Algoritmos para minimização do *makespan* em ambiente de máquina única com tempos de *setup* dependentes da sequência

Giuliano Frascati
Universidade Federal de São Carlos
São Carlos - SP
giuliano@dep.ufscar.br

Gabriel Fujiwara
Universidade Federal de São Carlos
São Carlos - SP
gabrielfugi@hotmail.com

Michele Amaral
Universidade Federal de São Carlos
São Carlos - SP
miicheleamaral@gmail.com

Roberto F. Tavares Neto
Universidade Federal de São Carlos
São Carlos - SP
tavares@dep.ufscar.br

RESUMO

Problemas de *scheduling* em ambiente de máquina única com *setup* dependente da sequência frequentemente são alvo de pesquisas que buscam algoritmos velozes. Tratando-se de um problema NP-Difícil, os custos computacionais das soluções desenvolvidas usando métodos exatos conhecidos são muito altos, o que impossibilita sua aplicação prática em caso reais de larga escala. Este projeto tem por objetivo implementar duas soluções velozes para o problema, uma heurística e a meta-heurística baseada em otimização por colônia de formigas. Uma formulação exata, baseada em programação inteira mista, será aplicada para validação dos algoritmos estocásticos implementados. Heurísticas e meta-heurísticas representam meios eficientes e flexíveis, com baixo custo computacional, capazes de atingir respostas com alta qualidade. Os resultados demonstram que, para problemas de pequena escala, o algoritmo exato apresenta dificuldades em comprovar respostas ótimas para problemas de 12 tarefas. Enquanto as soluções estocásticas demonstram alto potencial para encontrar boas respostas com baixíssimo tempo de computação.

PALAVRAS CHAVE. Pesquisa Operacional, *Scheduling*, Heurísticas, Meta-heurísticas.

PM – Programação Matemática; IND – PO na Indústria; MH – Meta-heurísticas

ABSTRACT

The scheduling problem in a single machine environment with sequence-dependent setup times has been the subject of researchers aiming for fast algorithms. Being a NP-Hard problem, the computational costs of solutions developed using known exact methods are very high, which makes it impossible to apply them in real large-scale cases. This project aims to implement fast solutions to the problem, a heuristic and the meta-heuristic of optimization by ant colony. An exact

formulation, based on mixed integer programming, will be applied for validation of implemented stochastic algorithms. Heuristics and meta-heuristics represent efficient and flexible methods, with low computational cost, capable of reaching high quality responses. The results show that for small scale problems, the exact algorithm presents difficulties in proving optimal answers to problems of 12 tasks. While stochastic solutions demonstrate high potential to find good answers with very low computing time.

KEYWORDS. Operational Research, Scheduling, Heuristics, Meta-heuristics.

MP - Mathematical Programming; IND - PO in Industry; MH - Meta-Heuristics

1. Introdução

Diante da atual disponibilidade e rapidez de informações, inserir-se ou manter-se no mercado competitivo é um grande desafio para o planejamento estratégico de qualquer organização. Portanto, muitas delas elaboram modelos matemáticos para auxiliar na tomada de decisão, visando a minimização de um ou mais objetivos de desempenho. Diversos autores como Sipper e Bulfin Jr. [1997] , Slack et al. [2015] e Velez-Gallego et al. [2016] mencionam que investimentos em otimização estão relacionados com melhorias na função de programação da produção, demonstrando um grande potencial para a redução de custos e inclusive tempo para produção de seus produtos.

Problemas de *scheduling* se incluem nos temas de estudo da Pesquisa Operacional. Ambientes produtivos determinam o fluxo da produção, e podem ser definidos em quatro grande grupos: máquina única, máquinas paralelas, *flowshop* ou *jobshop* de acordo com o fluxo que as ordens das tarefas a serem executadas devem percorrer para estarem completas [Pinedo, 2012].

Este trabalho trata do problema que envolve o *scheduling* para minimização do *makespan* em um ambiente de máquina única, cujos tempos de preparação dependem da sequência de operações a serem realizadas.

No caso de um grupo considerável de problemas de *scheduling*, o custo computacional de se aplicar alguns métodos exatos é proibitivo, requerendo o uso de estratégias alternativas de solução. A literatura sugere que muitas soluções aproximadas foram encontradas com sucesso por heurísticas (ex. Geiger [2010]). Tais métodos podem ser aplicados, por exemplo, para resolver problemas de sequenciamento e roteamento de veículos. Heurísticas são utilizadas por oferecer rapidamente uma solução de alta qualidade em relação aos objetivos otimizados, porém não necessariamente as respostas ótimas [Fernandes e Godinho Filho, 2010].

Outra estratégia que vêm ganhando destaque para a resolução de problemas NP-Difíceis são as meta-heurísticas, como mostrado em Tavares Neto [2010]. O ACO (*Ant Colony Optimization*) é uma meta-heurística que vem sendo amplamente utilizada para a resolução de problemas combinatórios complexos, por exemplo, Xu et al. [2008] que propõem uma variação do ACO para resolver o sequenciamento em ambiente de máquina única a partir da comparação com *benchmarks* de outros algoritmos que resolvem o mesmo problema. Já Cheng et al. [2008] propõe uma estratégia caótica ao ACO para resolver o problema de máquina única, mas para produção de lotes não idênticos, comprovando a eficiência do ACO ao comparar suas respostas do duas outras metas-heurísticas.

Para a presente pesquisa, foi desenvolvida uma heurística inspirado no algoritmo NEH (originalmente proposta por Nawaz et al. [1983], ver também Lu e Ying [2014], Li [2012] e Fernandes e Godinho Filho [2010], entre outros). Este método representa uma opção capaz de fornecer soluções próximas às ótimas para o problema de sequenciamento em um ambiente de *flowshop*, onde todas as tarefas devem passar por todas as máquinas, na mesma ordem. Apesar de ser apresentada originalmente como um método de solução para problemas em *flowshop*, adaptou-se essa estratégia para o problema aqui tratado, ou seja, para o ambiente de máquina única com *setup* dependente. Além disso, foi desenvolvida uma técnica de busca local, com o intuito de melhorar as respostas obtidas pela NEH adaptada.

Um segundo algoritmo baseado na meta-heurística ACO também foi implementado. De acordo com Dorigo e Stützle [2002], o ACO depende fortemente da parametrização dos valores aplicados aos parâmetros ajustáveis que definem sua operação. Para tal, a parametrização da meta-heurística implementada foi conduzida automaticamente por um software (IRACE de López-Ibáñez et al. [2016]) para este propósito específico.

Os resultados encontrados pelos algoritmos estocásticos foram comparados aos de um método exato, avaliando-se tanto a qualidade das respostas obtidas quanto a velocidade dos métodos. A seguir é apresentada a formulação matemática para o problema. A seção 3 revisa trabalhos semelhantes encontrados na literatura e descreve a implementação dos métodos rápidos propostos. Na seção 4 estão definidos os experimentos que compuseram este trabalho e na seção 5 os resultados

das simulações são analisados. Por fim, a seção 6 faz as considerações finais e direciona pesquisas futuras.

2. Formulação do Problema

Este trabalho trata um problema NP-difícil, como evidenciado por Thiruvady et al. [2009]. A formulação envolve o sequenciamento de tarefas e a minimização do *makespan*, em um ambiente de máquina única. O modelo tem como restrição os tempos de preparação entre as tarefas, que dependem da sequência na qual as operações serão realizadas. Formalmente, a tabela 1 define as variáveis utilizadas para a formulação dos algoritmos analisados adiante.

Tabela 1: Variáveis do Modelo

Variável	Descrição
n	Número de tarefas a serem sequenciadas
C_j	Tempo de término, para $j = 1...j$
p_j	Tempo de processamento, para j = 1...j
s_{ij}	Tempo de preparação para execução da tarefa j imediatamente após a tarefa i, para $i, j = 1...n$
x_{ij}	Variáveis de decisões binárias para determinar que uma tarefa j é sequenciada imediatamente após a tarefa i, para $i, j = 1...n$
G	Valor muito grande (usado para restrições do tipo "Big-M")

2.1. Modelo de Programação Inteira Mista

O modelo MIP (*Mixed Integer Programing*) tem como objetivo minimizar o tempo para conclusão de todas as tarefas (*makespan*). O objetivo do problema é: $min(C_{max})$. Assim, as equações 1 estabelecem qual o valor máximo no vetor C_j.

$$C_{max} \geq C_j \qquad \forall j \in (0 < j \leq n) \tag{1}$$

As equações 2 definem o tempo de términos das tarefas (C_j), elas consideram o tempo de término da tarefa anterior, o tempo de preparação entre as duas e o tempo de processamento da tarefa que está sendo processada. A equação 3 definem G, que é um número sempre maior que C_j utilizado para a formulação inteira mista. A equação 4 define que o início da sequência deve ser zero, pois não há nenhuma tarefa precedente.

$$C_j \geq C_i + p_j + s_{ij} - G \cdot (1 - x_{ij}) \qquad \forall \begin{Bmatrix} \{i, j\} \in (0 \leq i \leq n, 0 < j \leq n) \\ i \neq j \end{Bmatrix} \tag{2}$$

$$G = \sum_j (p_j + \sum_i s_{ij}) \tag{3}$$

$$C_0 = 0 \tag{4}$$

As equações 5 e 6 restringem a posição do sequenciamento para cada uma das tarefas, juntas essas equações formam uma restrição do fluxo de rede. Estas equações, com a somatória das linhas e colunas, estão presentes para garantir que as tarefas sejam sequenciadas uma única vez. As equações 7 e 8 garantem o balanço da sequência e evitam a formação de ciclos fechados.

$$\sum_i x_{ij} = 1 \qquad \forall j \in (0 \leq j \leq n) \tag{5}$$

$$\sum_j x_{ij} = 1 \qquad \forall j \in (0 \leq j \leq n) \tag{6}$$

$$\sum_i x_{ki} = \sum_j x_{jk} \qquad \forall k \in (0 \leq k \leq n) \tag{7}$$

$$x_{ii} = 0 \qquad \forall i \in (0 \leq i \leq n) \tag{8}$$

Além disso, as equações 9 definem x_{ij}, uma variável binária de decisão determina que a tarefa j é imediatamente precedida pela tarefa i. As equações 10 garantem que os tempos de processamento e preparação não sejam negativos.

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in (0 \leq i \leq n, 0 \leq j \leq n) \tag{9}$$

$$p_j, s_{ij} \geq 0 \qquad \forall i,j \in (0 \leq i \leq n, 0 \leq j \leq n) \tag{10}$$

3. Estratégias de solução

Inicialmente, implementou-se o modelo matemático (MIP), que serviu de base para análise do desempenho dos demais algoritmos. Com as respostas desse algoritmo dadas, os algoritmos apresentados a seguir foram implementados.

3.1. Algoritmo Baseado na NEH

A heurística NEH (Nawaz et al. [1983]) foi proposta inicialmente para problemas de minimização de *makespan* em ambientes *flowshop*. Seu comportamento se baseia em duas etapas principais: a primeira se utiliza de um algoritmo de ordenação para gerar uma sequência intermediária de tarefas; a segunda, se utiliza de um algoritmo de inserção para construir a sequencia final. No caso dos algoritmos a seguir, será usado o mesmo conceito, mesmo sabendo que o problema em questão adota um ambiente de manufatura composto por uma única máquina. Neste caso, determinou-se 4 (quatro) modelos de ordenação para as tarefas, utilizados pela heurística para iniciar seu processo de construção, como pode ser observado na tabela 2.

Tabela 2: Ordens prévias utilizadas

Nome	**Descrição**
FIFO	Segue a ordem de entrada das tarefas (*First In First Out*)
SPT	Ordena as tarefas de acordo com os tempos de processamento (p_j) em ordem crescente
SIJZERO	Ordena as tarefas de acordo com os tempos de *setup* apenas da tarefa inicial em ordem crescente
SIJSUM	Somam-se os tempos de *setup* para cada tarefa em suas possíveis posições sendo ordenadas em ordem crescente por esses somatórios

Tendo as políticas de ordenação já estabelecidas, o algoritmo 1 apresenta o pseudo-código da adaptação da técnica NEH implementado nesse artigo. Esta foi implementada a partir de cada uma das 4 ordens previamente estabelecidas. Por fim, foi desenvolvida uma técnica de busca local para melhoria das respostas geradas pela heurística. Uma troca de tarefas (*Swap*), como visto em Gorczyca et al. [2015]; Wu et al. [2009] e Wang e Wang [2012], trocando uma posição por outra, mas também em duplas e em trios de tarefas, pode ser uma alternativa de otimização. No presente caso, a técnica utilizada retira aleatoriamente 4 tarefas da resposta obtida pelo NEH adaptado e insere essas tarefas nas posições que permitem o menor tempo de preparação disponível. Cada instância foi solucionada 20 vezes por esse método. As oito variações dessa heurística base são apresentadas na tabela 3.

Algoritmo NEH
Ordene as tarefas em um ordem específica (FIFO, SPT, SIJZERO ou SIJSUM)
Selecione as duas primeiras tarefas (j_1 e j_2) e ordene-as de forma a obter o menor *makespan*
para cada *tarefa restante* ($j_3, ..., j_n$) **faça**
    Fixe a ordem das tarefas previamente sequenciadas e insira a tarefa na posição da sequência que resulta no menor *makespan* parcial possível entre todas as possibilidades de posições para inserção
fim
fim Algoritmo NEH

Algoritmo 1: NEH. Adaptado de Nawaz et al. [1983]

Tabela 3: Algoritmos comparados

Algoritmo	Descrição
NEH_1	NEH adaptado + ordem 0
NEH_2	NEH adaptado + ordem 1
NEH_3	NEH adaptado + ordem 2
NEH_4	NEH adaptado + ordem 3
NEH_5	NEH adaptado + ordem 0 + busca local
NEH_6	NEH adaptado + ordem 1 + busca local
NEH_7	NEH adaptado + ordem 2 + busca local
NEH_8	NEH adaptado + ordem 3 + busca local

3.2. Algoritmo Baseado em Colônia de Formigas

Segundo Dorigo e Stützle [2002], a otimização por colônica de formigas (do inglês, *Ant Colony Optimization* ACO) é baseado na comunicação indireta (através de feromônios artificiais) por um conjunto de agentes computacionais (formigas). Diversos autores vem usando esse tipo de estratégia para resolução de problemas complexos, como Madureira et al. [2012] que demonstram a aplicabilidade do ACO para o problema de *scheduling* da minimização do atraso total ponderado em ambiente de máquina única. Eles combinaram o ACO com um algoritmo de busca local para obter melhores resultados.

Os autores em M'Hallah e Alhajraf [2015], propõem um sistema baseado em ACO com um número reduzido de formigas e colônias, implementando também com ações de refinamento, que exploram o espaço em torno das soluções geradas usando uma pesquisa de vizinhança variável (VNS). A meta-heurística ACO implementada neste projeto segue duas funções básicas apresentadas no algoritmo 2: a construção das respostas se utiliza de uma função de probabilidade que se utiliza de uma regra de transição. A comunicação indireta ocorre através de uma estratégia de deposição e evaporação dos feromônios artificiais.

Meta-heurística ACO para *Número de Iterações da Meta-Heurística* **faça**
    ConstruçãoDeRespostas()
    AtualizaçãoDeFeromônios()
    RefinamentoDeRespostas() [Opcional]
fim
fim Meta-heurística ACO

Algoritmo 2: Pseudocódigo ACO. Fonte: adaptado de Dorigo e Stützle [2002]

Segundo Gambardella e Dorigo [1996] a regra de transição é definida através de dois fatores: as informações heurísticas do problema na forma da função de visibilidade (η); e uma matriz (τ) que armazena as trilhas artificiais de feromônio. Para a atualização dos níveis de feromônio artificial esse autores apresentam duas regras: i) uma de atualização local, realizada por cada formiga (agente) da colônia; e ii) uma regra de atualização global a partir dos resultados de agentes

anteriores. Sendo essas regras, responsáveis por atualizar informações sobre as soluções exploradas na região factível.

As equações 11 e 12 definem a regra de visibilidade que compõe a regra de transição. Essas informações constituem o componente heurístico do algoritmo, que no caso deste trabalho são baseados nos dados de entrada: p_j e s_{ij}. Essas informações são ponderadas por fatores que determinam um viés para a construção de soluções. Além disso, η_{ij} assume zero caso a tarefa já tenha sido sequenciada.

$$\eta_{ij} = \frac{1}{p_j^{\alpha_1}} \cdot \frac{1}{s_{ij}^{\alpha_2}} \qquad \forall i,j \in (0 \leq i \leq n, 0 \leq j \leq n) \tag{11}$$

$$P_{ij} = \frac{\tau_{ij} \cdot (\eta_{ij})^{\beta}}{\sum_{j'} \tau_{ij} \cdot (\eta_{ij'})^{\beta}} \qquad \forall i,j \in (0 \leq i \leq n, 0 \leq j \leq n) \tag{12}$$

Um fator muito importante para a boa execução do algoritmo é a determinação dos parâmetros a serem utilizados no ACO. Neste projeto de pesquisa o algoritmo baseado em ACO possuí configurações ajustáveis e fixas. Os parâmetros ajustáveis são compostos pelo nível máximo (ω_u) e mínimo (ω_l) de feromônio, taxa de deposição (ρ_d) e evaporação (ρ_e) de feromônios artificiais e os expoentes α_1, α_2 e β. A determinação deles foi feita de forma automatizada pelo *software* IRACE, de López-Ibáñez et al. [2016].

Os parâmetros que controlam o esforço computacional do algoritmo foram fixados. O ACO foi implementado com bonificação elitista, sendo assim um baixo número de agentes (5) foi aplicado e experimentos preliminares indicaram que ocorre convergência em 150 iterações, logo para garantir uma maior precisão o número de ciclos utilizado foi de 250 gerações.

A segunda coluna na tabela 4 apresenta os valores sugeridos pela literatura para esse tipo de problema, e a partir de um experimento com o software IRACE foram extraídas três configurações do ACO para avaliação, determinadas ACO_1, ACO_2 e ACO_3 [López-Ibáñez et al., 2016].

Tabela 4: Parâmetros para a Meta-Heurística ACO

Parâmetros	**Domínio**	**ACO_1**	**ACO_2**	**ACO_3**
ω_l	{1; 5; 10; 15}	10	10	15
ω_u	{15; 20; 25; 30}	25	25	30
ρ_d	{2; 5; 10}	10	10	5
ρ_e	{0,8; 0,9; 0,95; 0,99}	0,9	0,95	0,99
α_1	{1; 2; 3}	3	3	1
α_2	{1; 2; 3}	2	2	2
β	{1; 2; 3; 5}	5	2	5

4. Experimentos Computacionais

Primeiramente, implementou-se um algoritmo exato MIP, solucionado com a biblioteca CPLEX para Python, versão 12.6.2, a fim de obter o máximo de respostas ótimas possível. Nesse caso, foi-se incluída a restrição de tempo de 3600s (1h). Os algoritmos heurísticos descritos foram implementados em Python 3.4. Um computador com 8 núcleos, modelo Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, foi utilizado para os experimentos.

4.1. Instâncias

Como apontado por Rabanal et al. [2015], a avaliação e validação algoritmos estocásticos, como heurísticas e meta-heurísticas, deve ocorrem sobre amplos conjuntos de instâncias estatisticamente bem distribuídas no domínio possível para as variáveis de entrada do problema. De acordo com Wolpert e Macready [1997], configurar algoritmos estocásticos para conjuntos limitados de instâncias é relativamente simples e potencializa as chances de falha do algoritmo resultante. Os autores em Rabanal et al. [2015] encorajam fortemente utilizar geradores de instâncias aleatórias.

Portanto, apensar de existirem conjunto limitados para o problema tratado, um gerador de instâncias foi utilizado para criar os casos de teste.

Neste trabalho, as instâncias testes foram geradas a partir da análise de relação entre as distribuições uniformes dos tempo de processamento (p_j) e tempo de *setup* (s_{ij}), e estes são dependentes da alocação dos trabalhos. Tem-se que s_{ij} e p_j são variáveis aleatórias e os seus valores obedecem uma distribuições uniformes, com valor mínimo de 1 e máximo de 10, 25, 50 ou 100. Os problemas em que s_{ij} pode assumir valores muito maiores que p_j foram descartados, totalizando 13 níveis de problemas teste. Como o mínimo das distribuições é constante para todos os níveis e assume o valor 1 (um), a tabela 5 mostra o valor máximo para as distribuições que definem cada nível para geração aleatória de instâncias.

Tabela 5: Valor máximo das distribuições uniformes para os tempos de processamento e *setup*

Nível	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
T. *Processamento*	100	100	100	100	50	50	50	50	25	25	25	10	10
T. *Setup*	10	25	50	100	10	25	50	100	10	25	50	10	25

Os experimentos foram compostos por 390 instâncias de teste para cada tamanho de problema avaliado, e foram gerados 30 para cada nível. Este experimento foi composto de instâncias com pequeno para as quais foi possível obter respostas com o CPLEX dentro da restrição de tempo estabelecida. Foram criadas instâncias de pequeno porte entre os tamanhos de 8 e 12 tarefas.

5. Análise de Resultados

Nas versões do algoritmo adaptado NEH, os tempos foram consideravelmente menores em relação ao MIP, estando, em sua maioria, próximos à média referente a cada tamanho de instância. Os tempos do algoritmo NEH são extremamente baixos, poucos milissegundos, como pode ser visto nas tabelas 6, 7 e 8.

Os resultados mostram que o CPLEX não foi capaz de completar a sua execução dentro de uma hora para 333 problemas de 12 tarefas. Apesar disso, nenhum dos algoritmos retornou respostas menores para este problema de minimização nestes casos. Portanto, para as 333 instâncias onde o CPLEX atingiu o limite de execução as respostas foram consideradas ótimas e assim todas as instâncias foram comparadas em conjunto.

Os tempos de execução da heurística implementada são insignificantes. Assim foram implementados métodos para melhoramento de resposta que também possuem baixa complexidade computacional. Para a heurística aplicada sem melhoramento todas as instâncias foram solucionadas em no máximo 11 milissegundos e aplicando a busca local para melhoramento nenhuma simulação tomou mais de 44 milissegundos.

Tabela 6: Análise de tempos das versões do algoritmo NEH adaptado – 8 e 9 tarefas

	Tempos para 8 tarefas			**Tempos para 9 tarefas**		
Algoritmo	**Mínimo (ms)**	**Médio (ms)**	**Máximo (ms)**	**Mínimo (ms)**	**Médio (ms)**	**Máximo (ms)**
CPLEX	216	890,06	1707	822	6149,61	13398
NEH_1	1	1,02	2	1	1,02	2
NEH_2	1	1,07	4	1	1,93	4
NEH_3	1	1,12	3	1	1,99	4
NEH_4	1	1,24	4	1	2,00	4
NEH_5	8	8,99	19	10	11,26	13
NEH_6	8	9,19	11	11	11,84	21
NEH_7	9	9,48	20	11	12,90	25
NEH_8	9	10,35	20	11	12,87	26
ACO_1	712	765,75	1120	874	937,92	1003
ACO_2	730	792,11	886	876	952,44	1064
ACO_3	737	792,23	914	875	952,15	1023

Tabela 7: Análise de tempos das versões do algoritmo NEH adaptado – 10 e 11 tarefas

	Tempos para 10 tarefas			**Tempos para 11 tarefas**		
Algoritmo	**Mínimo (ms)**	**Médio (ms)**	**Máximo (ms)**	**Mínimo (ms)**	**Médio (ms)**	**Máximo (ms)**
CPLEX	6085	73474,52	172669	48977	966155,4	2444613
NEH_1	1	1,24	3	2	2,12	4
NEH_2	2	2,18	5	2	2,60	6
NEH_3	2	2,17	6	2	2,46	6
NEH_4	2	2,23	4	2	2,61	5
NEH_5	13	14,09	28	16	18,15	35
NEH_6	13	15,99	31	16	18,88	37
NEH_7	13	15,70	32	16	17,58	31
NEH_8	13	14,76	29	16	17,36	34
ACO_1	1051	1130,38	1447	1241	1337,7	1432
ACO_2	1053	1137,47	1485	1242	1342,18	1457
ACO_3	1056	1136,85	1324	1247	1346,96	1487

Tabela 8: Análise de tempos das versões do algoritmo NEH adaptado – 12 tarefas

Algoritmo	**Tempo mínimo (ms)**	**Tempo médio (ms)**	**Tempo máximo (ms)**
CPLEX	800355	3401332,32	3600835
NEH_1	2	2,19	4
NEH_2	2	3,03	6
NEH_3	2	3,08	11
NEH_4	2	3,07	9
NEH_5	19	21,59	42
NEH_6	19	21,29	44
NEH_7	19	20,72	26
NEH_8	19	20,58	27
ACO_1	1455	1608,48	1848
ACO_2	1449	1566,56	2022
ACO_3	1448	1569,38	1752

As tabelas 6, 7 e 8 apresentam também os tempos de execução do ACO. Nota-se que a tempo do ACO é muito superior ao tempo da heurística, mas ainda assim o maior tempo de execução encontrado foi de 2022 milissegundos.

Quanto à qualidade das respostas, foram utilizadas as soluções ótimas encontradas pelo MIP como base de comparação para os demais algoritmos. As 8 variações da heurística proposta e três configurações distintas para o ACO foram comparadas. Os resultados, presentes nas tabelas 9, 10 e 11 mostram a resposta média encontrada por cada algoritmo, a porcentagem de respostas ótimas e o *Gap* médio para cada conjunto de instâncias. O *Gap* indica a diferença percentual entre as duas soluções e é calculado da seguinte maneira: $[Gap = \frac{RespostadoAlgoritmo - Respostatima}{Respostatima} \times 100]$

Para a heurística, apesar do percentual de respostas ótimas encontradas não ser alto, o *gap* médio é baixo, mostrando que as soluções encontradas pelas versões do NEH adaptado são próximas às ótimas. Pode-se observar que o uso das heurísticas com busca local aumenta significativamente a qualidade da solução do problema, dobrando o percentual de respostas ótimas e reduzindo ainda mais o *gap* médio. Ao observar os tempos computacionais envolvidos, é razoável dizer que o ganho na qualidade da resposta é resultado da alocação de maior tempo para execução.

Todas as diferentes configurações da meta-heurística ACO apresentaram resultados superiores as versões do NEH em relação a qualidade de resposta, tanto quanto a porcentagem de respostas ótimas como no *gap* observados para os resultados não ótimos.

Tabela 9: Análise da qualidade de respostas para 8 e 9 tarefas

	8 tarefas			**9 tarefas**		
Algoritmo	**Resp. Média**	**Resp. Ótimas**	***Gap* Médio**	**Resp. Média**	**Resp. Ótimas**	***Gap* Médio**
CPLEX/MIP	271,08	100 %	-	296,01	100 %	-
NEH_1	287,48	7,18 %	7,70 %	316,17	5,13 %	8,11 %
NEH_2	286,27	9,23 %	7,24 %	314,53	5,13 %	7,53 %
NEH_3	288,16	7,95 %	7,86 %	316,53	3,33 %	8,29 %
NEH_4	287,79	6,15 %	7,76 %	315,11	2,82 %	7,80 %
NEH_5	282,03	16,41 %	5,64 %	308,97	12,82 %	5,57 %
NEH_6	280,29	18,72 %	4,78 %	308,39	10,00 %	5,40 %
NEH_7	281,53	20,26 %	5,69 %	309,05	8,97 %	5,58 %
NEH_8	281,88	15,38 %	5,60 %	309,19	7,69 %	5,64 %
ACO_1	273,38	63,59 %	2,56 %	299,05	47,95 %	2,22 %
ACO_2	273,41	63,33 %	2,42 %	299,26	50,51 %	2,41 %
ACO_3	273,60	63,85 %	2,72 %	299,11	46,41 %	2,20 %

Tabela 10: Análise da qualidade de respostas para 10 e 11 tarefas

	10 tarefas			**11 tarefas**		
Algoritmo	**Resp. Média**	**Resp. Ótimas**	***Gap* Médio**	**Resp. Média**	**Resp. Ótimas**	***Gap* Médio**
CPLEX/MIP	323,18	100 %	-	354,44	100 %	-
NEH_1	345,54	2,82 %	8,22 %	378,84	1,54 %	8,27 %
NEH_2	344,00	4,36 %	7,98 %	378,25	0,26 %	8,22 %
NEH_3	345,55	3,59 %	8,46 %	380,15	1,28 %	8,40 %
NEH_4	345,19	2,31 %	8,24 %	381,94	1,03 %	9,05 %
NEH_5	338,52	6,41 %	5,89 %	372,39	2,82 %	6,14 %
NEH_6	337,30	9,23 %	5,73 %	372,42	2,56 %	6,24 %
NEH_7	337,91	7,69 %	6,12 %	372,49	3,85 %	6,07 %
NEH_8	338,70	6,67 %	6,08 %	373,15	3,85 %	6,40 %
ACO_1	328,56	28,46 %	2,57 %	362,40	14,36 %	2,90 %
ACO_2	328,67	29,49 %	2,69 %	362,25	15,90 %	2,96 %
ACO_3	328,50	32,05 %	2,74 %	361,84	16,92 %	2,87 %

Tabela 11: Análise da qualidade de respostas para 12 tarefas (390 instâncias)

12 tarefas			
Algoritmo	**Resposta Média**	**Respostas Ótimas**	***Gap* Médio**
CPLEX/MIP	381,64	100 %	-
NEH_1	410,04	0,77 %	8,71 %
NEH_2	409,38	0,51 %	8,52 %
NEH_3	411,29	0,26 %	8,96 %
NEH_4	410,31	1,28 %	8,87 %
NEH_5	401,03	2,31 %	6,24 %
NEH_6	402,45	1,79 %	6,34 %
NEH_7	403,36	2,56 %	6,57 %
NEH_8	401,53	3,59 %	6,38 %
ACO_1	392,33	7,95 %	3,47 %
ACO_2	392,37	7,69 %	3,46 %
ACO_3	392,42	6,67 %	3,49 %

Após a análise dos resultados apresentados, é possível fazer uma comparação entre os dois algoritmos de resposta rápida, concluindo que os tempos computacionais envolvidos do algoritmo baseado em ACO são um pouco maiores que o do algoritmo NEH adaptado, porém apresentam uma melhor qualidade de respostas. Fato que pode ser justificado pela maior alocação de recursos para execução.

6. Considerações Finais

O presente trabalho apresentou um conjunto de algoritmos simples para a minimização do *makespan* em ambiente de máquina única com tempos de *setup* dependentes da sequência. Após a realização dos testes computacionais foi possível concluir que o tempo de execução dos algoritmos propostos é muito melhor do que o tempo do modelo exato (MIP), que foi usado como base para validação dos mesmos.

A qualidade de respostas oferecida pelo ACO foi superior aos procedimentos construtivos de inserção. Entretanto, ambos os algoritmos se mostraram muito eficientes.

Foi possível comprovar a eficiência de ambas as estratégias estocásticas velozes, com respostas próximas às ótimas, comprovando que estes métodos são aplicáveis e interessantes para o meio empresarial, principalmente para problemas de curto prazo que necessitam de decisões rápidas.

Um dos próximos passos consiste em comparar os algoritmos desenvolvidos neste trabalho com outras heurísticas e meta-heurísticas para instâncias de larga escala deste mesmo problema de *scheduling*. Além disso, outros objetivos, como a minimização do atraso total, e outros ambientes produtivos, como máquinas paralelas, *flowshop* e *jobshop*, podem ser explorados.

Reconhecimento

Os autores gostariam de agradecer à organização brasileira de fomento CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e à FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) – processos 2016/05673-1 e 2016/05672-5 – por financiar esta pesquisa. Agradecendo também aos departamentos da UFSCar (Universidade Federal de São Carlos) pelo apoio para alcançar estes objetivos.

Referências

Cheng, B.-Y., Chen, H.-P., Shao, H., Xu, R., e Huang, G. Q. (2008). A chaotic ant colony optimization method for scheduling a single batch-processing machine with non-identical job sizes. *IEEE Congress on Evolutionary Computation*, p. 40–43.

Dorigo, M. e Stützle, T. (2002). *The ant colony optimization metaheuristic: Algorithms, applications, and advances. Handbook of Metaheuristics, p. 251-285*. Kluwer Academic Publishers.

Fernandes, F. e Godinho Filho, M. (2010). *Planejamento e controle da produção: dos fundamentos ao essencial*. Atlas.

Gambardella, L. M. e Dorigo, M. (1996). Solving symmetric and asymmetric tsps by ant colonies. *IEEE Press*.

Geiger, M. J. (2010). On heuristic search for the single machine total weighted tardiness problem: Some theoretical insights and their empirical verification. *European Journal of Operational Research*, 207:1235–1243.

Gorczyca, M., Janiak, A., Janiak, W., e Dymaski, M. (2015). Makespan optimization in a single-machine scheduling problem with dynamic job ready times - complexity and algorithms. volume 182, p. 162 – 168. URL `http://dx.doi.org/10.1016/j.dam.2013.10.003`.

Li, M., X.; Yin (2012). A discrete artificial bee colony algorithm with composite mutation strategies for permutation flow shop scheduling problem. *Scientia Iranica E*, 19 (6):1921–1935.

López-Ibáñez, M., Pérez Cáceres, L., Dubois-Lacoste, J., Stützle, T., e Birattari, M. (2016). The irace package: User guide. Technical Report TR/IRIDIA/2016-004, IRIDIA, Université Libre de Bruxelles, Belgium. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2016-004.pdf`.

Lu, S.-W., C-C; Lin e Ying, K.-C. (2014). Robust single machine scheduling for minimizing total flow time in the presence of uncertain processing times. *Computers & Operations Research*, 74: 102–110.

Madureira, A., Falcão, D., e Pereira, I. (2012). Ant colony system based approach to single machine scheduling problems. *World Congress on Nature and Biologically Inspired Computing, NaBIC 2012, p 86-91, 2012*.

M'Hallah, R. e Alhajraf, A. (2015). Ant colony systems for the single-machine total weighted earliness tardiness scheduling problem. *Journal of Scheduling, April 8, 2015*.

Nawaz, Enscore, e Ham (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11:91–95.

Pinedo, M. L. (2012). *Theory, Algorithms, and Systems*. Prentice-Hall.

Rabanal, P., Rodríguez, I., e Rubio, F. (2015). On the uselessness of finite benchmarks to assess evolutionary and swarm methods. In *GECCO 15*. DOI: http://dx.doi.org/10.1145/2739482.2764672.

Sipper, D. e Bulfin Jr., R. (1997). *Production: Planning, Control and Integration*. New York: Mc Graw Hill.

Slack, N., Chambers, S., e Johnson, R. (2015). *Administração da produção*. SÃ£o Paulo: Atlas, SÃ£o Paulo, 3 edition.

Tavares Neto, R. (2010). Proposta de solução de problemas de scheduling considerando possibilidade de terceirização usando a técnica de otimização por colônia de formigas. Master's thesis, Universidade Federal de SÃ£o Carlos.

Thiruvady, D., Blum, C., Meyer, B., e Ernst, A. (2009). Hybridizing beam-aco with constraint programming for single machine job scheduling. *Lecture Notes in Computer Science, Springer Link*, p. 30–44.

Velez-Gallego, M. C., Maya, J., e Montoya-Torres, J. R. (2016). A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan. *Computers and Operations Research*, 73:132 – 140. ISSN 03050548. URL `http://dx.doi.org/10.1016/j.cor.2016.04.009`.

Wang, M.-Z. e Wang, J.-B. (2012). Single-machine makespan minimization scheduling with nonlinear shortening processing times. *Computers and Operations Research*, 39(3):659 – 663. ISSN 03050548. URL `http://dx.doi.org/10.1016/j.cor.2011.05.001`.

Wolpert, D. H. e Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1.

Wu, C.-C., Shiau, Y.-R., Lee, L.-H., e Lee, W.-C. (2009). Scheduling deteriorating jobs to minimize the makespan on a single machine. *International Journal of Advanced Manufacturing Technology*, 44(11-12):1230 – 1236. ISSN 02683768. URL `http://dx.doi.org/10.1007/s00170-008-1924-4`.

Xu, R., Chen, H.-P., Huang, G. Q., Shao, H., Cheng, B.-Y., e Liu, B.-W. (2008). Minimizing the total completion time on a single batch processing machine with non-identical job sizesusing ant colony optimization. *Industrial Electronics and Applications*, p. 2211–2215.