Gest. Prod., São Carlos, v. 20, n. 1, p. 76-86, 2013

Otimização por colônia de formigas para o problema de sequenciamento de tarefas em uma única máquina com terceirização permitida

An ant colony optimization approach for the single machine scheduling problem with outsourcing allowed

Roberto Fernandes Tavares Neto[1]
Moacir Godinho Filho[1]

Resumo: Este artigo trata do problema de sequenciamento de tarefas em um ambiente de máquina única com possibilidade de terceirização. O problema apresentado busca minimizar a soma ponderada dos custos totais de terceirização e do somatório dos tempos de finalização de cada tarefa e é definido na literatura como 1 / *Budget* / $(1 - \delta) \Sigma C_j + \delta$. *OC*. Para esta resolução do referido problema, é proposto um método formado por dois estágios: no primeiro estágio, propõe-se que as tarefas sejam sequenciadas, utilizando-se a regra SPT (*Shortest Processing Time* - Menor Tempo de Processamento) para que se consiga a redução do espaço de busca no grafo gerado, enquanto que, no segundo estágio, é proposto um algoritmo baseado em ACO (*Ant Colony Optimization* – Otimização por Colônia de Formigas). O algoritmo aqui proposto incorpora ao ACO quatro características específicas do problema estudado, a saber: i) uma representação em forma de grafo para problemas de *scheduling* que envolvam terceirização, obtida por meio da aplicação do primeiro estágio do método; ii) uma regra de pré-seleção, que garante a viabilidade da solução; iii) uma nova regra de visibilidade específica para o problema; e iv) uma estratégia de busca local. Os resultados obtidos neste trabalho mostram que o algoritmo baseado em ACO desenvolvido é mais eficiente que o algoritmo de Lee e Sung (2008a) que foi o único trabalho encontrado na literatura que trata do problema em discussão. Adicionalmente, a busca local proposta melhorou o resultado para problemas de tamanho médio e grande.

Palavras-chave: Sequenciamento e programação de operações. Colônia de formigas. Terceirização.

Abstract: *This paper considers the problem of scheduling a set of tasks on a single machine environment with outsource allowed. This multicriteria problem, defined as 1 / Budget / (1 – δ) Σ C_j + δ . OC, aims to minimize the weighted sum of total outsourcing costs and the sum of individual completion times. To solve this problem, a two-stage method is proposed: the first stage orders the tasks using the SPT (Shortest Processing Time) rule. The second stage uses an ACO (Ant Colony Optimization) algorithm to decide whether the job will be outsourced by using four problem-specific variations: i) a graph-representation of scheduling problems for scheduling problems with outsourcing allowed; ii) a pre-selection rule that guarantees the viability of the solution; iii) a new problem-specific visibility rule; and iv) a local search strategy. The implementation of this method produces better results than those obtained by Lee and Sung (2008a), the only paper found in the literature that deals with this problem. Moreover, the local search procedure was able to improve the results for medium and large problem sets.*

Keywords: *Scheduling. Ant colony optimization. Outsourcing.*

1 Introdução

Desde o trabalho publicado por Colorni, Dorigo e Maniezzo (1991), muitos pesquisadores usam as chamadas "formigas artificiais" para resolver problemas combinatórios complexos. Esta classe de algoritmos, chamada ACO (*Ant Colony Optimization* - Otimização por Colônia de Formigas), se baseia no comportamento de formigas reais e já mostrou bons resultados para problemas combinatórios descritos na literatura como NP-difíceis. Como exemplo, podemos observar Dorigo, Maniezzo e Colorni, (1996) e Gambardella e Dorigo (1996) que propõem um algoritmo para a solução tanto do caixeiro viajante simétrico quanto assimétrico, Gambardella e Dorigo (2000) que tratam do problema de ordenamento sequencial, Bullnheimer, Hartl e Strauss (1997) que trabalham com o problema de roteamento de veículos com capacidade limitada e Bauer et al. (2000) que tratam de problemas de *scheduling*. Mais problemas

[1] Departamento de engenharia de produção, Universidade federal de são carlos. {tavares,moacir}@dep.ufscar.br
Rod Washington Luis, km 235, São Carlos, São Paulo. CEP 13565-905, e-mail: tavares@dep.ufscar.br; moacir@dep.ufscar.br

Recebido em 26/4/2009 — Aceito em 25/7/2011

Suporte financeiro: FAPESP.

e pesquisas sobre ACO podem ser encontrados em Dorigo e Stutzle (2004) que apresentam mais de 60 publicações no tema.

Existem também várias pesquisas que se utilizam de ACO para a resolução de uma grande variedade de problemas de *scheduling*. Como exemplo, pode-se citar pesquisas que focam o *scheduling* em máquina única (ZAPFEL; BOGL, 2008; HOLTHAUS; RAJENDRAN, 2005; MERKLE; MIDDENDORF, 2000); máquinas paralelas (ARNAOUT; MUSA; RABADI, 2008); *flow-shop* (AHMADIZAR; BARZINPOUR; ARKAT, 2007; MARIMUTHU; PONNAMBALAM; JAWAHAR, 2009) e *job-shop* (ZHOU; LEE; NEE, 2008; ZHUO; ZHANG; CHEN, 2007).

Porém, a literatura parece carecer de abordagens a problemas de *scheduling* que consideram a possibilidade de terceirização. Durante a fase de revisão bibliográfica, apenas dois trabalhos foram encontrados no que se refere a problemas de *scheduling*: o primeiro, de Lee e Sung (2008a), trata do problema de minimização do somatório ponderado entre tempos de finalização e custo total de terceirização em um ambiente de máquina única restrito a um valor de orçamento limite (representado por $1 \,/\, Budget \,/\, (1 - \delta) \Sigma C_j + \delta \,.\, OC$); o segundo trabalho, de Lee e Sung (2008b), trata de um problema semelhante ao anterior, porém com o objetivo de minimizar o somatório ponderado entre o *lateness* máximo e o custo total de terceirização (problema este representado por $1 \,/\, Budget \,/\, (1 - \delta) L_{max} + \delta \,.\, OC$). Alguns poucos trabalhos encontrados também tratam de terceirização em operações logísticas, por exemplo, o trabalho de Zäpfel e Bögl (2008).

Dentro do referido contexto, o presente trabalho apresenta e implementa a proposta de um método baseado em ACO para a resolução do problema $1 \,/\, Budget \,/\, (1 - \delta) \Sigma C_j + \delta \,.\, OC$. Este problema é definido por Lee e Sung (2008a) da seguinte forma: considere um conjunto de *n* tarefas que devem ser alocadas em uma máquina única ou realizadas por meio de terceirização. O objetivo é minimizar a soma dos tempos para finalização de cada tarefa (ΣC_j) e do custo total de terceirização (*OC*). Adicionalmente, existe um limite para o custo total de terceirização, definido por Lee e Sung (2008a), como *Budget*. A soma de ΣC_j e *OC* é ponderada por meio de um parâmetro de custo, $0 < \delta < 1$. Os resultados obtidos pelo método aqui proposto e a serem apresentados neste trabalho se mostraram superiores aos encontrados por Lee e Sung (2008a). Adicionalmente, é mostrado que, quando se trabalha com o método de duas fases proposto no presente estudo, a busca local não é necessária para problemas de dimensões pequenas. Para problemas de dimensões maiores, o procedimento de busca local proposto aumenta a qualidade da resposta obtida.

Este artigo é estruturado da seguinte forma: na seção 2, discute-se em detalhes o problema de *scheduling* abordado neste artigo. A seção 3 mostra os conceitos básicos de ACO, focando o algoritmo *Max-Min Ant System* (MMAS), que será usado como base de desenvolvimento do método aqui proposto. A seção 4 apresenta o método proposto neste trabalho para a resolução do problema $1 \,/\, Budget \,/\, (1 - \delta) \Sigma C_j + \delta \,.\, OC$. A seção 5 apresenta e analisa os resultados computacionais obtidos por este trabalho, já com a aplicação do método proposto, enquanto que na seção 6 são tecidas as conclusões deste trabalho.

2 Descrição do problema

Nesta seção são descritas a formulação do problema e as notações usadas neste artigo. Como mencionado antes, um problema de *scheduling* busca alocar algumas tarefas com o intuito de atingir uma medida de *performance*. Uma característica importante de qualquer problema de *scheduling* é o ambiente de manufatura. O mais simples de se descrever (mas não necessariamente de se resolver) é o ambiente de máquina única. Como o nome sugere, este ambiente é composto de um único processador ("máquina") que deve processar todas as tarefas. Embora seja difícil encontrar situações práticas em que exista apenas uma máquina, há algumas situações específicas em que se pode usar os resultados obtidos por pesquisas nestes ambientes, por exemplo, quando existe um gargalo bem definido em um processo ou um recurso muito caro. Alguns autores, por exemplo, Blazewicz et al. (2007), também defendem que soluções de problemas de máquina única podem ser usados, às vezes, como "blocos básicos" de construção para a resolução de problemas mais complexos. Este fato é muito relevante no contexto deste artigo, especialmente porque algumas estratégias usadas na resolução de problemas de *scheduling* sem terceirização são usados para se obter o algoritmo proposto para o problema $1 \,/\, Budget \,/\, (1 - \delta) \Sigma C_j + \delta \,.\, OC$.

O problema abordado neste artigo pode ser definido como:

Seja *J* um conjuto de *n* tarefas a serem sequenciadas para produção *in-house* ou terceirizadas, cada tarefa J_i é definida por: a) um tempo de processamento p_i; b) um custo de terceirização o_i; e c) um tempo de entrega de terceirização l_i.

Como mencionado anteriormente, estes problemas contêm um conjunto de tarefas. Cada tarefa pode ser sequenciada em uma máquina única ou terceirizada. O término de uma tarefa *i* é definido pela Equação 1. O custo total de terceirização é definido pela Equação 2.

$$C_i = \begin{cases} l_i & \text{se } i \in O_\pi \\ \sum\limits_{k \in SP_k} pk + p_i & \text{se } i \in S_\pi \end{cases} \qquad (1)$$

$$OC = \sum_{j \in O_\pi} o_j \quad (2)$$

em que:

- O conjunto O_π contém todas as tarefas terceirizadas;
- O conjunto S_π contém as tarefas a serem realizadas na máquina única;
- O conjunto SP_k contém todas as tarefas de S_π sequenciadas antes da tarefa k.

A função objetivo do problema a ser estudado é mostrada na Equação 3:

$$\min z = (1-\delta) \cdot \sum_{j=1}^{n} C_j + \delta \cdot OC \quad (3)$$

Adicionalmente, existe uma condição de contorno que indica que $OC \leq Budget$.

Este problema é descrito por Lee e Sung (2008a) como um problema NP-difícil e descrito como $1 / Budget / (1 - \delta) \Sigma C_j + \delta . OC$, $0 < \delta < 1$. Desta forma, esse problema pode ser entendido como um problema de *scheduling* de otimização multicritério, que busca a minimização de uma soma ponderada entre a soma dos tempos de término de cada tarefa e o custo total de terceirização. Para resolver tal problema, os autores propuseram um conjunto de algoritmos baseados em técnicas de programação dinâmica, sendo seus resultados comparados, na seção 6, com o método proposto nesse artigo.

O problema definido por Lee e Sung (2008a) não é o primeiro estudo que foca estratégias de terceirização. Porém, embora seja possível encontrar um conjunto de artigos que tratam do tema (LEE; JEONG; MOON, 2002; QI, 2008; ZAPFEL; BOGL, 2008), nenhum outro artigo encontrado na literatura focou o problema de *scheduling* para o problema $1 / Budget / (1 - \delta) \Sigma C_j + \delta . OC$ de tal modo como foi proposto no presente trabalho. Na próxima seção, são mostrados os conceitos básicos de ACO, focando o algoritmo *Max-Min Ant System* (MMAS), que será usado como base de desenvolvimento do método aqui proposto.

3 A otimização por colônia de formigas

Nesta seção é apresentada, no item 3.1, a estrutura geral do algoritmo ACO. Para se aplicar o ACO em um problema de *scheduling*, é necessário o desenvolvimento de uma representação em forma de um grafo.

3.1 Estrutura geral do algoritmo ACO

O algoritmo proposto por Colorni, Dorigo e Maniezzo (1991), denominado *Ant Colony Optimization* (ACO - Otimização por Colônia de Formigas), usa uma abordagem de inteligência coletiva (*swarm intelligence*) para resolver o problema do caixeiro viajante (*Travelling Salesman Problem* - TSP). Neste problema, já muito tratado pela literatura, um conjunto de n cidades devem ser visitadas por um único vendedor ambulante. O objetivo é minimizar o percurso da viagem. Para resolver o TSP, o algoritmo ACO busca imitar o comportamento de formigas reais indo do ninho até a fonte de alimento. Para isso, agentes computacionais ("formigas") são posicionados em um grafo e forçados a se movimentar pelos nós até que uma condição de parada seja satisfeita. O caminho feito pela formiga é uma solução do problema. Segundo Colorni, Dorigo e Maniezzo (1991), o comportamento deste algoritmo pode ser exemplificado na Figura 1. Como se pode notar, no instante $t = 0$, uma formiga escolhe de forma aleatória qualquer um dos caminhos disponíveis. Quando se chega no objetivo, uma função definida previamente analisa o caminho realizado, e se obtém a qualidade da solução gerada. De acordo com esta análise, é realizado um depósito de feromônio artificial. Este feromônio faz parte da estratégia de *reforço positivo* do algoritmo e busca privilegiar as melhores soluções. Para evitar saturação nos valores de feromônio (ou seja, que os níveis de feromônio depositados se tornem tão grandes que novos depósitos não sejam significativos), Colorni, Dorigo e Maniezzo (1991) desenvolveram também uma estratégia de reforço negativo, chamada evaporação dos feromônios. Esta estratégia normalmente é modelada como um decaimento linear do nível atual de feromônio existente. Com este ciclo de depósito-evaporação aplicado em cada trecho de um caminho, consegue-se que os melhores caminhos sejam escolhidos com mais frequência que os demais.

A heurística desenvolvida por Colorni, Dorigo e Maniezzo (1991) é mostrada no Algoritmo 1. Neste algoritmo, cada iteração se inicia com o posicionamento de cada formiga em um nó inicial, definido por uma regra específica (por exemplo, todas as formigas podem ser posicionadas em um mesmo

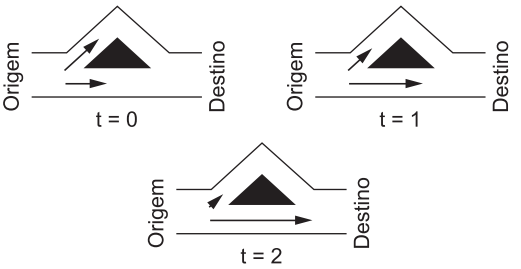

Figura 1. O comportamento básico da Otimização por Colônia de Formigas em diferentes momentos do tempo. O tamanho das setas indica a probabilidade da escolha de um caminho.

Algoritmo 1. O algoritmo ACO.

```
1  Inicialize
2  Repita Neste nível, cada execução é chamada iteração
3      Repita Neste nível, cada execução é chamada passo
4          Cada formiga aplica uma regra de transição para construir a próxima etapa da solução
5          Aplica-se a atualização local de feromônios
6      Até que todas as formigas tenham criado uma solução completa
7      Aplica-se o procedimento de busca local
8      Aplica-se o procedimento de atualização global de feromônios
9  Até que o critério de parada seja satisfeito
```

nó inicial ou serem alocadas de forma aleatória). Depois do posicionamento, todas as formigas se movem de acordo com uma *regra de transição* (a regra de transição normalmente contém um componente aleatório ponderado) até que se tenha uma solução completa (ou seja, uma condição de parada seja satisfeita). Uma solução é uma sequência de movimentos de uma formiga. Para a escolha do próximo nó, a *regra de transição* se utiliza de uma *função de visibilidade*. Depis de cada movimentação (chamada na literatura "passo" da formiga), os feromônios são atualizados de acordo com uma *regra de atualização local*. Depois de todas as formigas terem construído uma solução, dois procedimentos são executados: o primeiro, opcional, é um procedimento de busca local que tenta melhorar a solução criada pelas formigas; o segundo é a aplicação de uma *regra de atualização global de feromônios*. Este ciclo continua até que um segundo critério de parada seja satisfeito (por exemplo, o tempo computacional tenha sido atingido ou o número máximo de iterações tenha sido atingido).

Durante os últimos anos, foram propostas algumas variações do ACO. Dorigo, Maniezzo e Colorni (1996) e Gambardella e Dorigo (1996) propuseram o *Ant Colony System* (ACS), uma estratégia muito usada na literatura que serviu de base para o *Max-Min Ant System* (MMAS), proposto por Stützle e Hoos (2000). Neste artigo, o MMAS será usado como base do método aqui proposto.

O algoritmo ACS se utiliza do pseudocódigo do ACO (mostrado no Algoritmo 1) e incorpora o conceito de exploração/reforço. Segundo Gambardella e Dorigo (1996), o algoritmo ACS define a regra de transição conforme mostrado na Equação 4.

$$s = \begin{cases} \operatorname{argmax}\left\{ \tau_{ij} \cdot \left[\eta_{ij} \right]^{\beta} \right\} & \text{se } q \le q_0 \\ S & \text{caso contrário} \end{cases} \quad (4)$$

em que:

- i e j são nós de um grafo, conectados por arcos s;
- q é uma variável aleatória de distribuição uniforme entre 0 e 1;
- $0 \le q_0 \le 1$ e $0 \le \beta \le 1$ são parâmetros do algoritmo;
- $\tau(i,j)$ é a intensidade de feromônio entre os nós i e j;
- $\eta(i,j)$ é um valor determinado de acordo com o tipo de problema, chamado por alguns autores de *visibilidade*;
- S é um caminho escolhido conforme a Equação 5;

$$p_{ij}^{k} = \begin{cases} \dfrac{\left(\tau_{ij}^{\ \alpha}\right) \cdot \left(\eta_{ij}\right)^{\beta}}{\sum_{j \in N^*} \left(\tau_{ij}^{\ \alpha}\right) \cdot \left(\eta_{ij}\right)^{\beta}} & \text{se } j \in N^* \\ 0 & \text{caso contrário} \end{cases} \quad (5)$$

Em que:

- p_{ij}^{k} é a probabilidade da escolha da trilha que liga os nós i e j pela formiga k;
- N^* é um conjunto de nós ainda não visitados pela formiga.

Embora a regra apresentada na equação 5 tenha sido usada nos trabalhos de Colorni, Dorigo e Maniezzo (1991), Gambardella e Dorigo (1996) e Dorigo, Maniezzo e Colorni (1996), pesquisas posteriores a simplificaram ao atribuir ao parâmetro α o valor 1, encontrando bons resultados. Isso evita uma operação de exponenciação adicional e assim reduz o tempo computacional do algoritmo. A implementação do algoritmo presente neste artigo usa esta estratégia.

Conforme apontado por Stovba (2005), o ACS pode ser aplicado a problemas de baixa dimensionalidade. Porém, quando os problemas começam a ter um número maior de dimensões, o algoritmo original sem modificações não consegue obter soluções melhores do que algoritmos de propósito específico, desenvolvidos especialmente para o problema em questão. Este fato impulsionou a geração de um conjunto de algoritmos baseados no ACS, como o MMAS. Conforme descrito por Stützle e Hoos (2000), o MMAS é baseado no algoritmo ACO apresentado por Colorni, Dorigo e Maniezzo (1991) e no algoritmo ACS apresentado

por Dorigo, Maniezzo e Colorni (1996). A maior diferença do MMAS é a definição de um nível mínimo e máximo de feromônio em cada arco do grafo. Esta característica, em conjunto com uma estratégia elitista - que permite a atualização seguindo apenas a melhor solução encontrada - fez o algoritmo MMAS, conforme Dorigo e Blum (2005), se tornar uma das variações do ACO de grande sucesso (outras variações do ACO podem ser encontradas facilmente na literatura (BULLNHEIMER; HARTL; STRAUSS, 1997; CORDON et al., 2000). A estratégia de reforço usada é mostrada na Equação 6. A contribuição individual de feromônios é descrita na Equação 7. É importante notar que esta estratégia de atualização considera todas as soluções geradas pelas *m* formigas. Este comportamento é modificado quando se executa uma estratégia elitista. Esta estratégia, mostrada nas Equações 8 e 9, permite o depósito de feromônios apenas nas trilhas pertencentes à melhor solução encontrada até o momento.

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Sigma\Delta^{k}_{ij}(t) \quad (6)$$

em que:

- $\tau_{ij}(t+1)$ é o valor atualizado de feromônio na trilha que conecta os nós *i* e *j*;
- $0 \leq \rho \leq 1$ é uma constante responsável pela diminuição ("evaporação") do feromônio;
- $\Delta^{k}_{ij}(t)$ é a quantidade de feromônio depositada na trilha devido à influência da formiga *k* no instante *t*.

$$\Delta^{k}_{ij}(t) = \begin{cases} 1/L_k & \text{se a formiga } k \text{ vai do nó } i \text{ ao nó } j \\ 0 & \text{caso contrário} \end{cases} \quad (7)$$

em que:

- L_k representa um valor referente à qualidade da solução encontrada pela formiga *k* (por exemplo, a distância percorrida no caso da solução do TSP).

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta_{ij}(t) \quad (8)$$

em que:

- $\Delta_{ij}(t)$ é a quantidade de feromônio depositada na trilha.

$$\Delta_{ij}(t) = \begin{cases} 1/Q & \text{se o arco que conecta os nós } i\,j \in \text{melhor solução} \\ 0 & \text{caso contrário} \end{cases} \quad (9)$$

em que:

- Q é um parâmetro do algoritmo.

Como o MMAS vem mostrando um considerável sucesso nas pesquisas mostradas até o momento, este artigo usá-lo-á como base para o desenvolvimento do algoritmo aqui proposto.

A seguir, será apresentado o método proposto para a resolução do problema $1 \,/\, Budget \,/\, (1-\delta)\,\Sigma\, C_j + \delta \cdot OC$.

4 O método proposto para a resolução do problema

O método proposto neste trabalho, baseado na heurística ACO, é composto de dois estágios: o primeiro estágio ordena as tarefas usando a regra SPT (*Shortest Processing Time* - Menor Tempo de Processamento) buscando a redução do espaço de busca. Com isso, esta fase tem como resultado uma representação do problema em forma de grafo que permite ao ACO escolher apenas se uma tarefa deve ou não ser terceirizada. No segundo estágio, uma variação do algoritmo ACO é proposta e aplicada, selecionando quais tarefas devem ser terceirizadas e quais não devem. Esse algoritmo incorpora ao ACO quatro características específicas do problema estudado, a saber: i) uma representação em forma de grafo para problemas de *scheduling* que envolvam terceirização, resultado da aplicação do primeiro estágio do método; ii) uma regra de pré-seleção, que garante a viabilidade da solução; iii) uma nova regra de visibilidade, específica para o problema; e iv) uma estratégia de busca local.

Para apresentar o algoritmo, serão descritas a seguir as duas fases subsequentes do método proposto:

- Fase 1: O grafo é gerado por meio do pré-processamento do problema, conforme mostrado no Algoritmo 2;
- 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.

4.1 A Fase 1 do método proposto: A criação do grafo

Um ponto fundamental para o método proposto neste trabalho para a resolução do problema $1 \,/\, Budget \,/\, (1-\delta)\,\Sigma\, C_j + \delta \cdot OC$ é a criação do grafo que representa o problema. Para tal, levamos em consideração que a regra SPT (*Shortest Process Time* - menor tempo de processamento) pode ser usada para minimizar a soma de termos de individuais de finalização das tarefas (a regra SPT, conforme Baker (1943) e que minimiza o tempo de fluxo das tarefas. Se todas forem liberadas no mesmo instante, pode se mostrar que o SPT também minimiza os tempos de finalização). Essa mesma propriedade da sequência SPT é usada por Lee e Sung (2008a). 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 usada para a criação do grafo é mostrado no Algoritmo 1.

Algoritmo 2. O algoritmo proposto para a geração do grafo.

```
1 Ordene todos os trabalhos de acordo com a regra SPT
2 Para i = n até 1 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
```

A Figura 2 mostra um grafo representando um problema, este contendo 3 trabalhos, gerado pelo Algoritmo 1. 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.

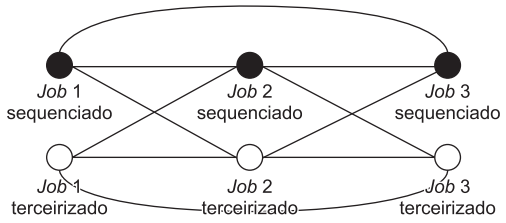

Figura 2. Grafo representando problema de *scheduling* de 3 trabalhos em um ambiente de máquina única com possibilidade de terceirização

4.2 A Fase 2 do método proposto: A proposição de um algoritmo MMAS para o problema estudado

Depois da realização da fase 1, é proposto um algoritmo no presente trabalho, baseado no MMAS, para resolver o problema 1 / *Budget* / $(1 - \delta) \Sigma C_j + \delta \, . \, OC$. Tal algoritmo apresenta as seguintes características específicas para o problema:

- Implementação de uma regra de pré-seleção, indicada no Algoritmo 3;
- Implementação da visibilidade (η) conforme indicado na Equação 10;
- Implementação de um procedimento de busca local conforme indicado no Algoritmo 4.

A seguir, estas características serão detalhadas.

4.2.1 A regra de pré-seleção

A 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 ao conjunto S_π e a formiga se move para o próximo trabalho. Este comportamento é mostrado no Algoritmo 3.

Desta forma, a regra de pré-seleção de nós define uma política de escolha 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.

4.2.2 A definição da visibilidade

Outra mudança realizada na heurística apresentada neste trabalho é a definição do parâmetro visibilidade (η). Para a resolução do problema 1 / *Budget* / $(1 - \delta) \Sigma C_j + \delta \, . \, OC$, este artigo define a visibilidade como sendo um parâmetro η^c_{ij}. Neste caso, i é o nó representando o trabalho atual, j é o nó

Algoritmo 3. A regra de pré-seleção de nós.

```
1 Se o_j + Σ_{k∈O_π} o_k então
2     Escolha entre os nós e use a regra de transição encontrada na equação 12
3 Senão
4     Escolha o trabalho J_ij^1
5 Fim
```

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^0_{ij}, o trabalho J_i é adicionado a S_π. Caso contrário, J_i é adicionado ao conjunto O_π. A definição de η^c_{ij} usada neste trabalho é mostrada na Equação 10. Nesta equação, $C^{S_\pi}_{max}$ é o tempo de finalização máximo das tarefas pertencentes ao conjunto S_π, e $N_{tarPosteriores}$ é o número de tarefas que não pertencem nem a $S\pi$ nem a O_π.

$$\eta^c_{ij} = \begin{cases} \dfrac{1}{\delta \cdot o_j + (1-\delta) \cdot \dfrac{l_j}{p_j}} & \text{se } c = 1 \\ \dfrac{1}{\left((1-\delta) \cdot \left(\dfrac{p_j}{l_j} \cdot \left(C^{S_\pi}_{max} + N_{tarPosteriores}\right)\right)\right)} & \text{caso contrário} \end{cases} \quad (10)$$

4.2.3 O procedimento de busca local proposto

Especialmente para instâncias de maior escala do problema, nota-se a necessidade do uso de uma regra de busca local. Desta forma, este artigo propõe

Algoritmo 4. O algoritmo de busca local proposto.

```
1  BuscaLocal (SI, Problema, Ponteiro, Budget, SP, SA)
2      Início
3      Se Ponteiro < 0 então
4          Retorna (SA)
5      Senão
6          Se Problema_k ∈ S_π e o_problema_k < Budget então
7              Seja costifo = δ · o_problema_k + (1 – δ) · l_problema_k
8              Seja costifs = (1 – δ) · (SP [Ponteiro] + p_problema_k · (n – p)
9              Se costifo > costifs então
10                 Retorna (BuscaLocal (SI, Problema, Ponteiro – 1, Budget, SP, SA))
11             Senão
12                 Seja caseS = BuscaLocal (SI, Problema, Ponteiro – 1, Budget, SP, SA)
13                 Mova Problema_Ponteiro de S_π para O_π e atualize Budget
14                 Seja caseO = BuscaLocal (SI, Problema, Ponteiro – 1, Budget, SP, SA)
15                 Se Fitness(caseS) > Fitness(caseO) então
16                     Retorna (caseO)
17                 Senão
18                     Retorna (caseS)
19                 Fim
20             Fim
21         Senão
22             Retorna (BuscaLocal (SI, Problema, Ponteiro – 1, Budget, SP, SA))
23         Fim
24     Fim
25 Fim
```

o Algoritmo 4 como estratégia de busca local. Este algoritmo considera as seguintes variáveis de entrada:

- *SI*: a sequência obtida inicialmente pelo algoritmo baseado em ACO, composta de duas subsequências distintas: O_π e S_π (trabalhos terceirizados ou não);
- *Problema*: a sequência dos *n* trabalhos originais ordenados por meio da regra SPT;
- *Ponteiro*: uma variável que indica um elemento a ser analisado $0 \leq Ponteiro \leq n$;
- *Budget*: o orçamento disponível;
- *SP*: um vetor de *n* elementos definidos como: $SP_i = \Sigma p_k$;
- *SA*: a solução atual.

As seguintes variáveis também são usadas no procedimento de busca local:

- *k*: uma posição de uma tarefa;
- *Problema*: o trabalho na posição *k* do conjunto *Problema.*

Os atributos de cada trabalho são indicados como O_{job} (custo de terceirização), l_{job} (*lead time* de terceirização) e p_{job} (tempo de processamento).

Estes elementos são obtidos quando a formiga construir a solução. O seguinte procedimento é executado: Um trabalho terceirizado é escolhido aleatoriamente, e movido de O_π para S_π. Estas são as sequências *SI* e *SA*. Adicionalmente, o orçamento disponível (*Budget*) e o vetor *SP* são calculados. *Ponteiro* é selecionado de forma a indicar o último elemento de *SI*.

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

5 Resultados computacionais

O método proposto foi desenvolvido usando a linguagem JAVA e executado em um *Pentium* D 2.80 GHz *dual core* com 2 Gb de memória. O sistema foi desenvolvido usando a JVN 1.6 e testado em uma plataforma *Ubuntu Linux* com gerenciador de janelas *Gnome*.

Para cada número de tarefas (10, 20, 30, 40, 50, 60 e 70), 20 instâncias diferentes foram geradas usando os seguintes parâmetros, baseados no trabalho de Lee e Sung (2008a): os tempos de processamento foram amostrados a partir de um intervalo [1, 10], assim como os custos de terceirização do intervalo [1, 40] e os *lead times* de terceirização do intervalo [1,30]. A execução desses procedimentos foi necessária devido à impossibilidade de se obter os dados usados originalmente por Lee e Sung (2008a).

Lee e Sung (2008a) obtiveram o valor ótimo por meio de uma técnica de *branch-and-bound*. Essa técnica consistiu na determinação de um limite da qualidade da solução por meio de um conjunto de técnicas de programação dinâmica. Foi então aplicada uma busca em profundidade.

Para cada problema estudado no presente trabalho, o valor ótimo ($Fitness_{otimo}$) foi encontrado, usando um procedimento de testes exaustivos. Este procedimento consiste em: i) ordenar os trabalhos na ordem SPT; e ii) verificar todas as 2^n possibilidades de terceirização e não terceirização dos trabalhos. A melhor solução viável é armazenada para posterior comparação com os valores encontrados pelo algoritmo aqui proposto. Durante a experimentação, todos os parâmetros do ACO foram estabelecidos como constantes, com exceção do parâmetro $\beta = \{0,2.5,5\}$, o qual possibilita a análise do impacto da visibilidade (η) na qualidade de solução. Quando $\beta = 0$, a visibilidade não é considerada no processo de geração de soluções. Os demais parâmetros utilizados foram: a) Níveis de feromônios: entre 15 e 20, iniciando em 20; b) $\rho = 0,99$; c) $1 / Q = 0,2$; d) 5 formigas por colônia; e) 10 iterações; f) $q_0 = 1$.

Para analisar o impacto da estratégia de busca local, todos os experimentos foram realizados duas vezes: a primeira não utiliza a busca 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:

- Na Tabela 1, são mostrados o *gap* médio para cada conjunto de instâncias e a comparação entre os resultados deste artigo e os resultados apresentados pelo melhor valor do *gap* médio encontrado por Lee e Sung (2008a). A Figura 3 mostra essa comparação. Vale ressaltar que os melhores valores encontrados por Lee e Sung (2008a) foram obtidos por meio de técnicas de programação dinâmica;
- Na Tabela 2, são mostrados o *gap* máximo de cada conjunto de instâncias e também uma comparação entre os resultados do presente trabalho e os apresentados por Lee e Sung (2008a);
- Na Tabela 3, é mostrado o número de instâncias em que o algoritmo conseguiu alcançar o valor $Fitness_{otimo}$;
- Na Tabela 4, é mostrado o número de instâncias em que ocorreu $Fitness_{encontrado} \leq 1.1 \, . \, Fitness_{otimo}$;
- Na Tabela 5, é mostrado 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.

Tabela 1. *Gap* médio usando o método proposto e os resultados encontrados por Lee e Sung (2008a).

n	Sem busca local			Com busca local			Por Lee e Sung (2008a) (%)
	$\beta = 0$ (%)	$\beta = 2,5$ (%)	$\beta = 5$ (%)	$\beta = 0$ (%)	$\beta = 2,5$ (%)	$\beta = 5$ (%)	
10	10,45	0,00	0,00	2,97	2,97	2,97	0,92
20	13,96	1,08	2,15	6,01	5,89	7,47	1,39
30	18,61	2,24	4,17	4,95	0,91	1,40	7,27
40	28,87	4,91	8,81	5,41	1,15	5,16	8,74
50	22,81	11,69	15,13	8,16	3,95	4,49	8,98
60	28,32	18,12	19,84	11,17	6,91	7,45	9,31
70	37,70	23,21	28,49	13,69	9,21	10,42	9,11

Tabela 2. *Gap* máximo encontrado pelo método proposto e por Lee e Sung (2008a).

n	Sem busca local			Com busca local			Por Lee e Sung (2008a) (%)
	$\beta = 0$ (%)	$\beta = 2,5$ (%)	$\beta = 5$ (%)	$\beta = 0$ (%)	$\beta = 2,5$ (%)	$\beta = 5$ (%)	
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

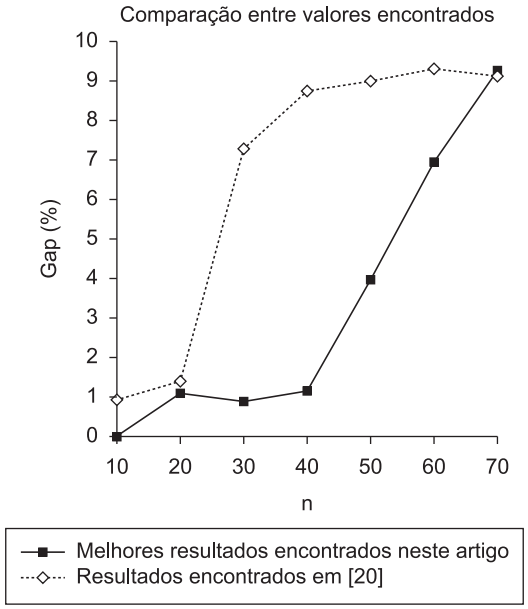

Figura 3. Comparação entre o melhor *gap* médio encontrado pelo método proposto e por Lee e Sung (2008a).

Vale notar que como não foi possível obter os exemplares dos problemas originais, é necessário cuidado ao realizar comparações sobre os dados apresentados tanto na Tabela 1 quanto na Tabela 2.

Analisando os dados apresentados nas Tabelas 1 e 2 e na Figura 3, pode-se observar o seguinte:

1) Para instâncias pequenas (n = {10, 20}), o algoritmo sem busca local e com valor de visibilidade β = {2,5;5} obteve 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 algoritomo que se utiliza deste procedimento;
2) Para instâncias médias e grandes (n = {30 – 70}), a busca local contribui para se encontrar a melhor solução. Para tais instâncias, os resultados obtidos usando a busca local são melhores que os resultados sem seu uso, independentemente do valor de β.
3) Ao se analisar o *gap* máximo encontrado, o algoritmo sem busca local gera resultados melhores que Lee e Sung (2008a) 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 (2008a).
4) Os resultados médios obtidos foram superiores aos citados por Lee e Sung (2008a) nas seguintes situações:
   - *n* = {10;20;30;40}, sem busca local e β = 2,5;
   - *n* = {10;30}, sem busca local e β = 5,0;
   - *n* = {30;40;50;60}, com busca local e β = 2,5;
   - *n* = {30;40;50;60}, com busca local e β = 5,0.
5) Para *n* = 70, os resultados encontrados pelo algoritmo com busca local e por Lee e Sung (2008a) são muito similares quando se observa o *gap* médio. Porém, o algoritmo proposto mostra valores melhores que Lee e Sung (2008a) quando se considera o *gap* máximo (Tabela 2).

Porém, apenas as análises do *gap* médio e máximo parecerem insuficientes, pois não permitem inferir a dispersão dos dados. Desta forma, visando analisar as aplicações práticas desta pesquisa, é apresentado nas Tabelas 3 e 4 quantas execuções atingiram 100% do valor ótimo e 90%, respectivamente.

Tabela 3. Número de resultados que alcançaram o valor ótimo.

n	Sem busca local			Com busca local		
	β = 0	β = 2,5	β = 5	β = 0	β = 2,5	β = 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. Número de resultados com variação de até 10% do valor ótimo.

n	Sem busca local			Com busca local		
	β = 0	β = 2,5	β = 5	β = 0	β = 2,5	β = 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 5. Tempo computacional médio para a execução do algoritmo proposto (segundos).

n	Sem busca local	Com busca local		
		β = 0	β = 2,5	β = 5
10	0,08	0,12	0,15	0,15
20	0,13	0,40	1,89	2,35
30	0,16	1,04	36,28	70,78
40	0,16	4,06	1,07	3,98
50	0,17	16,19	4,28	4,39
60	0,23	44,12	12,43	10,80
70	0,29	149,10	42,69	42,69

Quando se considera o número de resultados 100% precisos, percebe-se que dois conjuntos de dados definidos na análise da Tabela 1 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-se os valores apresentados na Tabela 4, 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 quando $\beta > 0$.

Desta forma, usando os dados apresentados nas Tabelas 1, 2 ,3 e 4, 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.

Na Tabela 5, foram encontrados 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, foi considerado que este algoritmo produziu uma solução viável.

6 Conclusões

Este artigo trata 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 ($1 / Budget / \delta . OC + (1 – \delta) \Sigma C_j$). Este problema, NP-difícil, proposto por Lee e Sung (2008a), foi resolvido com a proposta de um método baseado no algoritmo ACO. O método proposto incorpora cinco características específicas ao ACO: i) um algoritmo ACO composto de dois estágios, um primeiro estágio que busca reduzir o espaço de buscas e um segundo que identifica quais trabalhos devem ser terceirizados e quais devem ser realizados na máquina única; ii) uma representação em forma de grafo para problemas de *scheduling* que envolvam terceirização, resultado da aplicação do primeiro estágio do espaço de buscas; iii) uma regra de pré-seleção que garante a viabilidade da solução; iv) uma nova regra de visibilidade específica para o problema $1 / Budget / \delta . OC + (1 – \delta) \Sigma C_j$; e, por fim, (v) 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.

Dos dados gerados pelos experimentos, pode se verificar que as definições de visibilidade influenciaram positivamente o comportamento do algoritmo implementado. Sem o uso da estratégia de busca local e com um ajuste adequado dos parâmetros do algoritmo ($\beta = \{2,5;5,0\}$), os resultados encontrados foram melhores que aqueles encontrados por Lee e Sung (2008a) para problemas contendo 10 e 20 trabalhos. Para problemas contendo mais trabalhos, ($n = \{30;40;50;60\}$), 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 (2008a) 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 (2008a)

A estratégia de busca local proposta para o problema $1 / Budget / \delta . OC + (1 – \delta) \Sigma C_j$ 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 são tratados alguns problemas contendo 70 trabalhos, o tempo computacional requerido é viável.

Adicionalmete, 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. Entende-se, porém, que trabalhos futuros sobre esse tema devem abordar problemas com maiores dimensões, para verificar se essa característica do algoritmo ACO se mantém.

Concluindo, percebeu-se que os resultados obtidos das instâncias de teste indicam que o ACO pode ser usado para a resolução de problemas de *scheduling* que envolvam terceirização. Os resultados obtidos dão possibilidades de pesquisas futuras, como o estudo das propriedades do problema $1 / Budget / \delta . OC + (1 – \delta) . \Sigma C_j$, investigação de variações da função de visibilidade η e o desenvolvimento de novas técnicas de busca local para problemas de *scheduling* que envolvam terceirização.

Agradecimentos

Os autores agradecem à FAPESP por todo apoio dado na elaboração desse artigo

Referências

AHMADIZAR, F.; BARZINPOUR, F.; ARKAT, J. Solving permutation flow shop sequencing using ant colony optimization. In: IEEE INTERNATIONAL CONFERENCE ON. INDUSTRIAL ENGINEERING AND ENGINEERING MANAGEMENT, 2007.

Proceedings... Singapore, 2007, p. 753-757. http://dx.doi.org/10.1109/IEEM.2007.4419291

ARNAOUT, J.; MUSA, R.; RABADI, G. Ant colony optimization algorithm to parallel machine scheduling problem with setups. In: IEEE INTERNATIONAL CONFERENCE ON. AUTOMATION SCIENCE AND ENGINEERING, 2008, 4., 2008, Arlington. **Proceedings**... p. 578-582. http://dx.doi.org/10.1109/COASE.2008.4626566

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

BAUER, A. et al. Minimizing total tardiness on a single machine using ant colony optimization. **Central European Journal of Operations Research**, v. 8, p. 125-141, 2000.

BLAZEWICZ, J. et al. **Handbook of scheduling**: from theory to applications. Düsseldorf: Springer-Verlag, 2007.

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

COLORNI, A.; DORIGO, M.; MANIEZZO, V. Distributed optimization by ant colonies. In: EUROPEAN CONFERENCE OF ARTIFICIAL LIFE, 1991, Paris. **Proceedings**... Paris: Elsevier Publishing, 1992. p. 134-142.

CORDON, O. et al. A new aco model integrating evolutionary computation concepts: the best-worst ant system. In: INTERNATIONAL WORKSHOP ON ANT ALGORITHMS, 2000, 2., 2000, Bruxelas, Bélgica. **Proceedings**... p. 8-9.

DORIGO, M.; BLUM, C. Ant colony optimization theory: a survey. **Theoretical** , v. 344, n. 2-3, p. 243-278, 2005. http://dx.doi.org/10.1016/j.tcs.2005.05.020

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:** Cybernetics, v. 26, p. 29-41, 1996. PMid:18263004. http://dx.doi.org/10.1109/3477.484436

DORIGO, M.; STUTZLE, T. **Ant colony optimization**. MIT Press: 2004. http://dx.doi.org/10.1007/b99492

GAMBARDELLA, L. M.; DORIGO, M. Solving symmetric and asymmetric TSPS by ant colonies. In: EVOLUTIONARY COMPUTATION, 1996, Nagoya, Japan. **Proceedings**... Nagoya, Japan: 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. http://dx.doi.org/10.1287/ijoc.12.3.237.12636

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. http://dx.doi.org/10.1057/palgrave.jors.2601906

LEE, I. S.; SUNG, C. S. Single machine scheduling with outsourcing allowed. **International Journal of Production Economics**, v. 111, p. 623-634, 2008a. http://dx.doi.org/10.1016/j.ijpe.2007.02.036

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, 2008b. http://dx.doi.org/10.1016/j.ejor.2007.02.015

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. http://dx.doi.org/10.1016/S0360-8352(02)00079-7

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. http://dx.doi.org/10.1016/j.jmatprotec.2008.03.013

MERKLE, D.; MIDDENDORF, M. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In: REAL-WORLD APPLICATIONS OF EVOLUTIONARY COMPUTING, EVOWORKSHOPS, 2000, Berlin. **Proceedings**... Berlin: Springer-Verlag, p. 287-296, 2000.

QI, X. Two-stage supply chain scheduling with an option of outsourcing in stage one. In: INTERNATIONAL CONFERENCE ON. SERVICE SYSTEMS AND SERVICE MANAGEMENT, 2008, Melbourne, **Proceedings**... 2008. p. 1-6. http://dx.doi.org/10.1109/ICSSSM.2008.4598552

STOVBA, S. D. Ant algorthms: theory and applications. **Programming and Computer Software**, v. 31, n. 4, p. 167-178, 2005. http://dx.doi.org/10.1007/s11086-005-0029-1

STÜTZLE, T.; HOOS, H. H. Max-min ant system. **Future Generation Computer Systems,** v. 16, n. 9, p. 889–914, 2000.

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. http://dx.doi.org/10.1016/j.ijpe.2007.11.011

ZHOU, R.; LEE, H. P.; NEE, A. Y. C. Applying ant colony optimization (aco) algorithm to dynamic job shop scheduling problems. **International Journal of Manufacturing Research**, v. 3, n. 3, p. 301-320, 2008. http://dx.doi.org/10.1504/IJMR.2008.019212

ZHUO, X.; ZHANG, J.; CHEN, W. A new pheromone design in ACS for solving JSP. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION, Singapore, 2007. **Proceedings**... p. 1963-1969. http://dx.doi.org/10.1109/CEC.2007.4424714