DOI: 10.5585/ExactaEP.v15n4.6225

Estudo da influência da estimativa inicial na qualidade da solução no algoritmo genético binário em problemas de sequenciamento da produção em ambiente job shop

Study of initial guess influence on the quality of solutions on binary genetic algorithm in job shop scheduling problems

Valmir Ferreira da Cruz[1]

Fábio Henrique Pereira[2]

Resumo

A proposta deste trabalho é avaliar a influência da estimativa inicial, usada na geração de populações de soluções pelo Algoritmo Genético em problemas de sequenciamento, em relação à qualidade e à viabilidade das soluções. O problema de sequenciamento da produção consiste em encontrar a sequência da execução das tarefas no parque de máquinas que otimize, por exemplo, a utilização de recursos e o tempo total de processamento (makespan). É comum tratar problemas desta natureza com o uso de técnicas metaheuristicas de otimização como Algoritmo Genético principalmente em função da sua complexidade computacional. Neste trabalho foram realizados experimentos com os um conjunto de instâncias da literatura, variando a regra de sequenciamento utilizada na geração das soluções iniciais. Foram testadas regras usuais da literatura e identificada uma regra híbrida que gera um menor número de soluções não factíveis e um número maior de instâncias que atingiram o makespan ótimo.

Palavras-chave: Sequenciamento da produção. Job Shop. Algoritmo genético. Estimativa inicial.

Abstract

The purpose of this study is to evaluate the influence of the initial guess to generate the Genetic Algorithm population of solutions of scheduling problems in relation to the quality and feasibility of the solutions. The scheduling problem is defined as to find the sequence of operations on the machines that optimize some performance measure as, for example, the use of resources and the total processing time (makespan). It is common to treat such problems with the use of metaheuristics as genetic algorithm mainly due to its computational complexity. This work carried out experiments with a set of literature instances, varying the sequencing rule used in the generation of initial solutions. Usual rules from the literature have been tested and identified a hybrid rule that generates a smaller number of non-feasible solutions and the number of instances that have reached the optimal makespan.

Keywords: Scheduling. Job Shop. Genetic Algorithm. Initial Guess.

1 Programa de Mestrado e Doutorado em Engenharia de Produção na Universidade Nove de Julho – Uninove. São Paulo, SP [Brasil]
valmir.vfc@gmail.com

2 Programa de Mestrado e Doutorado em Informática, Gestão do Conhecimento e Engenharia de Produção na Universidade Nove de Julho – Uninove. São Paulo, SP [Brasil]
fabiohp@uni9.pro.br

1 Introdução

Empresas de todo o mundo buscam um aumento de eficiência em seus processos produtivos, visando produzir mais com os mesmos recursos e se tornarem cada vez mais competitivas, oferecendo produtos de qualidade a um custo cada vez menor. Essa eficiência pode ser alcançada com a otimização dos processos produtivos, com vistas a uma utilização racional de seus recursos disponíveis, para diminuir ao máximo a ociosidade e melhorar o aproveitamento de seu parque. Esta otimização tem sido objeto de estudo e pesquisa em várias áreas do conhecimento como, por exemplo, gestão empresarial, economia, logística, entre outros, e mais particularmente nos estudos da engenharia de produção, ou como também chamado engenharia industrial (Kunnathur et al., 2004; Heinonen & Pettersson, 2007).

Dentre os principais problemas da engenharia de produção, o problema de sequenciamento da produção vem sendo estudado atualmente por um grande número de pesquisadores (Kurdi, 2015; Asadzadeh, 2015; Amirghasemi & Zamani, 2015). Este problema consiste em encontrar qual a melhor sequência das operações a serem realizadas na linha de produção, buscando otimizar o tempo em cada máquina, de forma a reduzir ao máximo o tempo de ociosidade, bem como a melhor disposição das várias máquinas na produção, por exemplo. Segundo Bo Peng et. al (2014), o problema de agendamento do ambiente job shop (JSSP) é, além de um dos problemas mais notórios e intratáveis, um dos problemas de agendamento mais importantes que surgem em situações em que um conjunto de atividades, que seguem padrões de fluxo irregulares, devem ser realizados por um conjunto de recursos escassos.

O sequenciamento da produção, quando possui um número elevado de máquinas e/ou tarefas, exige muito tempo de processamento para avaliar todas as possíveis soluções, chegando a casos em que se torna impossível chegar à solução ótima em tempo computacional aceitável. Tal problema é classificado como NP-Difícil, que não é resolvido em tempo polinomial aceitável.

Problemas com este grau de complexidade permitem um número muito grande de combinações possíveis quando se faz o cruzamento entre jobs e máquinas. Segundo Lukaszewicz (2005), o espaço de soluções para um JSSP é formado por (n!)m sequências possíveis, sendo n o número de jobs e m o número de máquinas. Sendo assim, em um cenário com 10 jobs e 15 máquinas existem 2,49 * 1098 combinações possíveis. No entanto, muitas dessas combinações se apresentam como não factíveis, ou seja, não possíveis de serem executadas na prática.

Avaliar a viabilidade e o tempo total (makespan) de cada uma destas combinações à procura da melhor solução pode exigir um tempo computacional impraticável, podendo nem mesmo ser possível a conclusão da tarefa devido a recursos computacionais escassos. Assim, em geral são utilizadas técnicas que propõem a avaliação de um número limitado dessas combinações possíveis, com o intuito de selecionar apenas as melhores soluções dentre as avaliadas. Estas técnicas que se propõem a encontrar a solução subótima dentro de um tempo computacional aceitável, são chamadas de metaheurísticas.

Trabalhos com aplicação de metaheurísticas em problemas de sequenciamento são discutidos na próxima subseção. Na sequência do artigo é apresentada uma classificação dos ambientes de produção, com destaque para o ambiente job shop, os aspectos gerais do algoritmo genético, matérias e métodos e resultados e conclusões.

1.1 Aplicação de metaheurísticas em problemas de sequenciamento

Várias pesquisas que abordam o *JSSP* com a utilização de metaheurísticas vêm sendo publicadas ao longo dos anos. Gao et al. (2015) utilizaram a metaheuristica Colônia de Formigas para tratar o problema em remanufatura de engenharia; Saidi-Mehrabad et al. (2015) utilizaram a mesma metaheurística para tratar um problema de tempo de transporte entre as máquinas.

Como trabalhos recentes utilizando-se da metaheurística algoritmo genético no tratamento do *JSSP* pode-se citar: Asadzadeh (2015); Amirghasemi & Zamani (2015); Qing-dao-er-ji et al. (2013) e Kurdi (2015). O que ilustra o interesse da comunidade científica no uso dessa metaheurística. Nesse contexto, diversas abordagens têm sido desenvolvidas variando diversos parâmetros e operadores do AG, entre eles a representação da solução (Abdelmaguid, 2010; Grassi, 2014).

Abdelmaguid (2010) avaliou diferentes representações baseadas em números reais e lista de números inteiros. O autor destaca que as listas de números inteiros, que representam de forma direta uma solução para o *JSSP*, são as mais utilizadas, mas podem apresentar um elevado número de soluções não factíveis. Há, por outro lado, representações que produzem apenas soluções factíveis, mas que possuem deficiências em relação à geração de novas soluções diferentes das anteriores (Modolo, 2015).

O fato é que as diferentes representações produzem resultados diferentes no AG em problemas de sequenciamento, interferindo na geração de soluções factíveis. Nesse sentido, Grassi (2014) propôs um Algoritmo Genético com representação binária utilizando-se de um conceito de semente dinâmica (DSGA), com a proposta de redução do número de soluções não factíveis. No trabalho, o autor mostra que a representação proposta reduz a proporção de soluções não factíveis, em comparação ao AG tradicional com elitismo, desde que a semente inicial represente uma solução factível. Para garantir a factibilidade ds soluções o autor gera a semente inicial a partir de uma regra FIFO (*Fisrt In First Out*), mas não investiga quais os efeitos de gerar essa estimativa inicial com base em outras regras, em especial àquelas que, ao contrário da FIFO, consideram o tempo de processamento das tarefas no sequenciamento.

Portanto, o presente artigo tem como objetivo estudar os efeitos da estimativa inicial no algoritmo genético com representação binária na qualidade das soluções de sequenciamento, em relação à proporção de soluções não factíveis, o tempo de processamento e o valor de makespan obtido. São testadas regras de sequenciamento híbridas que consideram o tempo de processamento das tarefas além da ordem de chegada dessas tarefas: FIFO + SPT e FIFO + LPT (FIFO + *Shortest Processing Time* e FIFO + *Longest Processing Time*, respectivamente).

2 Cenários da produção e o problema no sequenciamento

Lukaszewicz (2005) definiu o sequenciamento da produção como o processo de atribuição de um ou mais recursos para a execução de certas atividades as quais, em sua execução, vão exigir certa quantidade de tempo. Em um ambiente industrial as máquinas podem representar os recursos, e as tarefas são as atividades que são processadas em cada máquina. Portanto, um *job* é um conjunto de uma ou mais tarefas.

De acordo com a forma como os *jobs* são distribuídos para diferentes máquinas e se os mesmos são diferentes entre si, os cenários de produção recebem diferentes classificações (Allahverdi et al., 2008).

Segundo (Graham et al., 1979; Pinedo, 2008), os ambientes de produção se classificam em:

- Simples etapa: no qual existe apenas uma operação para cada tarefa. Somente poderá existir uma máquina para executar cada tarefa, ou várias máquinas com a mesma funcionalidade trabalhando de forma paralela para executar as mesmas operações;
- Múltiplas etapas: no qual existe a necessidade de uma ordem para executar operações diferentes em máquinas diferentes. Este ambiente exige a execução das operações em várias máquinas com funcionalidades diferentes. Este grupo se subdivide em:
  - *Flow shop*: mesmas rotas para todas as ordens;
  - *Open shop*: rota das máquinas pode variar entre diferentes ordens;
  - *Job shop*: cada ordem é única, com rotas pré-estabelecidas sendo diferentes umas das outras. Este é o objeto de estudo deste trabalho.

O sequenciamento de tarefas de ambientes do tipo *job shop* possui as seguintes características:

- Ordem única com rota pré-estabelecida;
- Cada ordem é processada, no mínimo, uma vez em cada máquina.

Os problemas dentro deste ambiente são conhecidos como problemas de sequenciamento *job shop* (JSSP – *job shop schedulling problem*). Nos diferentes cenários do ambiente de produção pode-se classificar as principais características e limitações do JSSP (Bo Peng et. al, 2014):

- Cada job é composto por várias tarefas de um processo;
- Cada job deve ser processado em cada uma das máquinas;
- Cada job possui um tempo de processamento fixo;
- Cada máquina pode processar no máximo, uma operação de cada vez;
- Uma vez que uma operação de processamento começa em uma determinada máquina, ele deve concluir o processamento nessa máquina sem preempção;
- O job somente pode ser agendado se a máquina estiver disponível.

Levando em consideração todas as características e o comportamento descritos acima, um problema de sequenciamento em ambiente de produção *job shop* consiste em um processo de tomada de decisão a fim de otimizar alguns critérios, a partir de sequência de tarefas, tais como minimizar o tempo de entrega e maximizar a utilização de seus recursos, aumentando a capacidade produtiva. Para se atingir estes objetivos, são usadas técnicas de otimização.

JSSPs são problemas de otimização combinatória onde se procura encontrar o melhor resultado para a solução do problema testando cada uma das diferentes combinações. Devido a sua complexidade computacional, *JSSP* recebeu a classificação de problemas NP-Difícil, com difícil resolução (Bo Peng et. al, 2014; Jain & Meeran, 1999).

Na literatura especializada utilizam-se, em geral, problemas *benchmarks* no desenvolvimento dos trabalhos de pesquisa. Uma variedade de problemas é fornecida em (BEASLEY, 2005), e o *makespan* é o parâmetro mais utilizado para avaliar as soluções. *Makespan* se refere ao tempo total entre o processamento da primeira tarefa do primeiro *job* até a última tarefa do último *job* (Lukaszewicz, 2005; Pinedo, 2008).

Devido à complexidade computacional do *JSSP* são empregadas técnicas metaheurísticas como, por exemplo, o algoritmo genético apresentado na próxima seção.

3 Algoritmo genético

O Algoritmo Genético (AG), introduzido por Holland (1975), é uma técnica baseada na evolução das espécies, proposta por Charles Darwin, e vem sendo amplamente aplicada para resolver problemas de classe NP-Difícil. Em especial, a utilização desse algoritmo como técnica de otimização para problemas de sequenciamento do ambiente *JSSP*, é muito difundida (Asadzadeh, 2015; Amirghasemi & Zamani, 2015; Qing-Dao-Er-Ji et al., 2013; Kurdi, 2015).

AG diverge de outros métodos heurísticos por suas características distintas:

- Trata e opera um conjunto de pontos conhecidos como população, ao invés de pontos isolados;
- Opera em um espaço de soluções codificadas e não diretamente no espaço de busca, e com informações necessárias, usando o valor de uma função objetivo, chamado de aptidão;
- Utiliza regras de transição probabilística, em vez de determinística (Goldberg, 1989).

A técnica aplicada por um AG consiste em, a partir de uma população inicial, calcular o valor de aptidão de cada ponto, chamado de indivíduo ou cromossomo. A partir de operadores genéticos de cruzamento (*crossover*) e mutação, o AG cria novas gerações inserindo os cromossomos na população atual, promovendo mudanças aleatórias, a fim de acessar um menor espaço de busca e ao mesmo tempo evitar ficar preso a mínimos ou máximos locais. Esta passagem determina o mecanismo que combinará dois ou mais cromossomos existentes para criar dois ou mais filhos.

As soluções existentes no espaço de busca podem ser demonstradas a partir das seguintes representações:

Representação Binária – Na forma tradicional, utilizam-se *strings* (cadeia de caracteres) binárias de comprimento n para se codificar uma variável real e representar um indivíduo. Segundo Holland (1975; 1992) em sua teoria dos esquemas (*schemata theory*), a codificação binária é fácil de usar e manipular, simples de analisar teoricamente e não possui uniformidade nos operadores, em outras palavras, a mutação nos primeiros bits do gene afeta mais a aptidão que mutação nos últimos bits do gene. Daí a motivação para usá-la.

Representação Real – na qual se utiliza *string* de números reais para representar os indivíduos, ela é normalmente utilizada para otimização de parâmetros contínuos, pois, representação binária não é adequada por exigir bits para obter boa precisão numérica.

Representação de Permutação – na qual se utiliza *string* de números inteiros para representar os indivíduos, e são usados para indicar a ordem na qual uma sequência de eventos deve ocorrer. O AG tem por objetivo encontrar a solução subótima, ou até mesmo a solução ótima, manipulando uma população de indivíduos, que são candidatos a possíveis soluções para o problema. São realizados cruzamentos (*crossover*) entre os indivíduos, gerando filhos que poderão ou não sofrer mutação. Após sucessivas gerações, as populações evoluem, imitando o comportamento proposto por Charles Darwin.

4 Materiais e métodos

Este trabalho é baseado na versão do Algoritmo Genético com Semente Dinâmica (DSGA) apresentado em (Grassi, 2014). O DSGA

utiliza o AG clássico em um nível interno, no qual as soluções candidatas são geradas pela permutação uma solução inicial factível (chamada semente). No trabalho original, os autores utilizaram, como forma de garantir a viabilidade da solução inicial, uma semente criada com base na regra de despacho FIFO (*first in, first out*) e permutação com base no cromossomo binário.

A abordagem DSGA ainda é composta de um nível externo, na qual a melhor solução encontrada no nível interno, após um determinado número de gerações do AG clássico, é usada para atualizar a semente utilizada para as novas iterações, conforme Algoritmo 1 (Grassi, 2014).

Algoritmo 1. Método DSGA
Input: matriz de rotas R, número de iterações externas, *numExt*, parâmetros do AG, critério de parada do AG, Função Objetivo (FO)

1. k ← 0;
2. S ← Gera semente Inicial FIFO (R);
3. Inicia operações no Algoritmo Genético;
4. Enquanto k < *numExt* faça:
   4.1 Inicializa AG (parâmetros do AG);
   4.2 Gera população inicial;
   4.3 Enquanto CriterioParada = falso faça:
      a. Avalia soluções (FO);
      b. Aplica operadores genéticos;
      c. S_k ← melhor individuo da população k
   4.4 Fim enquanto.
   4.5 S ← S_k
5. Fim enquanto
6. Retorna Melhor Solução S

Os experimentos executados neste trabalho visam avaliar os efeitos causados nos resultados de:

- Makespan;
- Gap entre o makespan obtido e o melhor já alcançado na literatura;
- Tempo de processamento;
- Proporção de soluções não factíveis.

A partir de um conjunto de exemplares de *JSSP*, conhecidos como LA (Lawrence, 1984), foram geradas sementes iniciais, para os problemas de LA01 a LA10, com as seguintes características:

Semente não factível;

- FIFO (*First-In, First-Out*);
- FIFO + SPT (*shortest process time*);
- FIFO + LPT (*largest process time*);

A implementação da metodologia foi realizada com a confecção de um aplicativo escrito em linguagem C++ integrado à *GAlib*, que é uma biblioteca de AG escrita em C++ por Matthew Wall, do *Massachusetts Institute of Technology* (*GAlib*, 1996). No aplicativo foram implementadas as rotinas de geração da semente inicial com as abordagens propostas e da função objetivo, enquanto que as funções inerentes ao algoritmo genético ficaram a cargo dos métodos implementados na *GAlib*.

Os parâmetros utilizados no AG para a execução dos experimentos são mostrados a seguir na tabela 1. A função objetivo realiza a avaliação das soluções obtidas usando o algoritmo de Dijkstra modificado para o cálculo do caminho crítico, conforme apresentado no apêndice A.

Foram executados vinte experimentos de cada um dos exemplares de LA01 a LA10, sendo que cada exemplar foi executado individualmente e no final foram avaliadas as médias de cada um dos fatores estudados, bem como a média geral entre todas as instâncias.

5 Resultados

Os resultados obtidos foram comparados entre si com o intuito de identificar os efeitos da estimativa inicial na qualidade da solução. Os resultados da tabela 2 apresentam, para cada problema testado e cada abordagem de geração da semente

Tabela 1: Parâmetros adotados no AG clássico para execução dos experimentos

Parâmetro	Valor Adotado
Representação do Cromossomo	Binária (DSGA)
Seleção	Roleta
Substituição	Steady State
Taxa de Substituição da População	90%
Tamanho da População	10
Cruzamento	OnePointCrossover (OP)
Taxa de Cruzamento	90%
Mutação	Inversão de bit
Taxa de Mutação	1%
Número de Gerações	≥ 5.000 (25 internas e 200 externas)
Critério de Parada	25 gerações internas sem convergência + 200 gerações externas
Função de Aptidão	Algoritmo de Dijkstra modificado para o cálculo do caminho crítico (SHANKAR e SIREESHA, 2010).

inicial (*Seed*), os valores mínimos (min) e médios (mean) de makespan (mkp) obtidos em 20 execuções, e o *gap* (diferença) entre a melhor solução encontrada pelo algoritmo e o melhor resultado encontrado na literatura (best). Foi verificado quem em todos os experimentos iniciados com semente inicial não factível, o algoritmo não conseguiu convergir para uma solução factível, o que é sinalizado com o símbolo *. Os resultados da tabela 2 estão ilustrados graficamente na Figura 1.

No exemplar LA01 é possível notar que, partindo de uma semente não factível, o algoritmo não convergiu para o ótimo global e 100% de suas soluções foram não factíveis, conforme mostrado na tabela 3. Esse comportamento se repetiu em todos os demais problemas.

Em contrapartida, quando o processo se inicia com uma solução factível no LA01, apenas 9% do total das soluções foram não factíveis e não gerou *gap* em relação à melhor solução encontrada na literatura. O método também atingiu o ótimo (*gap* = 0) para os problemas LA02 e LA05-LA10. Vale destacar, no entanto, que o valor ótimo não é atingido necessariamente em todas as execuções. No exemplar LA02, por exemplo, a solução inicial com regra de despacho FIFO+LPT se sobressaiu às outras gerando um maior número de soluções factíveis e chegando à solução ótima no dobro de vezes com o mesmo tempo de processamento (Tabela 3).

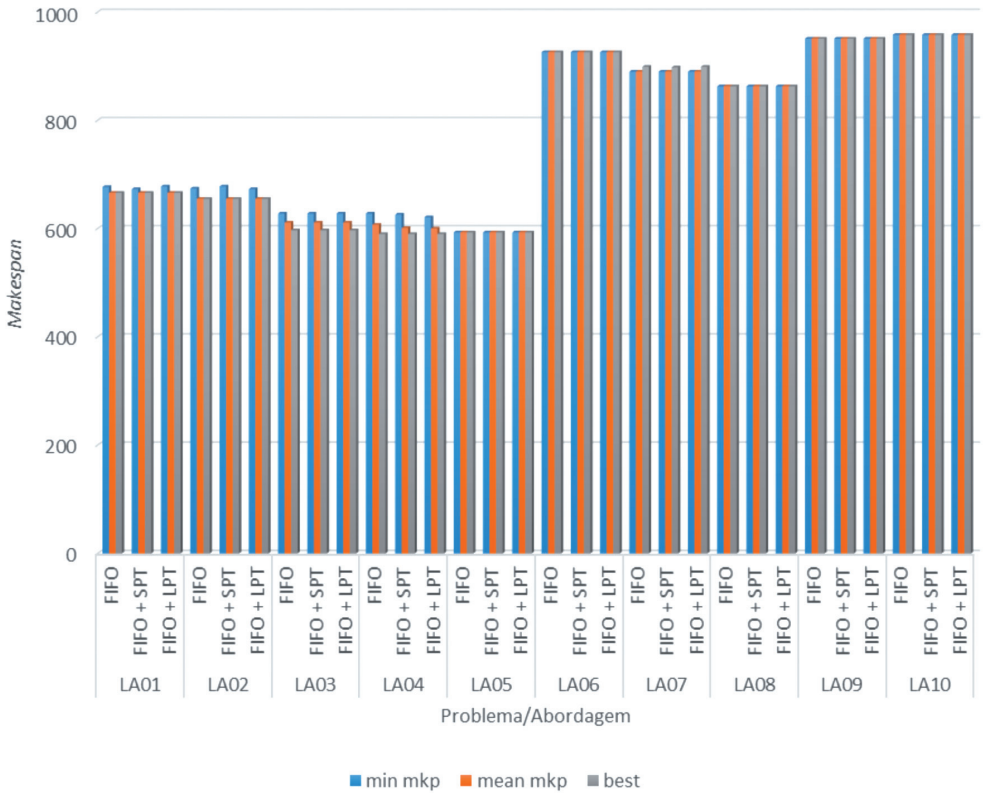

Figura 1: Resultados mínimos e médios de makespan para as diferentes abordagens.

Tabela 2: Resultados obtidos para as instâncias testadas. A última coluna indica o número de vezes que o valor ótimo foi obtido em 20 execuções do método

Problema	Seed	best mkp	min mkp	mean mkp	gap	#cv
LA01	Não factível	666	*	*	-	0
	FIFO		677	666	0	2
	FIFO + SPT		673	666	0	2
	FIFO + LPT		678	666	0	4
LA02	Não factível	655	*	*	-	0
	FIFO		674	655	0	0
	FIFO + SPT		678	655	0	0
	FIFO + LPT		673	655	0	0
LA03	Não factível	597	*	*	-	0
	FIFO		628	611	14	0
	FIFO + SPT		628	611	14	0
	FIFO + LPT		628	611	14	0
LA04	Não factível	590	*	*	-	0
	FIFO		628	607	17	0
	FIFO + SPT		626	601	11	0
	FIFO + LPT		621	600	10	0
LA05	Não factível	593	*	*	-	0
	FIFO		593	593	0	20
	FIFO + SPT		593	593	0	20
	FIFO + LPT		593	593	0	20
LA06	Não factível	926	*	*	-	0
	FIFO		926	926	0	20
	FIFO + SPT		926	926	0	20
	FIFO + LPT		926	926	0	20
LA07	Não factível	890	*	*	-	0
	FIFO		899	890	0	4
	FIFO + SPT		898	890	0	6
	FIFO + LPT		899	890	0	5
LA08	Não factível	863	*	*	-	0
	FIFO		863	863	0	20
	FIFO + SPT		863	863	0	20
	FIFO + LPT		863	863	0	20
LA09	Não factível	951	*	*	-	0
	FIFO		951	951	0	20
	FIFO + SPT		951	951	0	20
	FIFO + LPT		951	951	0	20
LA10	Não factível	958	*	*	-	0
	FIFO		958	958	0	20
	FIFO + SPT		958	958	0	20
	FIFO + LPT		958	958	0	20

Nos exemplares LA03 e LA04, o método não encontrou a solução ótima em todos os casos. Para o LA03 os experimentos que partiram de uma solução factível obtiveram resultados foram muito semelhantes independente da regra de despacho. No caso do LA04, porém, houve uma diferença a ser destacada, que é o menor *gap* na utilização da regra de despacho FIFO+LPT. Com isso é possível afirmar que a utilização de uma semente inicial factível com base na regra híbrida FIFO+LPT, mesmo não encontrando a solução ótima, encontrou a melhor solução subótima em um tempo computacional aceitável.

Por fim, para a instância LA07, apesar de uma pequena diferença na quantidade de soluções ótimas, ou seja, com *makespan* igual ao encontrado na literatura, pode-se dizer que houve empate técnico, principalmente se for levado em consideração o *makespan* médio, que teve variação irrelevante.

Importante ressaltar também que, iniciando o processo com a utilização de uma solução que obedece a regra de despacho FIFO, o algoritmo encontrou um número maior de vezes a solução ótima em relação às regras FIFO+SPT e FIFO+LPT com um tempo de processamento, se não igual, muito próximo. Um resumo de todos os resultados é apresentado na tabela 4.

6 Conclusões

Analisando os resultados dos experimentos é possível verificar que a escolha adequada da estimativa inicial tem influências positivas na qualidade das soluções obtidas. Além do algoritmo encontrar soluções ótimas na maioria dos experimentos, naqueles em que a semente inicial não era factível, o algoritmo genético não conseguiu convergir para uma solução factível. Neste cenário o problema da não convergência se repetiu em todos os exemplares de teste gerando 100% de soluções não factíveis.

Os experimentos realizados com a utilização de semente inicial factível apresentaram, dentro do universo de soluções geradas pelo algoritmo genético, uma proporção elevada de soluções factíveis, variando entre 91 e 96%.

Tabela 3: Valores de tempo médio e proporção de soluções não factíveis. Os tempos são apresentados no formato hora:minuto:segundo

Problema	Variável	Não factível	FIFO	FIFO + SPT	FIFO + LPT
LA01	Tempo médio	00:01:44	00:03:03	00:03:03	00:02:55
	Média de soluções factíveis	*	18.667	18.729	18.777
	Média de soluções não factíveis	20.289	1.890	1.872	1.800
	% soluções não factíveis	100%	9%	9%	9%
LA02	Tempo médio	00:01:42	00:03:27	00:03:30	00:03:29
	Média de soluções factíveis	*	18.825	18.835	18.765
	Média de soluções não factíveis	20	1.934	1.936	1.971
	% soluções não factíveis	100%	9%	9%	10%
LA03	Tempo médio	00:01:41	00:02:58	00:02:35	00:02:04
	Média de soluções factíveis	*	18.623	18.625	18.629
	Média de soluções não factíveis	20.293	2.088	2.063	2.041
	% soluções não factíveis	100%	10%	10%	10%
LA04	Tempo médio	00:01:54	00:02:09	00:02:02	00:02:13
	Média de soluções factíveis	*	18.603	18.714	18.749
	Média de soluções não factíveis	20.296	2.152	2.129	2.100
	% soluções não factíveis	100%	10%	10%	10%
LA05	Tempo médio	00:02:02	00:01:47	00:01:45	00:01:46
	Média de soluções factíveis	*	18.335	18.423	18.406
	Média de soluções não factíveis	20.292	2.053	2.045	2.009
	% soluções não factíveis	100%	10%	10%	10%
LA06	Tempo médio	00:10:10	00:13:43	00:12:33	00:13:04
	Média de soluções factíveis	*	35960	35930	35957
	Média de soluções não factíveis	37784	1820	1767	1796
	% soluções não factíveis	100%	5%	5%	5%
LA07	Tempo médio	00:07:58	00:12:24	00:11:29	00:12:03
	Média de soluções factíveis	*	36807	36691	36668
	Média de soluções não factíveis	37794	1824	1873	1853
	% soluções não factíveis	100%	5%	5%	5%
LA08	Tempo médio	00:08:55	00:07:51	00:07:30	00:07:33
	Média de soluções factíveis	*	36673	36825	36767
	Média de soluções não factíveis	37770	1641	1599	1567
	% soluções não factíveis	100%	4%	4%	4%
LA09	Tempo médio	00:12:17	00:11:15	00:08:50	00:08:31
	Média de soluções factíveis	*	36308	36285	36239
	Média de soluções não factíveis	37730	1636	1650	1681
	% soluções não factíveis	100%	4%	4%	4%
LA10	Tempo médio	00:08:30	00:41:18	00:13:22	00:12:37
	Média de soluções factíveis	*	36002	35914	35928
	Média de soluções não factíveis	37754	1776	1850	1808
	% soluções não factíveis	100%	5%	5%	5%

Tabela 4: médias gerais dos experimentos e ganho total (LA01 a LA10)

Parâmetro	Não factível	FIFO	FIFO + SPT	FIFO + LPT	Ganho
% soluções não factíveis	100%	7,2%	7,2%	7,1%	1,3%
Gap	*	3,1	2,5	2,4	22,6%
Makespan médio	*	779,6	779,4	779,0	0,1%
Média de soluções factíveis	*	27480,4	27497,2	27488,6	0,1%
Média de soluções não factíveis	29025,7	1881,2	1878,4	1862,5	1,0%
Menor Makespan	*	772	771,4	771,3	0,1%
Soluções ótimas	*	9,4	9,2	9	4,3%
Tempo médio	00:05:41	00:10:00	00:06:40	00:06:37	33,7%

A utilização de uma solução inicial factível e baseada na regra híbrida FIFO+LPT se sobressaiu em relação aos demais, demonstrando ganhos de desempenho em todos os parâmetros analisados se comparados com os resultados da abordagem com menor desempenho, com maior destaque para a média de tempo gasto na execução dos experimentos e para a diminuição do *gap* geral em 33,7% e 22,6%, respectivamente. Isso indica que uma estimativa inicial que utiliza todas as informações do problema (ordem de chegada das tarefas e tempos de processamento) pode produzir resultados melhores em comparação ao uso da regra FIFO.

Agradecimentos

Os autores agradecem à Universidade Nove Julho por tornar possível a elaboração deste trabalho, concedendo a bolsa de estudos ao primeiro autor, bem como permitindo o uso de suas instalações.

Referências

Abdelmaguid, T. F. (2010). Representations in genetic algorithm for the job shop scheduling problem: a computational study. *Journal of Software Engineering and Applications*, 3, 1155-1162.

Allahverdi A.; Ng C. T.; Cheng T. C. E.; Kovalyov M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187, 985-1032.

Beasley J. Operations research library (2005). Recuperado em 17 de dezembro, 2015 de http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

Bo Peng, ZhipengLü, T.C.E.Cheng (2014). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & OperationsResearch*, 53, 154–164.

GAlib: *A C++ library of genetic algorithm components*. Recuperado em 27 de outubro, 2015 de http://lancet.mit.edu/ga/dist/.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization and machine learning. New York: Addison-Wesley.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; KAN, A. H. G. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics. v. 5, 287-326.

Grassi, F. (2014). Otimização por algoritmos genéticos do sequenciamento de ordens de produção em ambientes job shop. Dissertação de mestrado, Universidade Nove de Julho.

Heinonen, J., e Pettersson, F. (2007). Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. *Applied Mathematics and Computation*, 187(2), 989-998.

Holland, J. H. (1992). Adaptation in Natural and Artificial Systems. 2. ed. The MIT Press.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press.

Jain, A. S., e Meeran, S. (1999). Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research*, 113(2), 390-434.

Kunnathur, A. S., Sundararaghavan, P. S., e Sampath, S. (2004). Dynamic rescheduling using a simulation-based expert system. *Journal of Manufacturing Technology Management*, 15(2), 199-212.

Lawrence, S. (1984). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques* (Supplement). PhD diss., Carnegie-Mellon University.

Leila Asadzadeh (2015). A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, 85, 376–383.

Lukaszewicz, P. P. (2005). Metaheuristics for job shop scheduling problem, comparison of effective methods. PhD diss., Aarhus School of Business.

Mehrdad Amirghasemi, Reza Zamani, (2015). An effective asexual genetic algorithm for solving the job shop scheduling problem. Computers & Industrial Engineering, 83, 123–138.

Mohamed Kurdi (2015). An effective new island model genetic algorithm for job shop scheduling problem. Computers & Operations Research.

Pinedo, M. L. (2008). Scheduling: Theory, algorithms, and systems. New York: Springer.

Ren Qing-dao-er-ji, Yuping Wang, Xiaoli Wang (2013). Inventory based two-objective job shop scheduling model and its hybrid genetic algorithm. Applied Soft Computing, 13, 1400–1406.

Saramago, S. P.; Steffen JR,. V. (2008). Introdução às técnicas de otimização em engenharia. Horizonte Científico, v. 1, 1-30.

Modolo, V. (2015). Estudo comparativo de diferentes representações cromossômicas nos algoritmos genéticos em problemas de sequenciamento da produção em job shop. Dissertação de mestrado, Universidade Nove de Julho.

Wang S. F.; Zou Y. R. (2003). Techniques for the job shop scheduling problem: a survey. Systems Engineering - Theory & Practice, 23, 49-55.

```
void findCriticalPath(vector<int> s)
{
   //
   int *p;
   vector<int> startVec;
   vector<int> endVec;
   vector<int> timeVec;

   // Cria Arestas do grafo
   for(int j=0; j<JOB; j++)
   {
         startVec.push_back(0);
         timeVec.push_back(0);

         for(int m=1; m<=MACHINE; m++)
         {
               //
               endVec.
push_back(j*MACHINE+m); //1-index
               startVec.
push_back(j*MACHINE+m); //1-index
               timeVec.push_
back(T[j*MACHINE+m-1]); //0-index
         }
         endVec.push_back(JOB*MACHINE+1);
   }
   //
   for(int m=0; m<MACHINE; m++)
   {
         for(int j=0; j<JOB; j++)
         {
               int job = s[m*JOB+j] - 1;
               // Vai na 'linha' de R referente ao job
               int start = job * MACHINE;
               int end = job *
MACHINE + MACHINE - 1;
               p = std::find(R+start,R+end, m+1);
               //Posicao
               int pos = p - (R+start);
               int no = job*MACHINE + pos + 1;
               // Coloca na lista de arestas
               if(j == 0)
               {
                  //seta apenas como nó inicial
                  startVec.push_back(no); //1-index
                  timeVec.
push_back(T[no-1]); //0-index
               }
               if(j == JOB-1)
               {
                  //seta apenas como nó final
                  endVec.push_back(no); //1-index
               }
               if(j != 0 && j != JOB-1)
               {
                  //seta como inicial e final
                  startVec.push_back(no); //1-index
                  endVec.push_back(no); //1-index
                  timeVec.
push_back(T[no-1]); //0-index
               }
         }
   }
   //
   int numberVertex = MACHINE*JOB
+ 2;   // number  of vertex
   int numberActivities = endVec.
size(); // number of edges
   int i, j;

   vector<int> indegree(numberVertex, 0);

   std::vector<int> v(startVec);
   std::vector<int> u(endVec);
   std::vector<int> d(timeVec);

   int project_duration=0;//project duration

   for (j=0; j<numberActivities; j++){
      indegree[u[j]]++;
   }
   queue<int> Q; //queue Q = empty queue
   int distance [numberVertex];
   memset(distance, 0, sizeof(int) * numberVer-
tex);//distance = array preenchido com zeros

   //vetor de conjuntos de nos
   vector< set<int> > vecSetCrit
icalPath(numberVertex);

   for (j=0; j<numberVertex; j++)
   {
      if (indegree[j]==0)
         Q.push(v[j]);
   }

   int first;

   while (!Q.empty()){ //while Q is not empty:

    first= Q.front();  //v = get front element from Q
    Q.pop(); //delete de first from queue

    //Para todo vizinho de first faca
    for(int i = 0;i < numberActivities;i++)
    {
       if(v[i] == first) //
       {
          //
          distance[u[i]]=std::max(dista
nce[u[i]], distance[v[i]]+ d[i]);
          indegree[u[i]]-=1;
          //
          if (indegree[u[i]]==0){
             Q.push(u[i]);
          }
       }
    }
   }//fim while
   //
   /*Selecione o vertice x com a maior distância.
   Esta é a mínima duração total do projeto.*/
   project_duration = *std::max_elemen
t(distance,distance+numberVertex);

   int criticalNode = std::find(distance,
distance+numberVertex, pro-
ject_duration) - distance;
   vector<int> criticalPath;
   //
   while(criticalNode != 0){
      vector<int> neighborhood;
      vector<int> neighDistances;
      //
      // varre recursivamente os
vizinhos e pega o máximo
      for(int i = 0;i < numberActivities;i++)
      {
            if(u[i] == criticalNode)
            {
                  neighborhood.push_back(v[i]);
                  neighDistances.
push_back(distance[v[i]]);
            }
      }
      int max = *std::max_
element(neighDistances.
begin(),neighDistances.end());
      int pp =  std::find(neighDistances.
begin(), neighDistances.end(),
max) - neighDistances.begin();
      if(max == 0) pp = neighDistances.size() - 1;
      criticalNode = neighborhood[pp];
      criticalPath.push_back(criticalNode);
   }

   currentCriticalPath = criticalPath;
   return;
}
```

Apendice A: Algoritmo de Dijkstra modificado para o cálculo do caminho crítico

Recebido em 17 dez. 2015 / aprovado em 8 nov. 2016

Para referenciar este texto

Cruz, V. F. da, & Pereira, F. H. Estudo da influência da estimativa inicial na qualidade da solução no algoritmo genético binário em problemas de sequenciamento da produção em ambiente job shop. *Exacta – EP*, São Paulo, v. 15, n. 4, p. 1-11, 2017.