MODELAGEM E SIMULAÇÃO DE EVENTOS DISCRETOS NA OTIMIZAÇÃO DO SEQUENCIAMENTO DA PRODUÇÃO EM JOB SHOP

Flávio Grassi
Programa de Pós-Graduação da Universidade Nove de Julho - PPGEP/UNINOVE
Av. Francisco Matarazzo, 612, São Paulo, SP, Brasil
flavio.grassi@uninove.edu.br

Marilda de Fátima Souza da Silva
Programa de Pós-Graduação da Universidade Nove de Julho - PPGEP/UNINOVE
Av. Francisco Matarazzo, 612, São Paulo, SP, Brasil
marilda.fatima1s@hotmail.com

Valdemar Módolo Junior
Programa de Pós-Graduação da Universidade Nove de Julho - PPGEP/UNINOVE
Av. Francisco Matarazzo, 612, São Paulo, SP, Brasil
vmodolo@uninove.br

Fabio Henrique Pereira
Programa de Pós-Graduação da Universidade Nove de Julho - PPGEP/UNINOVE
Av. Francisco Matarazzo, 612, São Paulo, SP, Brasil
fabiohp@uninove.br

RESUMO

O problema de sequenciamento da produção em ambientes Job Shop é um dos mais difíceis de resolver. Várias técnicas de otimização têm sido empregadas para solucionar tais problemas, entretanto o cenário e as restrições inerentes ao processo são frequentemente modelados utilizando expressões matemáticas, as quais nem sempre traduzem fácil e adequadamente o ambiente de produção real. Este artigo visa a modelagem de um cenário de produção por meio da simulação de eventos discretos, a fim de produzir um representação mais fidedigna de um ambiente de produção dinâmico e estocástico. O modelo é acoplado à Algoritmos Genéticos, com o objetivo de minimizar o *makespan*.

PALAVRAS CHAVE. Sequenciamento de Ordens, Simulação de Eventos Discretos, Algoritmos Genéticos.

Área Principal: SIM – Simulação.

ABSTRACT

Job Shop Scheduling Problems are one of the hardest problems to solve. Several optimization techniques have been employed in order to solve them, however the scenario and the constraints inherent to the process are often modeled using mathematical expressions, which do not always translate easily and accurately the actual production environment. This paper focuses on the modeling of production scenario through a discrete event simulation model, in order to provide a much more accurate representation of a dynamic and stochastic production environment. The resulting simulation model is applied to genetic algorithms, in order to minimize the makespan.

KEYWORDS. Sequencing Orders, Discrete Event Simulation, Genetic Algorithms.

Main Area: SIM – Simulation.

1. Introdução

Devido ao aumento da competitividade no mercado, a otimização dos processos em ambientes industriais têm sido objeto de estudo e pesquisa nas mais variadas áreas do conhecimento, como na gestão de negócios, economia, logística, dentre outros; sendo mais notadamente estudada dentro da engenharia de produção (Kunnathur *et al.*, 2004; Heinonen e Pettersson, 2007). Dentro desse contexto, um dos problemas que têm demandado mais pesquisas nas últimas décadas são os problemas de sequenciamento de produção.

Normalmente, os problemas de sequenciamento são resolvidos com o auxílio de algoritmos de otimização, os quais necessitam de uma representação do cenário do problema de sequenciamento. Uma maneira comumente utilizada é construir uma expressão matemática que envolva todo o comportamento e as restrições do cenário, e então apresentar essa expressão à técnica de otimização. Essa abordagem, no entanto, nem sempre é muito simples, especialmente quando se tratam de problemas de médio e grande porte com características estocásticas e dinâmicas como é o caso da maioria dos problemas reais. Na tentativa de resolver esse problema, este trabalho propõe um modelo de simulação eventos discretos que represente adequadamente a dinâmica, aleatoriedade e restrições do ambiente de produção, e possa trabalhar em conjunto a técnica de otimização.

2. Problemas de Sequenciamento

2.1 Cenários de sequenciamento de produção

De acordo com as definições encontradas na literatura, sequenciamento pode ser entendido como o processo de alocar um ou mais recursos para executarem certas atividades cujo processamento irá demandar certa quantidade de tempo (Lukaszewicz, 2005). Esses recursos são comumente apresentados como máquinas, e as atividades que serão processadas nas máquinas são conhecidas como tarefas. Assim sendo, um *job* pode ser entendido como um conjunto de uma ou mais tarefas.

Existem diferentes configurações de cenários de produção, com base na distribuição dos *jobs* nas máquinas e na semelhança entre esses *jobs*, as quais conduzem à diferentes classificações (Allahverdi *et al.*, 2008). Na classificação mais aceita, os ambientes de produção podem ser de uma única etapa, onde há somente uma operação para cada tarefa; ou de várias etapas, nos quais uma ordem de produção é composta por várias operações em diferentes máquinas (Graham *et al.*, 1979; Pinedo, 2008). Em ambientes de única etapa, tipicamente existe uma única máquina para executar cada tarefa, ou podem existir várias máquinas com a mesma funcionalidade rodando operações em paralelo. Já no ambiente de múltiplas etapas, cada ordem deve ser executada em diferentes máquinas, com diferentes características. Esse último grupo é subdividido em *flow shop*, *open shop* e *job shop*. Em um *flow shop*, todas as ordens possuem as mesmas rotas, o que significa que as operações em cada tarefa são realizadas em uma mesma sequência nas máquinas, para todas as ordens de produção. No *open shop*, um pouco mais complexo, a rotas das máquinas por qual as ordens passam pode não ser a mesma para todas as ordens. Por fim, no *job shop* – objeto desse estudo – cada ordem é única, com rotas pré-estabelecidas e diferentes entre si.

2.2 Problemas de Job Shop Scheduling em Ambientes de Produção

Em um *job shop*, cada ordem possui rotas diferentes e pré-estabelecidas, e é processada no mínimo uma vez em cada uma das máquinas. O problema dentro desse tipo de cenário é conhecido como *Job Shop Scheduling Problem* (JSSP). Pode-se listar as características e restrições do JSSP conforme segue, e que o difere de outros cenários de produção (Fan e Zhang, 2010):

- Todas as máquinas estão disponíveis no instante de tempo t_0=0 e todas as ordens são liberadas no instante t_0=0.
- Cada *job* pode ser processado em uma única máquina por vez.
- Não são permitidos processos simultâneos em qualquer uma das máquinas.

- Cada *job* é composto por várias tarefas a serem processadas.
- Cada *job* interage com apenas uma máquina por vez.
- Cada tarefa precise ser executada em máquinas específicas.
- A sequência de operações para cada *job* é sempre pré-definida e não pode ser alterada.
- Os tempos de processamento de todas as operações são conhecidos.
- Não existe restrições de precedência nas operações de diferentes *jobs*.
- Cada operação, desde seu início até o seu término, não pode ser interrompida por outros processos.

Portanto, considerando as características e o comportamento apresentado, o problema de sequenciamento em um ambiente de produção do tipo *job shop* é essencialmente um processo decisório, o qual busca identificar uma sequência que otimiza algum tipo de critério, como minimizar o tempo de entrega e maximizar o uso da capacidade produtiva. Normalmente são utilizadas técnicas de otimização para alcançar esses objetivos.

Relacionado à sua complexidade, os JSSP são classificados como problemas *NP-hard*, o que significa na teoria de complexidade computacional que são problemas difíceis de resolver (Fan e Zhang, 2010). Essencialmente os JSSP são problemas de otimização combinatória (nos quais o melhor resultado para resolver o problema é encontrado através do teste de cada uma das diferentes combinações possíveis).

Embora existam métodos determinísticos, que em geral trabalham testando e exaurindo todas as soluções possíveis na tentativa de encontrar a melhor entre elas, é quase impossível resolver esse tipo de problema dessa forma, exceto para problemas relativamente pequenos (Jain e Meeran, 1999). Em ambientes de produção reais, essa otimalidade não é necessariamente um critério a ser atingido. É suficiente obter resultados próximos do ótimo, mas com tempos razoáveis. Soluções próximas do ótimo podem ser obtidas por meio de métodos heurísticos, baseados em probabilidades, os quais não testam todas as possibilidades de solução no espaço de busca.

Assim, um JSSP é primariamente representado de alguma forma, tipicamente por expressões matemáticas analíticas, que são então submetidas a alguma das várias técnicas de otimização, com vistas encontrar um sequenciamento dos *jobs* que possa ser utilizado de forma satisfatória em problemas reais. Entretanto, como mostra a literatura especializada, o estudo e aplicação de técnicas para solucionar um JSSP é realizado com o uso de instâncias de problemas, conhecidas com *benchmarks*. Uma variedade de problemas está disponível em (Beasley, 2005), sendo o *makespan* o parâmetro mais utilizado para avaliar as soluções. O termo *makespan* refere-se ao tempo total entre o início do processamento da primeira tarefa do primeiro *job* e o fim processamento da última tarefa do último *job* (Lukaszewicz, 2005; Pinedo, 2008). No trabalho proposto foi utilizada a instância FT06.

2.3 Técnicas de Otimização

Uma técnica de otimização pode ser entendida com um algoritmo de computador usado para identificar, baseado em uma função objetivo que representa o problema, as melhores soluções possíveis (soluções sub-ótimas), ou a melhor delas (solução ótima). Na prática, o sucesso da técnica utilizada depende de vários fatores, os quais vão desde a adequada representação do problema até a escolha dos valores apropriados para os parâmetros próprios de cada técnica (Wang e Zou, 2003).

Uma revisão da literatura permite identificar quais técnicas são mais adequadas para cada tipo de problema, embora não haja uma regra bem definida e universalmente válida para isso. Vale ressaltar, no entanto, que não existe um senso comum em relação à especificidade de uma determinada técnica de otimização para cada problema a ser tratado. Uma referência sobre isso é apresentada pelo teorema proposto por (Wolpert e Macready, 1995), conhecido como teorema do "não existe comida de graça" (tradução nossa), o qual pode ser interpretado conforme segue: *se existe um determinado algoritmo A que é melhor que um determinado algoritmo B para uma certa instância de um grupo de problemas x, certamente haverá outro grupo instâncias do*

mesmo problema x no qual o algoritmo B tem um desempenho melhor do que o algoritmo A. É fato também que um algoritmo deve ser melhor para resolver um dado problema se ele foi parametrizado em função daquele problema específico, e esse mesmo algoritmo tende a não ser tão eficiente caso seja tentado generalizá-lo para solucionar outros problemas.

Uma das técnicas mais utilizadas na otimização de problemas de sequenciamento são os algoritmos genéticos, que são baseados na evolução natural das espécies encontradas na natureza.

Algoritmos Genéticos

Um Algoritmos Genético (AG) pode ser entendido como uma classe de técnicas metaheurísticas que consiste em encontrar soluções baseadas nos mecanismos de seleção natural e genética. Em linhas gerais, os AGs operam sobre um determinado conjunto de pontos, conhecido como população, a não sobre pontos isolados; trabalham em um espaço de soluções codificadas do problema, e não diretamente sobre o espaço de busca; e necessitam apenas da informação do valor de uma função objetivo, chamada de função de aptidão (*fitness*), usando regras probabilísticas ao invés de determinísticas (Goldberg, 1989).

Os procedimentos executados por um AG clássico são a criação uma população inicial e o cálculo do valor de aptidão para cada ponto, chamado de indivíduo ou cromossomo. Operadores genéticos são então aplicados para, probabilisticamente, selecionar os indivíduos com base no seu nível de aptidão, criando uma nova geração de indivíduos. Melhores níveis de aptidão significam maiores chances de seleção.

A evolução das novas gerações também é conduzida inserindo novos cromossomos na população atual utilizando os operadores genéticos de cruzamento e mutação. O cruzamento determina o mecanismo de combinação entre dois ou mais cromossomos para criar dois ou mais filhos; o operador de mutação promove mudanças aleatórias nos cromossomos, a fim de não ficar atrelado a algum máximo local e garantir acesso a todo o espaço de busca.

2.4 Modelagem e Simulação de Sistemas à Eventos Discretos

A modelagem e simulação tem sido utilizada ao longo de décadas como uma importante técnica para auxiliar nos processos de decisão em diversos segmentos, como sistemas de telecomunicação, ambientes de produção, manutenção, dentre outros. Trata-se de uma das ferramentas de apoio que permite construir modelos e analisar o desempenho de sistemas e processos complexos, permitindo aumentar a confiança durante o processo de desenvolvimento e tomada de decisão (Slack *et al.*, 2007). Existem abordagens clássicas que utilizam modelos matemáticos na modelagem de sistemas à eventos discretos, como redes de Petri e cadeias de Markov, mas dependendo da complexidade do sistema, a adoção de modelos de simulação de sistemas pode ser um recurso interessante em substituição às abordagens matemáticas clássicas.

A modelagem pode ser entendida como o processo de construção de um modelo que represente um sistema real, bem como a condução de experimentos com esse modelo a fim de entender melhor o comportamento do sistema e avaliar o impacto de operações estratégicas (Kelton *et al*., 2003). Como etapas para o desenvolvimento de uma simulação têm-se a formulação do problema, coleta de dados, construção, teste e validação do modelo de representação do problema; análises e documentação dos resultados. Observe que a validação é uma etapa fundamental para garantir que os resultados obtidos pela simulação são de fato aplicáveis ao problema. Nesse sentido, a submissão do modelo em um ambiente real, comparando o resultado entre eles, é a forma de validação mais adequada (Law *et al.*, 2009).

2.5 Simulação como Ferramenta para Métodos de Otimização

Existem diversos estudos que visam desenvolver melhores técnicas de otimização, seja gerando uma técnica inovadora através da combinação de abordagens de métodos já conhecidos ou melhorando uma técnica já existente, que sejam apropriadas para certos tipos de problemas com características próprias. Nesse contexto, exemplos de pesquisas que fazem uso do AG em ambientes de produção podem ser encontrados em diversos trabalhos atuais (Jinghua e Mianzhou, 2010; Raajan *et al.*, 2010; Datta *et al.*, 2011; Baker e Altheimer, 2012; Wang e Liu, 2013).

Entretanto, a fim de serem aplicadas para resolver um JSSP, as técnicas de otimização demandam uma representação do ambiente real de produção, contemplando suas restrições, sua dinâmica e aleatoriedades. Ou seja, é necessário representar o problema de maneira que a aptidão das possíveis soluções possa ser avaliada da forma mais fidedigna possível. Em geral, tal representação é obtida por meio de expressões matemáticas analíticas, as quais podem não ser completamente adequadas em situações mais realistas, nas quais a chegada de ordens de produção e o processamento dessas ordens pelo sistema possuem características dinâmicas e aleatórias.

O presente trabalho propõe o uso da modelagem e simulação de um sistema à eventos discretos para representação dos problemas de *job shop*, já que o uso de tais ferramentas permite uma representação mais simples e precisa do ambiente de produção real (Tavakkoli-Moghaddam e Daneshmand-Mehr, 2005; Su e An, 2010). Com vistas a sua validação, o modelo de simulação foi adaptado para uma instância específica de JSSP encontrada na literatura, chamada de FT06, com tempos de processamento determinísticos e valor de *makespan* conhecido. O modelo foi acoplado ao algoritmo genético e os resultados foram comparados e analisados.

3. Materiais

3.1 Software de Simulação

Este trabalho utilizou o software de simulação ARENA®, principalmente por possuir uma boa documentação, oferecer flexibilidade de comunicação com outros programas e por disponibilizar uma versão para estudantes, a qual permite a construção e simulação de modelos de pequeno e médio porte. O ARENA® é uma ferramenta de programação visual, flexível, orientada a objeto e que combina a construção visual de modelos de simulação com a interação de diferentes linguagens de programação. A linguagem de programação utilizada por este software é baseada na linguagem SIMAN (Rockwell Automation, 2007).

O software permite o uso de duas tecnologias Microsoft®: o *ActiveX Automation*, que disponibiliza facilidades de controle para aplicações do tipo *desktop* através de um *framework* disponível para algumas linguagens de programação; e o *Visual Basic for Applications* (*VBA*), que é uma linguagem de programação para escrita de códigos para implementar alguma funcionalidade mais avançada não disponível apenas fazendo uso dos objetos visuais, chamados de blocos dentro de software de simulação.

3.2 Ambiente de Algoritmo Genético

O processo de otimização foi realizado utilizando um ambiente de algoritmo genético disponibilizado através da biblioteca GAlib. Trata-se de uma biblioteca em C++ de componentes típicos de algoritmos genéticos, que inclui ferramentas de suporte a diferentes representações de soluções e diversas variações de operadores genéticos. Além de permitindo um acoplamento relativamente fácil com o modelo de simulação em ARENA®, a GAlib é uma biblioteca livre, largamente utilizada e, portanto, confiável (Wall, 1996). Vale lembrar que como o objetivo do trabalho era apresentar um modelo de representação de uma instância de JSSP, e não de focar na técnica de otimização, não serão apresentados detalhes da estrutura do algoritmo genético utilizado, nem dos parâmetros adotados, pois este não é o foco deste trabalho.

4. Métodos

4.1 Construção do Modelo de Simulação

Foi construído um modelo de simulação para o cenário de uma clássica instância de problema de JSSP conhecida como FT06, inicialmente proposta por H. Fisher e G.L. Thompson, em 1963 (Beasley, 2005). A fim de possibilitar a reprodução dos resultados, as configurações de cada módulo do modelo são apresentadas e explicadas. Nas tabelas, o estilo itálico significa nomes definidos pelo usuário, e outras palavras que aparecem em estilo padrão, indicam nomes, variáveis e opções de seleção disponíveis no ARENA®.

O primeiro elemento do modelo é o objeto chamado Create, que gera as entidades para o cenário (os *Jobs*, no caso). As principais configurações são mostradas na Tabela 1, definida de acordo com o JSSP adotado, conforme descrito na seção 2.2.

Tabela 1: Configurações do módulo Create

Type	Entities per Arrival	Max Arrivals
Constant	*6*	*1*

O próximo módulo é o Assign, onde são definidos alguns atributos utilizados durante o processo. Esses atributos incluem o índice do *job*, o índice das máquinas, a sequência que deverá ser seguida por cada *job* e o tempo de processamento da tarefa. O índice do *job*, que necessita ser único para cada *job*, é obtido pela propriedade SerialNumber da Entidade (Rockwell Automation, 2007), enquanto o índice da máquina depende do número de máquinas do especificado no JSSP.

No modelo de simulação, o tempo de processamento pode ser facilmente definido com uma matriz indexada pelos índices de *jobs* e máquinas, seja com valores constantes, como no caso da instância de teste usada, ou descritos por distribuições de probabilidades. Os principais ajustes desse módulo são ilustrados na Tabela 2.

Um importante atributo é o Priority, declarado no módulo de dados chamado Attribute, o qual é utilizado para definir as diferentes prioridades dos *jobs* durante a simulação. A avaliação das diferentes prioridades permite apresentar ao usuário a melhor sequência a ser utilizada, depois que o algoritmo de otimização a identifica.

Tabela 2: Configurações do módulo Assign

Type	Attribute Name	New Value
Attribute	*ArriveTime*	TNOW
Attribute	*JobIndex*	Entity.SerialNumber
Attribute	Entity.Sequence	*JobSequence(JobIndex)*
Attribute	*MachineIndex*	*(Machine1, … , Machine6)*
Attribute	*ProcessingTime*	*(JobIndex, MachineIndex)*

Um conjunto de entidades foi definido no módulo de dados Set, onde os elementos são exatamente os seis *jobs* existentes no problema. Naturalmente, esse número pode ser facilmente modificado de acordo com o problema a ser resolvido.

Depois do módulo Assign, utiliza-se um módulo Station que recebe os seis diferentes *jobs*, os envia para o primeiro módulo Route, que por sua vez direciona os *Jobs* para as estações de trabalho (máquinas). Para fazer isso, o campo Destination Type precisa ser definido como By Sequence, que significa que a sequência a ser obedecida depende do módulo de dados Sequence. Nesse módulo, as seis sequências diferentes são definidas, uma para cada *job*, de acordo com a instância FT06.

Cada máquina é representada por um módulo Process, com as principais definições demonstradas na Tabela 3. É importante alocar apenas um recurso para cada módulo Process, a fim de representar adequadamente o cenário de JSSP.

Tabela 3: Configurações do módulo Process

Action	Delay Type	Expression
Seize Delay Release	Expression	*ProcessingTime(JobIndex)*

Assim que o módulo Process é inserido, é necessário definir no módulo de dados Queue a informação referente ao comportamento da fila para cada processo. Esse módulo necessita ter o mesmo nome do processo seguido por um ponto e o nome Queue (como em *Machine1*.Queue). O campo Type, que está relacionado ao comportamento da fila em si, é definido como Lowest Attribute Value, sendo que o Attribute Name refere-se ao atributo de prioridade definido anteriormente no módulo de dados Attribute (ou seja, *Priority*).

Antes de cada módulo Process é necessário inserir um módulo Station, destinado a receber os *jobs* enviados pelo módulo Route.

Devido a um comportamento comum de ferramentas de simulação, foi observada durante as simulações iniciais a necessidade de um módulo Hold antes de cada processo no modelo. Quando não existem entidades sendo processadas, uma nova entidade que chega tem seu atendimento prontamente iniciado, sem ir para uma fila. Esse comportamento faz sentido do ponto de vista de um ambiente de produção, mas pode gerar um problema na simulação se o atributo de prioridade for ignorado. A inserção do módulo Hold imediatamente antes do módulo Process evita esse problema, ao reter o *job* em uma fila até que uma condição de prioridade seja satisfeita. Os principais ajustes desse módulo são ilustrados na Tabela 4. Além disso, cada módulo Hold precisa tem sua própria fila, configurada no módulo de dados Queue nas mesmas condições observadas para a fila do módulo Process.

Tabela 4: Configurações do módulo Hold

Type	Condition
Scan for Condition	*Priority*==(ResSeizes(*ResourceName*)+1

Cada módulo Process é conectado a um novo módulo Route, novamente com o campo Destination Type definido como By Sequence.

A última etapa da sequência de cada *job*, definido no módulo de dados Sequence, é definida por um novo módulo Station que representa a saída dos *jobs* do sistema. Nesse momento um módulo Record, definido conforme Tabela 5, é utilizado para coletar estatísticas de *makespan* que são enviadas ao AG para o cálculo da aptidão da sequência simulada.

Tabela 5: Configurações do módulo Record

Type	Attribute Name	Tally Name
Time Interval	*ArriveTime*	*Makespan*

Por fim, é necessário inserir um módulo Dispose, que indica o fim do sistema. O módulo Dispose não possui nenhum campo que precisa ser detalhado. A única recomendação é certificar-se que a caixa Record Entity Statistics está marcada.

Para fazer a integração do modelo de simulação construído e o algoritmo genético, é necessário criar um código utilizando o VBA e os controles *ActiveX Automation*. O *ActiveX* é utilizado para permitir que a aplicação externa do AG abra, inicie e pare o modelo de simulação. Por outro lado, os recursos do VBA permitem coletar os valores do arquivo de texto do AG (valores de prioridades) e inseri-los corretamente nos campos apropriados do modelo de simulação. Também permitem gerar um arquivo de texto no modelo de simulação com os valores de *makespan* a cada iteração da simulação. Uma visão geral do acoplamento entre o modelo de simulação e o AG pode ser visto na Figura 1.

No ambiente de programação do ARENA® existem procedimentos pré-definidos de lógica do modelo, os quais definem momentos específicos da simulação para a execução das instruções contidas no código VBA desenvolvido. Assim, a fim de abrir o arquivo de texto gerado pela GAlib e preencher os campos *Priority* do modelo adequadamente, foi utilizado o procedimento ModelLogic_RunBeginSimulation() que, como sugere, é executado antes da

simulação; e o procedimento ModelLogic_RunEndReplication() para escrever os valores de *makespan* em um arquivo de texto.

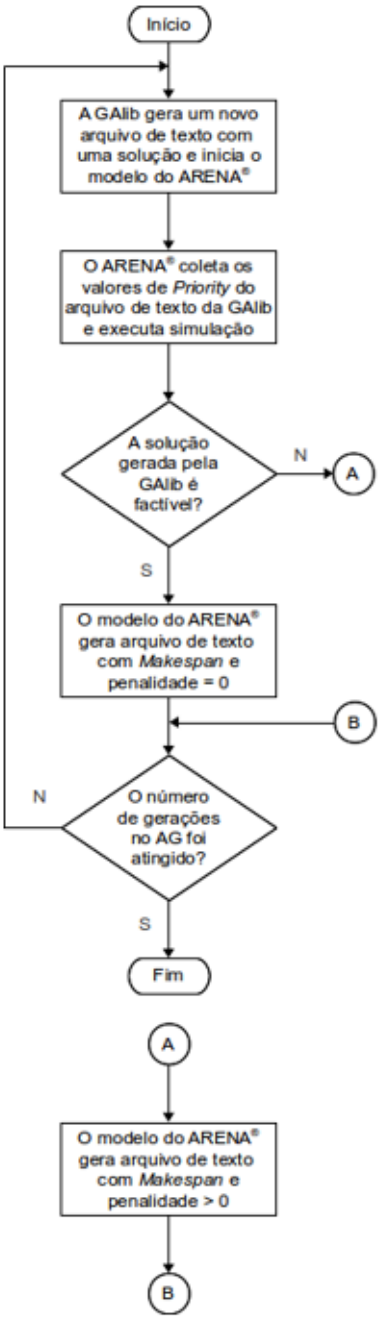

Figura 1: Fluxograma do acoplamento

4.2 Verificação do Modelo

O objetivo principal deste trabalho é a utilização de um modelo de simulação na representação de problemas dinâmicos de sequenciamento de médio e grande porte, contemplando tempos entre chegadas e tempos de processamento aleatórios, com soluções ótimas não conhecidas. Nesse contexto, o uso de expressões matemáticas analíticas não é a opção ideal devido à dificuldade de reproduzir matematicamente o cenário real de um ambiente de sequenciamento dinâmico e estocástico de um *job shop*, com todas as restrições inerentes.

O modelo resultante é usado para avaliar as soluções fornecidas por algoritmos de otimização. Foi constatado que o modelo simula e avalia as diversas soluções geradas pelo

algoritmo genético adequadamente. Entretanto, para que seja possível o uso do modelo proposto em cenários dinâmicos, é necessário submeter o modelo a uma instância determinística, para o qual seja conhecido o valor ótimo de *makespan* e a sequência que gera esse *makespan* ótimo. A sequência ótima para a instância FT06, usada neste trabalho e obtida por (Hou *et al.*, 2011), é apresentada na Tabela 6.

Tabela 6: Sequência dos *jobs* para cada máquina

Máquina	Sequência de processamento dos *jobs*
M1	4, 3, 1, 6, 2, 5
M2	2, 4, 6, 5, 3, 1
M3	3, 1, 2, 5, 4, 6
M4	3, 6, 4, 1, 2, 5
M5	2, 5, 3, 4, 6, 1
M6	3, 6, 2, 5, 1, 4

Para entender melhor o modelo de simulação desenvolvido foram capturados três diferentes momentos da simulação, os quais são ilustrados a seguir com foco na máquina número três. Nas Figuras 2, 3 e 4, os números que aparecem acima dos módulos e sobre as linhas de conexão representam o número do *job*. Portanto a Figura 2 mostra que o *job* número um está aguardando, enquanto o *job* número três é encaminhado para processamento na máquina, de acordo com a sequência ótima do problema apresentada na Tabela 6.

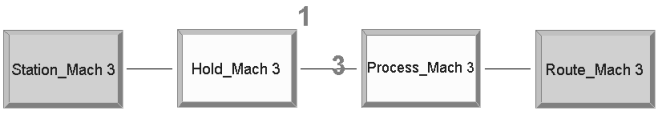

Figura 2: Processo na máquina 3 (instante 1)

Em um segundo momento, é possível verificar que o *job* número um é encaminhado para processamento e que um novo *job* número cinco aguarda para ser processado, conforme Figura 3.

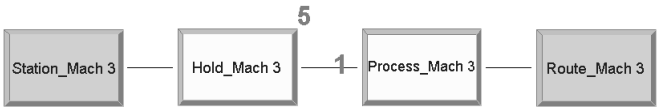

Figura 3: Processo na máquina 3 (instante 2)

Finalmente, na figura 4, observa-se que o recém-chegado *job* número dois toma prioridade em relação ao *job* cinco, respeitando a sequência de processamento definida para o problema nesta máquina (Tabela 6), ou seja, a sequencia dos *jobs* números três, um e dois.

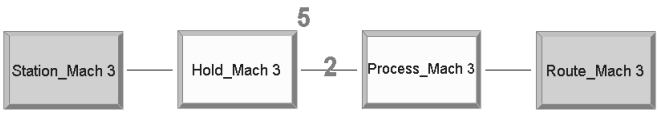

Figura 4: Processo na máquina 3 (instante 3)

O *job* número cinco, assim como os demais *jobs* para a máquina 3, é processado apenas após o processamento dos *jobs* três, um e dois, como programado. Resultados análogos para as outras cinco máquinas demonstram a confiabilidade do modelo de simulação.

Soluções factíveis

Uma importante característica do modelo de simulação proposto é a capacidade de tratar soluções não factíveis, que podem ser geradas pelo algoritmo genético. Como o modelo deve respeitar estritamente as prioridades pré-definidas, um indivíduo do AG pode conter uma sequência de prioridades impossível de ser executada nas máquinas por gerar espera simultânea de *jobs*. Um exemplo simples desse tipo de situação é o caso de um cenário com duas máquinas e dois *jobs*, com as rotas definida de acordo com o problema como {M2, M1} e {M1, M2}, para os *jobs* um e dois, respectivamente. Nesse caso, um conjunto de prioridades que indique o processamento dos *jobs* na ordem {1,2} e {2,1}, nas máquinas M1 e M2, faz com que o *job* 1 espere na fila da máquina M2 pelo *job* prioritário 2 enquanto este, por sua vez, espera na fila da máquina M1 pelo *job* prioritário 1.

O modelo apresentado neste trabalho trata essa situação associando um valor de penalidade, que é comumente utilizado dentro de técnicas de otimização, à soluções não factíveis ao problema. Como parâmetro de penalidade, foi utilizado a variável WIP (*Work-In-Process*), a qual refere-se ao número de entidades que permaneceram no sistema ou, em outras palavras, os *jobs* que não foram processados ao final da simulação.

Ao valor de aptidão gerado a cada iteração do modelo de simulação (*makespan*) é somado um fator igual a 100 vezes o WIP, podendo variar entre 200 e 600 para sequências não factíveis (em sequências factíveis todos os *jobs* são atendidos e o WIP é igual a zero). Essa expressão foi inserida no código VBA do modelo de simulação.

Resultados

Após a conclusão do modelo, a sequência ótima do problema FT06 foi avaliada. O resultado obtido com a simulação foi um valor de *makespan* de 55 unidades de tempo, conforme ilustrado na Figura 5 extraída diretamente das telas de relatório do ARENA®. Ressalta-se que esse é o valor do *makespan* ótimo reportado em (Hou *et al.*, 2011) e em diversos outros trabalhos recentes (Paneerselvam e Sumathi, 2011; Gao *et al.*, 2011; Qing-dao-er-ji e Wang, 2012). Conforme esperado, o valor do *makespan* é o mesmo que o tempo total máximo de permanência das entidades no sistema.

Tally

Interval	Average	Half Width	Minimum	Maximum
Makespan	51.0000	(Insufficient)	37.0000	55.0000

Time	Average	Half Width	Minimum	Maximum
NVA Time	0.00	(Insufficient)	0.00	0.00
Other Time	0.00	(Insufficient)	0.00	0.00
Transfer Time	0.00	(Insufficient)	0.00	0.00
Total Time	51.0000	(Insufficient)	37.0000	55.0000
VA Time	32.8333	(Insufficient)	25.0000	47.0000
Wait Time	18.1667	(Insufficient)	3.0000	30.0000

Figura 5: Resultados da simulação

5. Limitações e Trabalhos Futuros

Os autores veem como limitações desse trabalho o fato da validação ter utilizado apenas uma instância de JSSP (FT06). Apesar dos resultados indicarem que o modelo de fato representa o problema, a sua adaptação a outros JSSP determinísticos poderia garantir a robustez do modelo. Além disso, a minimização do tempo computacional durante a otimização não foi objetivo desse trabalho. Como a transferência de informação entre o modelo de simulação e o algoritmo genético é feita por arquivos de texto, acredita-se que é possível reduzir o tempo total de execução para atingir um resultado satisfatório alterando a forma de acoplamento.

Trabalhos futuros irão envolver analises sobre a melhor forma de representação do cromossomo para problemas de permutação como os JSSP, visando descartar soluções não

factíveis antes de uma avaliação pelo modelo de simulação, ou mesmo possibilitando gerar apenas sequências factíveis.

6. Reconhecimentos

Os autores agradecem a bolsa de estudos oferecida pelo Programa de Pós-Graduação em Engenharia de Produção da Universidade Nove de Julho – PPGEP/UNINOVE, bem como as valiosas sugestões dos revisores.

Referências

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

Baker, K. R. e Altheimer, D. (2012), Heuristic solution methods for the stochastic flow shop problem, *European Journal of Operational Research*, 216, 172-177.

Beasley, J. (2005), Operations research library. Available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt

Datta, D., Amaral, A. R. S. e Figueira, J. R. (2011), Single row facility layout problem using a permutation-based genetic algorithm, *European Journal of Operational Research*, 213, 388-394.

Fan, K. e Zhang, R. (2010), An analysis of research in job shop scheduling problem (2000-2009), IEEE International Conference on Advanced Management Science, Chengdu (China), 1, 282-288.

Gao, L. *et al.* (2011), An efficient memetic algorithm for solving the job shop scheduling problem, *Computers & Industrial Engineering*, 60, 699-705.

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. e Kan, A. H. G. R. (1979), Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 5, 287-326.

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, 989-998.

Hou, S., Liu, Y., Wen, H. e Chen, Y. (2011), A self-crossover genetic algorithm for job shop scheduling problem, IEEE International Conference on Industrial Engineering and Engineering Management, Taiyuan (China), 549-554.

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

Jinghua, W. e Mianzhou, C. (2010), Research of an improved genetic algorithm for job shop scheduling, International Conference on Measuring Technology and Mechatronics Automation, Changsha (China), 2, 1076-1078.

Kelton, W. D., Sadowski, R. P. e Sturrock, D. T. (2003), Simulation with Arena. New York: McGraw-Hill Professional.

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

Law, A. M., *et al.* (2009), How to build valid and credible simulations models, Proceedings of Winter Simulation Conference, Austin (USA), 24-31.

Lukaszewicz, P. P. (2005), Metaheuristics for job shop scheduling problem, comparison of effective methods, Master Thesis - Aarhus School of Business, Aarhus, 123p.

Paneerselvam, S. e Sumathi, S. (2011), Solution to the job shop scheduling problem using hybrid genetic swarm optimization based on (λ, 1)-interval fuzzy processing time, *European Journal of Scientific Research*, 64, 168-188.

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

Qing-dao-er-ji, R. e Wang, Y. (2012), A new hybrid genetic algorithm for job shop scheduling problem, *Computers & Operations Research*, 39, 2291-2299.

Raajan, P. R. M., Surekha, P. e Sumathi, S. (2010), A methodology to schedule and optimize job shop scheduling using computational intelligence paradigms, International Conference on Intelligent Control and Information Processing, Dalian (China), 809-814.

Rockwell Automation. (2007), Arena User's Guide. Milwaukee: Rockwell Software. Available at http://literature.rockwellautomation.com/idc/groups/literature/documents/um/arena-um001_en-p.pdf

Slack, N., Chambers, S. e Johnston, R. (2007), Operations management. Madrid: Prentice Hall.

Su, C. e An, Z. (2010), Job-shop scheduling considering dispatching rules of machines and material handling devices based on simulation, International Conference on Computer Application and System Modeling, Taiyuan (China), 8, 550-553.

Tavakkoli-Moghaddam, R. e Daneshmand-Mehr, M. (2005), A computer simulation model for job shop scheduling problems minimizing makespan, *Computers & Industrial Engineering*, 48, 811–823.

Wall, M. (1996), GALIB: A C++ library of genetic algorithm components. Mechanical Engineering Departament, Massachussetts Institute of Technology. Available at http://lancet.mit.edu/ga/dist/

Wang, S. e Liu, M. (2013), A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem, *Computers & Operations Research*, 40, 1064-1075.

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

Wolpert, D. H. e Macready, W. G. (1995), No free lunch theorems for search. Technical Report SFI-TR-95-02-010. Santa Fe: Santa Fe Institute.