura e (ii) problema inédito de *scheduling* em ambientes *flowshop*.Em ambos os casos, são propostos algoritmos ACO inéditos (um para o problema de *scheduling* em ambientes de máquina única e um para o problema de *scheduling* em ambientes *flowshop*), que são comparados com valores ótimos obtidos através de métodos exatos. Para permitir essa comparação, são propostos três métodos exatos: (i) Um modelo de programação matemática para o problema que trata do ambiente de máquina única; (ii) Um algoritmo *branch and bound* para o mesmo problema e (iii) Um modelo de programação matemática para o problema que trata do ambiente *flowshop*. Os resultados obtidos no trabalho mostraram que ambos os algoritmos ACO propostos conseguiram respostas próximas ao ótimo. Quando se tratando de problemas de maiores dimensões, o tempo computacional necessário para a execução do algoritmo foi muito menor que o tempo computacional requerido pelos métodos exatos. No caso do problema de *scheduling* em ambientes de máquina única, ainda pode-se ressaltar que a qualidade dos resultados (em termos de resultados e tempos computacionais) foram melhores que os relatados na literatura pesquisada.

Palavras-chave: Sequenciamento e Programação de Operações; Otimização por Colônia de Formigas; *Scheduling*; Terceirização; ACO; Metaheurísticas;

Abstract

Althought the scheduling-related literature has a high level of diversity, just a small group have been considering the possibility of outsource a set of tasks. During a literature review, only two papers related to this theme were found, both dealing on scheduling projects with outsource possibilities on single-machine environments. Along with this scenario, it was possible to stablish the ACO (Ant Colony Optimization) algorithm as a promissing tecnique to solve combinatorial problems, including scheduling problems. This thesis approaches two scheduling problems with outsourcing allowed: (i) a scheduling problem in single machine manufacturing environment and (ii) a scheduling problem in a flowshop environment. For each problem, a new ACO algorithm is proposed and implemented. To verify the quality of the results, are also proposed and implemented: (i) a mathemetical programming model for the single machine environment problem; (ii) a branch and bound algorithm for the single machine environment problem and (iii) a a mathemetical programming model for the flowshop environment problem. The results shown that both ACO algorithms generate close-to-optimal results in a shorter computational time. In the case of the single machine environment problem, the presented results are best than the results related on the literature.

Keywords: Scheduling; Ant Colony Otimization; *Scheduling*, Outsourcing

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de símbolos

$batch$ Formação de lotes de processamento

$Budget$ Orçamento total disponível para terceirização

β Constante que indica a importância da informação heurística

C_i Momento de finalização da tarefa i

$\overline{C}$ Tempo médio de finalização das tarefas de uma sequência

CTV Variância das datas de finalização de uma sequência de tarefas

d_i Data de entrega da tarefa i

δ Parâmetro de custo

E_i Adiantamento da tarefa i

F_i Tempo de fluxo da tarefa i

F_m Ambiente *flowshop*

$\overline{F}$ Tempo médio de fluxo das tarefas de uma sequência

$idle$ Tempo ocioso das máquinas de um ambiente de manufatura ao executar uma sequência de tarefas

$incompatible$ Indica a existência de tarefas que não podem ser processadas no mesmo lote

J_i Tarefa de índice i

J Conjunto de tarefas

$J_{n,m}$ Ambiente *jobshop*

L_i *Lateness* da tarefa i

l_i *Lead-time* de terceirização

M **ou** C_{max} *Makespan* de uma sequência

m Número de formigas

N^* Conjunto de nós que podem ser sequenciados

η_{ij} Visibilidade da trilha que liga os nós i e j

OC Custo incremental total de terceirização

o_i Custo de terceirização da tarefa i

O_π Conjunto não ordenado de tarefas a serem terceirizadas

OD Orçamento disponível

p_{ij} Tempo de processamento da tarefa i na máquina j

$\overline{p}$ Tempo médio de processamento das tarefas de uma sequência

P_m Ambiente de máquinas paralelas idênticas

$prmu$ Indica um ambiente *flowshop* permutacional

Q_m Ambiente de máquinas paralelas uniformes/proporcionais

Q_i *Qual-run time* da tarefa i

$1/Q$ Quantidade de feromônio a ser depositada em cada trilha

R_m Ambiente de máquinas paralelas não-relacionadas/diferentes

r_i Data de liberação da tarefa i

$rush$ Indica a existência de tarefas urgentes

ρ Constante de evaporação de feromônio

s_{ij} Custo de *setup* entre as tarefas i e j

$\overline{s}$ Tempo médio de *setup* das tarefas de uma sequência

S_π Conjunto ordenado de tarefas a serem executadas no ambiente de manufatura

T_i Atraso da tarefa i

τ_{ij} Quantidade de feromônio depositada na trilha que liga os nós i e j

τ_{min} Limite inferior do nível de feromônio em cada trilha

τ_{max} Limite superior do nível de feromônio em cada trilha

w_i Importância (peso) da tarefa i

Sumário

1 Introdução

1.1 Contextualização

O presente trabalho trata de forma conjunta os assuntos terceirização e programação de operações (*scheduling*). Segundo Jian e Xu (2006), a terceirização - ou seja, a prática de atribuir tarefas para terceiros - tem sua importância reconhecida em várias empresas. Por exemplo, Cabral (2004) relata uma tendência da cadeia produtiva de pneus em se reorganizar, buscando cada vez mais a terceirização; Paulillo (1999) trata dos impactos da tercerização na agroindústria; Weihua e Xiaoqing (2007) tratam da terceirização no desenvolvimento de software. De acordo com Yadav e Gupta (2008), essa importância da terceirização tem crescido muito nos últimos anos, se tornando, portanto um tema importante de pesquisas. Esse fenômeno também é percebido por Gonzalez *et al.* (2006).

Pode-se perceber que o tema terceirização é bastante amplo. Muitos trabalhos, como o de Antelo e Bru (2010), defendem uma forte relação entre áreas estratégicas da empresa e terceirização. Os autores defendem por exemplo, que a terceirização pode ser usada como parte de uma estratégia de crescimento da empresa, que permite o aumento da capacidade produtiva com um baixo custo. Outras justificativas são encontradas na literatura. Por exemplo, Lee *et al.* (2000) definem o papel da terceirização na empresa como uma forma de permitir à empresa se forcar em suas competências centrais.

Lee *et al.* (2000) definem cinco enfoques diferentes no estudo acadêmico da terceirização em empresas: (i) **Organizacional**, que coloca a terceirização no plano estratégico da empresa, e busca entender benefícios e riscos da mesma; (ii) **Performance**, que busca medir o impacto da terceirização nas empresas; (iii) **Decisão**, que objetiva determinar as características do processo a ser terceirizado, como pessoal envolvido, forma de controle, elementos econômicos, e outros; (iv) **Contratos**, que trata especificamente do gerenciamento de contratos de terceirização; e (v) **Relações**, que busca entender as relações existentes entre a empresa e o prestador de serviço.

Dentro da literatura a respeito da terceirização, talvez o aspecto mais importante e que

está relacionado de alguma forma aos pontos (i), (iii) e (v) citados por Lee *et al.* (2000), é a decisão de terceirizar ou não uma operação. Dentro desse contexto, a grande maioria dos estudos trata apenas os fatores estratégicos e/ou econômicos dessa decisão, não levando em consideração aspectos de tempo e menos ainda a relação entre terceirização e a programação de operações(*scheduling*) na empresa-alvo.

Scheduling ou programação de operações é definida, de acordo com Morton e Pentico (1993) como a realização de um conjunto de atividades que competem entre si por um conjunto de recursos escassos. Graves *et al.* (2005) enfatizam que o estudo de problemas de *scheduling* possuem uma importância prática óbvia, e têm sido exaustivamente pesquisados desde 1950. Trabalhos como os de Gupta (2002) e Ouelhadj e Petrovic (2008) apresentam revisões da literatura sobre o assunto, ilustrando quão ampla é a literatura de *scheduling*.

Apesar disso, da mesma forma que a literatura de terceirização, que não considera aspectos do *scheduling*, também a literatura de *scheduling* pouco trata da possibilidade de terceirização. Somente recentemente, Lee e Sung (2008b) propõem um problema de *scheduling* nunca antes abordado pela literatura. Neste problema, n tarefas devem ou ser sequênciadas para execução em um ambiente de máquina única ou enviados para execução em uma planta ou terceirizada. O objetivo é minimizar a soma ponderada entre os custos de terceirização e a soma de todos os tempos de finalização das tarefas, sendo a formação do conjunto de tarefas a serem terceirizadas limitado por um custo de orçamento máximo. Para resolver este problema, Lee e Sung (2008b) se utilizam de um algoritmo de programação dinâmica. Os mesmos autores também realizam um estudo sobre o problema de minimizar o somatório ponderado entre custo total e *lateness* máximo com restrições semelhantes de orçamento disponível (LEE; SUNG, 2008a).

É exatamente nesse contexto que esse trabalho se insere, pois pretende atuar com a proposição de algoritmos para a solução de problemas de *scheduling* com possibilidade de terceirização.

Para o estudo de problemas de *scheduling*, é comum que se dividam os ambientes produtivos em quatro grupos principais: (i) ambientes de máquina única, onde todas as tarefas devem ser realizadas por uma única máquina; (ii) ambientes de máquinas paralelas, onde as tarefas devem ser realizadas em uma máquina, existindo um conjunto de máquinas que podem realizar a mesma operação; (iii) *flowshop*, onde uma tarefa possui o operações a serem realizadas em máquinas distintas, sendo que a seqüência de máquinas utilizadas por uma tarefa é a mesma; (iv) *jobshop*, onde uma tarefa possui o operações a serem realizadas em máquinas distintas, em que a sequência de máquinas utilizadas por cada tarefa não necessariamente é a mesma. No presente trabalho, serão estudados os ambientes (i) e (iii), ou seja, máquina única e

flowshop.

1.2 Problema de Pesquisa e Objetivo Geral do Trabalho

A presente tese de doutoramento trabalha com a proposta de solução para o problema de *scheduling* em que exista possibilidade de terceirização. Os problemas de *scheduling* tratados no presente trabalho são problemas em ambiente de máquina única e *flowshop* permutacional. Em ambos os casos, cada tarefa pode ser terceirizada ou feita no ambiente de manufatura específico. O objetivo de desempenho a ser avaliado é a minimização do somatório ponderado entre o custo total de terceirização e o somatório dos tempos de finalização das tarefas não-terceirizados (para o caso do problema de máquina única) e o *makespan* (para os casos dos problemas de *flowshop* permutacional). A única restrição existente é a quantidade de recursos financeiros disponíveis para terceirização. Outras restrições, como capacidades produtivas dos prestadores de serviço, são desconsideradas neste trabalho.

Ao se considerar os problemas de *scheduling* como problemas combinatórios, várias técnicas denominadas *meta-heurísticas* foram aplicadas para sua resolução. Meta-heurística é um método definido formalmente como um processo de busca em um espaço de soluções com as seguintes características: (i) é um processo iterativo; (ii) combina diferentes aspectos do problema para reforçar soluções já existentes e explorar novas soluções; (iii) utilizam estratégias de aprendizado para a determinação de soluções quase-ótimas (OSMAN; LAPORTE, 1996). Estas técnicas, inclusive, já despertaram o interesse de empresas de consultoria e serviços em pesquisa e desenvolvimento de soluções em *software* como a Tessella Support Services (http://www.tessella.com), por conseguir alcançar resultados próximos ao ótimo em tempos computacionais aceitáveis. Como exemplos de meta-heurísticas, temos algoritmos genéticos (GOLDBERG, 1989), algoritmos meméticos (MOSCATO, 1989), colônia de partículas (KENNEDY; EBERHART, 1995) (do inglês, *Particle Swarm Optimization*, PSO), colônia de formigas artificiais (do inglês *ant colony optimization*, Colorni *et al.* (1991), ACO), entre outros.

Como alguns exemplos da utilização especificamente de meta-heurísticas para *scheduling* tem-se trabalhos como Fang *et al.* (1993), que utilizam algoritmos genéticos para o problema de *scheduling* em ambientes de *open shop* e *job shop*; Xia e Wu (2006), que se utilizam de técnicas de enxames de partículas para o problema de *scheduling* no *job-shop*, Stutzle (1998) que aplica a técnica de otimização através da colônia de formigas para o problema do *flow shop*, entre outros.

Neste contexto, o presente trabalho tem como objetivo principal propor algoritmos para

a solução de problemas de *scheduling* em ambientes de máquina única e *flowshop* em que exista possibilidade de terceirização por meio da utilização da meta-heurística ACO. Esta técnica, proposta por Colorni *et al.* (1991), conforme será mostrada no capítulo 2, busca inspiração na forma de movimentação e busca por alimentos de formigas reais, que se orientam baseados em níveis de feromônios que percebem em seu ambiente, para a solução de problemas combinatórios.

1.3 Objetivos Específicos

Diante disso, os objetivos específicos do presente trabalho são:

i Determinar, dentre os principais algoritmos ACO apresentados na literatura, um algoritmo a ser utilizado para o desenvolvimento das etapas seguintes.

ii Propor um algoritmo ACO para a resolução do problema de minimização da soma poderada do somatório dos tempos de finalização de tarefas e custo total de terceirização em um ambiente de máquina única com restrição de custo máximo de terceirização (representado por $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$). Esse problema, de característica NP-Difícil, é proposto e solucionado por Lee e Sung (2008b). Nessa tese, são comparados os resultados gerados pelo algoritmo com os resultados apresentados por Lee e Sung (2008b) e por um modelo de programação matemática.

iii Propor uma extensão da abordagem baseada em ACO usada para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, propondo e resolvendo um problema que trabalhe no ambiente *flowshop* com o objetivo de minimizar o somatório ponderado entre o *makespan* das tarefas não terceirizadas e o custo total de terceirização com restrição de custo máximo de terceirização (representado por $F/prmu, Budget/(1-\delta) \cdot M + \delta \cdot OC$). A escolha do *makespan* se deu devido à ênfase que este objetivo vêm recebendo na literatura especializada. Este problema, conforme é apresentado no capítulo 5, é um problema NP-Difícil. Nessa tese, os resultados obtidos pelo algoritmo ACO implementado são comparados com valores obtidos por um segundo algoritmo baseado na heurística NEH proposta por Nawaz *et al.* (1983) e por um modelo de programação matemática.

Ao se observar os objetivos específicos, nota-se que este trabalho é o segundo trabalho que atua sobre o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, e o primeiro que se utiliza de ACO para tal. Não foi encontrada na literatura nenhuma menção ao problema mencionado no ítem (iii).

1.4 Importância do Tema

Conforme mostrado anteriormente, o tema terceirização tem sido trabalhado na comunidade científica com várias abordagens, como gerenciamento de contratos, determinação de estratégias empresariais, entre outros. A importância do próprio tema "terceirização" é visto como algo de crescente importância, seja como uma forma de delegar tarefas que não pertencem ao núcleo de competências-chave da empresa, seja como uma forma de delegar tarefas que a empresa não possui capacidade produtiva para realizar (por exemplo, durante um processo de crescimento). Conforme visto na seção 1.1, apesar da ampla literatura sobre o tema faltam trabalhos que considerem outros aspectos de terceirização com tempo ou a programação de operações. O presente trabalho busca auxiliar no preenchimento dessa lacuna.

Este trabalho também trata da minimização do tempo necessário para se realizar um conjunto de tarefas. Percebe-se que a redução do tempo na manufatura é alvo de diversas pesquisas, inclusive através de várias técnicas que não envolvem de algoritmos de *scheduling*, como por exemplo Fahimnia *et al.* (2009) e Leng e Parlar (2009). A importância de execução de tarefas em um menor tempo em um ambiente de manufatura é reconhecida por vários pesquisadores. Por exemplo, Leng e Parlar (2009) indica que tempos menores podem gerar, entre outros efeitos benéficos, previsões mais precisas e menores níveis de estoque. A competição baseada no tempo (proposto por Stalk (1988) e colocada em prática por Suri (1998) e Suri (2010)), é o principal paradigma de gestão atual que defende o tempo um objetivo de desempenho fundamental nos tempos atuais.

Adicionalmente, a possibilidade de terceirização é muito pouco explorada na literatura de *scheduling*. A partir de uma revisão de literatura, somente encontrou-se os trabalhos de Lee e Sung (2008b) e Lee e Sung (2008a), que definem um indicador que leva em consideração custos e *lead-times* de terceirização e indicadores de desempenho do ambiente de manufatura interno à empresa (por exemplo, somatório de atrasos). O presente trabalho contribui nessa direção.

Por fim, apresenta-se, como visto no capítulo 2, o uso de algoritmos ACO para a resolução de problemas de *scheduling* tem sido uma tendência crescente na literatura mundial. Este trabalho segue essa tendência.

1.5 Metodologia de Pesquisa

Para se determinar a abordagem de pesquisa do trabalho proposto, é necessário que se entenda que este trabalho parte de um modelo de um problema a ser resolvido. Este modelo

é composto por variáveis mensuráveis, como: tempo de processamento, capital disponível para terceirização e outras, que são alteradas e cujo efeito sobre a resolução do problema é mensurado, neste caso, através de uma *função objetivo*.

Ao analisar as variáveis presentes nos problemas a serem estudados, uma das premissas dessa pesquisa é que as mesmas são definidas por meio de processos externos ao processo de resolução. Da mesma forma que o problema é descrito, assume-se que não é possível interferir em suas relações e ao mesmo tempo, é possível a mensuração de todas as variáveis do problema.

Outra característica importante desse trabalho é a *replicabilidade*. Desta forma, espera-se que os algoritmos apresentados nos capítulos 4 e 5, se novamente implementados e aplicados a um conjunto de dados semelhantes, produzam resultados equivalentes.

Ao delimitar essas características da pesquisa, e observando as características apresentadas por Martins (2010), é possível definir essa pesquisa como uma pesquisa quantitativa. A pesquisa quantitativa é de grande importância para a gestão de operações. Segundo, por exemplo, Ferreira (2006), esta abordagem é vantajosa quando:

- O problema é complexo e se faz necessária uma análise quantitativa para se chegar a uma conclusão.
- O problema possui elementos de importância ou urgência – como por exemplo problemas que levam em consideração questões de segurança.
- O problema é novo, e como tal, não existe uma "base empírica" para se tomar decisões.
- O problema é facilmente automatizado, minimizando assim o tempo gasto para sua resolução.

Martins (2010) apresenta alguns dos métodos mais apropriados para a pesquisa em Engenharia de Produção utilizando uma abordagem quantitativa, como:

i Experimento.

ii Quase-Experimento.

iii Pesquisa de Avaliação (*surveys*).

iv Modelagem/Simulação.

Tanto no caso do Experimento quanto do Quase-Experimento, o foco da pesquisa se dá na delineação de uma estratégia para a determinação da relação entre um conjunto de variáveis estabelecidas na hipótese do trabalho. Ao manipular as variáveis independentes e observar o resultado obtido nas variáveis dependentes, é possível determinar uma relação de causa e efeito entre elas. No caso do Experimento, é possível que a pesquisa isole as variáveis que não fazem parte do contexto do experimento. Já no Quase-Experimento, isso não é possível.

A pesquisa de avaliação consiste em tomar o(s) respondente(s) de um questionário como instrumento de pesquisa. A pesquisa então, com posse dos dados obtidos pelas respostas dos questionários aplicados, os analisa e assim obtém informações importantes para a organização.

Por fim, na Modelagem/Simulação, tem-se uma manipulação de variáveis em um modelo abstrato, sem contato direto na realidade. O relacionamento com a realidade se dá no estabelecimento do modelo, não em sua resolução ou análise. Assim, observando a proposta desse trabalho, percebe-se que o método Modelagem/Simulação é o método mais apropriado para a condução da pesquisa.

O método de pesquisa quantitativa Modelagem/Simulação possui duas abordagens principais: a abordagem empírica e a abordagem axiomática (BERTRAND; FRANSOO, 2002). A abordagem empírica quantitativa se foca na elaboração de um modelo baseado em observações da realidade e garantia de que esse modelo é de fato representativo. Por outro lado, a abordagem axiomática quantitativa tem como norte a produção de conhecimento sobre o comportamento de variáveis do modelo estudado (MORABITO; PUREZA, 2010). No caso da presente tese de doutorado, a investigação tem como foco o modelo proposto e busca entender seu comportamento. Como não se está analisando uma situação real e se extraindo o modelo, é natural que se escolha a abordagem axiomática quantitativa para esse trabalho.

Bertrand e Fransoo (2002) subdividem as pesquisas axiomáticas em dois grupos: a axiomática descritiva e a axiomática normativa. Segundo so autores, “Na pesquisa axiomática descritiva, o processo de modelagem é central. O pesquisador obtém o modelo conceitual – geralmente da literatura – e cria um modelo científico dele. Mais ainda, o pesquisador realiza algumas análises no modelo científico para obter compreensão do comportamento deste modelo. O pesquisador tipicamente não se move para a fase de resolução do modelo. Esta extensão é realizada na pesquisa axiomática normativa, onde a resolução do modelo é o ponto central da pesquisa reportada.” (BERTRAND; FRANSOO, 2002)

Morabito e Pureza (2010) mostram alguns exemplos desses dois grupos. No caso da pesquisa axiomática descritiva, é descrito um sistema analisado por meio de um modelo analítico de teoria de filas, em que a abordagem normativa permitiria fornecer medidas de

desempenho como utilização de servidores, número médio de clientes na fila, entre outros. No caso da pesquisa axiomática normativa, os autores usam como exemplo o problema clássico de dimensionamento de lotes, onde, parâmetros de entrada como o número de tipos de produtos, o período de tempo, a demanda, a capacidade produtiva, e outros, permitem determinar os tamanhos dos lotes de produção de cada produto em cada período do horizonte de planejamento buscando obter valores mais favoráveis em um conjunto de medidas de desempenho.

Para cumprir os objetivos desse trabalho, percebe-se que o método Modelagem/Simulação, com uma abordagem axiomática normativa é o mais indicado, pois permite analisar os modelos estudados e obter estratégias para determinar soluções que melhor favoreçam os critérios de desempenho estudados.

Arenales *et al.* (2007) mostram o processo de modelagem conforme a Figura 1.1. Segundo os autores, o processo de modelagem se inicia com a análise de um problema real, que é analisado e, após a formulação (modelagem), gera um modelo matemático contendo variáveis e relações matemáticas relevantes ao problema. Com o modelo matemático descrito, são realizadas deduções e análises para se obter conclusões, como a solução do modelo e decisões provenientes do mesmo. A próxima etapa realiza a interpretação(inferência) dos dados obtidos com a resolução do modelo. Com isso, se discute qual a aplicabilidade das conclusões obtidas pela análise do modelo no sistema real. Por fim, a fase de avaliação (julgamento) analisa essas conclusões reais e verifica quais serão implementadas na situação real e indica a necessidade de se alterar a modelagem.

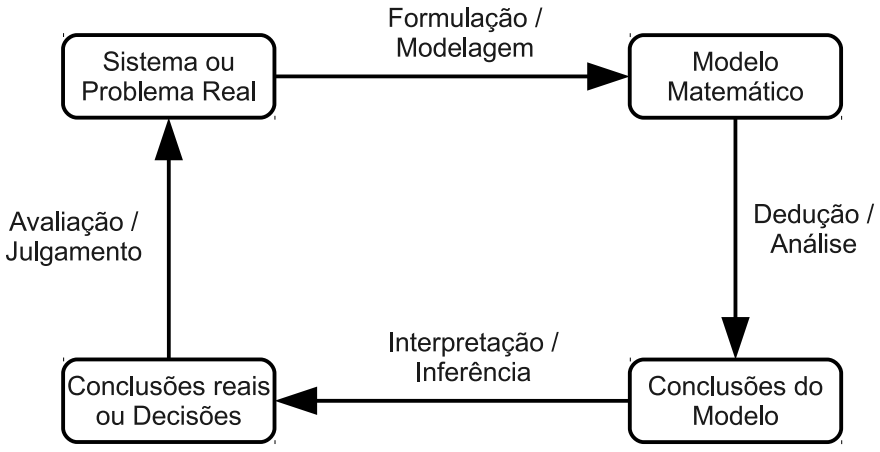


Figura 1.1: O processo de modelagem (fonte: Arenales *et al.* (2007))

Morabito e Pureza (2010) e Bertrand e Fransoo (2002) indicam que a resolução do modelo é uma das fases centrais da pesquisa axiomática. Isso se confirma na proposta desse trabalho, em que o foco será buscar uma estratégia de resolução de problemas de *scheduling* com terceirização.

Para isso, a técnica de pesquisa seguida neste trabalho consiste nos seguintes passos:

Passo 1 Determinar o estado da arte e as principais escolhas de implementação realizadas na literatura para a aplicação da meta-heurística ACO em problemas de *scheduling*;

Passo 2 Estudar o algoritmo ACO, suas principais variações e buscar estratégias de implementação, o que inclui determinar qual a variação do ACO a ser utilizada nesta pesquisa;

Passo 3 Estudar o problema de *scheduling* proposto por Lee e Sung (2008b), e propor e implementar um algoritmo ACO que permita resolvê-lo;

Passo 4 Com base no algoritmo desenvolvido no item Passo 3, e no levantamento do estado da arte realizado no Passo 2, propor um problema de *scheduling* em ambiente *flowshop* com terceirização e desenvolver uma variação de um algoritmo ACO para sua resolução;

Passo 5 Elaboração de conclusões e propostas de estudos futuros.

1.6 Estrutura do Trabalho

As etapas da pesquisa definidas anteriormente têm uma forte relação com a estrutura do trabalho em si, conforme é ilustrado na Figura 1.2.

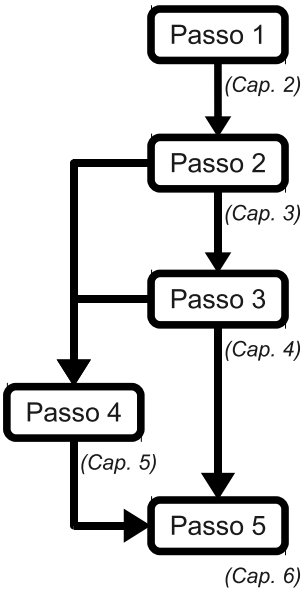


Figura 1.2: Estrutura do trabalho e sua relação com os passos da pesquisa

Resumidamente, a descrição dos próximos capítulos é a seguinte: no capítulo 2, é mostrado o referencial teórico, com o objetivo de classificar e analisar o estado da arte do tema "aplicação da meta-heurística ACO e suas variações em problemas de *scheduling*". No capítulo 3, é proposto um *framework* computacional, resultante de um estudo sobre as variações do algoritmo ACO, que nos permitiu tomar algumas decisões de projeto que influenciaram significativamente as implementaçoes realizadas no restante do trabalho. No capítulo 4, é apresentado o algoritmo ACO proposto para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$. No capítulo 5, é apresentado o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ e o algoritmo ACO implementado para sua resolução. Por fim, no capítulo 6, são tecidas as considerações finais do trabalho.

2 *Referencial Teórico: Scheduling, ACO e uma Revisão de Literatura a Respeito da Aplicação de ACO em Scheduling*

Este capítulo apresenta o referencial teórico necessário para o desenvolvimento do trabalho proposto nesta tese de doutorado. Para isso, ele é dividido em seções: na seção 2.1, são apresentados conceitos de *scheduling*; na seção 2.2, é apresentado o algoritmo ACO e suas variações; na seção 2.3 é apresentada uma revisão da literatura a respeito da aplicação de ACO em *scheduling*; e na seção 2.3.2 são tecidas algumas análises a respeito dessa revisão; por fim, a seção 2.3.5 traz algumas considerações finais do capítulo.

2.1 Programação e Sequênciamento de Operações

Inicialmente, deve-se entender que problemas de seqüênciamento e programação de operações (*scheduling*) dizem respeito à alocação em um período de tempo de tarefas em recursos produtivos (BAKER, 1943). Segundo Brucker (2007), a teoria de *scheduling* corresponde a um número virtualmente ilimitado de problemas. Alguns exemplos de livros recentes no tema são: Brucker (2007), LEUNG (2004),Pinedo (2008) e PINEDO (2009).

Brucker (2007) caracteriza um problema de *scheduling* da seguinte forma: sejam m máquinas $M_j, j = 1,2,...,m$ e n tarefas $J_i, i = 1,2,...n$. O problema de *scheduling* consiste em alocar todas as n tarefas nas m máquinas de forma a otimizar uma determinada medida de desempenho. Cada tarefa J_i possui um número de operações $O_{i1},...,O_{n_o}$, em que n_o é o número de operações das tarefas. Pode ou não haver restrições de procedência entre as operações. Uma operação n só pode ser alocada em um subconjunto de máquinas. Cada tarefa possui um conjunto de atributos, como por exemplo:

- p_{ij} que define o tempo de processamento da operação j da tarefa i. Normalmente, quando o problema de *scheduling* trata de alocação de tarefas com uma única operação, este atributo é citado na literatura como apenas p_i.
- d_i que define a data de entrega (do inglês, *due date*) da tarefa.
- r_i que define a data de liberação da tarefa (do inglês, *release date*), ou seja, a menor data em que se pode alocar uma operação da tarefa i em uma máquina qualquer.
- w_i define um peso, normalmente usado como medida de prioridade para uma função objetiva do problema.

Uma vez definidas as tarefas, é possível classificar os problemas de *scheduling*. Para tal, este trabalho usa como base o trabalho de Graham *et al.* (1979), no qual estes autores iniciam a classificação estabelecendo critérios para identificação do ambiente produtivo. Segundo estes autores, um conjunto de máquinas do ambiente produtivo podem ser:

1. **Idênticas:** se processam tarefas idênticas na mesma velocidade;
2. **Uniformes:** se processam tarefas idênticas em velocidade dependente da máquina;
3. **Não relacionadas:** se processam tarefas idênticas cujo tempo de processamento varia de acordo com a máquina e com a tarefa que está sendo executada.

Essas três categorias são encontradas na literatura com uma nomenclatura diferente. Vários autores, como Pacheco e Santoro (1998), usam os termos **Iguais**, **Proporcionais** e **Diferentes** para o que Graham *et al.* (1979) chama de máquinas Idênticas, Uniformes e Não relacionadas, respectivamente.

Além disso, Graham *et al.* (1979) classificam os ambientes produtivos como:

1. **Ambiente de máquina única:** quando existe apenas uma máquina que realiza todas as tarefas, sendo que cada tarefa possui uma única operação;
2. **Ambiente de máquinas paralelas:** quando m máquinas podem ser usadas para realizar todas as tarefas com uma única operação. As máquinas paralelas podem ser idênticas ou uniformes;
3. **Ambiente *flow shop*:** quando as tarefas possuem mais de uma operação a ser realizada, e todas as operações de todas as tarefas devem ser realizados na mesma sequência;

4. **Ambiente *job shop*:** quando as tarefas possuem mais de uma operação a ser realizada, e as operações das tarefas não seguem a mesma sequência;

Ao atribuir uma operação de uma tarefa J_i em uma máquina M_j em um tempo $t_0 >= r_j$ qualquer, pode-se determinar um conjunto de critérios de desempenho. Entre eles:

- $C_i = t_0 + p_i$ é o instante de conclusão da tarefa;
- $L_i = d_i - C_i$ é o *lateness* da tarefa;
- $E_i = max\{L_i, 0\}$ é o adiantamento (do inglês *earliness*) da tarefa;
- $T_i = max\{C_i - d_i, 0\}$ é o atraso (do inglês *tardiness*) da tarefa;
- $M = max\{C_i\}$ é o *makespan* de uma sequência, às vezes também referenciado na literatura como C_{max};

Após apresentar estes elementos comuns em grande parte dos problemas de *scheduling*, a seguir é mostrado o padrão de classificação estabelecido por Graham *et al.* (1979), que codifica um problema de *scheduling* no formato $\alpha/\beta/\gamma$.

O campo α diz respeito ao ambiente e pode assumir os seguintes valores:

1. *1*: para ambientes de máquina única
2. *Pm*: para ambientes com *m* máquinas paralelas idênticas
3. *Qm*: para ambientes com *m* máquinas paralelas uniformes
4. *Rm*: para ambientes com *m* máquinas paralelas não-relacionadas
5. *Mm*[1]: para ambientes com *m* máquinas paralelas que trabalham com a formação de lotes
6. *Fm*: para ambientes de *flow shop* com *m* máquinas
7. *Jm*: para ambientes de *job shop* com *m* máquinas

[1] A notação *M* para representar máquinas que podem trabalhar com lotes é utilizada por Li *et al.* (2009). Porém, seguindo grande parte da literatura de *scheduling*, este trabalho usa *M* para representar o critério de desempenho *makespan*, exceto na descrição do campo α.

O segundo campo, β diz respeito às características especiais da tarefa. Se deixado em branco, significa que as tarefas estão disponíveis imediatamente, não podem ser interrompidas e não possuem restrições de procedência entre si (embora as *operações* de cada tarefa possam ter relações de precedência). Por fim, o campo γ diz respeito à função objetivo do problema, normalmente relacionada a critérios de desempenho que se deseja otimizar. Como exemplo, pode-se listar alguns problemas de *scheduling*:

1. $1//\sum T_i$ é o problema de minimização de atraso total em ambiente de máquina única;
2. $1//\sum w_i \cdot T_i$ é o problema de minimização do total de atraso ponderado em ambiente de máquina única;
3. $Pm//\sum w_i \cdot T_i$ é o problema de minimização do total de atraso ponderado em ambiente de máquinas paralelas idênticas;
4. $Fm//\sum w_i \cdot T_i$ é o problema de minimização do total de atraso ponderado em ambiente *flow shop*;

Outra codificação bastante usada em problemas de *scheduling* é a proposta por Conway *et al.* (1967). Os autores definem uma notação $A/B/C/D$ onde:

1. A descreve a forma com que as tarefas chegam ao sistema. Pode ser ou o número de tarefas que se deseja sequênciar (para problemas estáticos) ou uma distribuição probabilística que indica a política de chegada de novas tarefas (para problemas dinâmicos).
2. B descreve o número de máquinas existentes.
3. C descreve o padrão do fluxo das tarefas (*flow shop*, *job shop*, etc).
4. D descreve a função objetiva do problema

Embora a notação proposta por Conway *et al.* (1967) seja muito comum na literatura de *scheduling*, grande parte dos trabalhos encontrados que tratam do uso de colônia de formigas para a resolução destes problemas usam exclusivamente a notação $\alpha/\beta/\gamma$. Desta forma, este projeto adotará esta notação para futuras referências de problemas de *scheduling*.

O quadro 2.1 lista algumas das principais variáveis usadas para a definição de problemas de *scheduling* utilizadas no presente trabalho.

Sigla	Significado
m	Número de máquinas
M_j	Máquina j
n	Número de tarefas
J_i	Tarefa i
p_{ij}	Tempo de processamento da operação da tarefa i a ser executada na máquina j
p_i	Tempo de processamento da tarefa i (para problemas de ambientes de máquina única
d_i	Data de entrega da tarefa i
r_i	Data de liberação da tarefa i
w_i	Peso da tarefa i
C_i	Data de conclusão da tarefa i
L_i	*Lateness* da tarefa i
E_i	Adiantamento da tarefa i
T_i	Atraso da tarefa i
M ou C_{max}	*Makespan* da sequência da tarefa i

Quadro 2.1: Principais variáveis usadas na definição de problemas de *scheduling*

Na próxima seção, são apresentadas algumas características de algoritmos ACO presentes na literatura.

2.2 A otimização por colônia de formigas

2.2.1 Fundamento do algoritmo ACO

Morton e Pentico (1993) descrevem quatro abordagens distintas para a resolução de problemas de *scheduling*:

1. Métodos para treinamento de especialistas humanos.
2. “Sistemas especialistas” que imitam especialistas humanos.
3. Métodos matemáticos ou computacionais de *scheduling*, sejam estes exatos ou aproximados.

4. Abordagens híbridas, que combinam vários métodos distintos.

Dentro do conjunto de métodos computacionais utilizados para a resolução de problemas de *scheduling*, temos os métodos heurísticos. Segundo Colin (2007), heurísticas são procedimentos generalistas que são adaptados à cada problema que se deseja atuar. Sua aplicação, ainda segundo Colin (2007), não é decorrente da otimalidade da função (pois métodos heurísticos não garantem a otimização do problema em que são aplicadas), e sim por sua flexibilidade e menor custo computacional. Esse menor custo computacional torna o uso de heurísticas possível para a resolução de problemas cuja solução por meio de métodos exatos (como por exemplo programação inteira e métodos *branch and bound*) seria proibitiva devido ao tempo necessário para a produção da resposta.

O trabalho desenvolvido por Colorni *et al.* (1991), propôs um método heurístico para a resolução de problemas combinatórios baseados no mesmo mecanismo usado por formigas reais para a determinação do caminho mais seguro entre seu ninho e a fonte de comida. Quando se busca a definição de uma "formiga" na ótica de sistemas de formigas (AS - *Ant Systems*), é interessante citar o trabalho de Dorigo *et al.* (1996), que as define como agentes com capacidades muito modestas que, de alguma forma, buscam imitar o comportamento de formigas reais. Para compreender o conceito de agentes computacionais, cita-se Franklin e Graesser (1997), que definem um agente autônomo como um sistema simulado dentro e fazendo parte de um ambiente, que sente e age sobre este mesmo ambiente, buscando objetivos próprios e afetando sua percepção futura do meio.

No caso dos algoritmos otimizadores baseados em colônias de formigas, o meio em que os agentes estão inseridos é representado através de um grafo composto normalmente de duas informações numéricas:

1. Uma diretamente relacionada com características do problema a serem resolvidos (denominada pela literatura como a *visibilidade*). Esta informação η_{ij} é referente à ligação entre os nós i e j do grafo percorrido pelas formigas. Como exemplos de informação fixa, temos distâncias entre cidades (para modelagem de problemas como o caixeiro viajante – TSP), tempos necessários para realização de operações (para problemas de *scheduling*), entre outros.

2. Uma relacionada com a execução do algoritmo, independente da visibilidade e modificada conforme a formiga percorre o grafo. Esta informação é denominada na literatura como "feromônios artificiais", em uma clara analogia com feromônios liberados por formigas

durante sua movimentação. Da mesma forma que o η_{ij}, a trilha de feromônio τ_{ij} é uma característica da trilha que liga os nós i e j do grafo.

O funcionamento do algoritmo AS baseia-se no conceito de inteligência emergente: o problema não é resolvido diretamente por um ou mais agentes, mas sim é resultado de um comportamento emergente das várias interações entre os agentes e o meio em que estão inseridos. Esse comportamento se dá pela comunicação entre os agentes e o meio, não da comunicação entre os agentes entre si. Segundo Dorigo *et al.* (1996), sabe-se que as formigas reais se utilizam da liberação de substâncias químicas chamadas feromônios para se comunicar. Ainda segundo os autores, enquanto uma formiga isolada se move aleatoriamente, uma formiga que encontra uma trilha de feromônios depositada anteriormente pode detectá-la e, desta forma, a probabilidade da formiga se mover por uma trilha delimitada por feromônios é maior (esta é a regra de transição). Além disso, ao escolher o caminho com feromônios, a formiga deposita mais e reforça este caminho. Assim sendo, as trilhas mais percorridas irão ter cada vez mais feromônios. Ao possuir uma quantidade maior de feromônios, essas trilhas serão mais escolhidas pelos agentes, que, durante suas passagens, irão ter seus níveis de feromônios aumentados indefinidamente, caso não exista um mecanismo de regulação deste reforço positivo. Rajendran e Ziegler (2004) definem esta ação dos feromônios como uma forma de memória adaptativa de soluções encontradas anteriormente. De acordo com Blum e Dorigo (2004) os algoritmos otimizadores que se baseiam em formigas artificiais tentam, durante sua execução, atualizar os valores dos feromônios de forma que a probabilidade de se gerar soluções de alta qualidade aumente no decorrer do tempo. Isso é possível por meio da concentração dos esforços da resolução do problema em regiões do espaço de busca, em que se supõe existir soluções de alta qualidade.

Esse comportamento é ilustrado na Figura 2.1. Nesta figura, são mostrados dois caminhos que ligam dois pontos (“Origem” e “Destino”). Inicialmente, os níveis de feromônio de ambos os caminhos (representado pelo comprimento das setas correspondentes) estão equilibrados (Fig. 2.1a), e com isso, a chance de escolha de cada caminho é igual. Assim sendo, um número igual de formigas passará pelos dois caminhos. Porém, as formigas que escolherem o caminho maior demoram mais para voltar, o que causa uma maior evaporação do feromônio por elas depositado. Computacionalmente, isso é geralmente modelado de forma a penalizar os depósitos de feromônio em caminhos com valores desfavoráveis de uma função objetivo específica do problema estudado. Em ambos os casos, o nível de feromônio do caminho mais longo se reduz em relação ao nível de feromônio do caminho mais curto (mostrado na Fig. 2.1b). Após algumas passagens das formigas (“iterações”), tem-se uma dominância do caminho mais favorável (mostrado na Fig 2.1c).

Usando esse mesmo princípio, é possível aumentar a complexidade do problema, como por exemplo no caso da figura 2.2. Neste caso, tem-se claramente uma extensão do problema mostrado na figura 2.1. O comportamento do algoritmo é análogo neste caso, inicialmente, na figura 2.2a, tem-se todos os níveis de feromônio iguais, e a chance de todas as escolhas possíveis é igual. Após um conjunto de iterações, o caminho mais curto possuirá uma maior quantidade de feromônios e a probabilidade de sua escolha se torna muito maior que os demais caminhos (2.2b).

O mesmo problema mostrado na figura 2.2 pode ser representado no formato de um grafo, conforme mostrado na figura 2.3. Neste caso, entende-se cada caminho como um arco, ou seja, um trecho por onde um agente pode percorrer entre dois nós *i* e *j*.

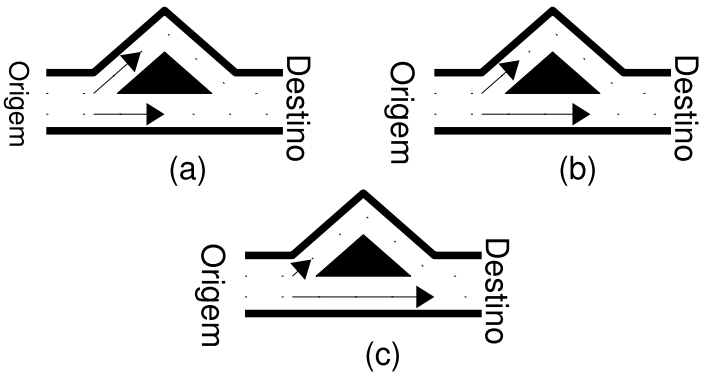


Figura 2.1: Ilustração da evolução do nível de feromônio em um caminho simples. O comprimento das setas é proporcional à probabilidade da escolha da rota.
Fonte: Adaptado de Dorigo et al. *(1996)*

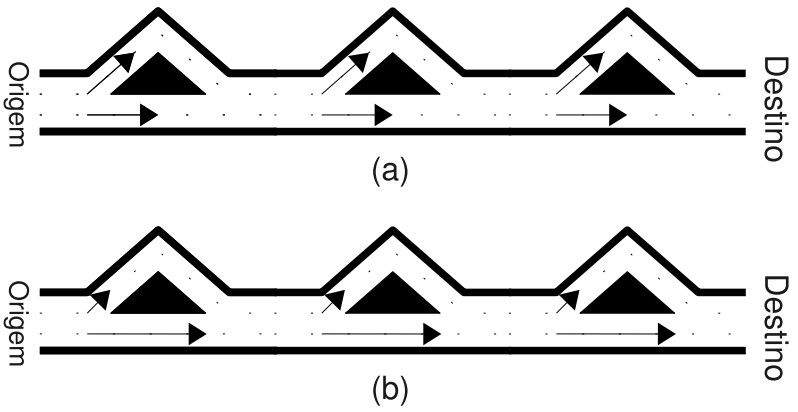


Figura 2.2: Ilustração da evolução do nível de feromônio em um caminho mais complexo.

A modelagem computacional do algoritmo AS é ilustrada no Algoritmo 2.1. Neste algoritmo, um conjunto de formigas constrói uma solução repetidas vezes. Essa solução é

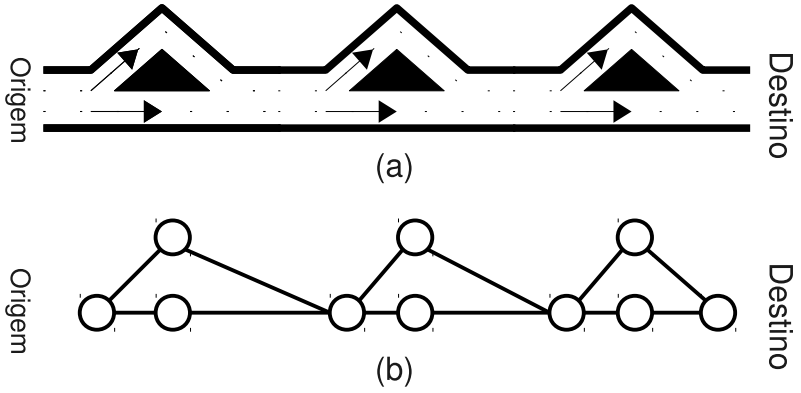


Figura 2.3: Transformação do problema apresentado na figura 2.3 em um grafo.

construída através de escolhas de caminhos seguindo uma **regra de transição**, em que após cada movimentação da formiga, é aplicada uma **regra de atualização local de feromônios**. Após todas as formigas construírem uma solução, é aplicada uma **regra de atualização global de feromônios**. Opcionalmente, aplica-se um algoritmo de **busca local**. Este ciclo continua até que um critério de parada seja satisfeito.

```
1  Inicialize
2  repita Neste nível, cada execução é chamada iteração
3  |  Cada formiga é posicionada no nó inicial
4  |  repita Neste nível, cada execução é chamado passo
5  |  |  Cada formiga aplica a regra de transição de estado para gerar a solução de forma
   |  |  incremental
6  |  |  Aplicação de regra de atualização local de feromônio
7  |  até que todas as formigas tenham construído uma solução
8  |  Aplicar regra de atualização global de feromônio
9  |  (opcional) Aplique o procedimento de busca local
10 até que a condição de parada seja satisfeita
```

Algoritmo 2.1: A estrutura geral do algoritmo AS

A **regra de transição** encontrada em vários trabalhos que tratam do algoritmo ACO é mostrada na equação 2.1 (DORIGO *et al.*, 1996):

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum\limits_{l \in N^*} \tau_{il}(t)^{\alpha} \cdot \eta_{il}^{\beta}} & \text{se} j \in N^* \\ 0 & \text{caso contrário} \end{cases} \tag{2.1}$$

Onde:

- p_{ij} representa a probabilidade da escolha do nó j pela formiga quando esta está no nó i;

- τ_{ij} representa a quantidade de feromônio depositada na trilha ij;
- η_{ij} representa a visibilidade da trilha ij;
- N^* representa o conjunto de nós que podem ser escolhidos;
- α e β são parâmetros.

A determinação de N^* é realizada de várias formas. Colorni *et al.* (1991) propuseram um método simples, utilizado em grande parte dos trabalhos que consiste em: a cada movimentação realizada, se atualiza um conjunto de nós já visitados pela formiga (ou seja, a solução já construída). Este conjunto de nós é denominado lista tabu. Neste caso, N^* é o conjunto de todos os nós existentes no grafo que não estão na lista tabu. A forma de determinação de N^* passou por algumas variações em pesquisas posteriores à de Colorni *et al.* (1991), ao incluir regras que permitem reduzir o espaço de buscas (por exemplo, Lin *et al.* (2008) utilizam critérios de dominância para reduzir o tamanho de N^* na aplicação do ACO para um problema de *scheduling* de *flowshop* permutacional).

Percebe-se que existem dois momentos onde o feromônio pode ser atualizado: o primeiro, ocorre durante a movimentação das formigas, antes que a construção da solução seja finalizada (**atualização local de feromônios**) e o segundo, ocorre após a obtenção, por todas as formigas, de uma solução (**atualização global de feromônios**).

As equações normalmente utilizadas para a atualização local e global dos feromônios são mostradas nas equações 2.2, 2.3, 2.4 e 2.5 (DORIGO *et al.*, 1996).

$$\tau_{ij}(t+1) = \rho_0 \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \delta_{ij}^k(t) \tag{2.2}$$

$$\delta_{ij}^k(t) = \begin{cases} Q & \text{se a formiga vai do nó i ao nó k} \\ 0 & \text{caso contrário} \end{cases} \tag{2.3}$$

$$\tau_{ij}(t+1) = \rho_1 \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta_{ij}^k(t) \tag{2.4}$$

$$\Delta_{ij}^k = \begin{cases} 1/L_k & \text{se na solução construída a formiga foi do nó i para o nó j} \\ 0 & \text{caso contrário} \end{cases} \tag{2.5}$$

Onde:

- $\tau_{ij}(t+1)$ é a quantidade de feromônio existente entre os nós i e j no instante seguinte;
- $\tau_{ij}(t)$ é a quantidade de feromônio existente entre os nós i e j no instante atual;
- $0 < \rho_0 < 1$ e $0 < \rho_1 < 1$ são denominadas constantes de evaporação;
- $\delta_{ij}^{k}(t)$ é a variação do feromônio na trilha que liga os nós i e j devido à ação da formiga k;
- L_k é o inverso do valor da função qualidade do problema;
- Q é uma constante.

É importante notar que as equações 2.2 e 2.4 se diferenciam em um ponto importante: o momento de atualização do feromônio. Enquanto as modificações da equação 2.2 propõem a atualização do feromônio sendo realizado durante a movimentação dos agentes no espaço de busca (chamado de regra de atualização local de feromônios), a equação 2.4 só atua após a construção de todas as soluções da iteração (regra de atualização global de feromônios).

O comportamento previsto pelos passos 6 e 7 do algoritmo 2.1 é crucial para todo o processo de comunicação do agente. Isso porque não existe comunicação direta, e sim uma comunicação sinergética que é instintivamente utilizada para a coordenação das ações entre as formigas.

Por fim, pode-se escolher aplicar um algoritmo de **busca local**. Algoritmos de busca local analisam uma sequência já existente e tentam melhorá-la, o que é uma solução para facilitar a exploração de espaços de busca de problemas NP-Difíceis (GOLDBARG; LUNA, 2005). A importância do uso de tais algoritmos em conjunto com o algoritmo ACO já é reconhecida na literatura, e muitas vezes consideradas abordagens complementares ao ACO (GAMBARDELLA; DORIGO, 2000).

A definição e a forma de implementação do algoritmo de busca a ser usado depende de características do problema a ser resolvido (DORIGO; STUTZLE,). Conforme já foi mostrado em diversas pesquisas, o acoplamento do algoritmo ACO com mecanismos de busca local pode trazer melhoras na resposta obtida. Isso pode ser explicado por que as duas abordagens são complementares: enquanto o ACO gera uma solução inicial, o algoritmo de busca local trata de tentar otimizar esta solução (DORIGO; STUTZLE, 2006). O uso de buscas locais em conjunto com algoritmos ACO podem ser facilmente encontrados na literatura: por exemplo, Dorigo e Gambardella (1997) se utilizam do algoritmo 3-opt em um algoritmo ACO para a resolução do problema do caixeiro viajante assimétrico; Gambardella e Dorigo (2000) propõem um algoritmo de busca local para o problema de atribuição quadrática; T'kindt *et al.* (2002) se utilizam do algoritmo *adjacend pairwise interchange* (API) na resolução de problemas *flowshop*; Xing *et*

al. (2008) se utilizam de um algoritmo próprio de busca local para a resolução de um problema de *scheduling* em ambiente *job shop*.

2.2.2 Os principais algoritmos ACO existentes na literatura

Nos casos estudados por Colorni *et al.* (1991), o algoritmo AS obteve resultados promissores. Porém, é bem conhecido que o mesmo, sem mudanças, possui uma eficiência menor que outras técnicas.

Stovba (2005) cita como causas da ineficiência do AS três fatores:

1. A melhor solução pode ser perdida devido à regra probabilística de seleção de rotas;
2. A convergência de uma solução próxima a ótima é baixa, devido à contribuição aproximadamente igual de soluções boas e ruins na atualização dos níveis de feromônios.
3. A memória da colônia armazena variantes obviamente não-promissoras, o que nos leva a um espaço de busca muito extenso quando tratamos de problemas multidimensionais.

Para resolver estes problemas, Stovba (2005) e Dorigo e Blum (2005) mencionam algumas técnicas para melhorar o algoritmo AS:

- "Elite Ants" (ASElite): Mencionada pela primeira vez em Dorigo *et al.* (1996), esta técnica se diferencia do AS original ao permitir o depósito de feromônio apenas nos melhores caminhos encontrados.
- Bullnheimer *et al.* (1997) propõem uma variação – o *Ant System with Elitist Strategy and Ranking* (ASrank). Este algoritmo permite que apenas um conjunto de melhores soluções sejam utilizadas para atualizar o feromônio.
- Gambardella e Dorigo (1996) e Dorigo e Gambardella (1997) propõem o *Ant Colony System* (ACS). Neste algoritmo, assim como o ASElite, altera-se a regra de depósito de feromônio original ao permitir a atualização apenas da melhor rota. Adicionalmente, a regra de transição original é modificada. Segundo Stovba (2005), a regra de transição do ACS força a formiga a buscar a solução ótima em um espaço de busca próximo da melhor solução encontrada anteriormente.
- Stutzle e Hoos (2000) desenvolveram o *MAX-MIN Ant System* (MMAS). De acordo com os autores, o MMAS impõe limites explícitos de máximo e mínimo aos níveis de

feromônio depositados nos caminhos do grafo a serem percorridos. Adicionalmente, assim como ocorre com o ASElite, apenas o agente com a melhor solução pode atualizar os níveis de feromônio. De acordo com Dorigo e Blum (2005), juntamente com o algoritmo ACS, o MMAS é um dos algoritmos de maior sucesso.

- *Best-Worst Ant System* (BWAS), apresentado por Cordon *et al.* (2000), se diferencia do algoritmo AS em três aspectos: primeiro, a atualização do feromônio se dá por meio do reforço positivo da melhor solução e do reforço negativo da pior solução. O segundo aspecto diz respeito ao reinício do processo de busca quando se nota a estagnação. Por fim, no algoritmo BWAS, é inserido uma alteração no tratamento da matriz de feromônios.

Além da mudança das regras de comportamento da formiga ou da colônia, existem pesquisas que vão mais além, ao tratar um conjunto de colônias que atuam em execuções paralelas para a resolução do problema. Como exemplo deste conjunto de algoritmos, cita-se o trabalho de Ellabib *et al.* (2007), que mostra seis algoritmos de ACS paralelos aplicados ao problema de roteamento de veículos com janela de tempo. Segundo os autores, a execução de um conjunto de colônias para a resolução de um problema, quando combinada com uma comunicação coordenada por meio de um módulo de troca (*exchange module* – EM), pode gerar resultados mais eficientes. O módulo de troca nada mais é que um trecho do algoritmo que determina como as informações serão passadas de uma colônia para outra – ou seja, ajuda a definir os processos de cooperação de algoritmos de colônias de formigas com múltiplas colônias.

Assim, ao se analisar o trabalho de Ellabib *et al.* (2007), define-se o Módulo de Troca como uma entidade computacional capaz de armazenar e processar informações presentes no espaço de busca e informações obtidas nas c colônias em execução simultânea, com o objetivo de:

- Permitir a ação coordenada entre diferentes formigas de diferentes colônias;
- e/ou alterar o espaço de busca de uma ou mais colônias de forma a permitir alguma vantagem na execução do algoritmo (rapidez de convergência, precisão, etc)

É interessante notar que Tavares Neto (2005) também se utiliza do conceito de módulo de troca, com regras baseadas em algoritmos culturais, mas desta vez para o planejamento de rotas de robôs de inspeção. Desta forma, define-se o módulo de troca como o elemento possibilitador da implementação de algoritmos AS com múltiplas colônias.

Os algoritmos até agora citados são mostrados no quadro 2.2. Ao analisar esse quadro, percebe-se que um mesmo mote inicial – o reforço positivo por meio de feromônio simulado regulado por um decaimento padrão (“evaporação”) - produziu um conjunto abrangente de heurísticas.

Algoritmo	Fonte	Número de Colônias	Usa regra de transição original?	Usa o ciclo depósito-evaporação original?	Usa alguma regra de atualização de feromônio Local/ Global?	Uso de EM?
AS	Colorni et al., 1991	1	Sim	Sim	Sim	Não
ASElite	Dorigo et al., 1996	1	Sim	Não	Sim	Não
ASrank	Bullnheimer et al., 1997	1	Sim	Não	Sim	Não
ACS	Dorigo e Gambardella, 1997	1	Não	Não	Sim	Não
MMAS	Stützle e Hoos, 2000	1	Sim	Não	Sim	Não
BWAS	Cordon et al., 2000	1	Sim	Não	Sim	Não
HCF	Blum e Dorigo (2004)	1	Sim	Não	Sim	Não
Múltiplo AS	Ellabib et al. (2007)	>1	Sim	Não	Sim	Sim
AS-CC	Tavares e Coelho (2005)	>1	Sim	Não	Sim	Sim

Quadro 2.2: Características de alguns dos principais algoritmos ACO encontrados na literatura Fonte:Tavares Neto e Godinho Filho (2009)

2.3 A Literatura que Aplica ACO em problemas de *Scheduling*: Revisão, Classificação e Análise

Para esta revisão da literatura, buscou-se nas bases de dados *Engineering Village*[2], *Scopus*[3] e *Scielo*[4] artigos que possuíssem as palavras "*Ant Colony*" e "*Scheduling*". Dos resultados obtidos, foram filtrados aqueles artigos que tratam de *scheduling* em ambientes de máquina única, paralela, *flowshop* e *jobshop*. Outros ambientes, como *open shop* e sistemas de manufatura flexível não foram considerados. Além disso, também consultou-se as referências dos artigos encontrados com o objetivo de coletar outras pesquisas que possivelmente não se encontraram na base de dados. Buscas na base de dados *Scielo* não revelaram nenhum artigo. Este procedimento resultou em um total de 47 artigos, que foram analisados e classificados conforme mostrado a seguir.

Para apresentar tais resultados, a seção 2.3.1 indica os critérios usados para classificação dos trabalhos; na seção 2.3.2 relata os resultados obtidos; a seção 2.3.3 apresenta os trabalhos analisados; por fim, a seção 2.3.4 analisa os resultados obtidos nessa revisão da literatura.

2.3.1 Proposta de Método para Classificação dos Trabalhos

Para facilitar a análise da literatura, propôe-se um método de classificação, baseado em 6 critérios, mostrado a seguir.

Critério 1 - Problema a ser resolvido: este critério utiliza a classificação proposta por Graham *et al.* (1979), segundo o formato $\alpha/\beta/\gamma$ já discutido na seção 2.1. O quadro 2.3 descreve as siglas utilizadas;

Critério 2 - Algoritmo base utilizado: foram consideradas as possibilidades presentes na primeira coluna do quadro 2.2. Quando os autores não especificaram o algoritmo, usando apenas os conceitos de otimização por colônia de formiga, é usado o termo "ACO";

Critério 3 - Uso de posicionamento absoluto: Um algoritmo ACO pode se preocupar com o posicionamento *relativo* das tarefas (ou seja, busca descobrir a tarefa a ser sequenciada após a última tarefa selecionada). Outra forma é fazer o ACO se preocupar com o posicionamento *absoluto* das tarefas (ou seja, o algoritmo deve escolher a tarefa que deve ocupar na próxima posição da sequência que está sendo construída, sem levar em

[2]http://www.engineeringvillage.com
[3]http://www.scopus.com
[4]http://www.scielo.org

consideração as tarefas que já foram sequênciados). Estas estratégias são definidas por Liao e Juan (2007) como tarefa-a-tarefa e tarefa-para-posição. Este critério analisa se o algoritmo leva em conta o relacionamento entre um trabalho e sua posição na seqüência final, ou seja, seu posicionamento absoluto. Este critério pode receber os valores "sim" (se a estratégia tarefa-para-posição é utilizada) e "não" (caso contrário).

Critério 4 - Uso de posicionamento relativo: Esse critério verifica se o algoritmo sequencia um trabalho observando os trabalhos recém-agendados, ou seja, seu posicionamento relativo em função de outros trabalhos. Liao e Juan (2007) chamam esta característica de tarefa-a-tarefa. Este critério pode receber os valores "sim"(se a estratégia tarefa-a-tarefa é utilizada) e "não" (caso contrário).

Critério 5 - Uso de critério de dominância: Nesse critério verifica-se se o algoritmo utiliza ou não um critério de dominância para agilizar a execução do algoritmo. Este critério pode receber os valores "sim" e "não".

Critério 6 - Forma de inicialização de feromônios: Esse critério verifica como é realizada a inicialização dos feromônios. Duas possibilidades existem com relação à este critério: atribuição de valor constante (*cte*) ou regra heurística (*heu*);

Critério 7 - Função de visibilidade utilizada pelo algoritmo: Esse critério verifica como o algoritmo define a função de visibilidade, podendo ser: um valor constante determinado na fase de inicialização do algoritmo e mantido durante sua execução(*cte*), um valor que se altera conforme características da sequência parcial já construída (*f(t)*), ou apenas a visibilidade não é usada na regra de transição (*1*), neste caso, as características do problema são usualmente consideradas de duas formas: (i) através da inicialização de feromônios através de uma heurística e/ou (ii) por meio do algoritmo de busca local.

Sigla	Significado
1	Usado para identificar ambientes de máquina única
P_m	Usado para identificar ambientes de máquinas paralelas idênticas
Q_m	Usado para identificar ambientes de máquinas paralelas uniformes/proporcionais
R_m	Usado para identificar ambientes de máquinas paralelas não-relacionadas/diferentes
F_m	Usado para identificar ambientes *flowshop*
$J_{n,m}$	Usado para identificar ambientes *jobshop*
r_i	Indica tarefas com data de liberação diferentes
$batch$	Indica a possibilidade de formação de lotes de processamento
$incompatible$	Indica que um conjunto de tarefas não pode ser agrupado com outro grupo no mesmo lote de processamento
s_{ij}	Indica a existência de *setup* dependente
$rush$	Indica a existência de ordens urgentes
$prmu$	Indica um ambiente *flowshop* permutacional
Q_i	Indica o *qual-run-time*
$window$	Indica que os trabalhos possuem limites inferiores e superiores de C_i
A_{ij}	Indica que as tarefas são conhecidas após o início da operação da sequencia
nwt	Indica que não é permitido espera entre o processamento de dois trabalhos
$\sum C_i$	Indica o somatório das datas de finalização das tarefas de uma sequência
$\sum T_i$	Indica o somatório de atrasos das tarefas de uma sequência
$\sum w_i \cdot T_i$	Indica o somatório ponderado de atrasos das tarefas de uma sequência
CTV	Indica a variância das datas de finalização das tarefas de uma sequência
M ou C_{max}	Indica o *makespan* de uma sequência
$\sum F_i$	Indica o somatório dos tempos de fluxo
$idle$	Indica o tempo ocioso (não utilizado) das máquinas ao operar uma sequência de tarefas

Quadro 2.3: Principais siglas encontradas na definição de problemas de *scheduling* identificados na revisão bibliográfica

2.3.2 Resultados Obtidos

O resultado da classificação dos 47 trabalhos encontrados utilizando o método proposto é mostrado na tabela 2.1. Na próxima seção, os pontos mais importantes desses trabalhos são apresentados.

Tabela 2.1: Trabalhos classificados conforme método de classificação proposto

	Artigo	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5	Critério 6	Critério 7
1	Bauer *et al.* (1999)	$1//\sum T_i$ e $1//\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	f(t)
2	Besten *et al.* (2000)	$1//\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	cte
3	Merkle e Middendorf (2000)	$1//\sum w_i \cdot T_i$	ACO	Não	Sim	Sim	cte	f(t)
4	Merkle e Middendorf (2003)	$1//\sum w_i \cdot T_i$	ACO	Não	Sim	Sim	cte	f(t)
5	Holthaus e Rajendran (2005)	$1//\sum w_i \cdot T_i$	ACO	Sim	Não	Não	heu	1
6	Cheng *et al.* (2009)	$1//\sum T_i$	ACO	Não	Sim	Sim	cte	f(t)
7	Gagné *et al.* (2002)	$1/s_{ij}/\sum T_i$	ACO	Não	Sim	Sim	cte	cte
8	Liao e Juan (2007)	$1/s_{ij}/\sum w_i \cdot T_i$	ACO	Sim	Sim	Sim	heu	f(t)
9	Anghinolfi *et al.* (2008)	$1/s_{ij}/\sum w_i \cdot T_i$	ACS	Não	Sim	Não	cte	f(t)
10	Kashan e Karimi (2008)	$1/batch, incompatible, s_{ij}/\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	cte
11	Thiruvady *et al.* (2009)	$1/s_{ij}/M$	MMAS	Não	Sim	Sim	cte	cte
12	Cheng *et al.* (2008)	$1/batch/C_{max}$	ACO	Não	Sim	Não	heu	cte
13	Sankar *et al.* (2005)	$P_m/s_{ij}/C_{max}$	ACO	Não	Sim	Não	cte	cte
14	Behnamian *et al.* (2009)	$P_m/s_{ij}/C_{max}$	MMAS	Não	Sim	Não	cte	cte
15	Raghavan e Venkataramana (2006)	$P_m/batch, incompatible/\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	cte
16	Li *et al.* (2008)	$P_m/A_{ij}, batch, incompatible/\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	f(t)
17	Li *et al.* (2009)	$P_m/A_{ij}, Q_i, batch, incompatible/\sum w_i \cdot T_i$	ACO	Não	Sim	Não	cte	f(t)
18	Zhou *et al.* (2007)	$R_m//\sum w_i \cdot T_i$	ACS	Não	Sim	Não	heu	f(t)
19	Monch (2008)	$R_m//\sum w_i \cdot T_i$	ACO	Sim	Não	Não	heu	f(t)
20	Arnaout *et al.* (2008)	$R_m/s_{ijk}/C_{max}$	ACO	Não	Sim	Não	cte	cte
21	Arnaout *et al.* (2009)	$R_m/s_{ijk}/C_{max}$	ACO	Não	Sim	Não	cte	cte
22	Ying e Liao (2003)	$F_m/prmu/M$	ACS	Não	Sim	Não	heu	cte

Continua na próxima página

Tabela 2.1: Trabalhos classificados conforme método de classificação proposto – (cont.)

	Artigo	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5	Critério 6	Critério 7
23	Rajendran e Ziegler (2004)	$F_m/prmu/M$	MMAS	Sim	Não	Não	heu	1
24	Ahmadizar *et al.* (2007)	$F_m/prmu/M$	ACO	Não	Sim	Não	heu	f(t)
25	Chen *et al.* (2008)	$F_m/prmu/M$	ACS	Sim	Não	Não	cte	1
26	Zhou e Qingshan (2009)	$F_m/prmu/M$	MMAS	Sim	Não	Não	cte	1
27	Ying e Lin (2007)	$F_m//M$	ACS	Não	Sim	Não	heu	cte
28	Shyu *et al.* (2004)	$F_2/nwt,setup/M$	ACO	Não	Sim	Não	heu	cte
29	Yan-hai1 *et al.* (2005)	$F_m//prmu,rush,M$	ACO	Não	Sim	Não	heu	cte
30	T'kindt *et al.* (2002)	$F_2//\sum C_i$ e $F_2//Lex(C_{max},\sum Ci)$	MMAS	Não	Sim	Não	cte	cte
31	Rajendran e Ziegler (2004)	$F_m/prmu/M$ e $F_m/prmu/\sum F_i$	MMAS	Sim	Não	Não	cte	1
32	Rajendran e Ziegler (2005)	$F_m/prmu/\sum F_i$	MMAS	Sim	Não	Não	heu	1
33	Gajpal e Rajendran (2006)	$F_m/prmu/CTV$	ACO	Sim	Não	Não	cte	1
34	Li e Zhang (2006)	$F_2//\alpha\cdot\sum C_i+(1-\alpha)\cdot M$	ACO	Sim	Sim	Sim	heu	f(t)
35	Pasia *et al.* (2006)	$F_m//\sum T_i,M$	ACO	Sim	Não	Não	cte	f(t)
36	Yagmahan e Yenisey (2008)	$F_m/prmu/\sum C_i,M,\sum F_i,idle$	ACS	Não	Sim	Sim	cte	cte
37	Al-Anzi e Allahverdi (2009)	$F_2/prmu/u\cdot M+v\cdot\sum Ci$	ACO	Não	Sim	Não	cte	f(t)
38	Lin *et al.* (2008)	$F_m/prmu/u\cdot M+v\cdot\sum C_i$	ACO	Sim	Sim	Sim	heu	f(t)
39	Marimuthu *et al.* (2009)	$F_m/batch/M$ e $F_m/batch/\sum F$	ACO	Sim	Não	Não	cte	cte
40	Huang e Yang (2009)	$F_m/r_j,S_{ij}/\alpha\sum_{i=1}^{m}MIT_i+\beta\sum_{i=1}^{m}\sum_{j=1}^{B_j}\sum_{t=1}^{n}JWT_{ijt}+\gamma\sum_{i=1}^{m}T_i$	ACS	Sim	Não	Não	cte	heu
41	Yoshikawa e Terai (2006)	$J_{n,m}//M$	ACO	Não	Sim	Não	cte	cte
42	Udomsakdigool e Kachitvichyanukul (2008)	$J_{n,m}//M$	ACO	Não	Sim	Não	cte	f(t)

Continua na próxima página

Tabela 2.1: Trabalhos classificados conforme método de classificação proposto – (cont.)

	Artigo	Critério 1	Critério 2	Critério 3	Critério 4	Critério 5	Critério 6	Critério 7
43	Heinonen e Pettersson (2007)	$J_{10,10}//M$	ACO	Não	Sim	Não	cte	f(t)
44	Zhuo *et al.* (2007)	$J_{10,10}//M$	ACS	Não	Sim	Não	heu	f(t)
45	Huang e Liao (2008)	$J_{10,10}//M$	ACO	Não	Sim	Não	heu	f(t)
46	Eswaramurthy e Tamilarasi (2009)	$J_{n,m}//M$	ACS	Não	Sim	Não	cte	cte
47	Huang e Yang (2008)	$J_{n,m}/window/\sum(u \cdot E_I + v \cdot T_i)$	ACO	Sim	Não	Não	cte	cte

2.3.3 Estruturação da Revisão da Literatura Usando o Método Proposto

A seguir, são apresentados os artigos listados na tabela 2.1, divididos pelo ambiente e pela função objetivo. Desta forma, na seção 2.3.3.1, são apresentados trabalhos que aplicam o ACO em problemas de *scheduling* em ambientes de máquina única; na seção 2.3.3.2, são apresentados trabalhos que tratam de máquinas paralelas; na seção 2.3.3.3, o ambiente tratado é o *flowshop*; por fim, na seção 2.3.3.4 são apresentados os trabalhos que abordam problemas de *scheduling* em ambiente *jobshop*. Dentro de cada uma dessas seções, procurou-se reunir os trabalhos utilizando-se como base a função objetiva tratada.

2.3.3.1 *Ant Systems* aplicado à problemas de *scheduling* de máquina única

Problemas de *scheduling* em ambientes de máquina única consistem em determinar a sequência em que um conjunto de n tarefas devem ser realizadas de forma a melhorar um indicador de desempenho. Em casos onde a preempção não é permitida - ou seja, não é possível que uma tarefa seja interrompida temporariamente para a execução de outra -, existem $n!$ possibilidades de sequenciamento diferentes.

A resolução destes problemas usando a heurística ACO se inicia com a sua representação em formato de um grafo. É muito comum na literatura encontrar esta representação como um grafo de n nós totalmente interconectados (por exemplo, Merkle e Middendorf (2000) e Liao e Juan (2007)). Cada par de nós N_i e N_j são conectados por um arco de tamanho η_{ij}. A figura 2.4 mostra um grafo que representa um problema de *scheduling* em ambiente de máquina única com três tarefas.

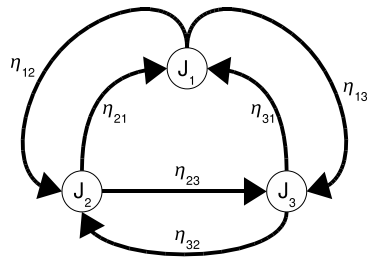


Figura 2.4: Exemplo de grafo construído para a representação de um problema de *scheduling* de 3 tarefas em ambiente de máquina única

Aplicações relacionadas à minimização de atraso

Dez artigos encontrados tratam do problema de *scheduling* em ambiente de máquina única e visam ou a minimização de atraso total ou a minimização de atrazo ponderado ($\sum T_i$ e $\sum w_i \cdot T_i$, respectivamente):

Bauer *et al.* (1999) tratam de ambos os problemas ($1//\sum T_i$ e $1//\sum w_i \cdot T_i$), usando regras heurísticas já conhecidas na literatura de scheduling para a definição de visibilidade, como a EDD (do inglês *Earliest Due Date* - menor data de entrega). Os autores usam como algoritmos de busca local as heurísticas *Adjacent Pairwise Interchange* (API) e a 2-opt.

Besten *et al.* (2000) tratam do problema $1//\sum w_i \cdot T_i$. Os resultados apresentados pelos autores mostraram que o algoritmo ACO desenvolvido foi capaz de obter a solução ótima em todas as instâncias encontradas. Os algoritmos de busca local utilizados, denominados pelos autores troca-inserção (do inglês *insert-interchange*) e inserção-troca (do inglês *interchange-insert*), aplicados cada um em metade das soluções construídas pelas formigas, mostraram serem cruciais para a eficácia do algorítmo.

Os trabalhos de Merkle e Middendorf (2000) e Merkle e Middendorf (2003) implementam melhorias no trabalho de Bauer *et al.* (1999) relacionado ao problema $1//\sum w_i \cdot T_i$. Estes autores criam uma nova regra de visibilidade (mostrada na equação 2.6) e adicionam um algoritmo que é incluído na regra de transição. Esse algoritmo estabelece critérios determinísticos para a alocação de parte dos trabalhos, que têm como objetivo limitar as possibilidades de escolhas da formiga ao evitar a construção de soluções ruins. Desta forma, pode-se dizer que este algoritmo restringe o espaço de buscas da formiga.

Um resultado interessante encontrado tanto em Merkle e Middendorf (2000) quanto em Merkle e Middendorf (2003) diz respeito à definição da visibilidade. Ambos os trabalhos se utilizam da equação 2.6. Cumpre informar que esta equação é uma adaptação da regra *Modified Due Date* (MDD) mostrada na equação 2.7. Segundo os autores, a equação 2.6 é vantajosa especialmente no seqüênciamento das últimas tarefas, quando o valor de T é muito grande.

$$\eta_{ij} = \frac{1}{max\{T + p_j, d_j\} - T} \tag{2.6}$$

$$\eta_{ij} = \frac{1}{max\{T + p_j, d_j\}} \tag{2.7}$$

Onde:

- η_{ij} representa a visibilidade do nó j quando a formiga está no nó i;
- T representa o atraso ponderado do seqüênciamento obtido até o momento;
- d_j representa a data de entrega do trabalho j
- p_j representa o tempo de processamento do trabalho j

O problema $1//\sum w_i \cdot T_i$ também é abordado por Holthaus e Rajendran (2005), que se utiliza do algoritmo *Fast Ant Colony Optimization* (FACO). Este algoritmo inicializa os feromônios usando uma heurística construtiva que leva em consideração as características das tarefas (como por exemplo, a data de entrega). A regra de transição, por outro lado, leva em consideração apenas os feromônios (ou seja, $\eta = 1$). Dessa forma, o tempo computacional necessário para a execução dessa regra de transição mais simples é menor do que, por exemplo, o tempo necessário para o cálculo da regra de transição de trabalhos como o de Merkle e Middendorf (2000).

Cheng *et al.* (2009) implementam um algoritmo híbrido para o problema $1//\sum T_i$ baseado nos resultados apresentados por Bauer *et al.* (1999). Segundo Cheng *et al.* (2009), o algoritmo por eles proposto inclui um conjunto de regras de eliminação. Estas regras são inseridas no algoritmo de Bauer *et al.* (1999) para reduzir o espaço de buscas, e melhoram a qualidade da solução final.

O trabalho de Gagné *et al.* (2002) trata de um problema semelhante, o $1/s_{ij}/\sum T_i$, ou seja, a minimização do atraso total em um ambiente de máquina única com setup dependente. Neste trabalho, faz-se uso de uma lista de candidatos, determinada através da seleção dos trabalhos com menor tempo de folga (definido na equação 2.8). Esta lista de candidatos tem como finalidade reduzir o espaço de busca, da mesma forma como Merkle e Middendorf (2000) e Merkle e Middendorf (2003) fazem com critérios determinísticos. A visibilidade η_{ij} determinada no algoritmo proposto por Gagné *et al.* (2002) é formada por: (i) uma informação relacionada ao tempo de *setup* da tarefa; e (ii) o tempo de folga da tarefa. A regra de transição se utiliza desta informação, em conjunto com a análise de informações das tarefas a serem processadas (denominadas pelos autores de *look-ahead information*).

$$m_{ij} = max\{(d_j - p_j - s_{ij}), 0\} \tag{2.8}$$

Também trabalhando com o problema $1/s_{ij}/\sum(w_i \cdot T_i)$, Liao e Juan (2007) usam o ACS de forma diferente. A primeira mudança se dá na definição da visibilidade: no caso, os autores se utilizam da regra Custo de Atraso Aparente com *Setup* (do inglês, *Apparent Tardiness Cost with Setup*, ATCS), mostrado na equação 2.9.

$$I_j(t,v) = \frac{w_j}{p_j} \cdot exp\left[-\frac{max(d_j - p_j - t, 0)}{k_1 \cdot \overline{p}}\right] \cdot exp\left[-\frac{s_{vj}}{k_2 \cdot \overline{s}}\right] \tag{2.9}$$

Onde:

- $I_j(t,v)$ é o índice do trabalho j no instante atual t no caso de que ele seja sequenciado após o trabalho v
- w_j é o peso do trabalho
- p_j é o tempo de processamento do trabalho
- d_j é a data de entrega do trabalho
- $\overline{p}$ é o tempo de processamento médio
- $\overline{s}$ é o tempo de *setup* médio
- k_1 é um parâmetro relacionado com a data de entrega
- k_2 é um parâmetro relacionado com o tempo de setup

Outra contribuição no trabalho de Liao e Juan (2007) diz respeito à forma de determinação do significado da trilha de feromônio. De acordo com a posição do trabalho em uma posição específica da sequência, não de acordo com o seu posicionamento em relação aos demais trabalhos. De acordo com os autores, esta abordagem (chamada pelos autores *job-to-position*, *trabalho na posição*) se mostrou mais eficiente que a abordagem *trabalho-a-trabalho* (do inglês, *job-to-job*) para a resolução do problema $1/s_{ij}/\sum T_i$.

O trabalho de Anghinolfi *et al.* (2008) trata do problema $1/s_{ij}/\sum w_i \cdot T_i$. Usando a mesma função de visibilidade usada por Liao e Juan (2007), esses autores apresentam um algoritmo ACS adaptativo, onde os parâmetros q_0 e β se ajustam automaticamente durante a execução das iterações. De acordo com os autores, a implementação da capacidade de ajuste automático desses parâmetros trouxe ao algoritmo a capacidade de explorar melhor o espaço de buscas.

Kashan e Karimi (2008) tratam do problema $1/batch, incompatible, s_{ij}/\sum(w_i \cdot T_i)$, ou seja, minimizar o atraso ponderado em um ambiente de máquina única que permite a formação de lotes de processamento com famílias de tarefas incompatíveis e *setup* dependente. Neste trabalho, os autores propõem dois algoritmos ACO, em que cada um trata da função de visibilidade conforme regras próprias.

Aplicações relacionadas à data de finalização das tarefas

Uma medida de desempenho muito comum em problemas de *scheduling* relacionada à data de finalização de tarefas é o *makespan* (M) de uma sequencia, por vezes também indicado

como a data de finalização máxima, C_{max}[5]. Durante o processo de busca, foram encontrados dois artigos que tratam dessa medida de desempenho. Thiruvady *et al.* (2009) abordam o problema de minimização do *makespan* em um ambiente de máquina única e tarefas com tempo de setup dependente, representado por $1/s_{ij}/M$. Para resolver o problema, os autores se utilizam de um algoritmo híbrido, composto por uma implementação do algoritmo MMAS e uma implementação da bem conhecida técnica heurística *Beam Search*[6]. Neste caso, o *Beam Search* é usado para limitar o espaço de buscas do MMAS, sendo auxiliado nesta tarefa por um terceiro algoritmo que estabelece restrições adicionais ao espaço de busca.

Cheng *et al.* (2008) apresentam um algoritmo para a resolução do problema de *scheduling* em máquina única visando a minimização do *makespan* com processamento em lote e trabalhos com tempo de processamento não-idênticos, representado por $1/batch/C_{max}$. Os autores propõem a heurística *Chaotic Ant Colony Optimization* (CACO), uma variação do algoritmo ACO que se utiliza de um operador baseado em teoria do caos como busca local. No algoritmo implementado, o ACO é utilizado para uma ordenação inicial dos trabalhos, que são posteriormente agrupados em lotes conforme a heurística BFF (*Batch First Fit*). Este algoritmo ainda define como visibilidade o inverso do tempo de processamento do trabalho respectivo.

A seguir, serão apresentadas algumas pesquisas que tratam do uso de colônias de formigas artificiais para a solução do problema de *scheduling* em máquinas paralelas.

[5] A apresentação dos artigos nesse capítulo se utiliza de ambas as nomenclaturas, buscando respeitar a simbologia encontrada nos trabalhos originais. Quando algum trabalho não indica a simbologia usada, o termo M é adotado.

[6] Para maiores informações sobre o uso da técnica *Beam Search* e sua aplicação em problemas de *scheduling*, veja por exemplo Valente e Alves (2008).

2.3.3.2 *Ant Systems* aplicado aos problemas de *scheduling* em ambientes de máquinas paralelas

A definição de problemas de *scheduling* em ambientes de máquinas paralelas é muito semelhante à definição realizada na seção 2.3.3.1. No ambiente de máquinas paralelas, tem-se um conjunto de trabalhos que devem ser processados uma única vez em quaisquer uma de um conjunto de máquinas. Estas máquinas podem ser idênticas, uniformes ou não-relacionadas, conforme descrito na seção 2.1.

Para a apresentação dos artigos referentes a *scheduling* de máquinas paralelas, esta seção é dividida em duas partes: a primeira, trata de artigos que abordam ambientes de máquinas paralelas idênticas; a segunda, trata de artigos que abordam ambientes de máquinas paralelas não relacionadas. Não foi encontrado nenhum artigo que trata da aplicação do ACO em problemas de *scheduling* em ambientes de máquinas paralelas uniformes.

***Ant Systems* aplicado aos problemas de *scheduling* em ambientes de máquinas paralelas idênticas**

Sankar *et al.* (2005) tratam do problema de minimização do maior tempo de finalização de trabalhos em um ambiente com m máquinas paralelas idênticas com *setup* dependentes, representado através da notação $P_m/s_{ij}/C_{max}$. Para isso, os autores abordaram o problema baseado no problema do caixeiro viajante n-dimensional e utilizaram, ao invés de matrizes de dimensão $n \times n$ para representar a quantidade de feromônio τ_{ij} e a visibilidade η_{ij} (como era o caso dos problemas de máquina única), m matrizes $n \times n$ para representar τ_{ij} e m matrizes $n \times n$ para representar η_{ij}. No algoritmo apresentado pelos autores, a cada passo, a formiga escolhe a tarefa para a próxima máquina disponível e determina o trabalho a ser sequenciado. Para esta determinação, os autores usaram a regra de visibilidade mostrada na equação 2.10.

$$\eta_{ij} = 1/d_j \tag{2.10}$$

O mesmo problema é tratado por Behnamian *et al.* (2009). Os autores aplicam um algoritmo híbrido, onde o ACO gera uma solução inicial, que é melhorada através de uma heurística composta pelas técnicas *Variable Neighborhood Search* (VNS) e *Simulated Annealing* (SA). Adicionalmente, os autores apresentam três algoritmos de busca local: (i) o primeiro consiste em alterar a posição de dois trabalhos sequenciados em uma única máquina; (ii) um algoritmo que busta trocar a posição de trabalhos sequenciados em máquinas diferentes; e (iii) um algoritmo que transfere trabalhos da máquina com maior *makespan* para a máquina com menor *makespan*.

Raghavan e Venkataramana (2006) tratam do problema de *scheduling* $P_m/Batch, incompatible/\sum w_i \cdot T_i$. Este problema busca a minimização do somatório de atrasos ponderado em um ambiente de máquinas paralelas idênticas que trabalham em lotes. Estes lotes são formados a partir de trabalhos com características semelhantes, como tempo de processamento. O algoritmo proposto pelos autores é composto de duas fases: a primeira, consiste em agrupar os trabalhos em possíveis lotes. Isso é feito através da regra de despacho *Apparent Tardiness Cost*, mostrada na equação 2.11. A segunda fase consiste em usar um algoritmo baseado em ACO para sequenciar estes lotes nas diferentes máquinas.

$$\pi_j^f = \frac{w_j}{P_f} \cdot e^{\left(\frac{-max(0;d_j-P_f-t)}{k\cdot\overline{p}}\right)} \tag{2.11}$$

Onde:

- π_j^f é o índice ATC do trabalho j na família f.
- P_f é a soma de todos os tempos de processamento de cada tarefa pertencente à família f.
- $\overline{p}$ é o tempo médio de todos os trabalhos ainda não sequenciados.
- t é o instante de finalização do último trabalho sequenciado.
- k é um parâmetro que varia entre os valores $\{0,5;1,0;1,5;...;5,0\}$.

Um problema semelhante, o $P_m/A_{ij}, Batch, incompatible/\sum w_i \cdot T_i$, é tratado por Li *et al.* (2008). Os autores justificam seu interesse por este problema devido à sua aplicabilidade em ambientes produtivos usados para a fabricação de semicondutores. Este problema busca a minimização do somatório de atrasos ponderado em um ambiente de máquinas paralelas idênticas que trabalham em lotes, com as tarefas sendo liberadas para processamento em momentos diferentes. Ao resolver este problema, os autores mostraram que sua implementação do algoritmo ACO trouxe resultados melhores do que a regra de despacho ATC (equação 2.11).

Uma terceira variação do problema busca a minimização do somatório do atraso ponderado em um ambiente de máquinas paralelas idênticas com formação de lotes de tarefas, considerando que existem tarefas incompatíveis (ou seja, que não podem ser colocadas no mesmo lote), tarefas com datas de chegada diferentes entre si e diferentes tempos de execução, é tratada por Li *et al.* (2009). Este problema, também descrito pelos autores como de grande importância para as indústrias de semicondutores, é representado por $P_m/A_{ij}, Q_i, Batch, incompatible/\sum w_i \cdot T_i$. Os resultados apresentados por Li *et al.* (2009)

mostraram que a heurística ACO implementada também se mostrou superior à regra de despacho ATC (equação 2.11).

A seguir, são apresentadas pesquisas que tratam do uso do ACO em ambientes de máquinas paralelas não relacionadas.

***Ant Systems* aplicado à problemas de *scheduling* em ambientes de máquinas paralelas não relacionadas**

As primeiras pesquisas encontradas para este ambiente tratam do problema $R_m//\sum w_i \cdot T_i$. Nesse problema, um conjunto de m máquinas não relacionadas processam um conjunto de n trabalhos. Cada trabalho deve ser processado apenas uma vez e o objetivo é minimizar o somatório do atraso ponderado. Para resolver o problema, Zhou *et al.* (2007) estendem o trabalho de Liao e Juan (2007), com algumas modificações:

- Os autores se utilizam de dois tipos de feromônios: um deles indica a escolha de se processar uma tarefa em uma máquina específica e o outro, indicando a escolha de se processar uma tarefa j após uma tarefa i;
- Na fase de inicialização do algoritmo, os autores estabelecem como valor inicial dos feromônios o inverso do valor do *fitness* encontrado, este problema é resolvido por meio de uma regra heurística;
- Os autores definem a visibilidade η_{ij} de acordo com a regra VMDD, mostrada na equação 2.12;

$$\eta_{ij} = \frac{w_j}{max\{P_i + p_{ij}, d_j\} - P_i} \tag{2.12}$$

Ainda neste problema, Monch (2008) trata de várias heurísticas usadas para resolvê-lo, entre elas o ACO. A regra de visibilidade é definida pela a regra ATC (vale lembrar que a regra ATC tambem é usada na resolução do problema $P_m/B, incompatible/\sum w_i \cdot T_i$ por Raghavan e Venkataramana (2006)).

Ainda, no que se refere às pesquisas em máquinas paralelas não-relacionadas, Arnaout *et al.* (2008) e Arnaout *et al.* (2009) atuam no problema $R_m/s_{ijk}/C_{max}$. Este problema, busca alocar tarefas em máquinas de forma a minimizar o tempo máximo de finalização (*makespan*), considerando *setup* dependente. Para resolver este problema, os autores se utilizam de um algoritmo de dois estágios: o primeiro estágio, atribui os trabalhos para as máquinas disponíveis e o segundo os sequência, usando um algoritmo semelhante ao utilizado para a resolução do

TSP. Cada um dos estágios é implementado em um único algoritmo ACO, com duas regras de transição distintas. Assim, a cada passo, a formiga realiza duas escolhas: (i) escolhe para qual máquina o problema será alocado; e (ii) escolhe qual a posição do próximo trabalho na máquina anterior. Para isso, são utilizadas para cada escolha um conjunto de feromônios e regra de visibilidade distintos.

2.3.3.3 *Ant Systems* aplicado a problemas de *scheduling* em ambientes *flowshop*

O problema de *scheduling* em ambientes *flowshop* consiste de n trabalhos a serem sequênciados em m máquinas. Cada trabalho J_i possui um conjunto de operações $o_{i1}; o_{i2}; ...; o_{im}$, em que cada operação o_{ik} corresponde à operação executada na máquina k referente ao trabalho J_i. Cada trabalho possui apenas uma operação a ser executada em uma máquina. Todas as máquinas devem obedecer à mesma sequência de execução das m operações.

Ainda sobre o ambiente *flowshop*, podemos classificá-lo como *permutacional* quando a sequência de execução de todos os trabalhos em todas as máquinas deve ser mantida. Quando não existe esta restrição, diz-se que o ambiente é um *flowshop não permutacional* (BAKER, 1943; PINEDO, 2008).

O algoritmo ACO e suas variações já foram usados para a resolução de problemas de *scheduling* em ambientes *flowshop* permutacional e não-permutacional. A seguir, são apresentados os trabalhos que tratam do uso do ACO em ambos os ambientes, conforme divisão por função objetivo do problema estudado. Inicialmente, são apresentados os artigos que tratam de minimização de *makespan*. Posteriormente, são apresentados artigos que tratam de demais problemas monocritérios. Por fim, na sequência são apresentados artigos que tratam do uso do ACO em problemas de *scheduling* em ambientes *flowshop* multicritério.

Artigos que tratam de minimização de *Makespan*

Dentre os problemas que tratam de minimização de *makespan* em ambiente *flowshop* usando ACO, provavelmente o mais tratado é o problema de minimização de *makespan* em *flowshop* permutacional ($F_m/prmu/M$). Na presente revisão de literatura, forma encontrados oito trabalhos que tratam desse tema: Ying e Liao (2003), Rajendran e Ziegler (2004), Ahmadizar *et al.* (2007), Chen *et al.* (2008), Zhou e Qingshan (2009), Ying e Lin (2007), Shyu *et al.* (2004) e Yan-hai1 *et al.* (2005).

Ying e Liao (2003) resolvem este problema através de um grafo muito semelhante ao grafo mostrado na figura 2.4. Os autores se utilizam da função de visibilidade mostrado na equação 2.13 para a escolha da sequência dos trabalhos no *flowshop* permutacional.

$$\eta_{ij} = -\sum_{t=1}^{m}\{(m-(2\cdot t-1))\cdot p_j^t\} - \min_j\{\eta_{ij}\} + 1 \tag{2.13}$$

Onde:

- p_j^t representa o tempo de processamento da tarefa j na máquina t

Ainda tratando do problema $F_m/prmu/M$, Rajendran e Ziegler (2004) apresentam dois algoritmos ACO. Estes dois algoritmos se diferenciam nos seguintes pontos:

- O primeiro algoritmo se utiliza de limites mínimos e máximos de feromônios, enquanto o segundo não;
- A regra de transição de ambos favorece a ordenação dos trabalhos conforme a melhor sequência encontrada. O segundo algoritmo ACO apresentado por Rajendran e Ziegler (2004) dá uma prioridade maior para o primeiro trabalho da melhor sequência encontrada que ainda não está sequênciado;
- A regra de transição do segundo algoritmo ACO leva em consideração a diferença entre a posição anterior e atual dos trabalhos (no primeiro algoritmo, é realizado apenas o ciclo depósito-evaporação normal, conforme descrito na seção 2.2.

Ahmadizar *et al.* (2007) também resolvem o problema $F_m/prmu/M$ desenvolvendo duas regras de prioridade e as combinando para formar a visibilidade η. As duas regras são:

1. Uma extensão da regra de Johnson ((JOHNSON, 1953)), descrita pelos autores como: "Se um trabalho possui tempo de processamento menor nas primeiras máquinas que nas últimas, este deve ser realizado primeiro". Esta regra é mostrada na equação 2.14.
2. Uma extensão da regra SPT, que dá prioridade aos trabalhos com menor tempo de processamento total. Esta regra é mostrada na equação 2.15.

$$R_j^{(1)} = \frac{(p_{j,(m-1)} + p_{j,m})/(p_{j,1} + p_{j,2})}{\sum_{l=1}^{N}((p_{l,(m-1)} + p_{l,m})/(p_{l,1} + p_{l,2}))} \tag{2.14}$$

$$R_j^{(2)} = \frac{1/(\sum_{i=1}^{m} p_{ji}}{\sum_{l=1}^{n}(1/\sum_{i=1}^{m} p_{jl})} \tag{2.15}$$

A visibilidade é então definida pelos autores como $\eta_{ij} = min\{R_j^{(1)}; R_j^{(2)}\}$.

Chen *et al.* (2008) tratam também do problema $F_m/prmu/M$, porém ignoram a função de visibilidade (ou seja, tratam $\eta = 1$). Na implementação do algoritmo ACS apresentada pelos autores, as características do problema influenciam a construção da solução apenas: (i) na atualização das trilhas de feromônio (a quantidade de feromônio depositado é inversamente proporcional ao *makespan* da solução encontrada) e (ii) pela ação da busca local (que tenta melhorar a solução encontrada através da troca de posição de pares de trabalhos.

O trabalho de Zhou e Qingshan (2009) apresenta um algoritmo MMAS para a resolução de problemas de *scheduling* $F_m/prmu/M$ de larga escala. Os problemas abordados tratam de instâncias com 100 e 200 tarefas, processados em 10 e 20 máquinas. Para resolver estes problemas, os autores apresentam um algoritmo híbrido entre MMAS e busca tabu, e assim como Chen *et al.* (2008), o valor de η é desconsiderado. A busca local aplicada no algoritmo MMAS consiste em reposicionar trabalhos individuais na sequencia obtida.

Ying e Lin (2007) também tratam do problema de minimização do *makespan*, porém agora aplicado ao ambiente de manufatura *flowshop* não permutacional, representado por $F_m//M$. Para isso, o primeiro passo dos autores é representar um problema com n trabalhos a serem processados em m máquinas em um grafo com $n \times m$ nós. A movimentação da formiga para um nó específico N_{im} significa o sequênciamento da tarefa J_i na máquina m. A visibilidade η_{im} é definida de acordo com a escolha aleatória entre um conjunto de 20 heurísticas construtivas simples, como SPT (do, inglês, *Shortest Processing Time*, menor tempo de processamento), LPT (do inglês, *Longest Processing Time*, maior tempo de processamento), e outras.

Shyu *et al.* (2004) tratam do problema $F_2/nwt,setup/M$, ou seja, buscam minimizar o *makespan* em um ambiente *flowshop* com duas máquinas onde não é permitida espera entre o processamento de um trabalho e outro, com setup dependente. O algoritmo utilizado é o ACS, conforme definido por Dorigo *et al.* (1996), sendo que a única modificação realizada por Shyu *et al.* (2004) diz respeito à visibilidade, que agora é definida em função dos tempos de *setup* e processamento das tarefas.

Por fim, Yan-hai1 *et al.* (2005) também tratam do problema de minimização de *makespan* em um ambiente *flowshop* não-permutacional, incluindo ordens urgentes (do inglês, *rush orders*). No algoritmo apresentado, os autores executam um algoritmo ACO para a geração de uma solução inicial. Após a geração desta solução, as ordens urgentes são inseridas no grafo, e o algoritmo continua sua execução, ajustando a solução ao novo problema.

Artigos que tratam de outras funções monocritérios

Dentre os demais trabalhos que tratam de *scheduling* em ambiente *flowshop* com funções objetivas monocritérios diferentes de minimizar o *makespan*, cita-se os trabalhos de T'kindt *et al.* (2002) (que buscam minimizar o somatório dos tempos de finalização das tarefas em um ambiente *flowshop* não permutacional com duas máquinas), Rajendran e Ziegler (2004) e Rajendran e Ziegler (2005) (que buscam minimizar o somatório dos tempos de fluxo) e Gajpal e Rajendran (2006) (que buscam minimizar a variância dos tempos de término das tarefas).

O trabalho de T'kindt *et al.* (2002) propõem um algoritmo MMAS para a resolução do

problema de minimização do somatório dos tempos de finalização de n tarefas em um ambiente *flowshop* não permutacional com duas máquinas ($F_2//\sum C_i$). Neste problema, a regra de transição é simplificada, fazendo a visibilidade η_{ij} idêntica para todos os trabalhos i e j. Adicionalmente, os autores se utilizam de um algoritmo de busca local baseado no algoritmo API.

Rajendran e Ziegler (2004) aplicam o ACO em dois problemas de *scheduling* em ambientes de *flowshop* permutacional: o primeiro, têm como objetivo tratar da minimização do *makespan* ($F_n/prmu/M$) e o segundo, do somatório dos tempos de fluxos de todas as tarefas ($F_n/prmu/\sum F_i$). Para ambos os problemas, o algoritmo proposto pelos autores se basearam no algoritmo MMAS com as seguintes modificações:

- São definidos dois nós i e k do grafo, onde i indica o índice do trabalho e k a posição do mesmo.
- A probabilidade de escolha de um trabalho i para ocupar a posição k é dada pela equação 2.16.
- O depósito de feromônios se dá não apenas na trilha pertencente à melhor sequência ik, mas também nos nós referentes à posições próximas à posição k.

$$p_{ik} = \frac{T_{ik}}{\sum_l \sum_{q=0}^{k} \tau_{iq}} \tag{2.16}$$

Rajendran e Ziegler (2005) se utilizam dos resultados obtidos por Rajendran e Ziegler (2004) e apresentam um novo algoritmo para a resolução do problema $F_m/prmu/\sum F_i$. Para isso, os autores se utilizam de um algoritmo semelhante ao de Rajendran e Ziegler (2004), inicializando os valores de feromônio através dos resultados obtidos através da heurística NEH, apresentada originalmente por Nawaz *et al.* (1983).

Gajpal e Rajendran (2006) tratam do problema $F_m/prmu/CTV$, ou seja, busca minimizar a variância das datas de finalização (CTV - do inglês *Completion-Time Variance*) de n trabalhos em um ambiente *flowshop* permutacional com m máquinas. Os autores usam a definição de CTV conforme equação 2.17.

$$CTV = \sum_{i=1}^{n} (C_i - \overline{C})^2/n \tag{2.17}$$

O algoritmo proposto por Gajpal e Rajendran (2006) se baseia no algoritmo MMAS, com algumas modificações:

- Uma solução inicial é construída usando o algoritmo NEH;
- Os limites máximos de feromônio são determinados através do resultado da função qualidade da seqüência da solução inicial;
- É implementada uma busca local denominada *random-job-insertion local search*, onde um trabalho j é escolhido aleatoriamente, tendo sua posição na sequência alterada para a posição onde se obtém o melhor valor do CTV.

Artigos que tratam de funções objetivos compostos de vários indicadores de desempenho

Além do desenvolvimento do algoritmo para a resolução do problema $F_2//\sum C_i$ mostrado anteriormente, T'kindt *et al.* (2002) também aplicaram o ACO para o problema $F_2//Lex(C_{max}, \sum Ci)$. Este problema, considera um problema de *scheduling* em um ambiente *flowshop* com duas máquinas, em que o objetivo é minimizar o tempo total de término mantendo o *makespan* com o menor valor possível. Para isso, os autores se utilizaram da mesma heurística já comentada.

Li e Zhang (2006) tratam do problema $F_2//\alpha \cdot \sum C_i + (1-\alpha) \cdot M$, ou seja, minimizar a soma ponderada dos tempos de finalização e do *makespan*. Para isso, os autores implementam um algoritmo ACO com as seguintes características:

- Os feromônios são inicializados com um valor constante, e depois atualizados conforme as sequências obtidas pelas heurísticas API e NAP;
- A regra de transição leva em consideração: (i) a trilha de feromônio τ_{ij} que diz respeito à alocação do trabalho j após o trabalho i; (ii) Um valor de visibilidade η_{ij} que está relacionada à alocação do trabalho j após o trabalho i; e (iii) Um valor de visibilidade ε_{jl} que indica a intensidade de uma trilha de feromônio independente de τ_{ij}, que indica a preferência da alocação da tarefa j na posição l da lista de tarefas sequenciadas.
- A quantidade de feromônio depositada depende da qualidade da função encontrada até o momento pelo algoritmo ACO.
- Os autores relatam que a atualização local de feromônios não mostrou resultados significativos; desta forma, tanto a trilha τ_{ij} quanto a trilha ε_{jl} são atualizadas apenas na fase de atualização global de feromônios.
- Os autores relatam o uso de critérios de dominância para impedir a construção de soluções não-promissoras e desta forma acelerar a execução do algoritmo.

Pasia *et al.* (2006) tratam do problema $F_m//\sum T_i, M$, ou seja, minimizar o somatório de atrasos e o *makespan* em um ambiente *flowshop* com m máquinas. Para isso, se utilizam do algoritmo *Pareto Ant Colony Optmization* (PACO), que estabelece um conjunto de feromônios para cada objetivo do problema. Os autores usam da definição de visibilidade mostrada na equação 2.18.

$$\eta_{ij} = \frac{1}{\max\limits_{i \in u}(d_i, M_j) - M_{j-1}} \tag{2.18}$$

Yagmahan e Yenisey (2008) tratam de quatro objetivos distintos quando se utilizam do ACO para a resolução do problema $F_m/prmu/\sum C_i, M, \sum F_i, idle$:

- $\sum C_i$. ou seja, minimizar o somatório dos tempos de finalização das tarefas;
- M, ou seja, minimizar o *makespan* da sequência final;
- $idle$, ou seja, minimizar o tempo de máquinas ociosas;
- $\sum F_i$, ou seja, minimizar o somatório de tempos de fluxo.

Para isso, os autores se utilizam de um algoritmo ACS com três importantes modificações:

- A visibilidade η_{ij} é determinada usando como base uma sequência de tarefas construída conforme uma heurística executada na fase de inicialização do ACS;
- Uma lista de candidatos é utilizada para restringir o espaço de buscas e assim tornar a execução do algoritmo mais rápida;
- Os autores implementa uma busca local que remove um trabalho escolhido aleatoriamente de uma posição k_0 e o insere em outra posição aleatória k_1.

Ainda tratando de problemas de *scheduling* em ambientes *flowshop* multiobjetivos, o trabalho de Al-Anzi e Allahverdi (2009) trata de um ambiente *flowshop* permutacional de dois estágios em que se deseja minimizar a soma ponderada entre o *makespan* e o tempo médio de finalização das tarefas ($F_2/prmu/u \cdot M + v \cdot \sum Ci/2$). Para isso, os autores desenvolvem uma heurística ACO com algumas modificações. Estas alterações, inexistentes nos demais trabalhos pesquisados foram:

- A regra de transição é alterada para a equação 2.19

- O depósito de feromônios (atualização global), mostrada na equação 2.20 leva em conta o número de iterações que já se passaram. Assim, de acordo com esta equação, os resultados formados no início do algoritmo contribuem menos para a criação da trilha de feromônios do que os resultados obtidos próximos ao fim da execução do algoritmo. Al-Anzi e Allahverdi (2009) descrevem a relação $\frac{\text{iteração atual}}{\text{número maximo de iterações}}$ como a *maturidade* do algoritmo.

- Ainda sobre a atualização global, a quantidade de feromônio depositada depende também da qualidade da solução encontrada (representada pelo termo $e^{(gap_{atual})}$.

- O número de formigas para a colônia e o número de iterações a serem realizadas é definido em função do número de trabalhos do problema.

$$p_{ij}^{k} = e^{(\tau_{ij} - \tau_{max})} \tag{2.19}$$

$$\tau_{ij} = \tau_{ij}^{0} + \frac{\text{iteração atual}}{\text{número maximo de iterações}} * e^{(gap_{atual})} \tag{2.20}$$

Onde:

- gap_{atual} é a diferença entre o melhor valor da função objetivo encontrada até o momento e o valor da mesma encontrada na iteração atual do algoritmo.

Lin *et al.* (2008) atuam no problema de minimizar a soma ponderada do *makespan* e do tempo total de finalização de tarefas em um ambiente *flowshop* permutacional, representado por $F_m/prmu/u \cdot M + v \cdot \sum C_i$. Para abordar o problema usando o algoritmo ACO, os autores criam uma nova regra de transição, que permite a construção da solução baseada em: (i) Em um conjunto de feromônios que representam o posicionamento relativo entre cada par de trabalhos; (ii) em um conjunto de feromônios que representam a ocupação de uma posição absoluta de um trabalho na seqüência final; e (iii) uma função de visibilidade. Adicionalmente, os autores analisam o uso de dois critérios de dominância, que se mostraram eficientes estratégias para melhorar o resultado final do algoritmo implementado.

Marimuthu *et al.* (2009) é o único trabalho pesquisado que aborda um ambiente *flowshop* com possibilidade de criação de lotes de produção, com objetivo de se minimizar o *makespan* ou o tempo total de fluxo ($F_m, lot//M$ e $F_m, lot//\sum F$, respectivamente). Para ambos os problemas, os autores desenvolveram um algoritmo ACO que usa unicamente da informação

dos feromônios e de uma busca local para a determinação da melhor solução. Esta busca local consiste em reposicionar cada trabalho em outra posição da sequência de forma a melhorar o valor da função objetivo (*makespan* ou tempo total de fluxo).

Por fim, Huang e Yang (2009) tratam do problema $F_m/r_j, S_{ij}/\alpha\sum_{i=1}^{m} MIT_i + \beta\sum_{i=1}^{m}\sum_{j=1}^{B_j}\sum_{t=1}^{n} JWT_{ijt} + \gamma\sum_{i=1}^{m} T_i$. Neste problema, é considerado um ambiente *flowshop* em que cada tarefa possui tempos de liberação diferentes e setup dependente. O objetivo é minimizar o somatório ponderado entre o tempo ocioso das máquinas, o somatório do tempo de espera de cada conjunto de tarefas em cada máquina e o somatório de atraso das tarefas. Para isso, é implementado um algoritmo ACS, sem busca local e com a visibilidade definida através de uma heurística própria,

2.3.3.4 *Ant Systems* aplicado a problemas de *scheduling* em ambientes *jobshop*

Da mesma forma como foi feita a apresentação dos trabalhos que se utilizam de ACO para a resolução de problemas de *scheduling* em ambiente *flowshop*, esta seção apresenta inicialmente os trabalhos que tratam do uso do ACO para minimização do *makespan* em ambientes *jobshop*. Em um segundo momento, são apresentados os demais trabalhos que se utilizam de funções objetivos monocritérios, e por fim, são apresentados os trabalhos que tratam de funções multicritérios.

Artigos que tratam de minimização de *Makespan*

A grande maioria dos trabalhos pesquisados que tratam do uso da heurística ACO para *scheduling* em ambiente *jobshop* tratam da minimização do makespan (representados por $J_{n,m}//M$). O primeiro trabalho encontrado foi o de Yoshikawa e Terai (2006). Para resolver o problema em questão, os autores implementam um algoritmo ACO. Neste algoritmo, cada nó do grafo representa a execução de uma operação de uma tarefa em uma máquina. A formação da sequência se dá através de uma heurística construtiva. Esta heurística, retorna um ou um conjunto de operações a serem sequenciadas imediatamente. Quando se tem um conjunto de operações a serem sequenciadas, o ACO é usado para realizar o desempate entre as operações candidatas.

Udomsakdigool e Kachitvichyanukul (2008) tratam do problema $J_{n,m}//M$ ao desenvolverem um conjunto de algoritmos ACO diferentes entre si na definição da visibilidade e de uma lista de tarefas candidatas a serem escolhidas para a movimentação da formiga. Neste trabalho, os autores definem como possibilidades de definição de visibilidade um conjunto de regras de despacho, apresentadas no quadro 2.4. A lista de tarefas candidatas é definida através de uma entre as três seguintes estratégias:

- Todas as tarefas pendentes (ainda não executadas) são consideradas;
- Todas as tarefas que podem ser executadas sem espera são consideradas;
- Um conjunto aleatório de tarefas são consideradas.

Udomsakdigool e Kachitvichyanukul (2008) criaram 30 variações do ACO usando cada combinação possível entre as possibilidades de lista de candidatos e de visibilidade. Através de análises estatísticas, os autores mostraram que a definição de melhor combinação dependeu de características, como o tamanho do problema.

Regra de Despacho	Descrição
EST	*Earliest Starting Time*, sequencia de acordo com o momento de início da tarefa, em ordem crescente.
EFT	*Earliest Finishing Time*, sequencia de acordo com o momento de finalização da tarefa, em ordem crescente
SPT	*Shortest Processing Time*, sequencia de acordo com o tempo de processamento da tarefa, em ordem crescente
LPT	*Longest Processing Time*, sequencia de acordo com o tempo de processamento da tarefa, em ordem decrescente
LWR	*Least Work Remaining in the Job*, sequencia de acordo com o tempo total de processamento das operações ainda não realizadas, em ordem crescente.
MWR	*Most Work Remaining in the job*, sequencia de acordo com o tempo total de processamento das operações ainda não realizadas, em ordem crescente.
SMT	*Shortest value obtained by Multiplying process time with Total process time*, sequencia de acordo com o produto do tempo de processamento da tarefa com o tempo total de processamento, em ordem crescente.
LMT	*Largest value obtained by Multiplying process time with Total process time*, sequencia de acordo com o produto do tempo de processamento da tarefa com o tempo total de processamento, em ordem decrescente.
SDT	*Shortest value obtained by Dividing process time with total process time*, sequencia de acordo com o quociente do tempo de processamento da tarefa com o tempo total de processamento, em ordem crescente.
LDT	*Largest value obtained by Dividing process time with Total process time*, sequencia de acordo com o quociente do tempo de processamento da tarefa com o tempo total de processamento, em ordem decrescente.

Quadro 2.4: Regras de despacho usadas para a definição de visibilidade por Udomsakdigool e Kachitvichyanukul (2008)

Heinonen e Pettersson (2007) também tratam do uso do algoritmo ACO para a minimização do *makespan* em um ambiente *jobshop*. Os autores se utilizam de um problema com 10 máquinas e 10 trabalhos (representado por $J_{10,10}//M$). Para a resolução do problema, os autores implementam um algoritmo híbrido baseado no MMAS, com busca local baseada em uma técnica de busca tabu.

A abordagem do problema $J_{10,10}//M$ por Zhuo *et al.* (2007) se inicia com a geração de um grafo composto de duas informações, conforme visto na figura 2.5. Nesta figura, são definidos dois tipos de nós: um nó inicial "0", e um conjunto de nós relativos às operações dos problemas. Cada nó o_{ij} se refere à execução da operação da tarefa i na máquina j. Todos os nós são interconectados (as ligações foram suprimidas para facilitar a visualização). Inicialmente, todos os nós o_{ij} estão desabilitados (ou seja, a formiga não pode se mover para eles). As setas da figura 2.5 indicam quais nós são habilidados de acordo com o posicionamento da formiga. Por exemplo, o nó "0" habilita os nós o_{11}, o_{22} e o_{33}. Para a execução do algoritmo, a formiga é posicionada no nó "0" se move conforme os nós habilitados.

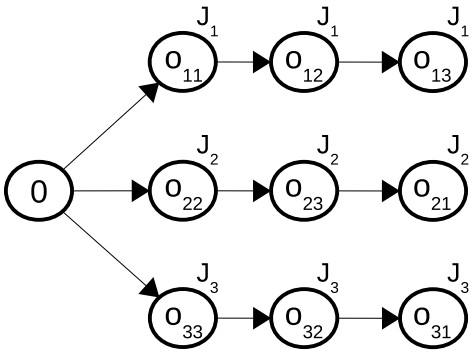


Figura 2.5: Grafo usado por Zhuo *et al.* (2007) para a representação de um problema de *jobshop*

Uma vez tendo representado o problema com o grafo mostrado na figura 2.5, Zhuo *et al.* (2007) aplicam um algoritmo ACS. Neste algoritmo, a visibilidade escolhida é definida através da equação 2.21

$$\eta_{ij} = \frac{TLJ}{n \cdot m} \tag{2.21}$$

Onde:

- TLJ é o tempo do trabalho que ainda não foi processado (do inglês, *Time-Left for Job*)

Huang e Liao (2008) tratam do mesmo problema que Zhuo *et al.* (2007) usando um algoritmo que se utiliza das técnicas ACO e busca tabu (este último como algoritmo de busca local para o ACO).

Por fim, Eswaramurthy e Tamilarasi (2009) também se utilizam de uma combinação do algoritmo ACO com busca tabu para o problema $J_{n,m}//M$. Provavelmente a maior contribuição deste algoritmo à literatura é o desenvolvimento do que foi chamada "estratégia de lista tabu de comprimento dinâmico" (do inglês, *dynamic tabu length strategy*). Esta abordagem busca aumentar e diminuir o tamanho da lista tabu conforme o número de iterações realizadas, e mostrou um aumento na performance do algoritmo.

Artigos que tratam de múltiplos objetivos

Huang e Yang (2008) tratam do problema de se minimizar a soma do adiantamento e do atraso de todas as tarefas em um ambiente *jobshop* com janelas de tempo. O algoritmo usado pelos autores se baseia na heurística básica do ACO. Neste trabalho, o posicionamento dos trabalhos é dado através apenas da alteração de seu posicionamento relativo, não sendo usado nenhum identificador da posição absoluta do trabalho na sequência final.

2.3.4 Análises

O estudo realizado nas seções anteriores permite algumas análise importantes:

A primeira análise diz respeito ao volume de publicações encontrado. Na figura 2.6, pode-se perceber claramente que nos últimos anos existiu um aumento significativo das publicações na área. Este aumento indica que o uso do algoritmo ACO em problemas de *scheduling* é um tema atual e de crescente importância nos periódicos pesquisados. Mais ainda, é possível perceber uma característica da evolução dos ambientes abordados, ao dividi-los em quatro estágios principais: no primeiro, correspondentes aos anos 1999 e 2000, as únicas publicações existentes tratavam do ambiente de máquina única. Em um segundo momento, apareceram artigos que se utilizam de ACO para *scheduling* em ambientes *flowshop* (a partir de 2002). A pesquisa em ambientes de máquinas paralelas se dá a partir de 2005 e, por fim, em 2006 é publicada a primeira pesquisa em ambientes *jobshop*. Percebe-se, assim, que a produção de trabalhos de *scheduling* usando o ACO é algo recente, e muito ainda pode ser trabalhado na área. Este resultado foi usado como guia na definição dos problemas a serem estudados e na estruturação deste trabalho, que segue a ordem estudo de problemas em ambiente de máquina única e *flowshop*.

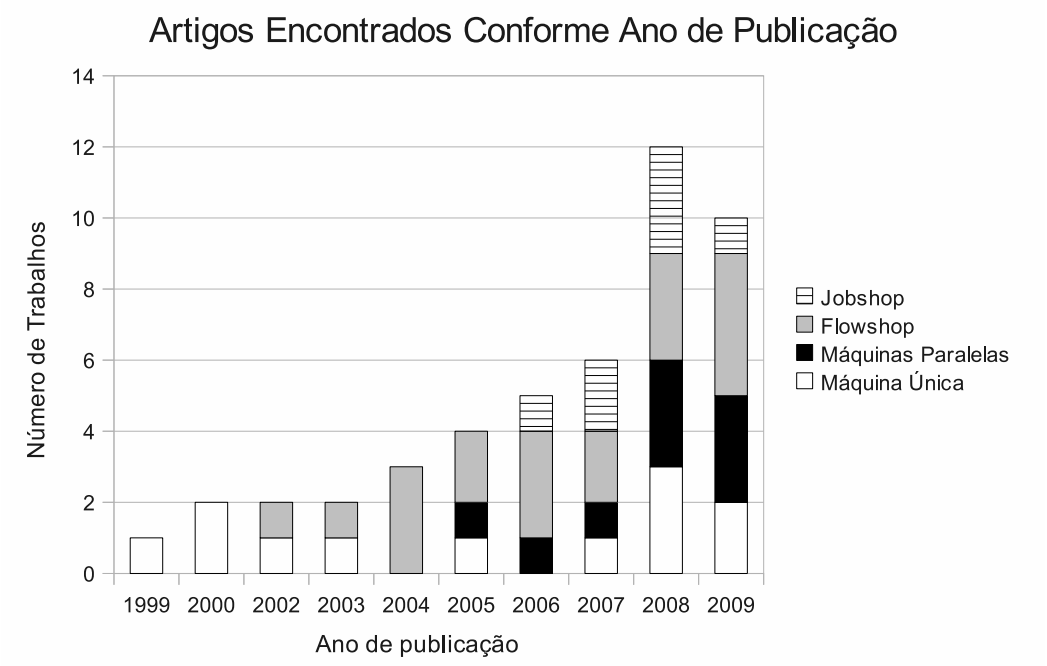


Figura 2.6: Número de artigos encontrados conforme ano de publicação

Além do ambiente, outra análise pertinente diz respeito ao número de critérios existentes na função objetivo, que permite dividir entre trabalhos que tratam de um único critério de desempenho (por exemplo, minimizar o atraso total de tarefas) e problemas que tratam de mais

de um critério de desempenho (por exemplo, minimizar o somatório ponderado dos tempos de finalização e do *makespan*).Na figura 2.7, são mostrados estes dois grupos. Nota-se que apenas 6 problemas de *flowshop* e 1 problema de *jobshop* abordam funções objetivos com mais de um critério de desempenho. Funções com apenas um critério de desempenho, por outro lado, foram encontradas em todos os ambientes de manufatura pesquisados. Mesmo quando se analisa as pesquisas de *flowshop*, percebe-se claramente que um maior número de trabalhos se focou em problemas de *scheduling* com um único critério de desempenho.

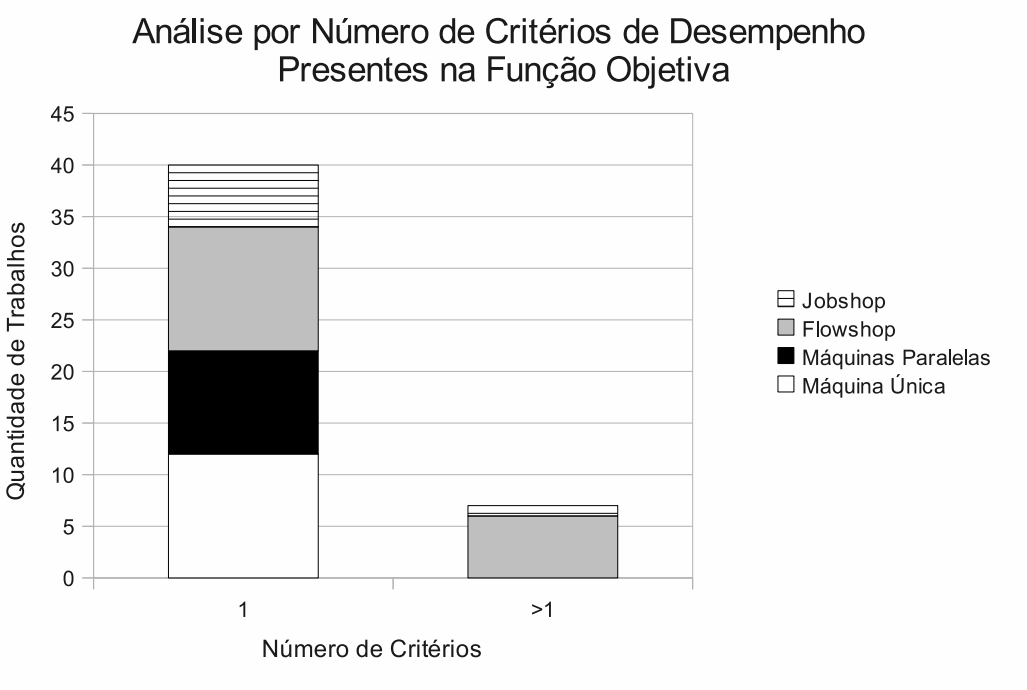


Figura 2.7: Número de artigos encontrados conforme ano de publicação

Na figura 2.8, são mostrados os algoritmos base utilizados nos trabalhos revisados. Vê-se que todos os trabalhos utilizam apenas os algoritmos ACO, ACS e MMAS nos trabalhos pesquisados. Utiliza-se o termo "ACO" quando os trabalhos, embora se utilizem do conceito fundamental da otimização por colônia de formigas, não estabelecem a variação utilizada, podendo implementar ou não algum dos recursos das variações existentes na literatura. Percebe-se então nessa figura que o algoritmo-base mais utilizado foi o ACO (31 trabalhos), seguido pelo ACS (9 trabalhos) e pelo MMAS (7 trabalhos). Isso indica claramente que a implementação de um algoritmo mais simples têm sido a preferência pelos pesquisadores, mesmo que sejam realizadas pequenas modificações. Uma modificação comum no ACO é a limitação dos níveis de feromônio, da mesma forma que é previsto no MMAS. Porém, como outras características do MMAS não foram implementadas (por exemplo, estratégia elitista), os próprios autores desses trabalhos assumem seu algoritmo como uma forma modificada do ACO. Quando analisamos qual técnica foi utilizada para a resolução de qual algoritmo, percebe-se que o ACO foi utilizado

para a resolução de problemas de *scheduling* nos quatro ambientes estudados, enquanto o MMAS apenas para problemas de *flowshop* e máquinas paralelas. Com isso, pode-se perceber um claro indicativo de que, em futuros trabalhos, os algoritmos ACO, ACS ou o MMAS devem ser considerados como uma possibilidade para o desenvolvimento de técnicas para a resolução de problemas de *scheduling*.

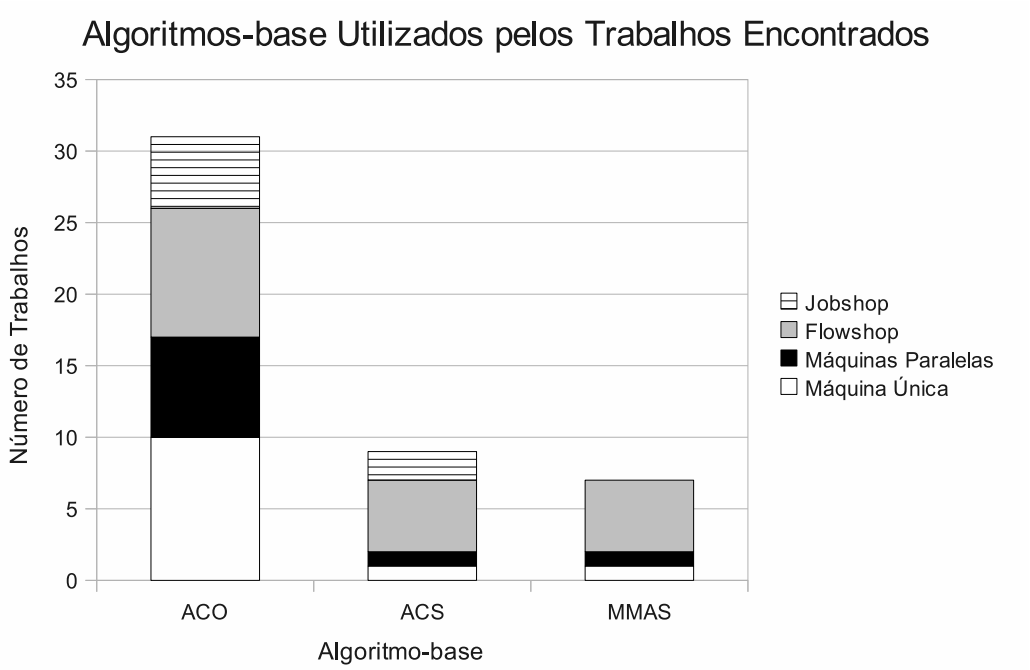


Figura 2.8: Análise do uso dos algoritmos ACO, ACS e MMAS nos trabalhos pesquisados

Um ponto interessante e também digno de nota é a forma como o posicionamento das tarefas é tratada pelos algoritmos: tarefa-para-posição (critério 3) e/ou tarefa-a-tarefa (critério 4). A figura 2.9 traz uma comparação entre o número de trabalhos que se utilizam de cada uma das técnicas observadas conforme o ambiente de manufatura estudado. Percebe-se que a estratégia tarefa-para-posição foi aplicada com mais frequencia em problemas de *flowshop* (11 de 20 artigos a usam). No caso dos demais ambientes, o uso desta estratégia foi menor: 1 artigo dos 12 de máquina única, 1 artigo dos 9 de máquinas paralelas, e 1 dos 10 artigos pesquisados que tratam do ambiente *jobshop*. A estratégia tarefa-a-tarefa é usada em grande parte dos trabalhos estudados. Isso indica que a estratégia tarefa-a-tarefa parece ser eficiente em grande parte dos problemas de *scheduling*. Porém, a estratégia tarefa-para-posição parece ter sido aplicada com sucesso principalmente em problemas de *flowshop*.

É interessante perceber que as estratégias trabalho-a-trabalho e trabalho-para-posição foram usadas em conjunto nos trabalhos de Liao e Juan (2007), Li e Zhang (2006) e Lin *et al.* (2008) (3 dos 48 trabalhos analisados). Isso permite concluir que estas duas estratégias não são excludentes e, conforme demonstrado nos trabalhos analisados, sua utilização conjunta é

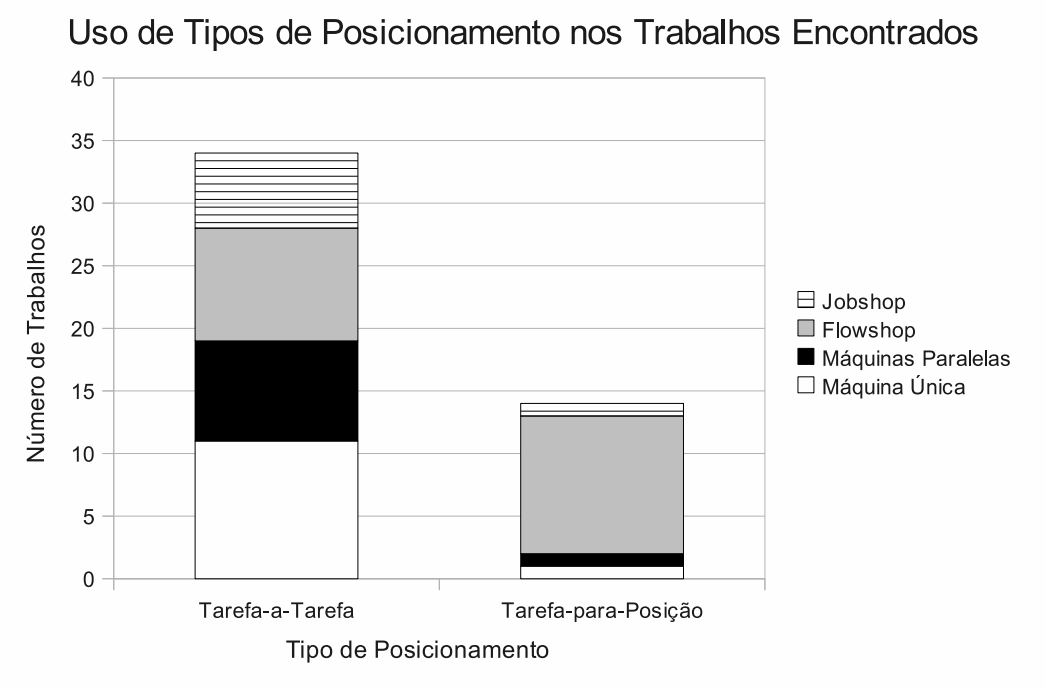


Figura 2.9: Análise do uso de posicionamento trabalho-a-trabalho e trabalho-para-posição nos trabalhos estudados

uma estratégia viável para a resolução de problemas de *scheduling*.

Com relação ao uso ou não de critérios de dominância, pode-se perceber na figura 2.10 que apenas trabalhos que tratam de ambientes de máquina única e *flowshop* se utilizaram desta estratégia. Nota-se que apenas no caso do ambiente de máquina única o número de trabalhos que se utilizam de critérios de dominância é igual ao número de trabalhos que não se utilizam do mesmo.

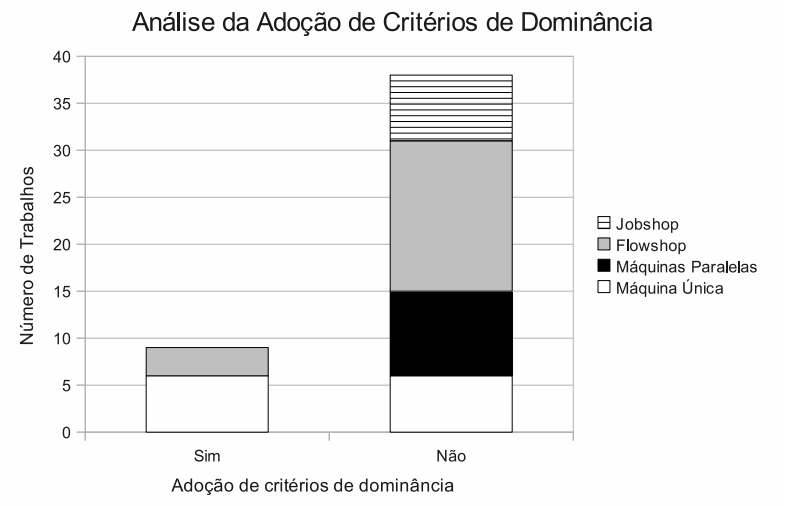


Figura 2.10: Análise da adoção de critérios de dominância nos artigos estudados

A presente análise da literatura também mostrou que, embora os algoritmos ACO, ACS

e MMAS originais inicializem todos os valores de feromônio com um mesmo valor inicial, alguns artigos se utilizam de uma heurística construtiva para a geração de uma trilha inicial. Com isso, o algoritmo já se inicia com uma solução inicial viável presente nas nas trilhas de feromônio no formato de um reforço positivo realizado durante a fase de inicialização do algoritmo. Conforme pode-se perceber na figura 2.11, o uso de uma regra heurística para a inicialização dos feromônios se deu em 3 artigos dos 12 analisados para ambientes de máquina única, 2 dos 9 analisados para ambientes de máquinas paralelas, 9 dos 19 analisados para ambientes *flowshop* e 2 dos 7 trabalhos que tratam de ambientes *jobshop*. Percebe-se que a inicialização através de heurísticas foi a estratégia mais utilizada para ambientes *flowshop*.

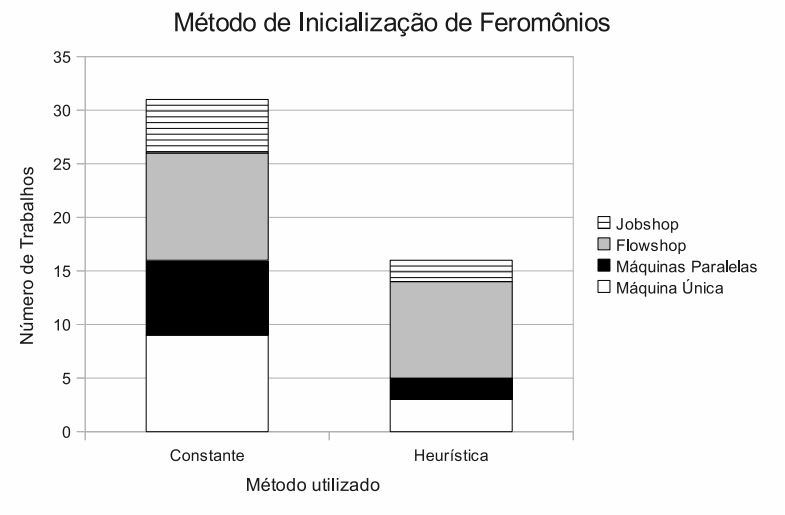


Figura 2.11: Análise da forma de inicialização dos feromônios nos trabalhos estudados

Ao se analisar a forma como a visibilidade é tratada pelos algoritmos implementados, percebe-se na figura 2.12 três estratégias distintas:

1. Os valores de η para todos os arcos do grafo são calculados na fase de inicialização e se mantém constante durante toda a execução do algoritmo. Isso ocorre em 5 dos 12 trabalhos de máquina única, 5 dos 9 trabalhos de máquinas paralelas, 7 dos 18 trabalhos que tratam de *flowshop* e 4 dos 8 trabalhos que tratam de *jobshop*.

2. Os valores de η são calculados durante a construção da solução, usando características da sequência parcial já construída em consideração (por exemplo, o *makespan* atual). Esta estratégia é usada em grande parte dos trabalhos, principalmente em ambientes de máquina única (6 dos 12 trabalhos) e *jobshop* (4 dos 8 trabalhos). No caso de máquinas paralelas, 4 dos 9 trabalhos analisados se utilizam desta estratégia. No caso de ambientes *flowshop*, 5 dos 18 artigos o fazem.

3. Não é usado o valor de visibilidade. Esta estratégia só foi utilizada em uma pesquisa em ambiente de máquina única e em 6 pesquisas de *scheduling* em ambientes *flowshop*.

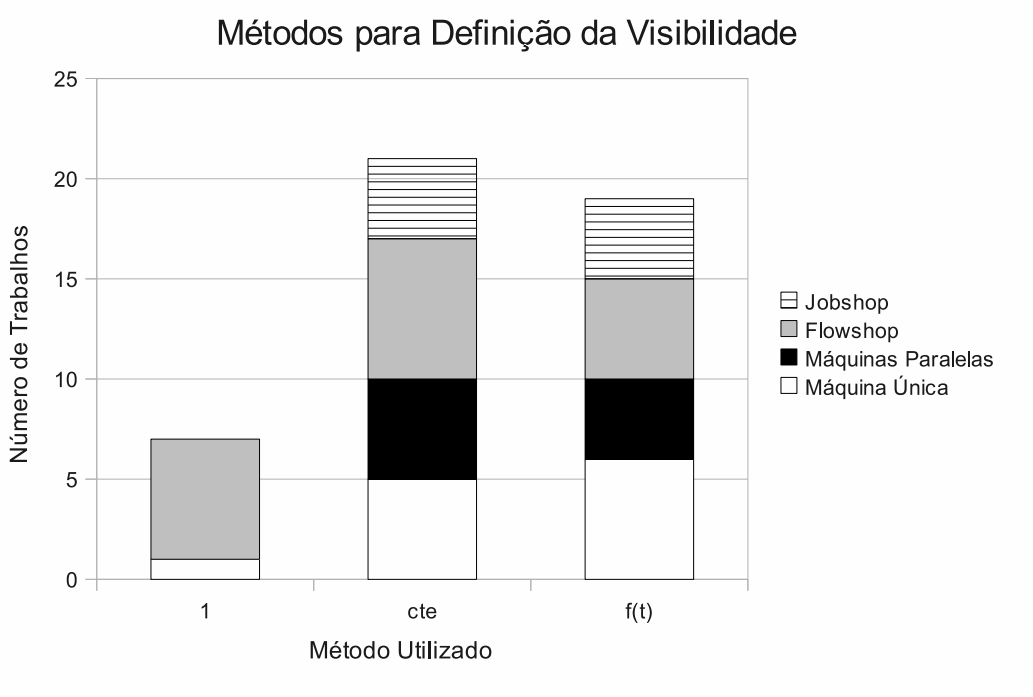


Figura 2.12: Análise da definição de visibilidade nos trabalhos estudados

2.3.5 Considerações Finais

Esse capítulo apresentou uma revisão bibliográfica sobre o uso do algoritmo ACO em problemas de *scheduling*. Durante essa análise, foi possível perceber que algoritmos baseados em formigas artificiais (no caso, ACO, MMAS, ACS) são estratégias válidas para a resolução de problemas de *scheduling*, pelo menos no que diz respeito aos quatro ambientes de manufatura estudados (máquina única, máquinas paralelas, *flowshop* e *jobshop*). Embora o algoritmo ACO seja muito simples, a possibilidade de mesclagem de heurísticas e conceitos já existentes referentes à problemas de *scheduling* abre um campo de pesquisa muito vasto. Isso explica a relevância de várias pesquisas produzirem diferentes técnicas baseadas em ACO para a resolução de problemas iguais (por exemplo, a resolução do problema $J_{10,10}//M$ por Heinonen e Pettersson (2007), Zhuo *et al.* (2007) e Huang e Liao (2008)).

As análises realizadas na seção 2.3.4 dizem respeito a uma área do conhecimento recente, cujos primeiros trabalhos estão sendo divulgados. Nota-se claramente que as primeiras pesquisas foram em problemas de *scheduling* em ambientes de máquina única, passando para *flowshop*, máquinas paralelas e *jobshop*. Com o amadurecimento no tema, características

adicionais foram aos poucos sendo incluídas ao algoritmo, como é o caso dos critérios de dominância, até o momento elaborados apenas para os ambientes de máquina única e *flowshop*.

Esta análise também nos pemite concluir que o algoritmo ACO é uma abordagem viável e atual para a resolução de problemas de *scheduling*. Para a adaptação do algoritmo para cada problema específico, todos os autores pesquisados se utilizaram de regras heurísticas e demais conceitos já estabelecidos na literatura de *scheduling*, principalmente para a definição de: (i) Uma forma de inicialização de feromônios que buscou aumentar a convergência do algoritmo; (ii) uma forma de se mensurar a visibilidade entre dois nós do grafo; (iii) critérios de dominância para reduzir o espaço de buscas do algoritmo; e/ou (iv) o algoritmo de busca local.

Durante o desenvolvimento deste capítulo, foi percebido que existiu um padrão no conjunto de alterações realizadas em um conjunto de algoritmos-base (ACO, ACS e MMAS). Porém, antes da incorporação de conceitos de *scheduling*, se mostrou necessário um estudo mais aprofundado do comportamento de algoritmos baseados em formigas artificiais em si. Assim, no capítulo seguinte, busca-se compreender a estrutura do algoritmo ACO e suas variações.

3 *Análises Computacionais*

Este capítulo traz o resultado de análises da implementação de várias heurísticas ACO. Este estudo teve como objetivo: (i) permitir uma maior compreensão do comportamento computacional dos algoritmos ACO presentes na literatura, (ii) gerar uma estrutura de implementação que possa ser reutilizada nesta e em futuras pesquisas. Este estudo preliminar teve como resultado final a proposta de um *framework* computacional que pode ser aplicado para a implementação de algoritmos ACO e (iii) ajudar a estabelecer um algoritmo-base a ser utilizado no restante da pesquisa.

Para apresentar os resultados obtidos, este capítulo é dividido da seguinte forma: na seção 3.1 é realizada uma análise do algoritmo ACO do ponto de vista computacional; na seção 3.2, são apresentados os experimentos computacionais para a implementação do algoritmo ACO; na seção 3.3 são apresentados os resultados obtidos; por fim, na seção 3.4 são realizadas algumas considerações finais.

3.1 Uma análise aprofundada do algoritmo ACO

Para o trabalho proposto nesta seção, o primeiro passo realizado foi determinar quais os principais blocos do algoritmo ACO que interagem entre si. Para isso, escolheu-se realizar uma releitura do algoritmo 2.1, conforme mostrado no algoritmo 3.1. Como o resultado da implementação deste conjunto de elementos tem como objetivo a geração de um componente reutilizável de software, estes serão referidos como um *framework* computacional.

É importante que seja observado que todos os passos previstos no algoritmo 2.1 são contemplados no algoritmo 3.1. As principais diferenças são:

- O algoritmo 3.1 divide a regra de atualização global do feromônio presente no passo 7 do algoritmo 2.1 em duas etapas: a primeira relaciona o efeito de cada formiga no feromônio após a construção de todas as soluções de todas as formigas de todas as colônias. A segunda trata de uma regra de atualização geral de todos os feromônios de

1 **Inicialização do algoritmo**
2 | Atribua a cada formiga o estado inicial para o agente, incluindo a propriedade de posição inicial
3 | Se necessário, crie mais colônias
4 **Fim da inicialização**
5 **repita** *Neste nível, cada execução é chamada* ***iteração***
6 | Se necessário, reinicialize todas as formigas ao seu estado inicial
7 | **repita** *Neste nível, cada execução é chamado* ***passo***
8 | | Faça todas as formigas de todas as colônias escolher o próximo passo e mova-as
9 | | Se necessário, execute a regra de atualização local de feromônios
10 | **até** *Até que todas as formigas tenham construído uma solução*
11 | **Execute a regra de atualização global de feromônio em dois estágios:**
12 | | Um reforço individual é aplicado à trilha
13 | | Um reforço global é aplicado
14 | **Fim da atualização global**
15 | Se aplicável, execute procedimentos de busca local.
16 | Se necessário, execute a regra de comunicação entre colônias, respeitando o descrito no modulo de troca
17 | Até que a condição de parada seja satisfeita
18 **até** *que a condição de parada seja satisfeita*

Algoritmo 3.1: Releitura do algoritmo ACO

todas as colônias. Como exemplo de uma regra de atualização geral de feromônios, temos a evaporação, quando todos os valores são apenas multiplicados por uma constante.

- O algoritmo 3.1 inclui um procedimento opcional de busca local, mostrado no passo 15.

Englobando e agrupando todas as operações e entidades computacionais até agora descritas, temos a estrutura mostrada na figura 3.1. Esta estrutura foi pensada como um componente independente de *software* que recebe: i) informações de configuração; ii) um problema de otimização combinatória representado através de um formato conhecido; e obtém a resposta do problema por meio de seus algoritmos internos. Este componente é composto internamente por pelo menos uma Colônia, que contém pelo menos uma Formiga. Possui também um Espaço de Busca e, caso necessário pode possuir um Módulo de Troca.

O relacionamento entre os elementos é mostrado na Figura 3.1. Neste diagrama em blocos, podem ser percebidas algumas características importantes:

- A Formiga não está diretamente conectada ao Espaço de Buscas ou ao Módulo de Troca. Sua ligação é com o Gerenciador da Colônia. Com isso, objetiva-se permitir que cada colônia possa ter um filtro diferente do estado atual do espaço de buscas. Desta forma, a implementação e experimentação de algoritmos de múltiplas Colônias de comportamento

heterogêneo se torna possível.

- Da mesma forma, percebe-se que cada colônia c possui kc formigas. Com isso, tem-se que a composição de cada Colônia pode ser diferente, tanto em número quanto na diversidade dos agentes.
- Nota-se também que o Espaço de Busca, o Módulo de Troca e a Colônia estão interligados.
- O Espaço de Busca é alterado e lido pela Colônia durante a criação da solução pelas formigas
- O Módulo de Troca é acionado: caso necessário, são lidas informações sobre a Colônia e/ou as Formigas, alterando o espaço de buscas conforme requerido pelo algoritmo.
- O Módulo de Troca consegue alterar trechos do espaço de busca específicos de cada Colônia. O valor do feromônio percebido por Formigas de uma Colônia c_1 em um trecho qualquer deve ser independente do valor do feromônio percebido por Formigas de uma Colônia c_2 no mesmo trecho.
- Para gerenciar estes fluxos de dados, foi criado um elemento denominado "Gerenciador do *Framework*". Este gerenciador tem como objetivo apenas implementar o algoritmo apresentado no algoritmo 3.1, deixando a execução das ações para os elementos que já discutidos.

A tabela 3.1 relaciona os itens de ação do algoritmo 3.1 com as ações tomadas pelos componentes presentes na figura 3.1. Nesta tabela, foi usada a seguinte nomenclatura para identificar os elementos da figura 3.1: FM – Gerenciador do *Framework*; SS – Espaço de Busca; CM – Gerenciador de Colônia; A – Formiga; EM – Módulo de Troca.

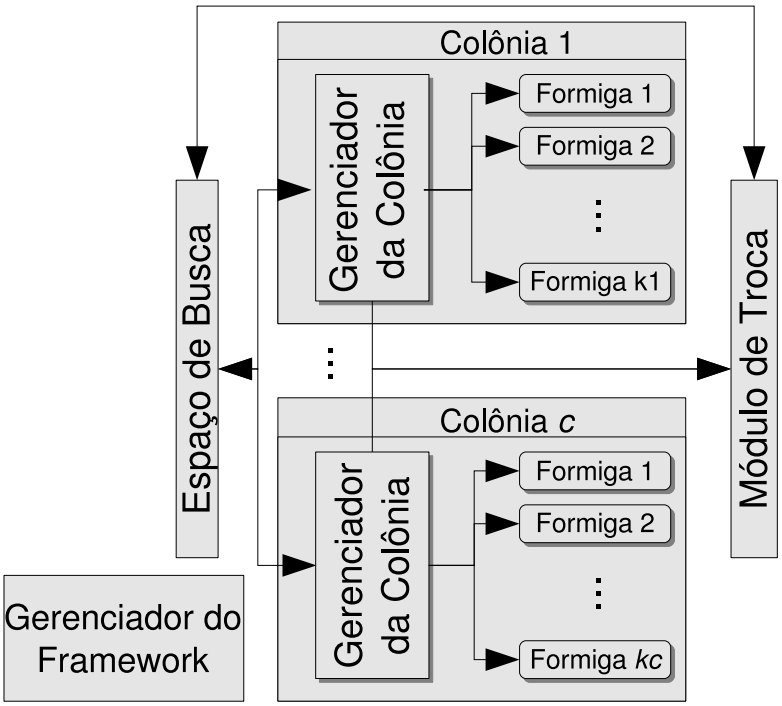


Figura 3.1: Diagrama em Blocos dos elementos de um algoritmo ACO genérico

Tabela 3.1: Ações tomadas pelos elementos definidos na figura 3.1 relativos a cada passo do algoritmo 3.1

	Item da listagem 2	**Ações Relacionadas**
1	Inicialize	
1.1	Atribua a cada formiga o estado padrão para o agente, incluindo a propriedade de posição inicial.	
2	Execute	
3	Reinicialize todas as formigas ao seu estado padrão	- FM – Envia um sinal de reinicialização para cada Colônia - CM – Cada colônia envia um sinal de reinicialização para cada formiga - A – Volta ao estado estabelecido no item 1.1
4	Execute	

Continua na próxima página

Tabela 3.1 Ações tomadas pelos elementos definidos na figura 3.1 relativos a cada passo do algoritmo 3.1– cont.

	Item da listagem 2	**Ações Relacionadas**
5	Faça todas as formigas de todas as colônias escolher o próximo passo e as mova	- FM – Solicita que cada CM realize uma movimentação de um passo em cada formiga - CM – Move cada formiga um passo - A – Move um passo
6	Se necessário, execute a regra de atualização local de feromônios	- CM – Após receber a movimentação da formiga, realize o depósito do feromônio correspondente
7	Até que todas as formigas tenham construído uma solução completa	
8	Execute a regra de atualização global de feromônio em dois estágios:	
8.1	Um reforço individual é aplicado à trilha.	- FM – Solicita que cada colônia atue sobre os respectivos feromônios de acordo com suas próprias regras - CM – Aplica a regra de atualização para cada formiga
8.2	Um reforço global é aplicado	- FM – Solicita que cada colônia atue sobre os respectivos feromônios de acordo com suas próprias regras - CM – Redireciona a atualização do feromônio - SS – Aplica a regra de atualização geral para os feromônios relacionados à colônia representada pelo CM

Continua na próxima página

Tabela 3.1 Ações tomadas pelos elementos definidos na figura 3.1 relativos a cada passo do algoritmo 3.1– cont.

	Item da listagem 2	**Ações Relacionadas**
9	Se aplicável, execute procedimentos de busca local.	- FM – Solicita a aplicação de procedimentos de busca local para cada colônia - CM – Aplica os procedimentos de busca local
10	Se necessário, execute a regra de comunicação entre colônias, respeitando o descrito no modulo de troca	- FM – solicita ao módulo de troca que realize a interação entre colônias - EM – Realiza a interação entre colônias
11	Até que a condição de parada seja satisfeita	

3.2 Experimentos computacionais

Para validar a estrutura presente na seção 3.1, escolheu-se realizar a implementação de uma estrutura computacional equivalente à figura 3.1. Desta forma, nesta seção é mostrada esta implementação.

Para testar a estrutura computacional (*framework*) proposta, decidiu-se implementar algumas variações do algoritmo ACO apresentadas no capítulo 2. Para a escolha de quais algoritmos deveriam ser implementados, considerou-se a tabela 3.1, e se buscou determinar um conjunto inicial que permitisse a demonstração de todas as possibilidades de variação mostradas nas cinco colunas da direita da tabela 3.1. A implementação tomou como ponto de partida o algoritmo Ant-Cycle, devido a dois motivos:

(a) O Ant-Cycle possui todas as rotinas de controle das entidades já apresentadas, como movimentação de formigas e alteração do feromônio;

(b) Como o depósito do feromônio é realizado após a determinação da solução, a conferência da execução correta por meio da depuração do programa se tornou mais simples;

Desta forma, foram escolhidos os algoritmos mostrados na tabela 3.2. A execução destes algoritmos foi realizada com o uso de 8 instâncias do problema do caixeiro viajante assimétrico encontradas em TSPLib (2007), com 17, 33, 35, 38, 53, 70 e 170 cidades. O número de formigas utilizadas foi $k = 10$, seguindo o observado em Dorigo e Gambardella (1997). Para o caso de algoritmos que apresentam múltiplas colônias, escolheu-se aleatoriamente o número de 5 colônias. Para a implementação do *Framework*, escolheu-se a linguagem JAVA, sendo implementada uma classe para cada elemento descrito anteriormente. Conforme será mostrado a seguir, o uso de uma linguagem orientada a objeto se mostrou uma forma eficiente e segura de se realizar novas implementações e modificações de métodos já existentes tendo como base o Ant-Cycle.

Algoritmo	Fonte	Justificativa
Ant-Cycle	Colorni et al. (1991)	O Ant-Cycle foi utilizado como implementação-base.
AS (Ant-Quantity e Ant-Density)	Colorni et al. (1991)	Demonstrar a viabilidade da arquitetura proposta para o uso exclusivo de regra local de depósito de feromônio (passo 6 da listagem 3)
ACS (Ant Colony System)	Gambardella e Dorigo (1996)	Demonstrar o uso da arquitetura proposta para o uso conjunto de regras locais e globais para depósito de feromônio
MMAS (Max Min Ant System)	Stützle e Hoos (2000)	Demonstrar o uso da arquitetura proposta para possíveis evoluções de algoritmos já existentes. A escolha do algoritmo MMAS é justificado neste trabalho com a análise de Dorigo e Blum (2005), que indicam este como uma das variações mais bem-sucedidas de algoritmos otimizados por formigas artificiais.
Múltiplo AS (Homogênio e heterogênio)	Baseado nos resultados de Ellabib et al. (2007)	Demonstrar o uso do módulo de troca para aplicações com múltiplas colônias.

Tabela 3.2: Algoritmos escolhidos para a implementação.

3.2.1 Decisões de projeto

O uso de linguagens como JAVA, que usam o conceito de orientação a objeto, permite o encapsulamento do código em classes. De forma resumida, uma classe é um trecho de código que contém um conjunto de dados e funções necessários para lidar com estes dados (DEITEL; J., 2005, p. 68)). Uma das etapas do projeto de um software orientado a objeto é separar

quais dados e operações serão realizados por quais classes. Para isso, criaram-se classes que representam cada um dos blocos mostrados na Figura 3.1. Para simplificar a documentação e a manutenção do código, escolheu-se dividir as classes em três grupos (implementados na linguagem JAVA como pacotes):

- Classes **base** (pacote *core*), que possuem a implementação base da meta-heurística. Não existe previsão para alteração de código nestes pacotes. Como exemplo, tem-se as classes Ant e Colony.
- Classes **intermediárias** (pacote *mutable*), que possuem a implementação das classes que fazem parte do *framework*, mas que podem sofrer modificações caso sejam adicionadas novas implementações. Como exemplo, tem-se a classe *Ant Factory*, que deve ser alterada a cada nova classe implementada.
- Classes **de implementação** (pacote *implementations*), que nada mais são do que classes que realmente implementam os comportamentos das diferentes heurísticas no *framework*.

A implementação de estruturas computacionais em linguagens que se utilizem de orientação a objeto como JAVA e C++ é muitas vezes guiada por padrões de projeto, com o intuito de tornar o software desenvolvido mais flexível e reutilizável (GAMMA *et al.*, 1995). Assim, outra decisão tomada foi a de se utilizar padrões de projeto (do inglês, *design patterns*). Neste projeto, é usado o padrão de projeto Fábrica (do inglês, *Factory*) (Gamma *et al.* (1995)) para a criação de vários objetos. Na Figura 3.2, o diagrama UML de classes deste padrão de projeto é apresentado, já o aplicando na geração dos objetos "*Ant*" (cada instância da classe "*Ant*" é uma formiga da colônia). O diagrama da Figura 3.2 possui os seguintes elementos: (i) uma classe *Colony*, que representa uma colônia; (ii) uma classe abstrata *Ant*, que representa uma formiga; (iii) uma classe *ASAnt*, classe derivada de *Ant*, que representa uma formiga que segue as regras estabelecidas pelo algoritmo AS; (iv) uma classe *AntFactory*, que gera formigas para a colônia. As demais classes do projeto, até mesmo as que implementam outros algoritmos, foram suprimidas para simplificar a visualização.

A aplicação do padrão Fábrica na criação de formigas de uma colônia é benéfica, principalmente quando se percebe que uma colônia pode requerer objetos que representem formigas instanciadas de diferentes classes, ou seja, com comportamentos diversos. Neste padrão de projeto, não se instancia *Ant*, mas sim suas classes derivadas. As instâncias das classes derivadas de *Ant* são obtidas pelo método estático *createAnt* da classe *AntFactory*. Ao enviar a solicitação de criação da formiga para o método *createAnt*, é passado um identificador numérico do tipo da formiga a ser criada. Esse método cria o objeto indicado e o devolve

convertido em uma instância de *Ant*. Desta forma, a colônia pode abrigar um conjunto heterogêneo de formigas.

Foi realizada ainda uma alteração no padrão de projeto Fábrica definido por Gamma *et al.* (1995). Se o projeto fosse seguir na totalidade o diagrama de classes apresentado pelos autores, o método *createAnt* estaria localizado na classe *Ant*. Desta forma, a inclusão de nova classe significaria a necessidade da alteração da classe *Ant*. Assim, para simplificar futuras mudanças, escolheu-se remover o método de criação das formigas e colocá-lo em uma classe específica.

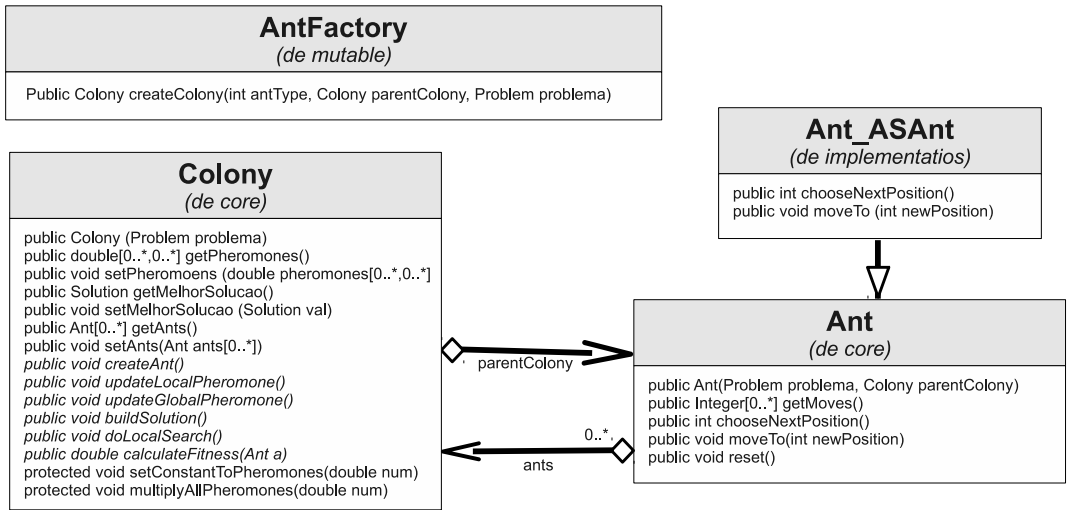


Figura 3.2: *Diagrama de classes para a geração e uso de formigas*

O comportamento encontrado com o uso do padrão Fábrica para a criação de formigas é muito similar ao que ocorre na criação de colônias pelo gerenciador de colônias, mostrado na Figura 3.3. Nesta figura, onde foram representados apenas os elementos relacionados à criação da colônia, existem os seguintes elementos:

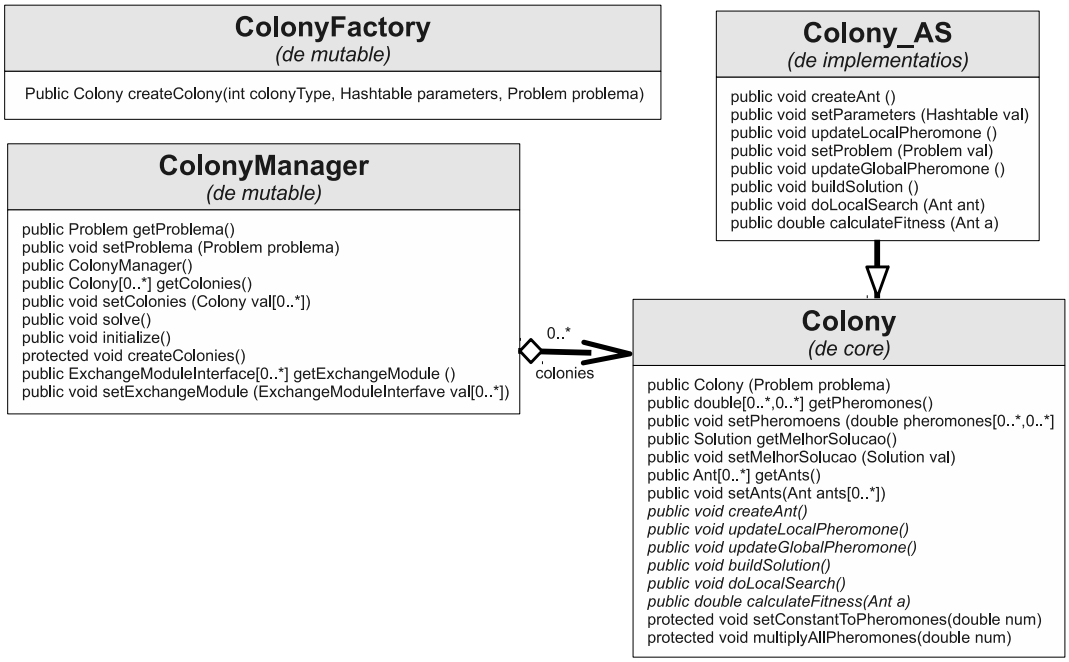


Figura 3.3: *Diagrama de classes para a geração e uso de colônias*

(i) uma classe *ColonyManager* que além de representar o gerenciador de colônias também as armazena;

(ii) a mesma classe *Colony* da figura 3.2;

(iii) uma classe *Colony_AS* que representa uma colônia de formigas que respeita a regra do AS;

(iv) uma classe *ColonyFactory*, análoga à classe *AntFactory* da figura 3.2.

Assim, para criar um conjunto de colônias e suas respectivas formigas, são seguidos os seguintes passos:

1. O Gerenciador do *Framework* inicializa a classe *ColonyManager*. Durante o processo de inicialização, o *ColonyManager* recebe parâmetros para a criação das colônias.

1.1. Seguindo os parâmetros determinados no passo anterior, o *ColonyManager* cria as colônias a serem usadas através da classe *ColonyFactory*.

2. O Gerenciador de *Framework* solicita à classe *ColonyManager* a criação das formigas para todas as colônias.

2.1. A classe *ColonyManager* repassa esta solicitação a todas as colônias.

2.2. Cada colônia cria suas próprias formigas, conforme mostrado anteriormente.

Caso seja necessário, a implementação de novas variações do algoritmo AS é realizada utilizando os seguintes passos:

1. Se necessário, crie uma nova classe derivada da classe abstrata *Colony* (ou de uma subclasse de *Colony*) que implementa a manipulação de feromônios;
2. Se necessário, crie uma nova classe derivada da classe abstrata *Ant* (ou de uma subclasse de *Ant*) que implementa o comportamento da formiga;
3. Caso tenha criado uma nova classe representando uma colônia, modifique a classe *ColonyFactory* para que seja possível a obtenção de uma nova instância da nova classe;
4. Caso tenha criado uma nova classe representando uma formiga, modifique a classe *AntFactory* para que seja possível a obtenção de uma nova instância da nova classe;

A seguir, tratar-se-a da implementação dos algoritmos já citados no item anterior.

3.2.1.1 A implementação do *Ant-Quantity* e *Ant-Density*

Com a implementação do funcionamento descrito na tabela 3.2, a implementação do algoritmo *Ant-Quantity* descrito anteriormente é realizada usando como base a entidade "Colônia". A classe referente a esta entidade é herdada pela nova classe para a implementação do algoritmo *Ant-Quantity*. As mudanças realizadas foram:

- A implementação da regra local de depósito de feromônio descrita na equação 2.2;
- A eliminação de regras de depósito de feromônio global.

A implementação do algoritmo *Ant-Density* foi muito semelhante à implementação do algoritmo *Ant-Quantity*. A única diferença foi na implementação das regras locais de depósito de feromônio, cujo valor se torna função da visibilidade.

3.2.1.2 A implementação do *Ant Colony System*

A implementação do *Ant Colony System* se deu por meio da criação de duas classes distintas:

- Uma classe estende a classe referente à entidade “Colônia”, de forma semelhante ao descrito na implementação dos algoritmos *Ant-Quantity* e *Ant-Density*. Porém, esta classe implementa a atualização do feromônio tanto localmente quanto globalmente, conforme descrito por Gambardella e Dorigo (1996) (equações 2.2 a 2.4);

- Uma segunda classe, que estende a entidade “Formiga”, implementa a regra de transição do ACS.

3.2.1.3 A implementação do *Max-Min Ant System*

Com a implementação do *Max-Min Ant System* (MMAS), o uso da implementação usando uma linguagem orientada a objeto se mostra uma boa escolha. Segundo Stutzle e Hoos (2000), o algoritmo MMAS se diferencia do Ant-System por dois pontos principais:

- A única atualização de feromônios existente é realizada apenas pela formiga que possui a melhor solução;

- A quantidade de feromônio existente em uma trilha é limitada entre dois valores (um mínimo e outro máximo, derivando daí o nome MMAS).

Ao recordar a regra de atualização global de feromônios do algoritmo ACS, percebe-se que a mesma também realiza a atualização de feromônios do melhor caminho. Desta forma, a implementação do MMAS foi realizada estendendo à classe Colônia referente ao algoritmo ACS (descrita no item 4.2). A implementação do MMAS foi simplificada nos seguintes passos:

- A atualização local do feromônio (equação 2.2) é eliminada (ou seja, um método vazio sobrescreve o método da classe referente à entidade colônia ACS);

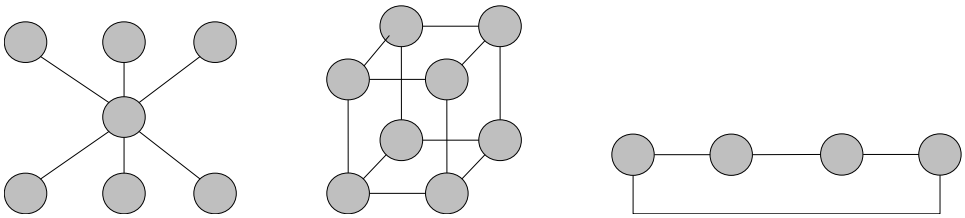
- A atualização global do feromônio (equação 2.4) é realizada conforme a regra do ACS (bastando para isso uma chamada ao método da superclasse).

Após a atualização global dos feromônios, os valores destes são limitados entre os valores determinados pelo algoritmo. Note que esta é a única implementação necessária para o MMAS.

A regra de transição descrita por Stutzle e Hoos (2000) pode ser entendida como sendo a mesma regra do ACS. Assim, para a implementação da classe Formiga do MMAS, é suficiente o uso da a classe referente à Formiga original.

3.2.1.4 A implementação de estratégias de otimização multicolônias de formigas com colônias homogêneas

Ellabib *et al.* (2007) descrevem três estratégias de troca de informação entre colônias, conforme mostrado na Figura 3.4. São elas: estrela (Figura 3.4a), hipercubo (Figura 3.4b) e anel (Figura 3.4c). Como exemplo, foi feita a troca de informação usando a estratégia de ligação por anel. O exemplo implementado se inspirou nos resultados apresentados por Ellabib *et al.* (2007), criando o seguinte comportamento: seja c colônias ligadas conforme mostrado na Figura 3.4c, e c_m e c_n duas colônias imediatamente adjacentes, tal que $m = n - 1$. Caso o melhor valor da função objetivo encontrado na colônia m seja melhor que o valor encontrado na colônia n, o melhor caminho de m passa para n. Para experimentação, utilizou-se colônias e formigas que implementam o algoritmo ACS (apresentado na seção 3.2.1.2). É interessante notar que, embora este algoritmo seja complexo, a estrutura computacional proposta neste trabalho, em conjunto com a implementação por meio de técnicas de programação orientada a objeto, reduziu a implementação necessária a uma evolução do código obtido anteriormente. Desta forma, conseguiu-se implementar um algoritmo de múltiplas colônias de formigas reescrevendo um método de uma classe. Neste trabalho este algoritmo foi denominado de *Multiple Ant Colony – Homogenius* (MAC-Ho).



(a) Estrela (b) Hipercubo (c) Anel

Figura 3.4: Três estratégias de troca de informação entre algoritmos ACS de múltiplas colônias. Nos três casos, cada círculo representa uma colônia

3.2.1.5 A implementação de estratégias de otimização multicolônias de formigas com colônias heterogêneas

Por fim, buscou-se implementar o caso de maior variabilidade: um algoritmo baseado em múltiplas colônias de formigas artificiais diferentes atuando em conjunto para a resolução de um problema. Para isso, definiu-se o seguinte cenário, usando três colônias distintas:

- A primeira colônia obterá a solução com o algoritmo ACS
- A segunda colônia obterá a solução com o algoritmo MMAS

- A terceira colônia obterá a solução com o algoritmo *Ant-Cycle*

Adicionalmente, será usada a mesma estratégia de troca de informações presente no algoritmo MAC-Ho.

É importante notar que, para a execução deste cenário, não foi necessária nenhuma alteração em código-fonte. A única alteração foi em arquivos de configuração que definem a composição do otimizador.

3.3 Resultados obtidos

Nesta seção, são avaliados resultados da implementação dos algoritmos vistos na seção anterior utilizando-se o *framework* proposto. Para tal, toma-se como base o trabalho de Chidamber e Kemerer (1994) que propõe um conjunto de métricas para o esforço de implementação de sistemas orientados a objeto. Os autores propõem alguns indicadores, entre eles:

- Métodos ponderados por classe (WMC – *Weighted Method per class*), definido como a soma da complexidade de cada método implementado na classe. Nesta análise, supõe-se a complexidade de todos os métodos unitária, sendo que esta métrica torna o número de métodos por classe.
- Profundidade da árvore de herança (DIT – *Depth of inheritance tree*), definido como o maior número de classes-pai existentes no sistema.

Para realizar esta análise, é necessário entender como o sistema foi implementado permitindo o reuso, mostrado na Figura 3.5.

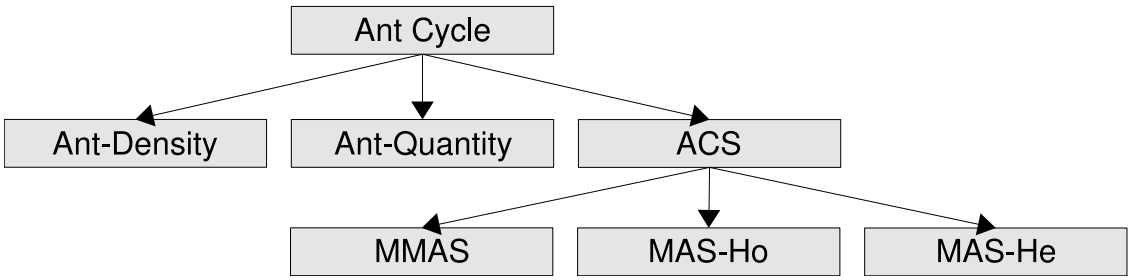


Figura 3.5: Reuso de código na implementação dos algoritmos estudados

Os resultados obtidos são interessantes, pois demonstram que, ao se obter um conjunto de algoritmos implementados por meio do *framework* proposto, implementações de sistemas muito mais complexos se tornam possíveis com um esforço ínfimo de codificação. Na verdade, o maior esforço detectado por meio das métricas analisadas foi para gerar o algoritmo *Ant-Cycle*, um dos mais simples conceitualmente entre todos os implementados. Com esta estrutura, foram obtidos os resultados mostrados na tabela 3.3 . Nesta tabela, realizou-se duas análises da métrica WMC: na primeira, o número total de métodos (WMC Total) é um indicador da complexidade da implementação do algoritmo sem a utilização do *framework* proposto; a segunda, refere-se à quantidade de métodos efetivamente implementados para o algoritmo analisado.

Algoritmo	WMC sem se considerar o framework proposto	WMC requerido ao se usar o framework proposto	DIT
AntCycle	38	38	0
AntDensity	38	3	1
AntQuantity	38	3	1
ACS	39	5	1
MMAS	40	3	2
Mac-Ho	38	1	2
Mac-He	42	0	2

Tabela 3.3: Análise comparativa dos algoritmos estudados

Ao observar a tabela 3.3, pode-se notar que o maior número de métodos necessários para a implementação das variações dos algoritmos AS se deu na implementação do *Ant-Cycle*, seguido pelo ACS, *Ant-Density*/*Ant-Quantity*, MMAS/MAC-Ho e, por fim, o MAC-He. Para compreender este resultado, é importante que se entenda as etapas de desenvolvimento do sistema analisado. Cada algoritmo, com exceção do *Ant-Cycle*, implementou apenas as variações necessárias em relação a um algoritmo-base.

Estes algoritmos foram implementados e executados. Nas tabelas 3.4 e 3.5, são mostrados os resultados (média e desvio padrão, respectivamente) dos algoritmos aplicados a 8 instâncias distintas do problema do caixeiro viajante assimétrico (ATSP – *Asymetric Traveling Salesman Problem*) disponível na TSPLib[1]. Os melhores valores da média e desvio padrão foram destacados em negrito. O quadro 3.1 traz a dimensão dos problemas estudados. Cada problema foi resolvido com cada algoritmo 10 vezes. A parametrização foi a mesma para todos os algoritmos. O resultado é mostrado em termos de variação do valor ótimo disponível na biblioteca TSPLib. A variação do valor ótimo é definida conforme a equação 3.1. Os tempos médios de execução são mostrados na Tabela 3.6.

$$\Delta_{opt} = \frac{V_{encontrado} - V_{ótimo}}{V_{ótimo}} \tag{3.1}$$

Onde:

- Δopt indica a variação do ótimo
- $V_{encontrado}$ indica o valor encontrado (ou seja, a distância total percorrida) para a resposta do problema após a aplicação do algoritmo
- $V_{ótimo}$ indica o valor ótimo disponível (ou seja, a distância total percorrida) para a resposta

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

do problema

Para a análise dos valores de Δopt é importante notar que, embora o ACS e outras meta-heurísticas como algoritmos genéticos e busca tabu, dificilmente conseguirem alcançar resultados aceitáveis quando comparados com heurísticas desenvolvidas especificamente para o problema do ATSP. Porém, é conhecido que para problemas mais complexos, os resultados obtidos pelo ACS são competitivos.

Problema	**Número de cidades**
br17.atsp	17
ft53.atsp	53
ft70.atsp	70
ftv170.atsp	170
ftv33.atsp	33
ftv35.atsp	35
ftv38.atsp	38
ftv70.atsp	70

Quadro 3.1: Dimensão dos problemas analisados

Instância	**Ant Quantity**	**Ant Density**	**ACS**	**MMAS**	**MAS-Ho**	**MAS-He**
br17.atsp	135.90%	132.05%	135.90%	**39.23%**	135.90%	43.59%
ft53.atsp	39.43%	38.86%	33.88%	27.18%	32.11%	**26.66%**
ft70.atsp	17.47%	16.69%	11.06%	12.89%	**10.00%**	12.92%
ftv170.atsp	40.83%	39.76%	38.85%	**26.95%**	36.92%	28.41%
ftv33.atsp	20.86%	18.72%	24.31%	**9.35%**	22.56%	10.89%
ftv35.atsp	18.39%	18.16%	19.30%	**9.14%**	15.27%	9.34%
ftv38.atsp	19.39%	19.27%	12.96%	10.34%	13.14%	**8.87%**
ftv70.atsp	29.01%	29.06%	27.90%	**17.66%**	20.79%	17.68%

Tabela 3.4: Média dos resultados encontrados pelos algoritmos após execução de testes computacionais

Instância	Ant Quantity	Ant Density	ACS	MMAS	MAS-Ho	MAS-He
br17.atsp	11.38	4.74	**0**	5.38	**0**	**0**
ft53.atsp	263.53	221.84	142.07	119.36	145.99	**95.76**
ft70.atsp	595.72	496.96	**265.63**	310.23	298.93	405.74
ftv170.atsp	85.13	99.49	95.51	62.97	70.24	**60.65**
ftv33.atsp	70.76	58.77	62.46	19.31	45.47	**8.62**
ftv35.atsp	84.23	40.39	46.37	**18.16**	45.52	20.12
ftv38.atsp	100.8	48.21	25.06	28.41	18.8	**18.59**
ftv70.atsp	99.08	70.44	91.72	**17.41**	47.23	37.69

Tabela 3.5: Desvio padrão dos resultados encontrados pelos algoritmos após execução de testes computacionais

Instância	Ant Quantity	Ant Density	ACS	MMAS	MAS-Ho	MAS-He
br17.atsp	398	140	**93**	100	483	492
ftv33.atsp	586	573	**362**	368	1810	1925
ftv35.atsp	658	685	411	**409**	2017	2324
ftv38.atsp	780	838	508	**495**	2440	2720
ft53.atsp	1576	1691	**918**	939	4693	4777
ft70.atsp	2558	2810	1709	**1579**	8520	8535
ftv70.atsp	2563	2850	1781	**1617**	7887	8587
ftv170.atsp	15496	16139	9768	**9704**	47746	50950

Tabela 3.6: Tempos computacionais típicos para a obtenção do resultado (em milissegundos)

Aos observar os dados, percebe-se que os algoritmos conseguiram realizar otimização do problema em um tempo computacional relativamente pequeno (sempre menor que 1 minuto, mesmo para as maiores instâncias). Com a parametrização estabelecida, o algoritmo MMAS obteve melhores resultados em grande maioria dos problemas, embora semelhante ao MAS-He. Adicionalmente, os tempos computacionais necessários para o processamento dos algoritmos foi significativamente menor na execução do MMAS, principalmente quando comparados com os algoritmos que se utilizam de várias colônias (MAS-Ho e MAS-He), assim como a quantidade de parâmetros envolvidos(o MAS-He possui um conjunto independente de

parâmetros para cada colônia). Também nota-se claramente que a dificuldade de um problema do tipo caixeiro viajante assimétrico ser resolvido pelo algoritmo da colônia de formigas depende de fatores além do simples número de cidades. Isto é verificado quando se percebe que o ACS, aplicado de forma idêntica às instâncias br17 (17 cidades) e ft70 (70 cidades), teve um desvio do ótimo de 136% e 12%, respectivamente. A existência de um parâmetro adicional que reflita a complexidade do problema também é indicada quando se analisa apenas a resposta dos algoritmos nas instâncias de 70 cidades (ft70 e ftv70). Por fim, independentemente dos resultados de Δopt obtidos, a validação do *framework* como uma estrutura computacional capaz de suportar a implementação de vários tipos de algoritmos ACO com pouco esforço de computação também é um resultado importante nessa etapa do trabalho.

3.4 Considerações finais

Este capítulo apresentou um conjunto de experimentos realizados para o estudo inicial do algoritmo ACO e suas variações. Para tal, utilizaram-se os algoritmos AS mais conhecidos na literatura. Buscou-se então definir um conjunto mínimo de entidades independentes que, ao interagirem, pudessem estabelecer um conjunto de comportamentos presentes nos algoritmos apresentados.

Durante o desenvolvimento da arquitetura proposta, notou-se que existem ganhos em termos de custos computacionais quando se fornece à entidade Formiga informações sobre o problema a ser resolvido. No caso, percebeu-se que a execução da lista tabu – ou seja, a memória dos pontos do grafo já visitados pela formiga - se torna muito mais rápida quando se consegue estabelecer o tamanho do problema.

Ao se implementar a estrutura proposta pelo paradigma de orientação a objeto, notou-se que o conceito de herança de classes permite o reaproveitamento de código, tornando a evolução de novos algoritmos uma tarefa muito mais simples.

Com o estabelecimento das regras que são alteradas entre uma evolução e outra dos algoritmos baseados em AS, consegue-se fazer uma análise comparativa como a apresentada na tabela 3.3.

A definição e validação inicial do *framework* computacional aqui apresentada permitem a proposição de um trabalho a ser desenvolvido no futuro: a implementação desta em sistemas dedicados. A implementação desta estrutura em vários núcleos de sistemas dedicados pareceu aos autores ser totalmente viável, sendo necessário um conjunto maior de experimentações para a comprovação.

Percebeu-se que o *framework* proposto permitiu a rápida implementação e caracterização de algoritmos complexos, como o MMAS e até mesmo estruturas multicolônias com um esforço reduzido – foi necessária apenas a implementação de 2% a 13% dos métodos necessários para o funcionamento do sistema (desconsiderando o algoritmo Mac-He que não requereu nenhuma implementação, apenas mudança em arquivos de configuração). Assim, acredita-se que a proposta para um sistema de prototipagem de sistemas baseados em algoritmos AS foi alcançada.

Os experimentos realizados aqui mostraram que o algoritmo MMAS, embora de implementação mais trabalhosa quando se compara com algoritmos como o ACS, permitiu resultados superiores nos problemas estudados, quando se analisou os valores obtidos e os tempos computacionais. Adicionalmente, a dificuldade na parametrização deste algoritmo é menor que a encontrada em sistemas com múltiplas colônias. A grande diferença entre os resultados obtidos com o algoritmo MMAS, quando relacionados com os demais algoritmos, nos indicou que a implementação deste seria suficiente para o desenvolvimento das técnicas apresentadas nos capítulos seguintes.

4 *Proposta de solução para o problema de scheduling em ambiente de máquina única com terceirização permitida usando a técnica de otimização por colônia de formigas*

Este capítulo propõe e implementa um método baseado em ACO para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$. Este problema é inicialmente mostrado na literatura por Lee e Sung (2008b) e definido da seguinte forma: considere um conjunto de n tarefas que devem ser alocados em uma máquina única ou realizadas através de terceirização. O objetivo é minimizar a soma dos tempos de finalização de cada tarefa ($\sum C_j$) e do custo total de terceirização (OC). Adicionalmente, existe um limite para o custo total de terceirização, definido por Lee e Sung (2008b) como $Budget$. A soma de $\sum C_j$ e OC é ponderada através de um parâmetro de custo $0 < \delta < 1$.

Para resolver esse problema, são propostas, nesse capítulo 4, estratégias distintas: as duas primeiras abordam o problema usando modelos de programação inteira; a terceira, usa um algoritmo *branch and bound* para obter a solução ótima do problema; por fim, um algoritmo ACO é proposto. Conforme será mostrado, o algoritmo ACO proposto nesse capítulo foi capaz de gerar resultados próximos do ótimo em um tempo computacional muito menor do que o necessário para as demais abordagens.

Este capítulo está dividido da seguinte forma: na seção 4.1, o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ é definido; na seção 4.2, são mostradas as abordagens para o problema usando programação inteira; na seção 4.3 é apresentado o algoritmo *branch and bound*; seção 4.4, o algoritmo ACO proposto é apresentado e implementado; na seção 4.5 são mostrados os resultados obtidos com a implementação do algoritmo; por fim, na seção 4.6 são mostradas as considerações finais deste capítulo.

4.1 Definição do problema

Neste capítulo, será abordado o problema de minimizar o somatório dos tempos de finalização de um conjunto de tarefas em um ambiente com uma única máquina. Este problema pode ser definido como: seja J_i um conjuto de n tarefas a serem sequenciadas para produção *in-house* ou terceirizadas. Cada tarefa J_i é definida por:

- Um tempo de processamento p_i;
- Uma data de entrega d_i;
- Um custo adicional existente quando se ocorre a terceirização da tarefa, o_i;
- Um tempo de entrega de terceirização l_i;

Conforme mencionado anteriormente, o problema possui adicionalmente dois parâmetros:

- Um parâmetro de custo $0 < \delta < 1$;
- Um parâmetro *Budget* que indica o valor total de orçamento disponível;

Cada tarefa pode ser sequenciada em uma máquina única ou terceirizada. A resposta é dada em função de dois conjuntos, a saber:

- O conjunto não ordenado O_π, contendo todas as tarefas terceirizados;
- O conjunto ordenado S_π, contendo todas as tarefas a serem realizados na máquina única;

Adicionalmente, para a definição do problema, estabelece-se o conjunto SP_k, que contém todas as tarefas S_π sequenciadas antes da tarefa k.

A função objetivo do problema pode então ser definida da seguinte forma: seja o término de uma tarefa i definido pela equação 4.1. O custo total de terceirização, $OC \leq Budget$, é definido pela equação 4.2.

$$C_i = \begin{cases} l_i & \text{se } i \in O_\pi \\ \sum\limits_{k \in SP_k} p_k + p_i & \text{se } i \in S_\pi \end{cases} \tag{4.1}$$

$$OC = \sum_{j \in O_\pi} o_j \tag{4.2}$$

Este problema é descrito por Lee e Sung (2008b) como um problema NP-difícil e descrito como $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC,\ 0 < \delta < 1$. Desta forma, este problema pode ser entendido como um problema de *scheduling* que busca a minimização de uma soma ponderada entre a soma dos tempos de término de cada tarefa e o custo total de terceirização.

O problema definido por Lee e Sung (2008b) não é o primeiro estudo que se foca em estratégias de terceirização. Porém, embora seja possível encontrar um conjunto de artigos que tratam do tema (por exemplo, Lee *et al.* (2002) usa uma abordagem baseada em algoritmos genéticos para tratar do planejamento e controle da produção em uma cadeia de suprimentos sem consideração de custos, Qi (2008) trata do *scheduling* em uma operação de dois estágios com *scheduling* permitido no primeiro estágio e Zapfel e Bogl (2008) aborda um problema real de atribuição de veículos e pessoal em uma agência dos correios com possibilidade de terceirização de operações), nenhum outro artigo encontrado na literatura se focou no problema de *scheduling* $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$.

Este problema é abordado de quatro formas nessa tese: na seção 4.2, são propostos modelos de programação inteira. Na seção 4.3, é proposta uma abordagem usando um algoritmo *branch-and-bound*. Na seção 4.4, é proposto o algoritmo ACO. Posteriormente, na seção 4.5, são mostrados resultados obtidos e na seção 4.6 são tecidas algumas considerações sobre os resultados obtidos pelos três métodos.

4.2 A abordagem ao problema usando um modelo de programação inteira

O problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ pode ser representado usando modelos de programação inteira mista. Assim, um modelo, baseado nos modelos encontrados em Arenales *et al.* (2007), para ambientes *flowshop*, é proposto. Esse modelo é apresentado na seção 4.2.1.

4.2.1 O modelo de programação matemática

Em resumo, a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ contém dois conjuntos: (i) O_π, um conjunto não ordenado que continha todas as tarefas a serem terceirizadas e (ii) S_π, um conjunto ordenado que possui todas as tarefas a serem realizadas na máquina única.

O conjunto S_π possui as tarefas sequenciadas de forma a minimizar o somatório dos tempos de processamento. Assim sendo, a resolução desse problema tem como objetivos: (i) separar as tarefas nos conjuntos S_π e O_π e (ii) ordenar as tarefas de S_π de forma a minimizar o somatório dos tempos de finalização de suas tarefas.

Existe porém, um resultado bem conhecido proveniente da literatura de *scheduling* que pode ser usado para simplificar a resolução do problema: de acordo com Baker (1943), a regra SPT (*Shortest Process Time* - menor tempo de processamento) minimiza o tempo de fluxo médio das tarefas. Quando todas as tarefas são liberadas no mesmo instante $r_i = 0$, o tempo de fluxo de uma tarefa $F_i = C_i - r_i$ é igual ao seu momento de finalização. Com isto, como se está trabalhando com tarefas que possuem o mesmo momento de liberação, minimizar o tempo de fluxo médio $\overline{F}$ também minimiza o tempo médio de finalização $\overline{C}$. Como $\overline{C} = \frac{\sum_{i=0}^{n} C_i}{n}$, em que n é o número de tarefas, nesse caso minimizar $\overline{C}$ é o mesmo que minimizar $\sum C_i$.

Com esse resultado, o modelo de programação inteira mista a ser desenvolvido pode ser simplificado, ao se incluir uma fase de pré-processamento que ordena as tarefas usando a regra SPT, gerando assim um conjunto S_{spt}. O modelo simplificado é mostrado a seguir:

As únicas variáveis necessárias são as variáveis y_i, que indicam se a tarefa é terceirizada ou não. A definição das variáveis y_i, idêntica ao modelo anterior, é mostrada novamente na equação 4.3:

$$y_i = \begin{cases} 1 & \text{Se a tarefa na posição } i \text{ da sequência inicial é terceirizada} \\ 0 & \text{Caso contrário} \end{cases} \tag{4.3}$$

A função objetivo do problema é mostrada na equação 4.4:

$$min\ z = (1-\delta) \cdot \left(\sum_{i \in S_{spt}} (y_i \cdot l_i) + \sum_{i \in S_{spt}} C_i \right) + \delta \cdot \sum_{i \in S_{spt}} (y_i \cdot o_i) \tag{4.4}$$

E as seguintes restrições se aplicam:

$$C_i \geq C_i^{'} - G \cdot y_i \tag{4.5}$$

$$C_i^{'} \geq C_i^{'} + p_i - G \cdot y_i \tag{4.6}$$

$$\sum_{i=1}^{n} y_i \cdot o_i \leq Budget \tag{4.7}$$

Sendo que G representa um número muito grande. As equações 4.5 e 4.6 definem o valor de C_i. A equação 4.7 define que o total a ser gasto com terceirização não deve passar do limite $Budget$.

O modelo apresentado nessa seção foi implementado, e usado para a resolução de um conjunto de testes, conforme será mostrado na seção 4.5.

4.3 O algoritmo *branch and bound*

Conforme poderá ser visto na seção 4.5, não foi possível obter o valor ótimo de problemas de maior escala usando procedimentos de programação matemática devido ao custo computacional. Diante disso, escolheu-se realizar o desenvolvimento de um algoritmo *branch and bound* para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$. Este procedimento se mostrou uma saída para a obtenção de valores ótimos de referência para o algoritmo ACO proposto na seção seguinte, ainda que seu tempo computacional para problemas de maior escala seja proibitivo.

Para a criação do algoritmo *branch and bound* foi utilizado o mesmo resultado que permitiu a simplificação do modelo de programação inteira apresentado na seção 4.2.1. Assim, a primeira premissa desse algoritmo é que o conjunto de tarefas do problema é ordenada previamente, gerando o conjunto ordenado de tarefas S_{spt}. Cada nível da árvore de busca se relaciona com uma tarefa de S_{spt}, sendo possível duas ramificações: uma indica que a tarefa é terceirizada e a outra indica que a tarefa é executada no ambiente de máquina única.

Determinou-se duas regras principais para este algoritmo:

1. A primeira regra, diz respeito à interrupção da busca em determinado ramo da árvore. Essa interrupção ocorre quando o valor da função objetivo da subsequência já definida é maior que o valor do limite atual. Esse limite é definido como o valor mínimo obtido entre: (i) o valor indicado pela equação 4.8 e (ii) os valores da função objetivo das sequências completas encontradas durante a execução do algoritmo;

$$F_{max} = min\{((1-\delta)\cdot\sum C_i);(\delta\cdot\sum o_i + (1-\delta)\cdot\sum l_i)\} \tag{4.8}$$

2. A segunda, diz respeito ao procedimento de ramificação da árvore. Para garantir que serão geradas apenas soluções viáveis, a tentativa de se terceirizar um trabalho J_i ocorrerá apenas quando $\sum_{j \in O'_\pi} o_j + o_i \leq Budget$. A tentativa de se incluir um trabalho no conjunto S_π ocorrerá sempre, desde que respeite a regra indicada no ítem 1.

Embora tenha sido obtido os valores ótimos para os problemas propostos, o tempo computacional necessário para a execução desse algoritmo é proibitivo para problemas de médio e grande porte conforme será mostrado na seção 4.5.3. Para resolver essa deficiência, na seção 4.4 é proposto e implementado um algoritmo ACO que mostrou ser capaz de trazer resultados próximos ao ótimo em tempos significativamente menores em comparação com as abordagens apresentadas até o momento.

4.4 O algoritmo ACO proposto

O método proposto neste trabalho, baseado na heurística ACO, é composto de dois estágios: o primeiro estágio apenas ordena as tarefas usando a regra SPT (*Shortest Processing Time* - Menor Tempo de Processamento) buscando a redução do espaço de busca. No segundo estágio, uma variação do algoritmo ACO é proposta aplicada, selecionando quais tarefas devem ser terceirizadas e quais não devem. Esse algoritmo apresenta algumas características específicas ao problema estudado: (i) uma representação em forma de grafo que permite ao ACO escolher se uma tarefa deve ou não ser terceirizada; (ii) uma regra de pré-seleção que garante que todas as soluções geradas pelo algoritmo sejam viáveis; (iii) uma nova regra de visibilidade para o ACO; (iv) uma busca local específica para o problema. O método proposto foi implementado em JAVA e executado em 7 grupos distintos de instâncias de teste. Os resultados obtidos se mostraram superiores aos encontrados por Lee e Sung (2008b). Adicionalmente, é mostrado que, quando se trabalha com o algoritmo de duas fases proposto, a busca local não é necessária para problemas de dimenões pequenas. Para problemas de dimensões maiores, o procedimento de busca local proposto aumenta a qualidade da resposta obtida.

O algoritmo ACO proposto é composto de duas fases distintas:

Fase 1 O grafo é gerado através do pré-processamento do problema, conforme mostrado no algoritmo 4.1;

Fase 2 É proposto um algoritmo MMAS específico para a resolução do problema representado por este grafo;

A seguir, cada fase será examinada em detalhes.

4.4.1 Fase 1: Criando o grafo

Um ponto fundamental para o método proposto neste trabalho para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ é a criação do grafo que representa o problema. Para isso, se utiliza da mesma estratégia aplicada nos procedimentos das seções 4.2.1 e 4.3: uma fase de pré-processamento das n tarefas que gera uma sequência S_{spt}. Ao aplicar este resultado, consegue-se reduzir o espaço de busca, permitindo que o grafo $n \times n$, seja substituído por um grafo $2 \times n$. A estratégia completa usada para a criação do grafo é mostrado no algoritmo 4.1.

1 Ordene todos os trabalhos de acordo com a regra SPT
2 **para** *i = 1 a n* **faça**
3 | Gere dois nós: N_i^0 que representa a terceirização do trabalho J_i e N_i^1 que representa a não terceirização de J_i
4 | Conecte o nó N_i^0 aos nós N_{i-1}^0, N_{i-1}^1, N_{i+1}^0 e N_{i+1}^1
5 | Conecte o nó N_i^1 aos nós N_{i-1}^0, N_{i-1}^1, N_{i+1}^0 e N_{i+1}^1
6 **fim**

Algoritmo 4.1: O algoritmo proposto para a geração do grafo

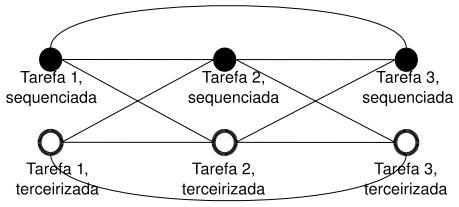


Figura 4.1: Um grafo representando um problema de *scheduling* de 3 trabalhos em um ambiente de máquina única com possibilidade de terceirização.

A figura 4.1 mostra um grafo representando um problema contendo 3 trabalhos gerado pelo algoritmo 4.1. Ao se comparar as figuras 2.4 e 4.1, pode se perceber que, na figura 4.1, o número de nós do grafo é duplicado e o número de arcos é reduzido. Ou seja, como a sequência de trabalhos é conhecida, o problema se torna em apenas determinar se o trabalho J_i deve ou não ser terceirizado.

4.4.2 Fase 2: A proposta de um algoritmo MMAS para o problema estudado

Após a realização da fase 1, o algoritmo baseado no MMAS proposto foi implementado. O algoritmo proposto é específico para o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, apresentando as seguintes características:

1. Implementação de uma regra de pré-seleção, indicada no algoritmo 4.2;
2. Implementação da visibilidade (η) conforme indicado na equação 4.9;
3. Implementação de um procedimento de busca local conforme indicado no algoritmo 4.3;

A seguir, estas modificações serão detalhadas.

4.4.2.1 A regra de pré-seleção

Esta regra de pré-seleção garante que todas as soluções construídas pelas formigas sejam viáveis. Para tal, é realizada uma comparação entre o custo de terceirização do trabalho e o orçamento disponível. Se existe orçamento disponível, a formiga se utiliza da regra de transição. Caso contrário, o trabalho é adicionado à S_π e a formiga se move para o próximo trabalho. Este comportamento é mostrado no algoritmo 4.2.

```
se Σ_{k∈O_π} (o_k + o_j) ≤ Budget então
    Escolha entre os nós j^0_ij e j^1_ij usando a regra de transição encontrado na equação 4.9
senão
    Escolha o trabalho j^1_ij
fim
```

Algoritmo 4.2: A regra de pré-seleção de nós

Desta forma, a regra de pré-seleção de nós define uma política de escolha de nós que permite que a terceirização ocorra apenas quando existe orçamento disponível para tal. Com isso, consegue-se reduzir o espaço de busca e evitar a geração de soluções inviáveis.

4.4.2.2 A definição da visibilidade (η)

Outra mudança realizada na heurística apresentada neste trabalho é o definição do parâmetro *visibilidade* (η). Para a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$,

o algoritmo proposto define a visibilidade como sendo um parâmetro η_{ij}^{c}. Neste caso, i é o nó representando o trabalho atual, j é o nó representando o próximo trabalho a ser sequenciado, e c representa a terceirização ou não de j. Se $c = 0$, j não é terceirizado e $c = 1$ representa um trabalho terceirizado. Se a formiga se move para o nó n_{ij}^{0}, o trabalho J_i é adicionado a S_π. Caso contrário, J_i é adicionado ao conjunto O_π. A definição de η_{ij}^{c} usada nestre trabalho é mostrada na equação 4.9. Nesta equação, $C_{max}^{S_\pi}$ é o tempo de finalização máximo das tarefas pertencentes à S_π, e $N_{tarPosteriores}$ é o número de tarefas que não pertencem nem a S_π nem a $O\pi$.

$$\eta_{ij}^{c} = \begin{cases} \frac{1}{\delta \cdot o_j + (1-\delta) \cdot \frac{l_j}{p_j}} & \text{se } c = 1 \\ \frac{1}{((1-\delta) \cdot (\frac{p_j}{l_j} \cdot (C_{max}^{S_\pi} + N_{tarPosteriores}))} & \text{caso contrário} \end{cases} \tag{4.9}$$

A equação 4.9, mostra que, se se desconsiderar a ação de feromônios o algoritmo buscará favorecer a terceirização de trabalhos com menor custo de terceirização o_j, menor *lead-time* de terceirização l_j e maior tempo de processamento p_j. Inversamente, os trabalhos terão uma chance maior de serem alocados em S_π quando a razão $\frac{p_j}{l_j}$ for maior. Adicionalmente, o termo $(C_{max}^{S_\pi} + N_{tarPosteriores})$ tem como objetivo penalizar a criação de uma sequência S_π com *makespan* muito grande.

4.4.2.3 O procedimento de busca local proposto

Especialmente para instâncias de larga escala do problema, nota-se a necessidade do uso de uma regra de busca local. Conforme mencionado anteriormente, o uso de estratégias de busca local é comum quando trabalhamos com algoritmos ACO. Como busca local, é proposto o algoritmo 4.3 como estratégia de busca local.

Esse procedimento consiste em: (i) mover um trabalho escolhido aleatoriamente de O_π para S_π, atualizando o valor do total de orçamento disponível e (ii) aplicar o procedimento recursivo do algoritmo 4.3, que verifica a existência de um conjunto de trabalhos que podem ser movidos de S_π para O_π de forma a melhorar a solução encontrada até o momento. Os parâmetros iniciais de entrada do procedimento recursivo de busca local são:

- $SeqModificada$: A sequência original com um trabalho aleatório J_f movido de O_π para S_π. O primeiro elemento dessa sequência possui índice 0, e o último n-1.;
- OD: O orçamento disponível, $OD = Budget - \sum_{i \in O_\pi, i \neq f} o_i$;
- $Ponteiro$: Um valor que indica o índice da tarefa que está sendo analisada na etapa atual

do procedimento recursivo. Em um primeiro instante, $Ponteiro = n - 1$.

Adicionalmente, os seguintes valores são utilizados como variáveis globais:

- *bestFitness*: O melhor valor da função objetivo encontrada pelo algoritmo ACO até o momento;
- *bestSequence*: A melhor sequência encontrada até o momento.

```
1  buscaLocal (SeqModificada, OD, Ponteiro)
2  início
3      se Ponteiro = −1 então
4          fit ← o valor da função objetivo de SeqModificada
5          se fit < bestFitness então
6              Atualize o valor de bestFitness e de bestSequence
7          fim
8      senão
9          seqAtual ← SeqModificada
10         se Ponteiro ≠ f e seqAtual[Ponteiro] = 0 então
11             o_Ponteiro ← o custo de terceirização do trabalho J_Ponteiro
12             se OD ≥ o_Ponteiro então
13                 seqAtual[Ponteiro] ← 1
14                 buscaLocal(seqAtual, OD − o_Ponteiro, Ponteiro − 1)
15                 seqAtual[Ponteiro] ← 0
16             fim
17         fim
18         buscaLocal(seqAtual, OD, Ponteiro − 1)
19     fim
20 fim
```

Algoritmo 4.3: O algoritmo de busca local

4.4.2.4 A parametrização do algoritmo

Conforme já apresentado anteriormente, algoritmos ACO possuem um conjunto de parâmetros a serem estabelecidos antes de sua execução. A seleção correta dos parâmetros é diretamente relacionado com a qualidade dos resultados obtidos pelo algoritmo. Nesse trabalho, a parametrização foi realizada aplicando o seguinte procedimento em uma das instâncias de teste, selecionada de forma aleatória:

- O número de formigas foi estabelecido em 10, seguindo o relatado em muitos trabalhos da literatura (por exemplo, Lin *et al.* (2008) e Hu *et al.* (2005)).

- Os valores dos níveis de feromônio máximo e mínimo foram estabelecidos entre 10 e 20, obtidos em trabalhos anteriores (por exemplo, Tavares Neto (2005)). Diferentes valores foram testados (5 e 7,5 para o valor mínimo de feromônio e 22,5 e 25 para o valor máximo), mas $[10,20]$ foram mantidos.

- O valor inicial de todos os feromônios foi selecionado como sendo o valor do limite superior de feromônios. Este valor foi escolhido ao observar os resultados existentes em Stutzle e Hoos (2000), onde essa seleção de parâmetros resultou em uma melhor qualidade de solução.

- Para determinar a taxa de evaporação de feromônio, os valores $[0,999;0,99;0,95;0,91;0,5]$ foram analisados, seguindo valores observados na literatura (por exemplo, Thiruvady *et al.* (2009) usa $\rho = 0,99$).

- O total de feromônio a ser depositado na trilha é determinado através da equação $\tau_{max} \cdot \rho$. Isso garante que a o valor da melhor trilha terá o valor de τ_{max}.

- O valor de β foi selecionado seguindo o seguinte procedimento: analisando a equação 2.1, pode-se perceber que quando $\beta = 0$, a informação heurística (η) é descartada. Quando β é muito alto (maior que um limite β_{alto}), a informação heurística se torna dominante sobre a informação do feromônio, que é descartada. Assim, para que tanto a informação sejam utilizadas, o valor de β deve estar entre 0 e β_{alto}. Além disso, experimentos empíricos apresentados pela literatura (por exemplo, Heinonen e Pettersson (2007)), indicam que existe um único valor ótimo de β. Seguindo essas informações, selecionou-se o valor inicial de β como 1. Esse valor foi aumentado gradativamente até que a qualidade da resposta piore. Quando apenas se obteve valores piores de β, esse parâmetro foi diminuído de 0,1 até que o melhor valor fosse encontrado.

Na próxima seção, serão analisados os resultados computacionais do algoritmo proposto.

4.5 Resultados computacionais

4.5.1 Um exemplo de resolução do problema proposto

Um exemplo do problema analisado nesta seção, com 10 tarefas é mostrado a seguir. Na tabela 4.1, as tarefas pertencentes ao problema são mostradas. Para cada linha da tabela, são mostradas 4 colunas, a saber:

Tarefa: indica o nome da tarefa;

Tempo de Processamento: indica o tempo de processamento da tarefa

Custo: Indica o custo de terceirização da tarefa;

***Lead-Time*:** Indica o *lead-time* de terceirização da tarefa.

Considera-se, neste caso, que as colunas *Tempo de Processamento* e *Lead-Time*, da mesma forma como o problema apresentado por Lee e Sung (2008b), utilizam-se da mesma escala de tempo.

Tarefa	Tempo de Processamento	Custo	*Lead-Time*
J0	9	3	50
J1	5	12	9
J2	5	26	62
J3	7	15	39
J4	3	20	15
J5	8	4	5
J6	7	19	1
J7	5	1	66
J8	2	27	33
J9	4	25	92

Tabela 4.1: Tarefas para o exemplo de problema de sequenciamento em ambiente de máquina única

Adicionalmente, dois parâmetros do problema devem ser definidos:

- $\delta = 0,42$;
- O orçamento total disponível ($Budget$) é de 94.

Uma das possíveis respostas para esse problema (não necessariamente a ótima), define os conjuntos S_π e O_π da seguinte forma:

- O conjunto S_π é definido como o conjunto ordenado de tarefas $S_\pi = \{J0, J1, J2, J3, J4, J5, J6, J7\}$;
- O conjunto O_π é definido como o conjunto de tarefas $O_\pi = \{J8, J9\}$.

Com isso, é possível determinar o valor da função objetiva do problema:

- O conjunto ordenado S_π define o sequenciamento de tarefas mostrado na figura 4.2. Os valores de C_i são definidos na tabela 4.2.
- O conjunto não ordenado O_π define que as tarefas $J8$ e $J9$ serão terceirizada. Com isso, temos o custo total de 52 e o *lead-time* máximo de terceirização de 92.

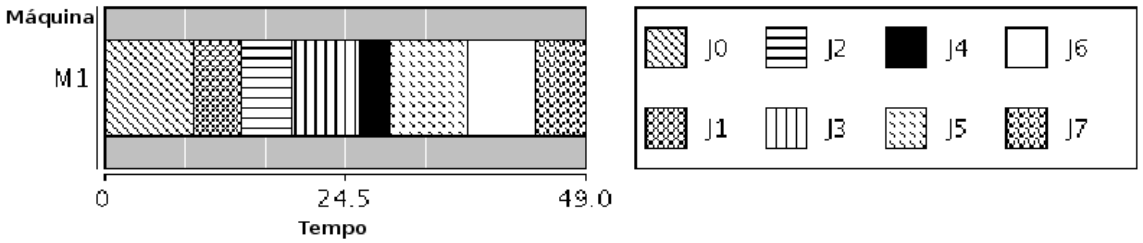


Figura 4.2: Gráfico de Gantt representando a sequência obtida com o sequenciamento proposto

Tarefa	Tempo de Processamento	Cj
J0	9	9
J1	5	14
J2	5	19
J3	7	26
J4	3	29
J5	8	37
J6	7	44
J7	5	49
	$\sum C_j$	227

Tabela 4.2: Cálculo dos valores de (C_j) para as tarefas do conjunto proposto

Desta forma, substituindo os valores na função objetivo $(1-\delta)\sum C_j + \delta \cdot OC$, tem-se o valor da função objetivo para este problema como sendo $226,0$.

4.5.2 Algumas considerações sobre os parâmetros do problema

A seguir, são analisados os efeitos da variação dos parâmetros δ e *Budget* na resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$.

O valor do orçamento disponível *Budget* limita quais trabalhos podem pertencer ao conjunto O_π. Se $Budget = 0$, o conjunto O_π não poderá possuir nenhum trabalho, e o problema se torna apenas como alocar as tarefas em S_π de forma a minimizar $\sum C_i$. Seguindo o que foi mostrado anteriormente, a solução desse problema é idêntica à solução dos problemas $1//\sum C_i$ e $1//\overline{F}$, bastando ordenar as tarefas seguindo a regra SPT.

Por outro lado, quando $Budget = \infty$, nada impede a alocação de qualquer trabalho em O_π. Com isso, a decisão da alocação ou não de tarefas em S_π e O_π ocorre apenas devido à função objetiva $(1-\delta)\sum C_j + \delta \cdot OC$.

O outro parâmetro do problema é o parâmetro de custo δ. Com $\delta = 1$, a função objetiva do problema se torna $z = \delta \cdot OC$. Dessa forma, a melhor solução será alcançada quando todos os trabalhos estiverem no conjunto S_π, em uma ordem qualquer (como o termo $\sum C_i$ é eliminado, a ordem dos trabalhos de S_π é irrelevante para a função objetiva). Nesse caso, nenhuma tarefa é terceirizada.

Quando $\delta = 0$, por outro lado, o custo de terceirização é eliminado, fazendo $z = \sum C_i$, sendo C_i o valor descrito na equação 4.1.

4.5.3 Implementação do algoritmo e testes em maior escala

Buscando a resolução do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ através dos métodos descritos anteriormente, gerou-se um conjunto de problemas-teste, seguindo o procedimento descrito por Lee e Sung (2008b): para cada número de tarefas (10, 20, 30, 40, 50, 60 e 70), 20 instâncias diferentes foram geradas usando os seguintes parâmetros: os tempos de processamento foram amostrados a partir de um intervalo [1, 10], os custos de terceirização do intervalo [1, 40], *lead times* de terceirização do intervalo [1,30].

Posteriormente, o modelo de programação matemática apresentado na seção 4.2.1 foi implementado usando a linguagem AMPL e o *solver* CPLEX versão 11. Foi estabelecido o tempo máximo de execução de cada problema como de 90 minutos. A tabela 4.3 mostra os tempos computacionais e o número de resultados ótimos obtidos pelo CPLEX usando o limitante de tempo estabelecido.

Analisando a tabela 4.3, percebe-se que foi possível obter o valor ótimo para todas as

Tamanho	Tempo Médio (s)	Desvio Padrão	% de Soluções Ótimas Encontradas
10	0,01s	0,01	100,00%
20	0,17s	0,09	100,00%
30	9,89	0,73	100,00%
40	1min54s	185,54	100,00%
50	20min41s	1606,81	70,00%
60	34min37s	1607,78	50,00%
70	52min3s	1058,21	20,00%

Tabela 4.3: Análises de tempos computacionais requeridos para a execução do modelo de programação matemática e número de soluções ótimas encontradas

instâncias com $n \leq 40$. Para instâncias maiores, o tempo computacional necessário para o CPLEX obter a solução ótima superou os 90 minutos estipulados.

Outro ponto importante diz respeito ao desvio padrão obtido, que indica a alta variação do tempo computacional necessário para a obtenção do ótimo. Isso reforça a análise realizada na seção 4.5.2: existe uma relação entre o conjunto de soluções viáveis e os parâmetros específicos de cada problema. Assim, alguns problemas foram mais simples de serem resolvidos (e obtiveram um tempo computacional menor), enquanto outros demandaram um número maior de iterações.

Percebeu-se também que, para os problemas que conseguiram ser resolvidos pelo CPLEX, o resultado ótimo foi o mesmo que o resultado obtido pelo método *branch and bound*, o que nos permitiu validar nossa implementação.

O algoritmo *branch and bound* foi implementado usando a linguagem JAVA e executado em um Pentium D Core Duo 2.53 GHz com 4GB de memória. O mesmo foi utilizado para a resolução do problema, sendo que o tempo máximo de execução encontrado foi de aproximadamente 1 hora.

Por fim, o algoritmo ACO proposto na seção 4.4 foi implementado e executado. Para cada problema, o valor ótimo ($Fitness_{otimo}$) foi encontrado usando o procedimento de *Branch and Bound* [1]. A melhor solução viável é armazenada para posterior comparação com os valores encontrados pelo algoritmo proposto. Durante a experimentação, os parâmetros foram selecionados como todos os parâmetros do ACO foram estabelecidos como constantes, com exceção do parâmetro $\beta = \{0, 2.5, 5\}$, que possibilita a análise do impacto da visibilidade (η) na qualidade de solução. Quando $\beta = 0$, a visibilidade não é considerada no processo de geração

[1] Como esse algoritmo foi desenvolvido especialmente para este problema, buscando-se minimizar o uso de procedimentos computacionais conhecidamente custosos, como acesso a disco e redimensionamento de matrizes, com ele foi possível obter o valor ótimo dos problemas analisados em um tempo menor que o tempo requerido usando o procedimento de programação matemática.

de soluções. Os demais parâmetros são selecionados como:

- Níveis de feromônios: entre 15 e 20, iniciando em 20;
- $\rho = 0.99$
- $1/Q = 0.2$
- $m = 5$
- 10 iterações
- $q_0 = 1$

Estes valores foram obtidos através de experimentos iniciais conforme explicado na seção 4.4.2.4. Para analisar o impacto da estratégia de busca local, todos os experimentos foram realizados duas vezes: a primeira não utiliza a busa local, e a segunda sim.

Cada problema foi executado 5 vezes, e os melhores valores encontrados foram armazenados como $Fitness_{encontrado}$. Para se analisar a qualidade da solução, definiu-se um parâmetro $\%gap$, $\%gap = (Fitness_{encontrado} - Fitness_{otimo})/Fitness_{otimo})$. Os resultados foram analisados usando-se quatro critérios diferentes, objetivando a comparação com Lee e Sung (2008b):

- Na tabela 4.4 é mostrada o *gap* médio para cada conjunto de instâncias. Nesta tabela também é mostrado uma comparação entre os resultados deste artigo e os resultados apresentados pelo melhor valor do *gap* médio encontrado por Lee e Sung (2008b)[2]. A figura 4.3, mostra uma comparação entre os melhores valores encontrados neste artigo e os melhores valores de Lee e Sung (2008b).
- Na tabela 4.5 mostra-se o *gap* máximo encontrado em cada conjunto de 5 experimentos de cada conjunto de instancias. Nesta tabela também é mostrada uma comparação entre nossos resultados o os resultados apresentados por Lee e Sung (2008b)
- Na tabela 4.6 é mostrado o número de instâncias que o algoritmo conseguiu alcançar o valor $Fitness_{otimo}$.
- Na tabela 4.7 é mostrado o número de instâncias onde $Fitness_{encontrado} \leq 1.1 \cdot Fitness_{otimo}$

[2]Infelizmente, mesmo após vários contatos com os autores, não foi possível a obtenção das instâncias originais ou do código usado para a obtenção dos valores originais. Porém, a geração dos problemas utilizados foi realizada conforme descrito em Lee e Sung (2008b).

- Na tabela 4.9 é mostrada o tempo computacional médio necessário para se executar o algoritmo. Os tempos computacionais do algoritmo que não utiliza de busca local são muito similares e independem do valor usado de β, portanto esses dados foram colocados em uma só coluna para facilitar a visualização dos dados.

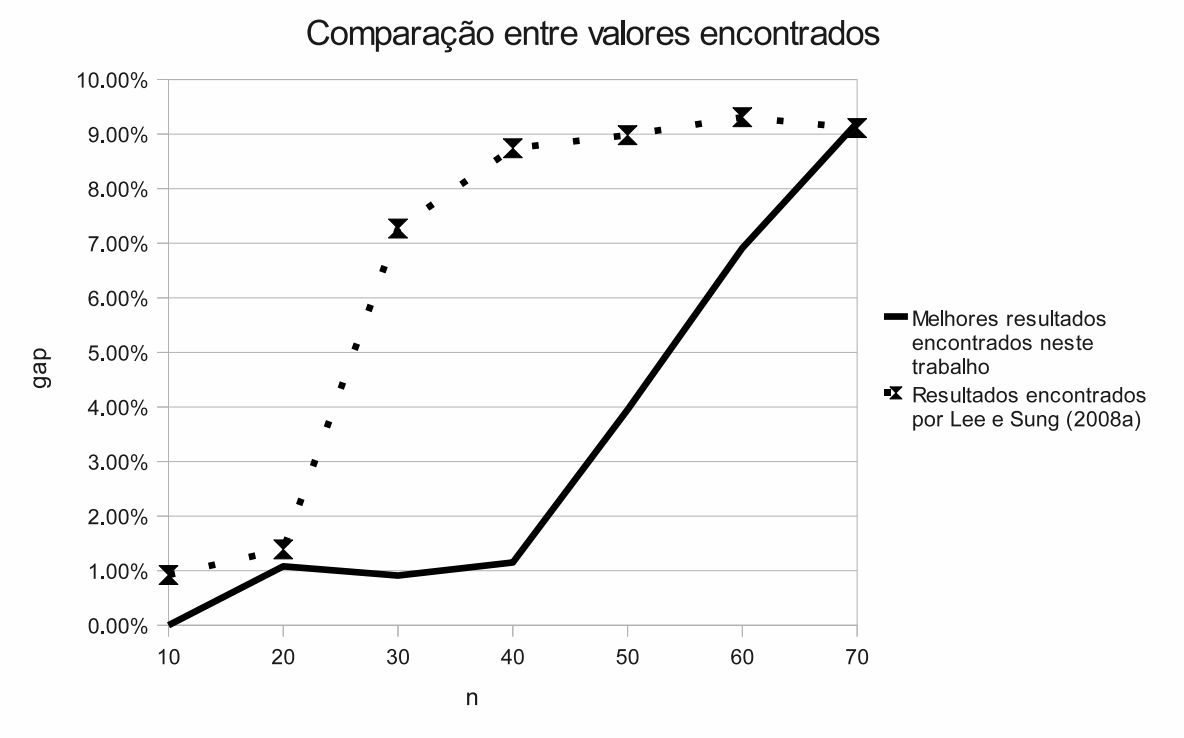


Figura 4.3: Comparação entre o melhor *gap* médio encontrado pelo método proposto e por Lee e Sung (2008b)

Analisando os dados apresentados nas tabelas 4.4 e 4.5 e na figura 4.3, pode-se observar o seguinte:

1. Para instâncias pequenas (n=$\{10,\ 20\}$), o algoritmo sem busca local e com valor de visibilidade $\beta = \{2.5;5\}$ obtiveram os melhores valores. Aqui, a busca local influencia o comportamento do algoritmo e converge em um ótimo local, explicando o alto valor do *gap* médio do algoritmo que se utiliza deste procedimento.

2. Para instâncias médias e grandes (n=$\{30\text{-}70\}$), a busca local contribui para que se encontre a melhor solução. Para tais instâncias, os resultados obtidos usando a busca local são melhores que os resultados obtidos sem o uso da busca local, não importando o valor de β.

3. Ao se analisar o *gap* máximo encontrado, o algoritmo sem busca local gera resultados melhores que Lee e Sung (2008b) para problemas de tamanho $n = \{10,20\}$. Para

valores maiores de n, o algoritmo com busca local mostrou resultados melhores que os encontrados por Lee e Sung (2008b).

4. Os resultados médios obtidos foram superiores aos citados por Lee e Sung (2008b) nas seguintes situações:
   - $n = \{10; 20; 30; 40\}$, sem busca local e $\beta = 2.5$
   - $n = \{10; 30\}$, sem busca local e $\beta = 5.0$
   - $n = \{30; 40; 50; 60\}$, com busca local e $\beta = 2.5$
   - $n = \{30; 40; 50; 60\}$, com busca local e $\beta = 5.0$
5. Para $n = 70$, os resultados encontrados pelo algoritmo com busca local e por Lee e Sung (2008b) são muito similares quando se observa o *gap* médio. Porém, o algoritmo proposto mostra valores melhores que Lee e Sung (2008b) quando se considera o *gap* máximo (tabela 4.5)
6. Por fim, ainda com relação ao *gap* máximo, todos os resultados encontrados foram melhores que os apresentados por Lee e Sung (2008b).

Porém, apenas as análises do *gap* médio e máximo parece ser insuficientes, pois não permite inferir a dispersão dos dados. Desta forma, visando analisar as aplicações práticas desta pesquisa, é apresentado nas tabelas 4.6 e 4.7 quantas execuções atingiram 100% do valor ótimo e 90%, respectivamente.

Quando se considera o número de resultados 100% precisos, percebe-se que dois conjuntos de dados definidos na análise da tabela 4.4 continuam válidos. Desta forma, para problemas com $n = \{10, 20\}$, o algoritmo sem busca local foi mais preciso que o algoritmo que se utiliza da busca local. Para instâncias maiores ($n = \{30 - 70\}$), o uso da busca local gerou melhores soluções.

Analisando os valores apresentados na tabela 4.7, encontram-se resultados similares: problemas menores são solucionados de forma mais eficiente com o algoritmo sem busca local. Porém, existe uma nova informação apresentada na tabela: Quando $n = 30$, ambos os algoritmos obtiveram resultados iguais.

Ainda como uma forma de realizar uma análise da dispersão dos dados, na tabela 4.8 mostra-se o desvio padrão dos resultados obtidos. Percebe-se que, para os experimentos realizados sem a busca local e com $\beta = 0$ (ou seja, contando apenas com a ação dos feromônios), o desvio padrão encontrado foi maior, com excessão dos resultados encontrados com $n = 50$ e

$n = 60$. Com o uso da busca local e $\beta = 0$, o valor do desvio padrão é maior ou igual aos demais experimentos, com excessão de quando $n = 20$. Nos dois grupos de experimentos - aqueles que se utilizavam de busca local e os que não se utilizavam desse mecanismo - grande parte dos resultados com menor dispersão de dados (ou seja, menor desvio padrão) são encontrados quando $\beta = 2,5$.

Desta forma, usando os dados apresentados nas tabelas 4.4, 4.5, 4.6 e 4.7, pode-se concluir que o procedimento utilizado de busca local é eficiente para problemas de médio e grande porte. Para problemas pequenos, a busca local não é necessária, podendo inclusive influenciar negativamente os resultados.

Finalmente, na tabela 4.9, encontra-se os tempos computacionais necessários para executar o algoritmo. Como esperado, o procedimento de busca local foi responsável pelo aumento do tempo necessário para a execução do algoritmo. Como para a maior instância analisada (n=$\{70\}$), o tempo computacional foi menor que 43 segundos, considera-se que este algoritmo produziu uma solução viável.

n	Valor médio da função objetivo	Resultados usando ACO Sem busca local			Resultados usando ACO Com busca local			Resultados Obtidos por Lee e Sung (2008b)
		$\beta = 0$	$\beta = 2.5$	$\beta = 5$	$\beta = 0$	$\beta = 2.5$	$\beta = 5$	
10	88,64	10.45%	0.00%	0.00%	2,97%	2,97%	2,97%	0.92%
20	281,54	13.96%	1.08%	2.15%	6,01%	5,89%	7,47%	1.39%
30	569,77	18.61%	2.24%	4.17%	4,95%	0,91%	1,40%	7.27%
40	765,19	28,87%	4,91%	8,81%	5,41%	1,15%	5,16%	8.74%
50	1230,14	22,81%	11,69%	15,13%	8,16%	3,95%	4,49%	8.98%
60	1822,67	28,32%	18,12%	19,84%	11,17%	6,91%	7,45%	9.31%
70	2615,65	37,70%	23,21%	28,49%	13,69%	9,21%	10,42%	9.11%

Tabela 4.4: *Gap* médio usando o método proposto e os resultados encontrados por Lee e Sung (2008b)

n	Resultados usando ACO Sem busca local			Resultados usando ACO Com busca local			Resultados Obtidos por Lee e Sung (2008b)
10	71.63%	0.00%	0.00%	27.29%	27.29%	27.29%	10.28%
20	32.59%	4.06%	5.39%	24.78%	28.69%	29.51%	13.35%
30	35.56%	6.68%	8.72%	21.73%	3.93%	3.93%	41.89%
40	47.87%	15.14%	19.58%	12.65%	6.02%	16.43%	43.55%
50	37.52%	27.55%	38.39%	16.09%	11.27%	15.97%	58.96%
60	58.35%	35.02%	41.16%	30.82%	16.75%	15.43%	63.73%
70	67.02%	59.81%	63.07%	26.16%	20.20%	18.66%	48.41%

Tabela 4.5: Gap máximo encontrado pelo método proposto e por Lee e Sung (2008b)

n	Resultados Usando ACO Sem busca local			Resultados Usando ACO Com busca local		
	$\beta = 0$	$\beta = 2.5$	$\beta = 5$	$\beta = 0$	$\beta = 2.5$	$\beta = 5$
10	5	20	20	14	14	14
20	1	7	1	2	4	2
30	0	1	1	1	6	4
40	0	0	0	2	6	2
50	0	0	0	0	2	2
60	0	0	0	0	2	2
70	0	0	0	0	0	0

Tabela 4.6: Número de resultados que alcançaram o valor ótimo

n	Resultados Usando ACO Sem busca local			Resultados Usando ACO Com busca local		
	$\beta = 0$	$\beta = 2.5$	$\beta = 5$	$\beta = 0$	$\beta = 2.5$	$\beta = 5$
10	13	20	20	18	18	18
20	6	20	20	14	15	15
30	2	20	20	18	20	20
40	1	17	12	17	20	17
50	0	10	9	12	18	18
60	0	7	3	9	14	14
70	0	4	1	7	11	8

Tabela 4.7: Número de resultados com variação de até 10% do valor ótimo

n	Resultados Usando ACO Sem busca local			Resultados Usando ACO Com busca local		
	$\beta = 0$	$\beta = 2.5$	$\beta = 5$	$\beta = 0$	$\beta = 2.5$	$\beta = 5$
10	0,1	0	0	0,04	0,04	0,04
20	0,07	0,01	0,01	0,06	0,06	0,08
30	0,05	0,01	0,02	0,04	0,01	0,01
40	0,07	0,04	0,06	0,04	0,01	0,04
50	0,07	0,07	0,09	0,04	0,03	0,03
60	0,09	0,09	0,1	0,05	0,05	0,04
70	0,1	0,1	0,09	0,06	0,04	0,04

Tabela 4.8: Desvio padrão dos resultados encontrados

Usando o Médodo ACO proposto					Usando Métodos Exatos	
n	Sem busca local	Com busca local			Programação Matemática	Branch and Bound
		Betha = 0	Betha = 2.5	Betha = 5		
10	0,08s	0,12s	0,15s	0,15s	0,009s	0,0007s
20	0,13s	0,40s	1,89s	2,35s	0,178s	0,023s
30	0,16s	1,04s	36,28s	1min 10,78s	9,887s	2,439s
40	0,16s	4,06s	1,07s	3,90s	1min54,041s	19,242s
50	0,17s	16,19s	4,28s	4,39s	20min41,786s	8min27,219s
60	0,23s	44,12s	12,43s	10,80s	34min37,769s	1h0min12,164s
70	0,29s	2min 29,1s	42,69s	42,69s	52min3,31s	18h50min6,708s

Tabela 4.9: Tempo computacional médio para a execução dos algoritmos analisados

4.6 Análises e Considerações Finais do Capítulo

Este capítulo tratou do problema de minimizar a soma ponderada dos tempos de finalização e de terceirização de um conjunto de tarefas em um ambiente de máquina única ($1/Budget/\delta \cdot OC + (1-\delta) \cdot \sum C_i$). Este problema, NP-difícil, proposto por Lee e Sung (2008b), foi resolvido usando por meio da proposta de um método baseado na heurística ACO. O método proposto incorpora cinco características específicas ao ACO, a saber: uma representação em forma de grafo para problemas de *scheduling* que envolvam terceirização; um algoritmo ACO composto de dois estágios, onde o primeiro estágio busca reduzir o espaço de buscas e o segundo estágio apresenta uma regra de pré-seleção, que garante a viabilidade da solução; uma nova regra de visibilidade, específica para o problema $1/Budget/\delta \cdot OC + (1-\delta) \cdot \sum C_i$; e uma estratégia de busca local. Na solução proposta, as tarefas são sequenciadas usando a regra SPT e, posteriormente, o ACO é usado para indicar quais delas devem ser terceirizadas ou não.

A primeira conclusão que foi obtida durante os experimentos realizados é que existe uma forte relação entre os parâmetros do problema e o número de soluções a serem analisadas, tanto pelo algoritmo *branch and bound* quanto no modelo de programação matemática.

Também dos dados gerados pelos experimentos, pode se verificar que as definições de visibilidade influenciaram positivamente o comportamento do algoritmo ACO implementado. Sem o uso da estratégia de busca local e com um ajuste adequando dos parâmetros do algoritmo ($\beta = \{2.5; 5.0\}$), os resultados encontrados foram melhores que aqueles encontrados por Lee e Sung (2008b) para problemas com 10 e 20 tarefas. Para problemas contendo mais tarefas, ($n = \{30; 40; 50; 60\}$), o algoritmo desenvolvido com a estratégia de busca local gerou melhores resultados. Para $n = 70$, o *gap* médio encontrado pelo algoritmo proposto com busca local e os resultados encontrados por Lee e Sung (2008b) são muito semelhantes. Porém, o *gap* máximo encontrado no algoritmo proposto mostra que, mesmo quando $n = \{70\}$, o algoritmo ACO é mais adequado para a resolução do problema do que o algoritmo proposto por Lee e Sung (2008b)

A estratégia de busca local proposta para o problema $1/Budget/\delta \cdot OC + (1-\delta) \cdot \sum C_i$ melhorou significativamente os resultados obtidos na resolução de problemas com 30, 40, 50 60 e 70 trabalhos. Percebeu-se que a busca local aumenta o tempo computacional necessário para a execução do algoritmo. Porém, mesmo quando trata-se de problemas com 70 trabalhos, o tempo computacional requerido é viável.

Outra análise importante diz respeito ao tempo computacional necessário para a execução dos métodos apresentados: percebe-se que algoritmos exatos, sejam eles os baseados

em modelos matemáticos os mesmo o método *branch-and-bound* implementado, o tempo computacional cresceu muito mais que o tempo do algoritmo ACO. Por exemplo, no caso dos problemas com 70 tarefas, temos que, no melhor caso, o tempo necessário para a obtenção de uma resposta através de métodos exatos foi mais que 730% maior que o tempo necessário para a execução do algoritmo ACO.

Adicionalmente, é importante que se perceba que os problemas analisados foram resolvidos com poucas iterações do ACO, mesmo no caso de problemas maiores. Isso pode ser explicado devido às reduções do espaço de busca já mencionadas.

Concluindo, percebe-se que os resultados obtidos das instâncias de teste nos indicaram que o ACO pode ser usado para a resolução de problemas de *scheduling* que envolvem terceirização. Os resultados obtidos dão possibilidades de pesquisas futuras, como o estudo das propriedades do problema $1/Budget/\delta \cdot OC + (1-\delta) \cdot \sum C_i$ e investigação de variações da função de visibilidade η.

No próximo capítulo, o problema $1/Budget/\delta \cdot OC + (1-\delta) \cdot \sum C_i$ é estendido no problema $F/prmu, Budget/(1-\delta) \cdot M + \delta \cdot OC$, que busca minimizar o somatório ponderado entre *makespan* e o custo total de terceirização em um ambiente *flowshop* permutacional com restrição de orçamento máximo disponível.

5 *Proposta de solução para o problema de scheduling em ambiente* **flowshop** *com terceirização permitida usando a técnica de otimização por colônia de formigas*

Este capítulo apresenta o problema de *scheduling* $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$. Este problema, inspirado no problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ proposto por Lee e Sung (2008b), não foi encontrado na literatura pesquisada. Para sua resolução, é proposto um algoritmo ACO composto de dois estágios. Este algoritmo foi implementado e executado, com seus resultados comparados por um procedimento heurístico criado nesta pesquisa especificamente para este problema.

Este capítulo está dividido da seguinte forma: na seção 5.1, o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ é definido; na seção 5.2, é apresentado um modelo de programação matemática para a resolução desse problema; na seção 5.3, é proposto um algoritmo ACO para o problema em questão; na seção 5.4 são apresentados os resultados obtidos pelo algoritmo ACO; por fim, na seção 5.5 são apresentadas análise e considerações finais deste capítulo.

5.1 Definição do problema

Nesta seção é descrita a formulação do prpblema e as notações usadas neste capítulo. Como mencionado anteriormente, um problema de *scheduling* busca alocar algumas tarefas em alguns recursos para que se consiga obter uma determinada medida de desempenho. Como apresentado por Brucker (2007), o sequenciamento de um trabalho é a alocação de intervalos de tempo em um conjunto de uma ou mais máquinas. Uma característica importante de um problema

de *scheduling* é o ambiente de manufatura. Nesta seção, o ambiente tratado é o *flowshop* permutacional com m máquinas. Todos os trabalhos devem executar uma única operação em cada máquina. A sequência de uso das m máquinas é a mesma para todos os trabalhos. Da mesma forma, a sequência de execução de cada trabalho em cada máquina é a mesma. Por fim, o problema abordado neste trabalho não permite preempção.

O problema abordado aqui pode ser definido como: seja J_i um elemento de um conjunto de n trabalhos que podem ser realizados em um ambiente *flowshop* ou terceirizados. Este trabalho é realizado em m máquina Cada trabalho J_i é caracterizado por:

- Um tempo de processamento p_{ik} para o trabalho i na máquina k;
- Um custo adicional existente quando se ocorre a terceirização da tarefa, o_i;
- Um *lead time* de terceirização l_i;

Para representar as duas decisões - sequenciar o trabalho no *flowshop* ou terceirizá-lo, dois conjuntos-resposta são definidos, de forma análoga ao apresentado no capítulo 4:

1. Um conjunto S_π, que contém a sequência de operações a serem realizadas no *flowshop* permutacional;
2. Um conjunto O_π, que contém os trabalhos a serem terceirizados.

Assim como o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, o custo de outsourcing é definido pela equação 4.2. O *makespan* M deste problema é definido na equação 5.1.

$$M = max(max(l_i | i \in O_\pi), M_{S_\pi}) \tag{5.1}$$

Onde:

- $max(l_i | i \in O_\pi)$ é o valor máximo do *lead time* de terceirização de todos os trabalhos pertencentes ao conjunto O_π;
- M_{S_π} é o *makespan* dos trabalhos pertencentes ao conjunto S_π.

Adicionalmente, existe uma condição de contorno relacionada ao capital disponível para a terceirização: o custo total de terceirização de todos os trabalhos de O_π deve ser menor ou igual ao orçamento total, representado pelo parâmetro $Budget$.

A função objetiva para este problema é mostrada na equação 5.2.

$$f = (1-\delta)\cdot M + \delta \cdot OC \tag{5.2}$$

Onde:

- $0 \leq \delta \leq 1$ é um parâmetro específico do problema. Seguindo a nomenclatura de Lee e Sung (2008b) para o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, definimos este como sendo o *parâmetro de custo*.

Para representar este problema, é adotado a formulação $\alpha/\beta/\gamma$ de Conway *et al.* (1967). Neste caso, tem-se:

α deve representar um ambiente *flowshop* com n trabalhos e m máquinas. Assume o valor $\alpha = F_{n,m}$. Esta sigla é simplificada para F;

β deve representar duas restrições: (i) Trata-se de *flowshop* permutacional; e (ii) Existe um limite dado pelo parâmetro $Budget$. Assim, $\beta = \{prmu, Budget\}$;

γ deve representar a função objetiva do problema. Desta forma, $\gamma = \{(1-\delta)\cdot M + \delta \cdot OC\}$.

Assim, este problema é apresentado pela notação $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$, podendo ser considerado como uma extensão do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, definido por Lee e Sung (2008b) como um problema NP-difícil.

Para resolver esse problema, nesse capítulo são propostas duas abordagens: na seção 5.2, é apresentada uma forma de solução usando um modelo de programação inteira mista; e, na seção 5.3, o problema é resolvido usando uma abordagem baseada na heurística ACO.

5.2 A abordagem do problema usando um modelo de programação matemática

Assim como o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ apresentado no capítulo 4, o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ também pode ser resolvido através de modelos de programação inteira mista. Para a definição do modelo a ser apresentado, definiu-se duas

variáveis, x_{ij} e y_i (equações 5.3 e 5.4 respectivamente), de forma análoga à seção 4.2:

$$x_{ij} = \begin{cases} 1 & \text{Se a tarefa } i \text{ é alocada na posição } j \\ 0 & \text{Caso contrário} \end{cases} \tag{5.3}$$

$$y_i = \begin{cases} 1 & \text{Se a tarefa } i \text{ é terceirizada} \\ 0 & \text{Caso contrário} \end{cases} \tag{5.4}$$

A função objetivo do problema é representada na equação 5.5:

$$minimizar\ z = (1-\delta) \cdot Makespan + \delta \sum_{i=1}^{n} (y_i \cdot o_i) \tag{5.5}$$

O conjunto de restrições é dado pelas equações 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12 e 5.13:

$$Makespan \geq l_i \cdot y_i \text{ para } i = 1..n \tag{5.6}$$

$$Makespan \geq C_{m,n} \tag{5.7}$$

$$C_{1,1} = \sum_{i=1}^{n} (p_{i,1} \cdot x_{i,1}) \tag{5.8}$$

$$C_{1,j} = C_{1,j-1} + \sum_{i=1}^{n} (p_{i,1} \cdot x_{i,j}) \text{ para } j = 2..n \tag{5.9}$$

$$C_{k,1} = C_{k-1,1} + \sum_{i=1}^{n} (p_{i,k} \cdot x_{i,1}) \text{ para } k = 2..m \tag{5.10}$$

$$C_{k,j} \geq C_{k-1,j} + \sum_{i=1}^{n} (p_{i,k} \cdot x_{i,j}) \text{ para } j = 2..n \text{ e } k = 2..m \tag{5.11}$$

$$C_{k,j} \geq C_{k,j-1} + \sum_{i=1}^{n} (p_{i,k} \cdot x_{i,j}) \text{ para } j = 2..n \text{ e } k = 2..m \tag{5.12}$$

$$\sum_{i=1}^{n} y_i \cdot o_i \leq Budget \tag{5.13}$$

Onde:

- A equação 5.5 define a nova função objetiva.
- As equações 5.6 e 5.7 definem os limites mínimos para o cálculo do *makespan*. No caso, a equação 5.6 define o limite inferior devido ao *lead time* de terceirização, e a equação 5.7 define o limite inferior defido ao *Makespan* das tarefas alocadas na sequência S_π.
- A equação 5.8 define o valor de $C_{1,1}$.
- A equação 5.9 aloca as operações das tarefas da sequência S_π primeira máquina.
- A equação 5.10 aloca as operações da primeira tarefa da sequência S_π em todas as máquinas.
- A equação 5.11 garante que nenhuma tarefa se inicialize em uma máquina $k \geq 2$ antes de finalizar sua operação na máquina anterior $(k-1)$.
- A equação 5.12 garante que nenhuma tarefa $j \geq 2$ se inicialize em uma máquina antes do fim do processamento da tarefa anterior $(j-1)$.
- A restrição mostrada na equação 5.13 garante que o limite *Budget* não será ultrapassado.

Esse modelo foi implementado e usado para a resolução de um conjunto de problemas de teste, conforme será mostrado na seção 5.4.

5.3 Resolução usando o algoritmo ACO proposto

Esta seção apresenta o algoritmo *Flowshop Ant Colony Optimization* (FSACO), uma meta-heurística baseada no ACO proposta nessa tese para a resolução do problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$. Com seu funcionamento baseado no ambiente de manufatura *flowshop*, esta nova abordagem é composta dois módulos ACO conectados seqüêncialmente cujo funcionamento sustitui o *loop* interno do algoritmo ACO já apresentado (algoritmo 2.1, linha 4). Nesta caso, cada algoritmo ACO resolve uma parte diferente do problema. A figura 5.1 ilustra a estrutura do algoritmo FSACO. O primeiro bloco (denominado ACO-1 na figura 5.1), aloca os trabalhos do problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ em dois conjuntos distintos: um conjunto O_π contendo os trabalhos a serem terceirizados e um conjunto S_π com os trabalhos a serem sequenciados. O segundo bloco analisa os trabalhos contidos em S_π e os ordena buscando minimizar o *makespan*, o que cria um conjunto ordenado

S^*_π. A solução construída pelos dois blocos ACO é dada pelos conjuntos O_π e S^*_π. Desta forma, para construir uma solução, são realizados os seguintes passos:

1. A colônia ACO-1 cria dois conjuntos não ordenados: O_π e S_π. O conjunto O_π será parte da solução final. O conjunto S_π é uma das entradas para a segunda colônia, ACO-2.
2. A colônia ACO-2 ordena o conjunto S_π em um nov conjunto S^*_π. S^*_π contém todos os trabalhos de S_π ordenados de forma a se minimizar o *makespan*.
3. A solução final é composta do conjunto não-ordenado O_π e do conjunto ordenado S^*_π.

Um fator chave nesta abordagem é que ambas as colônias são trechos de código independentes. Em termos computacionais, a primeira colônia usa a colônia ACO-2 como um otimizador "em caixa-preta", ou seja, não tem controle de suas variáveis internas, apenas das entradas e saídas. Desta forma, a única informação trocada entre as colônias ACO-1 e ACO-2 são os conjuntos S_π e S^*_π. Demais estruturas de dados, como níveis de feromônios e atribuições de parâmetros são geradas especificamente para cada colônia.

O pseudocódigo completo para o algoritmo FSACO é mostrado no algoritmo 5.1.

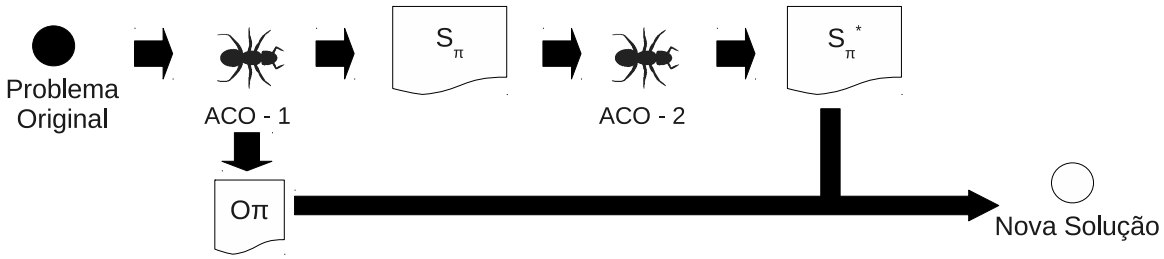


Figura 5.1: O fluxograma de geração de soluções para o FSCAO (isso corresponde à linha 5 do algoritmo 2.1)

As próximas seções irão discutir os aspectos-chave da implementação de ambos os estágios do algoritmo FSACO.

5.3.1 O primeiro estágio

Nesta seção, são apresentadas as escolhas de implementação realizadas para o primeiro estágio do algoritmo (bloco ACO-1 da figura 5.1). Este algoritmo é uma heurística MMAS implementada com algumas modificações (i) na estrutura do grafo, (ii) na regra de transição e (iii) na função de visibilidade.

```
1  Inicialize
2  repita Neste nível, cada execução é chamada iteração
3  |  Cada formiga é posicionada no nó inicial
   |  /* Início do ACO-1                                              */
4  |  repita Neste nível, cada execução é chamada passo
5  |  |  Cada formiga aplica uam regra de transição para determinar se um trabalho é
   |  |  inserido em O_π ou S_π
6  |  até todas as formigas tenham construído uma solução completa
   |  /* Fim do ACO-1                                                 */
7  |  Envie o conjunto S_π encontrado para a colônia ACO-2, e obtenha um conjunto
   |  ordenado S*_π que minimize o makespan
8  |  Aplique o procedimento de busca local (opcional)
9  |  Aplique a regra de atualização global de feromônios no grafo da colônia ACO-1
10 até o critério de parada seja satisfeito
```

Algoritmo 5.1: O pseudocódigo do FSACO

5.3.1.1 A estrutura do grafo para o primeiro estágio

Para abordar o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$, o primeiro passo é representá-lo em uma estrutura de grafo. Para tal, cada trabalho J_i produz dois nós não conectados: o nó N_i^0 que representa o trabalho J_i sendo atribuído para o conjunto S_π e o nó N_i^1 que representa o trabalho J_i sendo atribuído para o conjunto O_π. Neste grafo, N_i^0 e N_i^1 são conectados apenas aos nós N_{i-1}^0, N_{i-1}^1, N_{i+1}^0 e N_{i+1}^1. A figura 5.2 mostra um exemplo de grafo gerado usando este procedimento.

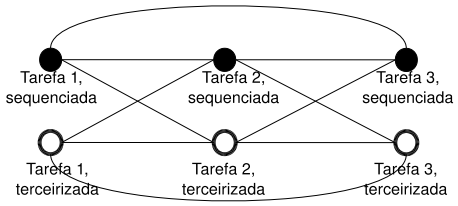


Figura 5.2: Um exemplo de grafo usado pelo primeiro estágio do algoritmo FSACO

5.3.1.2 A regra de transição e a função de visibilidade usada no primeiro estágio

Para se desenvolver a regra de transição do primeiro estágio do algoritmo FSACO, foi necessário incorporar a restrição *Budget*, evitando assim a geração de soluções inviáveis. Para tal, desenvolveu-se as equações 5.14, 5.15, 5.16 e 5.17. As equações 5.14 e 5.15 substituem a

equação 2.1 do algoritmo original.

$$P_{J_i}^0 = \begin{cases} 1 & \text{se } AvaliableBudget < o_i \\ \tau_i \cdot (\eta_i^0)^\beta & \text{caso contrário} \end{cases} \tag{5.14}$$

$$P_{J_i}^1 = \begin{cases} 0 & \text{se } AvaliableBudget < o_i \\ \tau_i \cdot (\eta_i^1)^\beta & \text{caso contrário} \end{cases} \tag{5.15}$$

Neste caso, os valores de η_i^0 e η_i^1 são dados pelas equações 5.16 e 5.17:

$$\eta_i^0 = \frac{p_i^0}{p_i^0 + o_i} \tag{5.16}$$

$$\eta_i^1 = \frac{o_i}{p_i^0 + o_i} \tag{5.17}$$

Onde:

- $P_{J_i}^0$ e $P_{J_i}^1$ são pesos a serem dados para a função probabilística que escolhe entre os nós N_i^0 e N_i^1 (equivale à equação 2.1)
- *AvaliableBudget* é o valor de recursos disponíveis para terceirização, conforme a restrição *Budget* e os trabalhos pertencentes à O_π. É calculado através da equação 5.18
- p_i^0 é o tempo de processamento da primeira operação do trabalho J_i

$$AvaliableBudget = Budget - \sum_{J_i \in O_\pi} o_i \tag{5.18}$$

Na próxima seção, serão mostradas as escolhas de implementação realizadas para o segundo estágio do algoritmo.

5.3.2 O segundo estágio

Nesta seção, são apresentadas as escolhas de implementação para o segundo estágio do algoritmo (representado com o"ACO-2" na figura 5.1). Este algoritmo também é uma heurística MMAS, com algumas modificações na geração do grafo, na função visibilidade e na busca local.

5.3.2.1 A estrutura de grafo e a função de visibilidade para o segundo estágio

No segundo estágio foi implementado uma heurística MMAS. Neste estágio, cada trabalho J_i contido no conjunto S_π é representado por um único nó totalmente interconectado N_i de um grafo, resultando em um grafo similar ao apresentado na figura 4.1. As trilhas que conectam qualquer nó ao nó N_i possui uma regra de visibilidade associada. Esta regra é mostrada na equação 5.19.

$$\frac{1}{\sum_{m=0}^{M} p_{im}} \tag{5.19}$$

5.3.2.2 A busca local do segundo estágio

Baseado no trabalho de Ahmadizar *et al.* (2007), o procediemnto de busca local usado neste estágio é mostrado no algoritmo 5.2. Este algoritmo seleciona aleatoriamente trabalhos e os reposiciona de forma aleatória, mantendo as posições relativas dos trabalhos resultantes. Se o *fitness* do novo conjunto for melhor, este é mantido.

```
1  para cada J_i ∈ S_π faça
2  |   para cada posição j de S_π faça
3  |   |   r é um número aleatório entre 0 e 1
4  |   |   se r < 0.2 então
5  |   |   |   se J_i não está na posição j então
6  |   |   |   |   Remova J_i de sua posição original e o insira na posição j
7  |   |   |   fim
8  |   |   fim
9  |   fim
10 fim
11 se o fitness da nova sequência for melhor que o fitness da sequência anterior então
12 |   substitua a sequência anterior pela nova sequência
13 fim
```

Algoritmo 5.2: O procediemnto de busca local pelo o segundo estágio

As demais características do algoritmo MMAS são mantidas.

5.4 Resultados computacionais

5.4.1 Um exemplo de resolução do problema proposto

Um exemplo do problema analisado nesta seção, contendo 10 tarefas e 4 máquinas é mostrado a seguir. Na tabela 5.1, as tarefas pertencentes ao problema são mostradas. Para cada linha da tabela, são mostradas 7 colunas, a saber:

Tarefa: indica o nome da tarefa;

P0/P1/P2/P3: indicam o tempo de processamento da tarefa nas máquinas 0 a 3;

O: Indica o custo de terceirização da tarefa;

L: Indica o *lead-time* de terceirização da tarefa.

Tarefa	P0	P1	P2	P3	O	L
J0	16	15	89	91	309	276
J1	40	64	68	21	247	336
J2	92	28	45	54	192	147
J3	95	14	83	46	76	380
J4	49	50	29	5	379	351
J5	9	11	4	43	270	11
J6	61	93	58	95	176	370
J7	36	28	40	4	377	127
J8	43	48	73	25	323	210
J9	85	29	32	73	39	50

Tabela 5.1: Tarefas para o exemplo de problema de sequenciamento em ambiente flowshop

Adicionalmente, dois parâmetros do problema devem ser definidos:

- $\delta = 0,57$;
- O orçamento total disponível (*Budget*) é de 1351.

Uma resposta possível pode ser apresentada pela definição de dois conjuntos distintos:

- O conjunto S_π é definido como o conjunto ordenado de tarefas $S_\pi = \{J0, J8, J5, J7, J6, J2, J3, J1, J4\}$;

- O conjunto O_π é definido como o conjunto de tarefas $O_\pi = \{J9\}$.

Com isso, é possível determinar o valor da função objetiva do problema:

- O conjunto ordenado S_π define o sequenciamento de tarefas mostrado na figura 5.3. Com isso, tem-se o valor de *makespan* de 558.
- O conjunto não ordenado O_π define que apenas a tarefa $J9$ será terceirizada. Com isso, tem-se o custo total de 39 e o *lead-time* máximo de terceirização de 50.

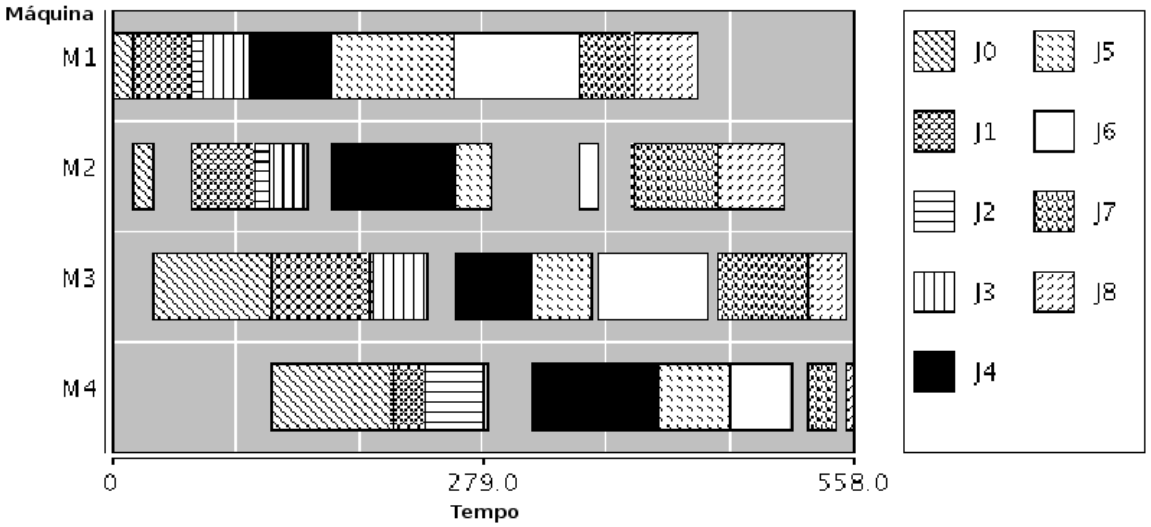


Figura 5.3: Gráfico de Gantt representando a sequência obtida com o método exaustivo

Desta forma, aplicando as equações 5.1 e 5.2, tem-se o valor total da função objetiva como sendo $258,78$.

5.4.2 Implementação dos algoritmos e testes em maior escala

O método proposto foi desenvolvido em JAVA e executado em um Pentium D 2.80GHz dual core com 2Gb de memória. O algoritmo ACO proposto foi desenvolvido usando o JVN 1.6 e testado em um ambiente Linux Ubuntu com ambiente gráfico Gnome. O modelo de programação inteira mista foi implementado no sofwtare CPLex versão 11 para Windows.

O conjunto de testes foi gerado através de um procediemnto semelhante ao usado por Ahmadizar *et al.* (2007). Os parâmetros específicos de terceirização foram gerados por

um procedimento inspirado no procedimento usado por Lee e Sung (2008b), adaptado para problemas de *scheduling* em ambiente *flowshop*. Para cada combinação de m número de máquinas $m \in \{2,3,4,5\}$ e n número de trabalhos $n \in \{5,10,15,20,25\}$, foram geradas 10 instâncias de problemas. Cada instância foi criada usando:

- O tempo de processamento P_{im} é um inteiro escolhido na distribuição uniforme $[1;99]$;
- O *lead time* de terceirização de cada trabalho l_i é um inteiro escolhido na distribuição uniforme $[m;99 \cdot m]$;
- O custo de *outsourcing* de cada trabalho c_i é um inteiro escolhido na distribuição uniforme $[m;99 \cdot m]$;
- O orçamento disponível $Budget$ é um inteiro escolhido na distribuição uniforme $[\frac{5 \cdot m}{3};495 \cdot m]$;
- O parâmetro de custo δ é um número real escolhido na distribuição uniforme $[0.3;0.7]$;

Para cada problema, o valor alvo ($Fitness_{ref}$) foi encontrado usando o método de programação matemática proposto na seção 5.2. Foi estabelecido um limite máximo de tempo de 90 miuntos.

Para grande parte dos problemas, o software CPlex foi capaz de obter o valor ótimo. A tabela 5.2 mostra quantos dos problemas de teste foram resolvidos pelo uso de programação matemática.

Posteriormente, o algoritmo FSACO proposto na seção 5.3 foi implementado e executado. Para o ACO-1, os seguintes parâmetros foram usados, determinados conforme a seção 4.4.2.4:

- Níveis de feromônio: entre 10 e 20, iniciando em 20;
- $\rho = 0.99$;
- $1/Q = 0.2$;
- 10 formigas;
- 20 iterações;
- $q_0 = 1$;
- $\beta = 50$;

Categoria de Problemas	% de Soluções Ótimas
5 x 2	100.00%
5 x 3	100.00%
5 x 4	100.00%
5 x 5	100.00%
10 x 2	100.00%
10 x 3	100.00%
10 x 4	100.00%
10 x 5	100.00%
15 x 2	100.00%
15 x 3	100.00%
15 x 4	100.00%
15 x 5	100.00%
20 x 2	100.00%
20 x 3	100.00%
20 x 4	100.00%
20 x 5	90.00%
25 x 2	90.00%
25 x 3	90.00%
25 x 4	80.00%
25 x 5	100.00%

Tabela 5.2: Porcentagem de problemas em que foi possível obter a solução ótima usando procedimentos de programação matemática

Para o ACO-2, os seguintes parâmetros foram usados:

- Níveis de feromônio: entre 10 e 20, iniciando em 20;
- $\rho = 0.9$;
- $1/Q = 0.2$;
- 10 formigas;
- 10 iterações;
- $q_0 = 1$;
- $\beta = 0.9$;

Estes parâmetros foram determinados usando o mesmo procedimento da seção 4.5: através de testes iniciais determinou-se um conjunto de valores que foram aplicados nos demais experimentos.

Para cada instância do problema, o algoritmo FSACO foi executado 10 vezes. Para os valores médios de *fitness* encontrados, o *gap* da solução definido como $\%gap = (Fitness_{ACO} - Fitness_{ref})/Fitness_{ref}$ foi calculado. Os tempos computacionais médios necessários para a resolução do problema usando o método de programação matemática e para a execução do algoritmo FSACO foram calculados. Os resultados obtidos são mostrados na tabela 5.3. Nesta tabela, são mostrados os *gaps* (média e desvio padrão) de cada conjunto de instâncias geradas de acordo com o procedimento mencionado anteriormente nessa seção. Também são mostrados os tempos computacionais requeridos para a execução do FSACO e do método de programação matemática.

Analisando os dados, percebe-se claramente que não existe uma relação entre o *gap* encontrado e o tamanho dos problemas. Da mesma forma, o desvio padrão não possui um acréscimo ou decréscimo linear de acordo com o tamanho dos problemas. Uma conclusão possível é que isso indica que a qualidade do algoritmo ACO implementado depende de características do problema, sendo claramente mais eficiente em alguns problemas da mesma família do que em outros.

Quando se analisa os tempos computacionais necessários para a resolução dos problemas de teste, percebe-se que os tempos computacionais do FSACO são menores que os tempos computacionais requeridos para a obtenção do ótimo quando se trata de instâncias com 15 tarefas e 2 máquinas ou de tamanho superior, com excessão do caso de instâncias de 20 tarefas e 2 máquinas.

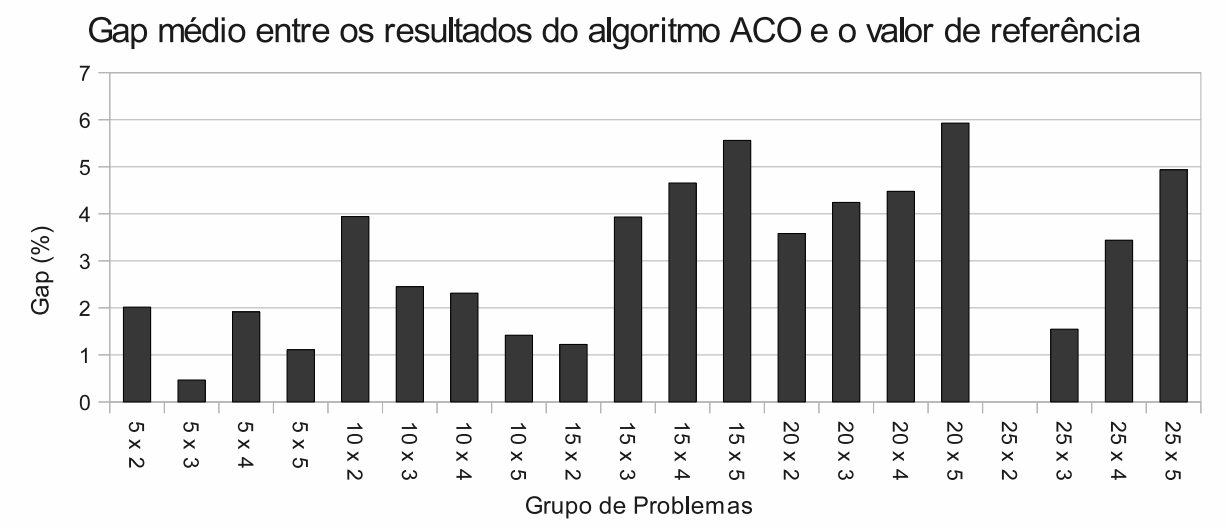


Figura 5.4: Comparação entre os *gaps* encontrados para todos os problemas analisados

Grupo de Problemas	Valor médio da Função Objetivo	*Gap* (médio)	*Gap* (desvio padrão)	Tempo req. usando Prog. Inteira Mista (s)	Tempo req. usando ACO
5 x 2	138,03	2,02%	5,16%	$<$ 2s	$<$ 2s
5 x 3	152,62	0,47%	1,40%	$<$ 2s	$<$ 2s
5 x 4	218,41	1,91%	2,96%	$<$ 2s	$<$ 2s
5 x 5	264,79	1,11%	2,97%	$<$ 2s	$<$ 2s
10 x 2	224,92	3,94%	9,36%	$<$ 2s	$<$ 2s
10 x 3	293,98	2,45%	2,42%	$<$ 2s	$<$ 2s
10 x 4	299,72	2,31%	2,44%	$<$ 2s	$<$ 2s
10 x 5	365,08	1,42%	1,21%	$<$ 2s	$<$ 2s
15 x 2	320,26	1,22%	2,54%	3,66s	$<$ 2s
15 x 3	403,57	3,93%	3,50%	4,5s	$<$ 2s
15 x 4	418,65	4,66%	2,53%	8,14s	2,05s
15 x 5	435,59	5,56%	4,12%	23,01s	2,12s
20 x 2	448,64	3,58%	5,93%	$<$ 2s	2,55s
20 x 3	504,81	4,24%	3,14%	35min20,67s	2,89s
20 x 4	474,94	4,48%	1,94%	5min22,91s	3,83s
20 x 5	623,30	5,93%	3,14%	43min5,15s	3,9s
25 x 2	546,83	0,00%	0,00%	5min16,69s	3,85s
25 x 3	585,75	1,55%	2,89%	8min47,28s	5,22s
25 x 4	752,44	3,44%	2,96%	19min6,76s	5,16s
25 x 5	749,79	4,94%	1,58%	3min8,27s	6,14s

Tabela 5.3: Resultados encontrados

5.5 Análises e Considerações Finais do Capítulo

Este capítulo tratou do problema de minimizar a soma ponderada do *makespan* e custos de terceirização em um *scheduling* de um ambiente *flowshop* ($F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$). Este problema, é baseado no problema NP-difícil $1/Budget/\delta \cdot OC + (1-\delta)\cdot \sum C_i$ proposto por Lee e Sung (2008b), e pode ser resolvido com o método ACO proposto. Este método, denominado *Flow Shop Ant Colony Optimization* (FSCAO), consiste de dois algoritmos ACO sequenciais, cada um especializado em resolver uma parte do problema.

O primeiro estágio é responsável por determinar se um trabalho deve ou não ser terceirizado, e usa uma função de visibilidade desenvolvida especificamente para este problema. Esta função de visibilidade, atuando em conjunto com o mecanismo de depósito-evaporação do MMAS, foi capaz de determinar bons valores para os conjuntos O_π e $S_{\pi,t=0}$. O segundo estágio, usa como visibilidade o inverso do tempo de processamento de cada trabalho e, com o procedimento de busca local implementado, também foi capaz de mostrar bons resultados.

Sobre o tempo computacional necessário para a execução dos métodos apresentados, percebe-se que a obtenção do ótimo através de algoritmos exatos necessitou de um tempo computacional muito maior que o tempo do algoritmo ACO. Por exemplo, no caso dos problemas com 20 tarefas e 5 máquinas, temos que o tempo necessário para a obtenção de uma resposta através de métodos exatos foi mais que 36.000% maior que o tempo necessário para a execução do algoritmo ACO. Percebe-se que, assim como o problema apresentado no capítulo 4, a performance do algoritmo exato depende muito do problema analisado, existindo casos onde ele se torna mais simples e casos onde o software CPlex teve mais dificuldade em resolver o problema. Porém, em todos os casos de instâncias de tamanho acima de 20 tarefas e 3 máquinas, o algoritmo ACO obteve uma resposta em tempo muito menor que o modelo de programação matemática. Um tópico a ser explorado em trabalhos futuros é o desenvolvimento de modelos matemáticos mais eficientes para a resolução desses modelos.

Também como futuros trabalhos, espera-se usar este algoritmo em instâncias maiores do problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$. Como o tempo computacional para o FSACO não se mostrou em nenhum momento proibitivo e os resultados obtidos forma bons, não se conseguiu identificar nenhum comportamento do algoritmo que indicasse a não possibilidade de sua aplicação em outras instâncias.

6 *Considerações Finais*

Este documento apresentou o trabalho de doutoramento intitulado "Resolução de problemas de sequenciamento de operações em máquinas considerando possibilidade de terceirização usando a técnica de otimização por colônia de formigas". Este trabalho, vem a ser o primeiro que se utiliza da meta-heurística ACO em problemas de *scheduling* com possibilidade de terceirização, e o primeiro que busca integrar terceirização com critérios de desempenho em ambientes *flowshop*. Para isso, dois problemas foram abordados inicialmente: $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, proposto por Lee e Sung (2008b) e o problema inédito $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$.

Durante esse trabalho, foi possível a obtenção de um algoritmo ACO que resolvesse o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, proposto por Lee e Sung (2008b). O desenvolvimento desse algoritmo gerou as seguintes características inéditas na literatura: (i) Uma representação em forma de grafo para problemas de terceirização; (ii) Um algoritmo ACO composto de dois estágios, em que o primeiro é uma heurística que reduz significativamente o espaço de buscas; (iii) uma regra de visibilidade específica para o problema; (iv) uma estratégia de busca local. Os resultados obtidos se mostraram melhores que os relatados por Lee e Sung (2008b).

Usando como base os conhecimentos adquiridos para o desenvolvimento do problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$, foi proposto o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$, como uma combinação do problema clássico $F/prmu/M$ e do conceito de terceirização em *scheduling* proposto por Lee e Sung (2008b). Para a resolução desse problema, foi utilizado um modelo de programação inteira mista e um algoritmo ACO, ambos inéditos e desenvolvidos especialmente para esse problema. O algoritmo ACO trouxe algumas inovações não existentes na literatura, a saber: (i) implementação de um algoritmo ACO composto de duas colônias de execução sequencial, sendo uma especializada em determinar quais tarefas devem ser sequenciadas ou terceirizadas e a outra especializada em reduzir o *makespan* das tarefas a serem sequenciadas; (ii) uma regra de pré-seleção de caminho para a primeira fase, que garante que todas as soluções encontradas serão factíveis; (iii) uma regra de visibilidade que permite a

decisão de se terceirizar ou não um trabalho; (iv) um procedimento de busca local específico para o problema.

Percebeu-se que os algoritmos propostos conseguiram obter resultados próximos ao ótimo em tempos computacionais muito menores do que os encontrados nos demais algoritmos utilizados como comparação. Em nossa avaliação, isso mostra que o ACO pode ser uma estratégia válida para a resolução de problemas de *scheduling* que levem em consideração indicadores de desempenho relacionados à terceirização de tarefas.

Devido a limitantes computacionais, não foi possível a geração de valores de *fitness* de referência para problemas do tipo $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ maiores do que 25 trabalhos e 5 máquinas. Assim, como futuros trabalhos, deseja-se aplicar o algoritmo ACO proposto para problemas de maior escala. Como o tempo computacional para o FSACO não se mostrou em nenhum momento proibitivo e os resultados obtidos forma bons, não se conseguiu identificar nenhum comportamento do algoritmo que indique que ele não possa ser aplicado para outras instâncias. Também como trabalhos futuros, entende-se que os métodos aqui descritos podem ser expandidos para problemas de *scheduling* que envolvam terceirização e que tratem de outros ambientes e outras funções objetivo, o que parece ser um campo promissor de pesquisas na área.

Também como pesquisa futura, entende-se que a parametrização dos algoritmos apresentados podem alvos de um estudo com maior rigor estatístico. Entende-se que o uso de técnicas como análise fatorial e planejamento de experimentos podem trazer mais informações sobre as relações entre os parâmetros e o comportamento dos algoritmos desenvolvidos nessa pesquisa.

Concluindo, entende-se que este trabalho contribuiu para a literatura de *scheduling* ao propor o problema $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$ e propor formas inéditas de resolução para o problema $1/Budget/(1-\delta)\sum C_j + \delta \cdot OC$ e $F/prmu, Budget/(1-\delta)\cdot M + \delta \cdot OC$. Adicionalmente, percebe-se que houve contribuições na literatura referente ao algoritmo ACO, devido à proposta de dois algoritmos ACO inéditos para a resolução dos problemas abordados. Mais que isso, percebe-se que, ao se obter um conjunto de algoritmos cuja resposta se aproxima de valores ótimos em tempos computacionais extremamente baixos (quando comparados com métodos exatos), entende-se que as contribuições deste trabalho têm grande interesse em aplicações práticas.

Referências

AHMADIZAR, F.; BARZINPOUR, F.; ARKAT, J. Solving permutation flow shop sequencing using ant colony optimization. In: *2007 IEEE International Conference on Industrial Engineering and Engineering Management*. [S.l.: s.n.], 2007. p. 753–757.

AL-ANZI, A. S.; ALLAHVERDI, A. Heuristics for a two-stage assembly flowshop with bicriteria of maximum lateness and makespan. *Computers & Operations Research*, v. 36, n. 9, p. 2682 – 2689, 2009.

ANGHINOLFI, D.; BOCCALATTE, A.; PAOLUCCI, M.; VECCHIOLA, C. Simulated evolution and learning. In: ____. [S.l.]: Springer Berlin / Heidelberg, 2008. cap. Performance Evaluation of an Adaptive Ant Colony Optimization Applied to Single Machine Scheduling, p. 411–420.

ANTELO, M.; BRU, L. Outsourcing or restructuring: The dynamic choice. *International Journal of Production Economics*, v. 123, p. 1–7, 2010.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. *Pesquisa Operacional*. [S.l.]: Elsevier, 2007.

ARNAOUT, J.; MUSA, R.; RABADI, G. Ant colony optimization algorithm to parallel machine scheduling problem with setups. In: *4th IEEE Conference on Automation Science and Engineering*. [S.l.: s.n.], 2008.

ARNAOUT, J.; RABADI, G.; MUSA, R. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 2009.

BAKER, K. *Introduction to Sequencing and Scheduling*. [S.l.]: John Wiley & Sons, 1943.

BAUER, a.; BULLNHEIMER, B.; HARTL, R. F.; STARUSS, C. *Applying Ant Colony Optimization to solve the Single Machine Total Tardiness Problem*. [S.l.], 1999.

BEHNAMIAN, J.; ZANDIEH, M.; GHOMI, S. M. T. F. Parallel-machine scheduling problems with sequence-dependent setup times using an aco, sa and vns hybrid algorithm. *Expert Syst. Appl.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 36, n. 6, p. 9637–9644, 2009. ISSN 0957-4174.

BERTRAND, J. W. M.; FRANSOO, J. C. Operations management research methodologies using quantitative modeling. *International Journal of Operations & Production Management*, v. 22, p. 241–264, 2002.

BESTEN, M. den; STUTZLE, T.; DORIGO, M. Ant colony optimization for the total weighted tardiness problem. In: *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 2000. p. 611–620. ISBN 3-540-41056-2.

BLUM, C.; DORIGO, M. Lecture notes in computer science. In: ____. [S.l.]: Springer Berlin / Heidelberg, 2004. cap. Deception in Ant Colony Optimization, p. 118–129.

BRUCKER, P. *Scheduling Algorithms*. [S.l.]: Springer Berlin / Heidelberg, 2007.

BULLNHEIMER, B.; HARTL, R.; STRAUSS, C. An improved ant system algorithm for the vehicle routing problem. *Operations Research*, 1997.

CABRAL, S. Analisando a reconfiguração da cadeia de produção de pneus no brasil pela economia dos custos de transação. *Gestão e Produção*, v. 11, p. 373–384, 2004.

CHEN, R.; LO, S.; WU, C.; LIN, T. An effective ant colony optimization-based algorithm for flow shop scheduling. In: *IEEE Conference on Soft Computing in Industrial Applications*. [S.l.: s.n.], 2008.

CHENG, B.-Y.; CHEN, H.-P.; SHAO, H.; XU, R.; HUANG, G. Q. A chaotic ant colony optimization method for scheduling a single batch-processing machine with non-identical job sizes. In: *IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2008. p. 40–43.

CHENG, T. C. E.; LAZAREV, A. A.; GAFAROV, E. R. A hybrid algorithm for the single-machine total tardiness problem. *Comput. Oper. Res.*, Elsevier Science Ltd., Oxford, UK, UK, v. 36, n. 2, p. 308–315, 2009. ISSN 0305-0548.

CHIDAMBER, S.; KEMERER, C. A metric suíte for object oriented designs. *IEEE Transactions on Software Engineering*, 1994.

COLIN, E. C. *Pesquisa Operacional - 170 Aplicações em Estratégia, Finanças, Logística, Produção, Marketing e Vendas*. [S.l.]: LTC, 2007.

COLORNI, A.; DORIGO, M.; MANIEZZO, V. Distribuited optimization by ant colonies. In: *European Conference of Artificial Life*. [S.l.: s.n.], 1991. p. 134–142.

CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. *Theory of Scheduling*. [S.l.]: Addison-Wesley, 1967.

CORDON, O.; VIANNA, I. F.; HERRERA, F.; MORENO, L. A new aco model integrating evolutionary computation concepts: The best-worst ant system. In: *Ants2000*. [S.l.: s.n.], 2000.

DEITEL, H. M.; J., D. P. *Java: Como Programar*. [S.l.]: Prentice-Hall, 2005.

DORIGO, M.; BLUM, C. Ant colony optimization theory: a survey. *Theor. Comput. Sci.*, Elsevier Science Publishers Ltd., Essex, UK, v. 344, n. 2-3, p. 243–278, 2005. ISSN 0304-3975.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, v. 1, p. 53–66, 1997.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, v. 26, p. 29–41, 1996.

DORIGO, M.; STUTZLE, T. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In: *Handbook of Metaheuristics*. [S.l.]: Kluwer Academic Publishers. p. 251–285.

DORIGO, M.; STUTZLE, T. Handbook of metaheuristics. In: ____. [S.l.]: Springer New York, 2006. cap. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, p. 250–285.

ELLABIB, I.; CALAMAI, P.; BASIR, O. Exchange strategies for multiple ant colony system. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 177, n. 5, p. 1248–1264, 2007. ISSN 0020-0255.

ESWARAMURTHY, V. P.; TAMILARASI, A. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. *International Journal of Advanced Manufacturing Technologies*, v. 40, p. 1004–1015, 2009.

FAHIMNIA, B.; MARIAN, R.; MOTEVALLIAN, B. Analysing the hindrances to the reduction of manufacturing lead-time and their associated environmental pollution. *International Journal of Environmental Technology and Management*, v. 10, p. 16–25, 2009.

FANG, H. lan; FANG, H. lan; ROSS, P.; ROSS, P.; CORNE, D.; CORNE, D. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: *Proceedings of the Fifth International Conference on Genetic Algorithms*. [S.l.]: Morgan Kaufmann, 1993. p. 375–382.

FERREIRA, P. A. V. Planejamento e análise de sistemas de produção. Notas de aula (2006). 2006.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*. London, UK: Springer-Verlag, 1997. p. 21–35. ISBN 3-540-62507-0.

GAGNé, C.; PRICE, W. L.; GRAVEL, M. Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. *Journal of the Operational Research Society*, v. 53, p. 895–906, 2002.

GAJPAL, Y.; RAJENDRAN, C. An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops. *International Journal of Production Economics*, v. 101, p. 259–272, 2006.

GAMBARDELLA, L. M.; DORIGO, M. Solving symmetric and asymmetric tsps by ant colonies. In: *In Proceedings of the IEEE Conference on Evolutionary Computation, ICEC96*. [S.l.]: IEEE Press, 1996. p. 622–627.

GAMBARDELLA, L. M.; DORIGO, M. An ant colony system with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, v. 12, p. 237–255, 2000.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Publisher., 1995.

GOLDBARG, M. C.; LUNA, H. P. L. *Otimização Combinatória e Programação Linear*. [S.l.]: Ed. Campus, 2005.

GOLDBERG, D. E. *Genetic algorithms in search, optimization, and machine learning*. [S.l.]: Addison-Wesley Professional, 1989.

GONZALEZ, R.; GASCO, J.; LLOPIS, J. Information systems outsourcing: A literature analysis. *Information and Management*, v. 43, p. 821–834, 2006.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. H. G. R. Optimization and approximation in deterministic machine scheduling: a survey. *Annals of Discrete Mathematics*, v. 5, p. 287–326, 1979.

GRAVES, S. C.; KAN, A. H. G. R.; ZIPKIN, P. H. *Logistics of Production and Inventory*. [S.l.]: Elselvier, 2005.

GUPTA, J. N. D. An excursion in scheduling theory: an overview of scheduling research in the twentieth century. *Production Planning and Control*, v. 13, p. 105–116, 2002.

HEINONEN, J.; PETTERSSON, F. Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, v. 187, n. 2, p. 989 – 998, 2007. ISSN 0096-3003. Disponível em: <http://www.sciencedirect.com/science/article/B6TY8-4M4TNHK-D/2/e1881949c42b94ac5aedb84afd39

HOLTHAUS, O.; RAJENDRAN, C. A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs. *Journal of the Operational Research Society*, v. 56, p. 947–953, 2005.

HU, Y.; YAN, J.; YE, F. Flow shop rescheduling problem under rush orders. *Journal of Zhejiang University*, v. 10, p. 1040–1046, 2005.

HUANG, K.; LIAO, C. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers and Operations Research*, v. 35, p. 1030–1046, 2008.

HUANG, R.; YANG, C. Ant colony system for job shop scheduling with time windows. *International Journal of Advanced Manufacturing Technologies*, v. 39, p. 151–157, 2008.

HUANG, R.; YANG, C. Solving a multi-objective overlapping flow-shop scheduling. *International Journal of Advanced Manufacturing Technology*, v. 42, p. 955–962, 2009.

JIAN, H.; XU, M. The value of production outsourcing: A real options perspective. In: *2006 International Conference on Management Science and Engineering*. [S.l.: s.n.], 2006.

JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, v. 1, p. 61–68, 1953.

KASHAN, A. H.; KARIMI, B. Scheduling a single batch-processing machine with arbitrary job sizes and incompatible job families: An ant colony framework. *Journal of the Operational Research Society*, v. 59, p. 1269–1280, 2008.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *Proceedings., IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1995.

LEE, I. S.; SUNG, C. S. Minimizing due date related measures for a single machine scheduling problem with outsourcing allowed. *European Journal of Operational Research*, v. 186, p. 931–952, 2008a.

LEE, I. S.; SUNG, C. S. Single machine scheduling with outsourcing allowed. *International Journal of Production Economics*, v. 111, p. 623–634, 2008b.

LEE, J.; HUYNH, M. Q.; CHI-WAI, K. R.; PI, S. The evolution of outsourcing research: What is the next issue? In: *33rd Hawaii International Conference in System Sciences*. [S.l.: s.n.], 2000.

LEE, Y. H.; JEONG, C. S.; MOON, C. Advanced planning and scheduling with outsourcing in manufacturing supply chain. *Computers and Industrial Engineering*, v. 43, p. 351–374, 2002.

LENG, M.; PARLAR, M. Lead-time reduction in a two-level supply chain: Non-cooperative equilibria vs. coordination with a profit-sharing contract. *International Journal Production Economics*, v. 118, p. 521–544, 2009.

LEUNG, J. Y. *Handbook of Scheduling*. [S.l.]: Chapman and Hall, 2004.

LI, J.; ZHANG, W. Solution to multi-objective optimization of flow shop problem based on aco algorithm. In: *2006 International Conference on Computational Intelligence and Security*. [S.l.: s.n.], 2006.

LI, L.; QIAO, F.; WU, Q. D. Aco-based multi-objective scheduling of parallel batch processing machines with advanced process control constraints. *International Journal of Advanced Manufacturing Technologies*, v. 44, p. 985–994, 2009.

LI, L.; QUIAO, F.; WU, Q. Ant colony optimization and swarm intelligence. In: ____. [S.l.]: Springer Berlin / Heidelberg, 2008. cap. ACO-Based Scheduling of Parallel Batch Processing Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, p. 219–226.

LIAO, C.-J.; JUAN, H.-C. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. *Comput. Oper. Res.*, Elsevier Science Ltd., Oxford, UK, UK, v. 34, n. 7, p. 1899–1909, 2007. ISSN 0305-0548.

LIN, B.; LU, C.; SHYU, S.; TSAI, C. Development of new features of ant colony optimization for flowshop scheduling. *International Journal of Production Economics*, v. 112, n. 2, p. 742–755, April 2008. Disponível em: <http://ideas.repec.org/a/eee/proeco/v112y2008i2p742-755.html>.

MARIMUTHU, S.; PONNAMBALAM, S. G.; JAWAHAR, N. Threshold accepting and ant-colony optimization algorithms for scheduling m-machine flow shops with lot streaming. *Journal of Materials Processing Technology*, v. 209, p. 1026–1041, 2009.

MARTINS, R. A. Metodologia de pesquisa em engenharia de produção e gestão de operações. In: MIGUEL, P. A. C. (Ed.). [S.l.]: Elsevier, 2010. cap. Abordagens Quantitativa e Qualitativa, p. 45–62.

MERKLE, D.; MIDDENDORF, M. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: *EvoWorkshops*. [s.n.], 2000. p. 287–296. Disponível em: <citeseer.ist.psu.edu/article/merkle00ant.html>.

MERKLE, D.; MIDDENDORF, M. Ant colony optimization with global pheromone evaluation for scheduling a single machine. *Applied Intelligence*, v. 1, p. 105–111, 2003.

MONCH, L. Heuristics to minimize total weighted tardiness of jobs on unrelated parallel machines. In: *IEEE International Conference on Automation Science and Engineering, 2008. CASE 2008.* [S.l.: s.n.], 2008.

MORABITO, R.; PUREZA, V. Metodologia de pesquisa em engenharia de produção e gestão de operações. In: ____. [S.l.]: Elsevier, 2010. cap. Modelagem e Simulação, p. 165–194.

MORTON, T. E.; PENTICO, D. W. *Heuristic Scheduling Systems*. [S.l.]: John Wiley & Sons, 1993.

MOSCATO, P. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Pasadena, California, USA, 1989.

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

OSMAN, I. H.; LAPORTE, G. Metaheuristics: A bibliography. *Annals of Operations Research*, v. 63, p. 513–623, 1996.

OUELHADJ, D.; PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, v. 12, p. 417–431, 2008.

PACHECO, R. F.; SANTORO, M. C. Abordagem de um problema de alocação de rotas de esteiras mecanizadas utilizando modelos de programação da produção. In: *ENEGEP - Encontro Nacional de Engenharia de Produção*. [S.l.: s.n.], 1998.

PASIA, J. M.; HARTL, R. F.; DOERNER, K. F. Solving a bi-objective flowshop scheduling problem by pareto-ant colony optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 4150, p. 294–305, 2006.

PAULILLO, L. F. Terceirização e reestruturação agroindustrial: avaliando o caso citrícola brasileiro. *Revista Administração*, v. 3, p. 87–103, 1999.

PINEDO, M. L. *Scheduling: Theory, Algorithms and Systems*. [S.l.]: Springer, 2008.

PINEDO, M. L. *Planning and Scheduling in Manufacturing and Services*. [S.l.]: Springer, 2009.

QI, X. Two-stage supply chain scheduling with an option of outsourcing in stage one. In: *2008 International Conference on Service Systems and Service Management*. [S.l.: s.n.], 2008.

RAGHAVAN, N. R. S.; VENKATARAMANA, M. Scheduling parallel batch processors with incompatible job families using ant colony optimization. In: *Proceeding of the 2006 IEEE International Conference on Automation Science and Engineering*. [S.l.: s.n.], 2006.

RAJENDRAN, C.; ZIEGLER, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, v. 155, n. 2, p. 426–438, June 2004. Disponível em: <http://ideas.repec.org/a/eee/ejores/v155y2004i2p426-438.html>.

RAJENDRAN, C.; ZIEGLER, H. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Comput. Ind. Eng.*, Pergamon Press, Inc., Tarrytown, NY, USA, v. 48, n. 4, p. 789–797, 2005. ISSN 0360-8352.

SANKAR, S. S.; PONNAMBALAM, S. G.; RATHINAVEL, V.; VISVESHVAREN, M. S. Scheduling in parallel machine shop: an ant colony optimization approach. In: *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*. [s.n.], 2005. p. 276–280. Disponível em: <http://dx.doi.org/10.1109/ICIT.2005.1600649>.

SHYU, S.; LIN, B.; YIN, P. Application of ant colony optimization for no-wait flowshop scheduling problem to minimize the total completion time. *Computers and Industrial Engineering*, v. 47, p. 181–193, 2004.

STALK, G. *Time – The next source of competitive advantage*. [S.l.]: Harvard Business Review, 1988.

STOVBA, S. D. Ant algorthms: Theory and applications. *Programming and Computer Software*, v. 31, n. 4, p. 167–178, 2005.

STUTZLE, T. An ant approach to the flow shop problem. In: *In Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98*. [S.l.]: Verlag, 1998. p. 1560–1564.

STUTZLE, T.; HOOS, H. H. Max-min ant system. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 16, n. 9, p. 889–914, 2000. ISSN 0167-739X.

SURI, R. *Quick Response Manufacturing*. [S.l.]: Productivity Press, 1998.

SURI, R. *It's About Time*. [S.l.]: CRC Press, 2010.

TAVARES NETO, R. F. *Planejamento de Vistorias através de robôs móveis utilizando o algoritmo de colônia de formigas*. Dissertação (Mestrado) — Pontificia Universidade do Paraná, 2005.

TAVARES NETO, R. F.; GODINHO FILHO, M. Proposta de um framework para prototipagem de sistemas heurísticos multi-agentes baseados em algoritmos de colônia de formigas. *Pesquisa Operacional*, 2009.

THIRUVADY, D.; BLUM, C.; MEYER, B.; ERNST, A. Lecture notes in computer science. In: ____. [S.l.]: SpringerLink, 2009. cap. Hybridizing Beam-ACO with Constraint Programming for Single Machine Job Scheduling, p. 30–44.

T'KINDT, V.; MONMARCHE, N.; TERCINET, F.; LAUGT, D. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, v. 142, n. 2, p. 250–257, 2002.

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

VALENTE, J. M. S.; ALVES, R. A. F. S. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Computers & Operations Research*, v. 35, p. 2388–2405, 2008.

WEIHUA, G.; XIAOQING, G. Empirical analysis on supply chain of offshore software outsourcing from china perspective. In: *IEEE International Conference on Service Operations and Logistics, and Informatics*. [S.l.: s.n.], 2007.

XIA, W.; WU, Z. A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, v. 29, p. 360–366, 2006.

XING, L.; CHENG, Y.; YANG, K. Double layer aco algorithm for the multi-objective fjssp. *New Generation Computing*, v. 26, p. 313–327, 2008.

YADAV, V.; GUPTA, R. K. A paradigmatic and methodological review of research in outsourcing. *Information Resources Management Journal*, v. 21, n. 1, p. 27–43, 2008.

YAGMAHAN, B.; YENISEY, M. M. Ant colony optimization for multi-objective flow shop scheduling problem. *Computers & Industrial Engineering*, v. 54, p. 411–420, 2008.

YAN-HAI1, H.; JUN-QI, Y.; FEI-FAN, Y.; JUN-HE, Y. Flow shop rescheduling problem under rush orders. *Journal of Zhejiang University - Science A*, v. 6, p. 1040–1046, 2005.

YING, K.; LIN, S. Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems. *International Journal of Advanced Manufacturing Technologies*, v. 33, p. 793–802, 2007.

YING, K.-C.; LIAO, C.-J. An ant colony system approach for scheduling problems. *Production Planning and Control*, v. 14, p. 68–75(8), 2003. Disponível em: <http://www.ingentaconnect.com/content/tandf/tppc/2003/00000014/00000001/art00006>.

YOSHIKAWA, M.; TERAI, H. A hybrid ant colony optimization technique for job-shop scheduling problems. In: *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*. [S.l.: s.n.], 2006.

ZAPFEL, G.; BOGL, M. Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, n. 113, p. 980–996, 2008.

ZHOU, H.; LI, Z.; WU, X. Scheduling unrelated parallel machine to minimize total weighted tardiness using ant colony optimization. In: *IEEE International Conference on Automation and Logistics*. [S.l.: s.n.], 2007.

ZHOU, P.; QINGSHAN, D. Hybridizing fast taboo search with ant colony optimization algorithm for solving large scale permutation flow shop scheduling problem. In: *IEEE International Conference on Granular Computing*. [S.l.: s.n.], 2009.

ZHUO, X.; ZHANG, J.; CHEN, W. A new pheromone design in acs for solving jsp. In: *IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2007.