LUCAS AUGSTEN GALVÃO

Desenvolvimento de um ambiente de simulação para o problema de dimensionamento do *time buffer* no método DBR em ambiente *flow shop* genérico com produção MTO

São Paulo

2017

LUCAS AUGSTEN GALVÃO

Desenvolvimento de um ambiente de simulação para o problema de dimensionamento do *time buffer* no método DBR em ambiente *flow shop* genérico com produção MTO

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenheiro de Produção

São Paulo
2017

LUCAS AUGSTEN GALVÃO

Desenvolvimento de um ambiente de simulação para o problema de dimensionamento do *time buffer* no método DBR em ambiente *flow shop* genérico com produção MTO

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de Engenheiro de Produção

Orientador: Prof. Dr. Marco Aurélio de Mesquita

São Paulo

2017

FICHA CATALOGRÁFICA

Galvão, Lucas

Desenvolvimento de um ambiente de simulação para o problema de dimensionamento do time buffer no método DBR em ambiente flow shop genérico com produção MTO / L. Galvão -- São Paulo, 2017.

136 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1.Controle da Produção 2.Simulação I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II.t.

AGRADECIMENTOS

Primeiramente, dedico este trabalho aos meus pais e minha irmã por todo o apoio e carinho sempre presentes durante estes anos, bem como pelos ensinamentos de vida. Também agradeço por sempre se esforçarem para que eu tenha usufruído da melhor educação possível, o que me é e sempre foi fonte de motivação.

Ao professor Marco Aurélio Mesquita, pela atenção, suporte e ensinamentos prestados para o desenvolvimento e aperfeiçoamento deste trabalho.

Ao corpo docente, direção e demais funcionários da Universidade de São Paulo, por me propiciar todas as oportunidades de aprendizagem e desenvolvimento que tive durante o período de graduação.

E a todos que aqui não foram mencionados, mas fizeram parte da minha formação profissional e acadêmica, meu profundo agradecimento.

“A ciência não é uma ilusão, mas seria uma ilusão acreditar que poderemos encontrar noutro lugar o que ela não nos pode dar.”

(Sigmund Freud)

RESUMO

O DBR (*drum buffer and rope*) é um método de controle da produção baseado na teoria das restrições que busca explorar o gargalo de um sistema de produção de forma a evitar sua ociosidade e maximizar a produtividade do sistema. O dimensionamento do *time buffer* é um dos grandes desafios do DBR e pouca importância tem sido dada a este problema na literatura atual. Desta maneira, o presente trabalho tem como objetivo central a elaboração de um ambiente de simulação por eventos discretos que permita o estudo do problema do dimensionamento do *time buffer* na aplicação do método de controle DBR. O estudo focou na simulação de um ambiente *flow shop* genérico dentro da lógica de produção MTO (*make-to-order*) e buscou avaliar o impacto de certos parâmetros do ambiente de simulação em conjunto com o *time buffer* em indicadores de desempenho escolhidos. Para a realização deste trabalho, realizou-se uma extensa revisão bibliográfica sobre o assunto, seguida pela concepção conceitual do modelo e desenvolvimento do modelo de simulação de acordo com esta concepção. Após a etapa de testes para verificação, o modelo foi utilizado para simular uma série de cenários. Os resultados mostraram que o desempenho do DBR está diretamente atrelado a um correto dimensionamento do *time buffer*. Por sua vez, este dimensionamento depende das características ou parâmetros do ambiente de simulação. Desta maneira, a simulação por eventos discretos se mostrou uma ferramenta poderosa na condução do dimensionamento do *time buffer*.

Palavras-chave: Controle da produção. Teoria das Restrições. DBR. Dimensionamento do *time buffer*. Simulação por eventos discretos.

ABSTRACT

DBR (drum buffer and rope) is a production control technique based on the Theory of Constraints, which explores the system's bottleneck to avoid idleness and maximize the performance of the production system. The determination of the time buffer size is a critical step in the DBR application, but little attention has been given to this issue in the current scientific literature. Therefore, the objective of the present study is the development of discrete events simulation environment to support the study of the time buffer size determination in DBR application. The study focused on the general flow shop routing configuration and MTO (make-to-order) strategy to assess the influence of certain configuration parameters in connection with the time buffer size over selected performance indicators. The methodology started with an extensive literature review followed by the conceptual design of the simulation model and the operational model development. After a verification period, the model was used to conduct a set of simulations with different scenarios. The simulation output was extensively analyzed in order to draw conclusions. The results showed that the DBR performance was directly related to the adjustment of the time buffer size. In addition, the time buffer size determination depends on a series of system configuration parameters. Thus, the discrete events simulation was found to be a powerful tool in the study of the time buffer size determination.

Keywords: Production control. Theory of Constraints. DBR. Time buffer sizing. Discrete events simulation.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATO	*Assemble to Order*
ConWIP	*Constant Work in Process*
CR	*Critical Ratio*
DBR	*Drum Buffer Rope*
EDD	*Earliest Due Date*
ETO	*Engineer to Order*
FIFO	*First In, First Out*
LIFO	*Last In, First Out*
LPT	*Longest Processing Time*
LS	*Least Slack*
LWQ	*Least Work Next Queue*
MRP	*Material Requirements Planning*
MTO	*Make to Order*
MTS	*Make to Stock*
PCP	Programação e Controle da Produção
SKU	*Stock Keeping Unit*
SPT	*Shortest Processing Time*
ToC	*Theory of Constraints*
WIP	*Work in Process*
WLC	*Workload Control*

SUMÁRIO

1. INTRODUÇÃO

1.1. Motivação

O *flow shop* genérico é uma linha de produção com múltiplos estágios, tendo cada estágio uma ou múltiplas máquinas idênticas. A linha produz múltiplos produtos em lotes, todos com o mesmo fluxo de produção. Em outras palavras, todos os *jobs* (ordens de produção) percorrem a linha no mesmo sentido, não necessariamente passando por todos os estágios. Desta forma, é um ambiente de produção existente na grande maioria das linhas de produção, podendo ser a linha como um todo ou somente uma parte desta (ENNS, 1995).

A estratégia de produção MTO (*make to order*) é aquela em que o pedido, ou a realização do ato de venda, é realizado antes do início da produção de determinado produto. É uma das estratégias de produção mais comumente empregadas em sistemas de produção, sendo extremamente útil principalmente para empresas com um nível maior de sofisticação, onde há a probabilidade de customização (KRAJEWSKI & RITZMAN, 1996).

O método DBR (*drum, buffer and rope*) é um dos métodos de controle da produção mais conhecidos e estudados recentemente. Ele se baseia no recurso gargalo do sistema de produção para realizar o controle da produção de forma a evitar ociosidade no recurso gargalo e elevado estoque em processo (UMBLE & SRIKANTH, 2002).

A ideia básica do DBR é controlar a liberação de ordens de tal forma que o recurso gargalo tenha sempre uma fila (estoque) de ordens para processar, evitando risco de parada por falta de material. Por outro lado, a liberação das ordens deve garantir que o estoque não seja desnecessariamente elevado, ao ponto de prejudicar a produtividade da linha. A taxa de liberação das ordens adequada é representada pelo ***drum*** (tambor) e o sinal de liberação que vem do ***buffer*** (pulmão) para o início do processo é representado pela ***rope*** (corda).

Para a efetiva implementação do método DBR, existem certos desafios que necessitam ser considerados e que são foco de diversas pesquisas e artigos científicos sobre o tema. Os principais desafios para a implementação do DBR são: identificação do recurso gargalo, programação da fila de entrada, dimensionamento do *time buffer* e definição das datas de entrega para os clientes (aplicável somente a sistema MTO).

Portanto, um dos pilares da aplicação do método DBR consiste em dimensionar o *time buffer*, ou pulmão da restrição, que consiste na antecedência programada para a chegada de *jobs* no gargalo. O dimensionamento do *time buffer* é responsável por regular a ocupação do recurso gargalo do sistema, controlando o ritmo do sistema de produção, como prega a teoria das restrições (DE SOUZA, 2005).

Isto é fator determinante para o desempenho operacional de uma linha de produção, impactando diretamente a produtividade, a capacidade de suprir a demanda, o custo de produção e a necessidade de capital para girar a operação de um determinado sistema de produção.

Enquanto valores muito baixos de *time buffer* podem resultar em perda de produtividade e *default* de ordens de produção, valores elevados levam para o caminho contrário. Porém, estes valores elevados geram um aumento de *jobs* em processamento no sistema de produção que pode inviabilizar operacionalmente a produção, devido ao custo de estoque elevado gerado por esta prática, que trará a empresa a necessidade de capital de giro.

Além disso, um número excessivo de *jobs* no sistema pode fazer com que pedidos com maior urgência de atendimento não possam ser atendidos, devido ao número excessivo de *jobs* que já estão no sistema e tem prioridade de processamento. Um elevado WIP pode ser prejudicial para a capacidade de entrega de um sistema de produção por aumentar o lead time de *jobs*, como demonstra a Lei de Little (1).

$$WIP = \lambda * Lead\ time \tag{1}$$

Desta maneira, a etapa de dimensionamento do *time buffer* durante a aplicação do método DBR deve ser feita cuidadosamente, com sua devida importância, de forma a maximizar indicadores de desempenho operacionais e financeiros que estejam alinhados com os objetivos estratégicos da companhia. Embora existam métodos empíricos para o dimensionamento do *time buffer* em uma implantação do DBR (THURER et al., 2017; SCHRAGENHEIM & RONEN, 1990; DANIEL & GUIDE, 1997), esses métodos não buscam atingir um ponto ótimo em relação ao desempenho operacional e/ou financeiro, mas sim, uma solução conveniente para casos específicos de sistemas de produção.

Além disso, em sistemas reais, com maior grau de complexidade do ambiente de produção e/ou maior grau de imprevisibilidade maior da entrada de pedidos, uma solução empírica pode ser difícil de ser achada e soluções existentes elaboradas pelos autores citados acima podem ser ineficientes.

Com exceção dos métodos empíricos, o autor deste trabalho identificou pouca produção científica na literatura acerca de métodos que busquem otimizar o *time buffer* ou que discutam o impacto de variações de parâmetros do ambiente de produção sobre valores de *time buffer* que impactem positivamente na operação do sistema. Em geral, os artigos identificados na literatura abordam casos muito específicos, como o trabalho de Radovilsky (1997), que estuda o caso de um *job shop* puro com tempo de chegada de *jobs* necessariamente igual ao tempo de processamento dos estágios ou o trabalho de Ye & Han (2008), que estuda o caso de um *assembly shop*, trabalho que se mostra complexo em termos de aplicação prática.

Além disto, estes métodos limitam-se a indicadores de *performance* financeiros, pouco avaliando aspectos operacionais, como o tempo de produção e o atraso de entrega. Um maior detalhamento sobre a literatura relativa ao método DBR será apresentado adiante.

Zhang & Du (2015) produziram o artigo mais abrangente sobre o tema, permitindo a adaptação a variedades de sistemas de produção dentro do ambiente de produção *job shop* puro. A abordagem consiste na implantação de um modelo de otimização de programação linear que, com o auxílio de um modelo de simulação para o sistema de produção, permite a determinação do tamanho ótimo de *time buffer* que minimiza os custos de produção. Desta maneira, a criação de um modelo de simulação que permita a implementação do método DBR é uma alternativa viável e eficiente para o dimensionamento do *time buffer*.

A larga existência de sistemas de produção com ambiente de produção *flow shop* genérico e estratégia de produção MTO, aliado ao fato de não terem sido identificados na literatura estudos sobre o impacto de parâmetros operacionais sobre o tamanho do *time buffer* que traz melhores valores de indicadores de desempenho operacionais, como atraso de entregas, produtividade e *lead time*, é o que motivou o desenvolvimento deste trabalho.

Adicionalmente, a utilidade da simulação para a implantação prática do método DBR e para o estudo do dimensionamento do *time buffer* motivou a criação de um ambiente de simulação adaptável como método para estudar o dimensionamento do *time buffer* em diferentes cenários e o impacto de certos parâmetros neste dimensionamento.

1.2. Objetivos

O presente trabalho tem como objetivo principal estudar o problema de dimensionamento de *time buffer* sob a ótica do DBR em um ambiente de produção *flow shop* genérico com estratégia de produção MTO.

Para isto, os seguintes objetivos específicos foram definidos:

- Desenvolver um modelo de simulação genérico que represente a aplicação do método de controle DBR e permita o estudo de diferentes configurações possíveis de sua aplicação em um sistema *flow shop* genérico com estratégia de produção MTO.
- Estudar o impacto do tamanho do *time buffer* em alguns indicadores de desempenho operacional do sistema de produção, em cada cenário estudado.
- Estudar o impacto de certos parâmetros, como severidade do gargalo e regras de sequenciamento no comportamento dos indicadores de desempenho estudados, para diferentes configurações com tamanhos variados de *time buffer*.
- Estudar como o tamanho do *time buffer* ideal se comporta conforme se variam os parâmetros do sistema de produção, como posição do recurso gargalo, número de estágios de produção no sistema, severidade do gargalo, etc.

1.3. Estrutura do Trabalho

O presente trabalho será composto de sete capítulos. Conforme já visto, o presente capítulo discutiu a motivação e os objetivos para a realização deste trabalho.

O segundo capítulo trará uma revisão bibliográfica extensa sobre os temas relacionados ao presente trabalho, como: (i) estratégias de produção existentes e suas características e particularidades; (ii) configurações de ambientes de produção existentes, definindo-os e apresentando suas características; (iii) métodos de controle da produção mais conhecidos na literatura, bem como suas características e diferenças; (iv) revisão sobre o método DBR, destacando sua lógica de funcionamento e literatura existente, com destaque para o dimensionamento do *time buffer*; (v) revisão da literatura sobre métodos de simulação.

O terceiro capítulo trata da metodologia utilizada para o desenvolvimento do trabalho, destacando os passos dos capítulos seguintes. O quarto capítulo discute o desenvolvimento do modelo de simulação, descrevendo seu funcionamento, testes de verificação e validação, e as instâncias que foram utilizadas no capítulo cinco.

O quinto capítulo traz o método para a escolha das instânias de simulação e apresenta em detalhe os parâmetros de entrada dos cenários gerados. O capítulo seis apresenta os resultados obtidos com a simulação das instâncias geradas no capítulo cinco e uma análise comparativa crítica destes resultados.

O capítulo sete resume a análise feita no capítulo seis para subsidiar a conclusão do presente trabalho. Por fim, o também discute os próximos trabalhos que possam resultar do tema aqui tratado, dos resultados obtidos do trabalho ou do modelo de simulação elaborado.

Além destes sete capítulos, o código do modelo de simulação, bem como as planilhas e macros utilizadas de suporte ao modelo, estão anexados ao final do trabalho.

2. REVISÃO BIBLIOGRÁFICA

Antes da construção do modelo de simulação e execução dos experimentos, é necessário apresentar uma revisão bibliográfica sobre o tema "controle da produção" e sobre o método DBR. As técnicas abordadas na literatura sobre o dimensionamento do *time buffer* são também discutidas.

Também é de fundamental importância definir conceitualmente o ambiente e a estratégia de produção, bem como os termos utilizados no desenvolvimento do presente trabalho. Todos estes aspectos serão discutidos no presente capítulo.

2.1. Estratégias de Produção

Os sistemas de produção de empresas de manufatura guardam semelhanças entre si. Portanto, estes sistemas são muitas vezes agrupados com base nestas características para fins de estudo e aplicação de técnicas específicas de programação e controle da produção.

Com este intuito, diversos autores apresentaram diferentes classificações para agrupar sistemas de produção semelhantes. Uma das classificações mais famosas é a classificação reproduzida por Krajewski & Ritzman (1996), que se refere às estratégias de produção para os sistemas de produção, dividindo-as em três grupos: MTS (*make-to-order*), ATO (*assemble-to-order*) e MTO (*make-to-order*).

Posteriormente, Pires (2004) destaca a existência de uma quarta estratégia, de forma a complementar a classificação apresentada por Krajewski & Ritzman (1996): a estratégia de produção ETO (*engineering-to-order*). Segue abaixo breve resumo descritivo das principais características destas quatro estratégias.

2.1.1. MTS (*"make to stock"*)

Krajewski & Ritzman (1996) definem a estratégia de produção MTS como sendo aquela em que a produção é voltada para estocagem, ou seja, em que a companhia produz itens padronizados que passam pelo estoque e posteriormente são destinados para venda.

Pires (2004) destaca que a interferência do cliente no processo produtivo é praticamente inexistente, com exceção da pesquisa de mercado, uma vez que a programação da produção é feita por previsão de demanda em sistemas MTS.

Segundo Arnold (1999), a estratégia de produção MTS é aquela que permite o menor grau de customização de produtos, mas que possui menor tempo de entrega (*lead time*) em comparação com as demais estratégias de produção. Nestas estratégias, o cliente possui pequeno envolvimento direto no projeto do produto. Por este motivo, a estratégia de produção MTS é muito empregada na produção de produtos mais padronizados, com vendas em massa e pouca customização.

2.1.2. ATO (*"assemble to order"*)

Segundo Krajewski & Ritzman (1996), a estratégia de produção ATO é aquela em que a empresa produz componentes padronizados que, após a realização efetiva do processo de venda, são combinados entre si para configurar o produto final conforme pedido pelo cliente. Em outras palavras, o processo de produção começa antes da venda, diferentemente da produção MTS, porém acaba após a venda, com a montagem de componentes padronizados.

Vale ressaltar que para um sistema de produção ser considerado ATO, não há a necessidade de todos os componentes serem pré-fabricados, mas apenas aqueles de maior uso. Pires (2004) destaca que em um sistema ATO, apesar do estoque de produtos acabados ser pequeno, comparativamente a estratégia MTS, ainda há elevado número de peças em estoque, no caso, principalmente no estoque intermediário, formado pelos componentes pré-fabricados.

Segundo Arnold (1999), na estratégia de produção ATO, a participação do cliente no processo produtivo se limita à configuração do produto final, ou seja, seleção do conjunto de componentes para a etapa de montagem. A seleção dos componentes que serão pré-fabricados é realizada com base no histórico de pedidos.

Arnold (1999) destaca que o tempo para entrega no sistema ATO é reduzido, quando comparado com as demais estratégias de produção (exceção feita a estratégia MTS), pois se limita ao tempo de montagem e fabricação de componentes específicos. Esta estratégia também permite certo nível de customização, uma vez que oferece ao cliente um leque de possibilidades de combinação de componentes.

2.1.3. MTO (*"make to order"*)

Segundo Krajewski & Ritzman (1996), a estratégia de produção MTO é aquela em que a empresa começa a produzir o pedido apenas após a concretização da venda, ou seja, o processo de venda antecede qualquer atividade relacionada à produção, consistindo em produção sobre encomenda. O que vai ser produzido pode variar desde um produto inédito, produzido de forma customizada, até um produto escolhido entre um conjunto de opções (PIRES, 2004).

Arnold (1999) afirma que a participação do cliente em relação às atividades produtivas em um sistema MTO é maior, uma vez que as atividades a serem executadas no sistema de produção dependem diretamente do pedido dos clientes.

Além disso, Arnold (1999) aponta que o projeto dos produtos em uma estratégia MTO pode sofrer influência, ou até mesmo se originar dos contatos iniciais com o cliente. Contudo, a etapa de produção só se inicia após o recebimento do pedido formal.

Segundo Arnold (1999), o tempo até a entrega em um sistema com estratégia MTO é proporcionalmente maior do que nas demais estratégias previamente apresentadas. Porém, este maior tempo até a entrega é o custo a ser pago com o ganho de customização do pedido que esta estratégia de produção traz para determinada companhia.

2.1.4. ETO (*"engineer to order"*)

A estratégia de produção ETO consiste em uma extensão da abordagem MTO, que se diferencia desta, pois a etapa de projeto do produto se inicia após a formalização da venda, ou seja, o projeto do produto, assim como a produção, é feito sob encomenda. Nesta estratégia de produção, os produtos são altamente customizados, variando conforme a especificação do cliente (PIRES, 2004).

Arnold (1999) destaca que o cliente é altamente envolvido no projeto dos produtos e o estoque de materiais em um sistema baseado na estratégia ETO normalmente só é adquirido até que haja necessidade de sua utilização. Desta forma, esta estratégia de produção apresenta nível reduzido de estoques de matéria-prima, produtos em processo ou produtos acabados.

Como consequência das características únicas de um sistema de produção ETO, temos o maior tempo até a entrega entre as estratégias de produção, pois passa a incluir a etapa de projeto dos produtos. Porém, a estratégia ETO é aquela que permite maior customização de produtos, e, portanto, é a mais apropriada para o desenvolvimento de projetos únicos e diferenciados, geralmente de grande porte, como navios, aeronaves, entre outros (ARNOLD, 1999).

A Tabela 1 resume as principais características de cada estratégia de produção, com base nos estudos feitos por Krajewski & Ritzman (1996), Arnold (1999) e Pires (2004).

Tabela 1 - Principais características das estratégias de produção

Estratégia de Produção	Grau de Customização dos Produtos	Tempo até a Entrega	Principal Tipo de Estoque
MTS	Baixo	Baixo	Produtos Acabados
ATO	Médio	Médio	Produtos em Processamento
MTO	Elevado	Elevado	Matéria Prima
ETO	Produtos únicos	Elevado	-

Fonte: Elaborado pelo autor

2.2. Ambientes de Produção

Além da estratégia de produção, outro tema essencial que delimita o estudo de sistemas de produção é a configuração do ambiente de produção. Cada ambiente de produção permite um funcionamento diferente do caminho do *job* no sistema de produção e é elemento determinante para a implementação de ferramentas de controle da produção. Ou seja, determina o fluxo de materiais na fábrica. O ambiente de produção está diretamente associado a estratégia de produção, uma vez que determinadas estratégias possuem uma tendência maior de estarem associados a tipos de ambiente de produção (MTS com *flow shop* puro, MTO com *flow shop* genérico ou *job shop*).

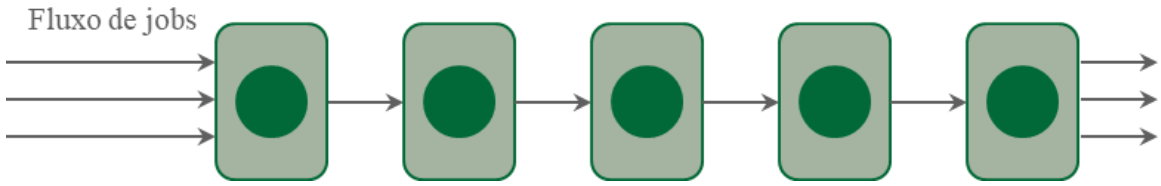
Pinedo (2002), em sua obra sobre *scheduling* em sistemas de produção, delimita os diferentes tipos de ambientes de produção e suas características e diferenças. Para a finalidade deste trabalho, procurou-se definir e apresentar as principais características que diferenciam cada ambiente de produção, segundo o modelo proposto por Pinedo (2002), adicionando a definição de Enns (1995) e Thurer et al. (2017) para *flow shop* genérico.

2.2.1. *Flow Shop* Puro

O *flow shop* puro é o ambiente de produção em linha mais básico. Consiste em um sistema de produção formado por *n* estágios (ou etapas produtivas) em série, em que cada *job* percorre cada um destes estágios, na mesma sequência, até a conclusão do processo. Todos os *jobs* devem percorrer o mesmo caminho em um *flow shop* puro. Cada estágio do ambiente de produção é composto por apena uma máquina (PINEDO, 2002).

A Figura 1 ilustra um *flow shop* puro com um total (*n*) de 5 estágios.

Figura 1 - *Flow shop* puro com 5 estágios



Fonte: Elaborado pelo autor

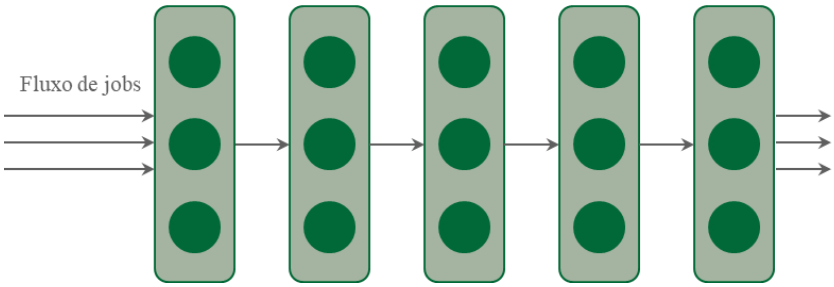
2.2.2. *Flow Shop* Flexível

O *flow shop* flexível é uma generalização do *flow shop* puro, de forma a permitir mais de uma máquina em cada estágio. Consiste em um sistema de produção formado por *n* estágios (ou etapas produtivas) em série, em que o *job* deve percorrer cada um destes estágios até a conclusão do processo produtivo. Da mesma forma que o *flow shop* puro, todos os *jobs* devem percorrer o mesmo caminho (PINEDO, 2002). Conforme discutido por este autor, a diferença de um *flow shop* flexível para o puro é que, diferentemente do puro, cada estágio do ambiente de produção pode ser composto por mais de uma máquina trabalhando em paralelo.

Em outras palavras, no *flow shop* flexível, cada estágio pode possuir uma quantidade *m* de máquinas idênticas, de forma que um *job* que passa por certo estágio pode ser processado por qualquer uma das *m* máquinas, sendo processado uma única vez apenas em cada estágio. Desta forma, o *flow shop* flexível permite que em um mesmo estágio, *m jobs* sejam processados simultaneamente. O número de máquinas em cada estágio não necessita ser igual para todos os estágios para que o sistema seja enquadrado como *flow shop* flexível (PINEDO, 2002).

A Figura 2 mostra um *flow shop* flexível com $n = 5$ estágios e $m = 3$ máquinas para cada estágio.

Figura 2 - *Flow shop* flexível com 5 estágios e 3 máquinas por estágio



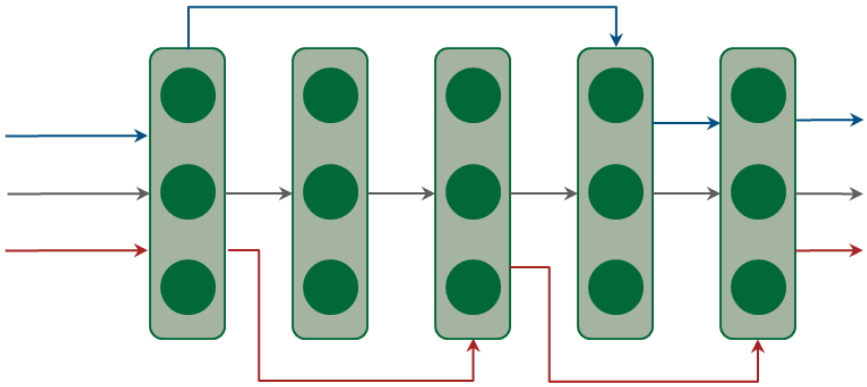
Fonte: Elaborado pelo autor

2.2.3. *Flow Shop* Genérico

O modelo de *flow shop* flexível foi modificado por Enns (1995) e, posteriormente, utilizado por Thurer et al. (2017), de forma a torná-lo ainda mais genérico do que o conceito previamente apresentado por Pinedo (2002).

De acordo com estes autores, o *flow shop* genérico possui uma única diferença em relação ao *flow shop* flexível. A diferença é o conceito de que não há a necessidade de um *job* passar necessariamente por todas as estações, ou seja, cada *job* pode ter seu próprio caminho, desde que a ordem do fluxo dentro do sistema seja mantida (Figura 3). Ou seja, o sentido do fluxo é o mesmo, mas determinado *job* pode pular um ou mais estágios. Para fins deste trabalho, utilizaremos o conceito apresentado por Enns (1995) e Thurer et al. (2017) para *flow shop* genérico como ambiente de produção.

Figura 3 - *Flow shop* genérico com 5 estágios com 3 máquinas por estágio e 3 padrões de fluxo



Fonte: Elaborado pelo autor

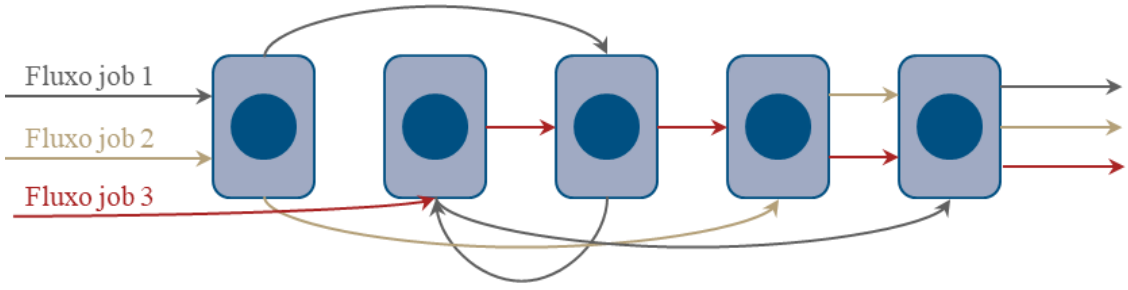
2.2.4. *Job Shop* Puro

De acordo com Pinedo (2002), em um *job shop* puro, da mesma forma que no *flow shop* puro, o sistema de produção é formado por *n* estágios, cada um destes com apenas uma máquina. A diferença existente é que no *job shop* puro, os *jobs* seguem rotas diferentes. Portanto, não há um caminho pré-determinado para o conjunto de *jobs*, mas sim um sequenciamento diferenciado para cada *job* que entra no sistema.

Há variações do modelo de *job shop* puro em que o *job* deve passar obrigatoriamente por cada estágio, podendo passar mais de uma vez por certo estágio, e variações em que o *job* deve passar apenas uma vez por cada estágio (PINEDO, 2002).

A Figura 4 exemplifica um *job shop* puro com $n = 5$ estágios e três configurações de fluxo. É possível notar que cada *job* percorre seu próprio roteiro, gerando múltiplos fluxos de produção.

Figura 4 - *Job shop* puro com 5 estágios



Fonte: Elaborado pelo autor

2.2.5. *Job Shop* Flexível

O *job shop* flexível é uma generalização do *job shop* puro. Consiste em um sistema de produção formado por *n* estágios (ou etapas produtivas) em série, em que cada *job* possui sua própria rota, ou seja, não há rota única (PINEDO, 2002).

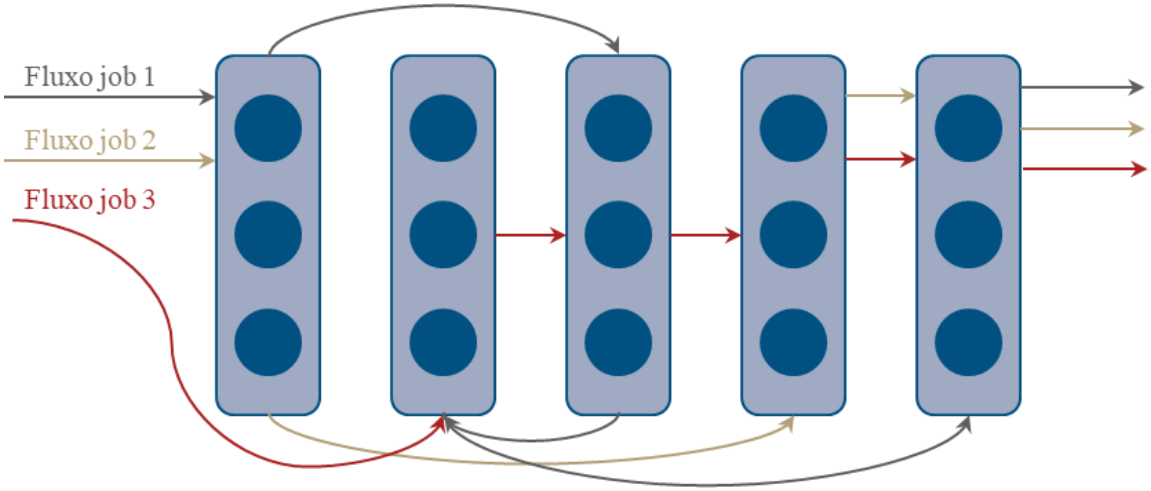
Conforme ressaltado por Pinedo (2002), a diferença de um *job shop* flexível para o puro é que, diferentemente do puro, cada estágio do ambiente de produção pode ser composto por mais de uma máquina trabalhando em paralelo.

Desta forma, cada estágio pode possuir uma quantidade m_j de máquinas idênticas, de maneira que um *job* que passa por certo estágio pode ser processado por qualquer uma das m_j máquinas, sendo processado uma única vez apenas em cada passagem pelo estágio. Portanto, o *job shop* flexível permite que em um mesmo estágio, até m_j *jobs* sejam processados simultaneamente (PINEDO, 2002).

Há variações do modelo de *job shop* flexível em que o *job* deve passar obrigatoriamente por cada estágio, podendo passar mais de uma vez por certo estágio, e variações em que o *job* deve passar apenas uma vez por cada estágio (PINEDO, 2002).

A Figura 5 destaca um *job shop* flexível com $n = 5$ estágios e $m = 3$ máquinas por estágio, além do caminho percorrido por três *jobs*. É possível notar que cada *job* percorre seu próprio caminho, sem um direcionamento único para o ambiente de produção.

Figura 5 - *Job shop* flexível com 5 estágios e 3 máquinas



Fonte: Elaborado pelo autor

2.2.6. Open Shop

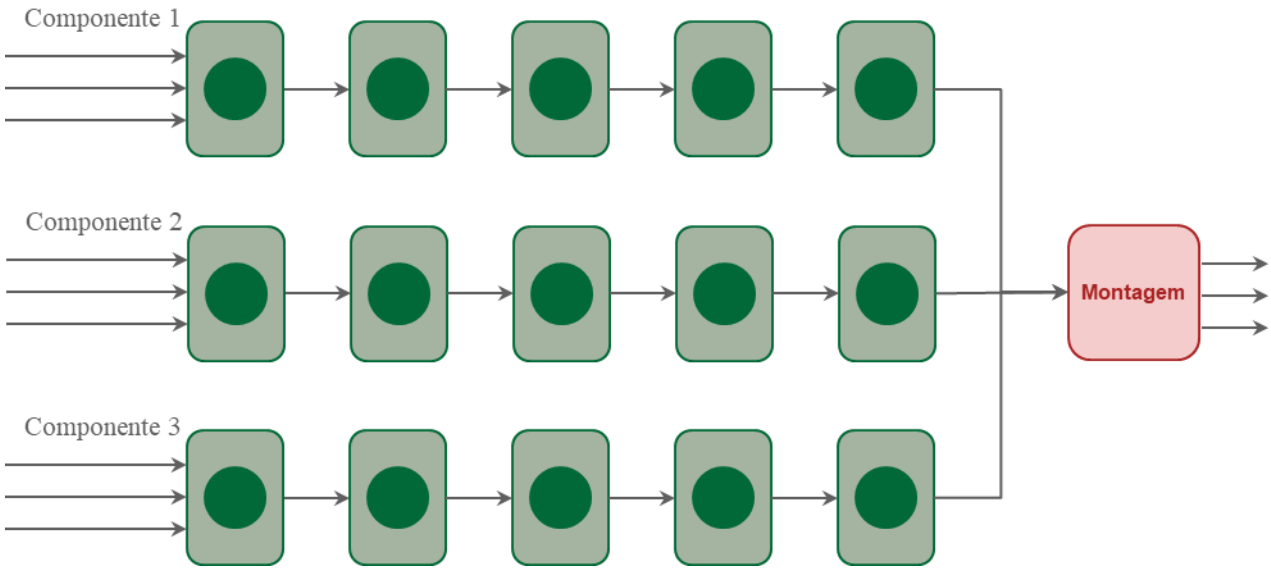
O *open shop* consiste na generalização do *job shop* flexível, de forma a permitir que um determinado *job* não necessite passar por todos os *n* estágios do processo produtivo (tempo de processamento zero é permitido). De resto, guarda as mesmas características do ambiente de produção do *job shop* flexível. É o ambiente de produção mais genérico existente, em que cada *job* possui seu próprio caminho podendo passar ou não por determinado estágio (PINEDO, 2002).

2.2.7. Assembly Shop

Assembly shop é um ambiente de produção em que certo produto é formado pela união de módulos e componentes após uma etapa de montagem, sendo que cada componente possui um processo de produção em *flow shop* ou *job shop*. Portanto, é composto por uma série de linhas de produção para cada componente, com uma etapa final de montagem (PINEDO, 2002).

A Figura 6 ilustra um *assembly shop* formado por três linhas, com montagem dos três componentes como etapa final, para a formação do produto. Cada linha possui características de um *flow shop* puro para cada componente, com $n = 5$ estágios.

Figura 6 - *Assembly shop* com 3 linhas



Fonte: Elaborado pelo autor

2.3. Métodos de Controle da Produção

Desde o início da década de 80, com o crescimento do estudo de como programar e controlar a produção sistemas de produção, diversas técnicas para planejamento e controle da produção vêm surgindo na literatura. Algumas destas técnicas destacam-se, sendo alternativas viáveis e eficientes para o planejamento e controle da produção (GUPTA & SNYDER, 2009).

Para o escopo deste trabalho, serão detalhadas a seguir as principais técnicas existentes que se destacam no âmbito do controle da produção, bem como desenvolvimentos recentes que tratam sobre o tema.

2.3.1. MRP: *Backward Scheduling*

O MRP (*"material requirements planning"*) é a técnica de programação e controle da produção mais utilizada atualmente no planejamento e controle da produção. A grande vantagem trazida pela implementação do MRP é a possibilidade de se lidar com ambientes com elevado número de SKUs de forma eficiente, diferentemente de grande parte dos sistemas de PCP existentes (FERNANDES et al., 2007).

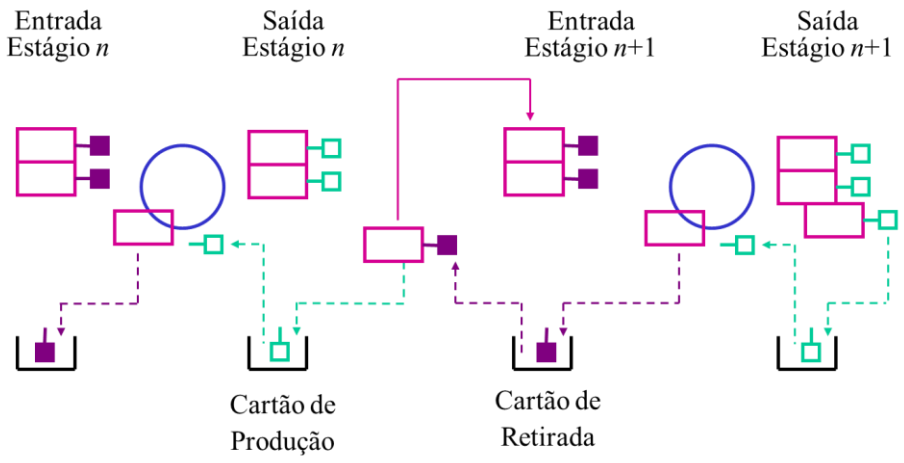
O MRP não é um sistema eficiente no dimensionamento de *lead time* e em programação de curto prazo (FERNANDES et al., 2007). Em termos de controle da produção, o MRP normalmente é um sistema empurrado, porém, existem mecanismos para o controle do estoque em processo. A programação é baseada em métodos de previsão de demanda e é formada a partir de lotes que visam minimizar o estoque em processo, garantindo o cumprimento de prazos. Portanto, o mecanismo de controle utilizado no MRP se mostra susceptível à variabilidade e imprevistos (CORRÊA & GIANESI, 1993).

2.3.2. Kanban

Kanban é uma técnica de controle da produção japonesa correlacionada ao famoso método JIT (*"Just-In-Time"*). *Kanban*, que em português significa "cartão", consiste em uma técnica de controle da produção que utiliza cartões para formalizar a movimentação de materiais dentro do sistema de produção (PRICE et al., 1994). Conforme destacado por este autor, é um método de controle da produção que gera em um sistema puxado, pois determinado estágio só irá operar o *job* se certo item for retirado do estoque entre esta etapa e a etapa posterior (produção para reposição de estoque). Desta forma, a produção será sempre regida pelo final do sistema produzido.

Existem diversas vertentes para o método *Kanban*. O *Kanban* composto por dois tipos de cartão é uma das vertentes mais tradicionais. O primeiro é um cartão de retirada (*"Withdrawl Kanban"*), que formaliza o pedido de material de um estágio de produção posterior para seu precedente e autoriza a movimentação de *jobs* entre estações. O segundo é um cartão de produção (*"Production Kanban"*), que formaliza o pedido de produção para determinada estação com o objetivo de repor o estoque movimentado, autorizando a operação na estação (PRICE et al., 1994) (Figura 7).

Figura 7 - *Kanban* de dois cartões



Fonte: Elaborado pelo autor

Embora tenha se mostrado uma técnica efetiva e muito utilizada para controle da produção, com extensa produção bibliográfica associada, o método *Kanban* tem recebido críticas, sendo considerado adequado para produção repetitiva (MTS), mas inadequado para produção intermitente (PRICE et al., 1994).

Por exemplo, Lambrecht & Decaluwe (1988) afirmam que o método *Kanban* reage tardiamente ao aparecimento de problemas de um sistema de produção. Adicionalmente, Gardiner et al. (1994) destacaram que o método *Kanban*, por bloquear a produção de produtos acabados quando o estoque final estiver completo, é um sistema menos susceptível a grandes incertezas associadas à geração de pedidos, o que é menos evidente no método DBR.

2.3.3. *Workload Control*

O método de controle da produção denominado *workload control*, ou WLC, é um método com mais de 30 anos de história. Sua principal característica é o uso de um mecanismo de entrada de *jobs* no sistema conectado ao nível de *jobs* já em processamento no sistema de produção (THURER et al., 2017).

O objetivo deste método consiste em nivelar a quantidade de *jobs* dentro do sistema, denominado "carga de trabalho", evitando ociosidade ou superlotamento do sistema. Apesar deste método possuir maior aplicação e ter resultados superiores em sistemas balanceados, sem um recurso gargalo evidente, há evidências de melhoria de performance com a adoção do método *workload control* em sistemas com um recurso gargalo evidente (THURER et al., 2017).

Thurer et al. (2017) mencionam que a utilidade do método *workload control* é maior em sistemas MTO, onde o uso de *buffer* permite maior proteção contra a variabilidade na chegada de *jobs* no sistema, bem como mantém níveis adequados de WIP no sistema. Os autores ressaltam que a técnica *workload control* é um método genérico, com diversas variações existentes na literatura sobre o tema. Por exemplo, existe a técnica *starvation avoidance*, que se comporta de forma similar ao DBR, ou seja, considerando a apenas a carga de trabalho antes do gargalo do sistema. Há também a técnica denominada *path aggregation*, introduzida por Fredendall et al. (2010), que controla a carga de trabalho em cada estação do sistema de produção.

Apesar das diversas técnicas existentes relativas ao método *workload control*, a mais conhecida e por mais vezes citada na literatura pelo autor é a técnica ConWIP (*"Constant Work In Process"*), que será detalhada no próximo tópico.

2.3.4. ConWIP

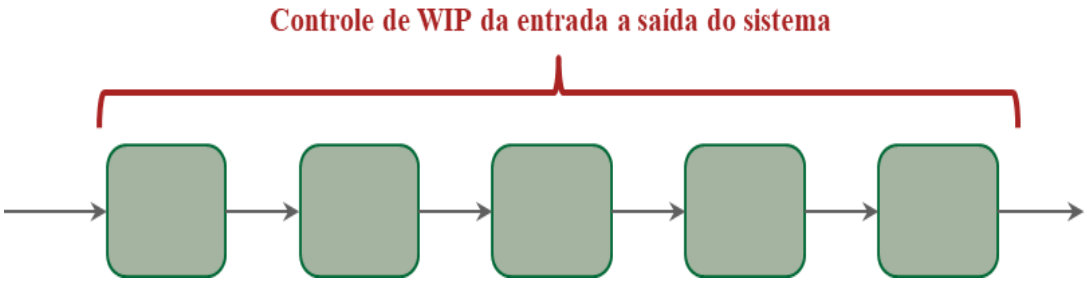
O método de controle da produção ConWIP é um caso específico do *workload control* introduzido por Spearman et al. (1990). Este método busca manter estável o nível de estoque em processo em todo o sistema de produção durante o período de operação (SPEARMAN et al., 1990).

O ConWIP monitora o número de *jobs* entre a primeira e a última estação na linha, esteja ele na fila ou sendo processado. Estabelece-se um nível máximo para esta carga de trabalho. Este valor não pode ser ultrapassado. Desta forma, pedidos gerados só entram no sistema quando a carga de trabalho for menor que este valor máximo, evitando a superlotação do sistema (SPEARMAN et al., 1990).

Spearman et al. (1990) ainda destacam a similaridade do ConWIP com o método *kanban* de controle de produção, afirmando que ambos buscam manter um nível constante de estoque em processo. Porém, enquanto o *kanban* busca manter esta constância de estação por estação, o ConWIP trata o sistema como um todo, buscando manter tal constância na linha, ou seja, entre a entrada e a saída do sistema.

Da mesma forma que o método *kanban*, o ConWIP também pode funcionar por meio de cartões, que conectam a saída de *jobs* do sistema de produção com a entrada de um novo *job* no sistema (SPEARMAN et al., 1990). Porém, sistemas computacionais também se habilitam a realizar o ConWIP (Figura 8).

Figura 8 - Método ConWIP de controle da produção



Fonte: Elaborado pelo autor

2.3.5. DBR

O método de controle da produção DBR (*"drum-buffer-rope")* é um método que segue a lógica introduzida por Goldratt & Cox (1986) em sua obra "A Meta": a Teoria das Restrições, ou ToC (*"Theory of Constrains"*). O método DBR parte do pressuposto de que existem apenas alguns estágios, dentro de um sistema de produção, que efetivamente possuem a capacidade de restringir a produção do sistema. Estes serão os estágios que efetivamente irão ser determinantes para o nível de produção global do sistema como um todo (UMBLE & SRIKANTH, 2002).

Por esta razão, o método DBR busca garantir que apenas nestes estágios limitantes a produção não seja interrompida durante a operação da fábrica. Desta maneira, o método DBR busca otimizar a produção da fábrica evitando que o gargalo do sistema de produção tenha ociosidade (UMBLE & SRIKANTH, 2002).

Devido à relevância deste tema para o desenvolvimento deste trabalho, um tópico a parte foi destinado para aprofundar este método, com o objetivo de descrever suas etapas, a literatura existente sobre o tema e a questão do dimensionamento do *buffer* ou pulmão do sistema, tema central do presente trabalho.

2.4. DBR: revisão da literatura

2.4.1. Relação com ToC

Como mencionado na última seção, o método DBR segue a lógica da Teoria das Restrições (ToC) de Goldratt & Cox (1986), que pauta seu mecanismo de programação e controle do sistema de produção com base no recurso gargalo do sistema, de forma a evitar ociosidade no gargalo.

Goldratt & Cox (1986) formularam um método em cinco passos que caracteriza a ToC:

- 1º Passo – Identificar o gargalo
- 2º Passo – Explorar ao máximo a capacidade do gargalo
- 3º Passo – Subordinar a programação e controle do resto do sistema de forma a evitar ociosidade no gargalo
- 4º Passo – Elevar a capacidade do sistema aumentando a produtividade do gargalo
- 5º Passo – Retornar ao 1º passo de forma a fechar o ciclo e balancear o sistema

Como resultado do método, a ToC busca constantemente balancear o sistema de produção de forma a eliminar recursos gargalos. Procura-se, então, implementar um ciclo contínuo de diagnóstico-melhoria até que o sistema esteja balanceado (DARLINGTON et al., 2015).

O DBR consiste em um método de programação e controle, ou seja, efetivamente busca implementar em um sistema de produção mecanismos capazes de realizar os três primeiros passos da ToC. O DBR se limita a estes três passos somente, uma vez que é apenas um mecanismo de controle, não promovendo melhorias no sistema de forma direta (WU et al., 2006).

Porém, indiretamente, o DBR é responsável pela promoção de melhorias operacionais, uma vez que identifica o gargalo do sistema e o ponto limitante a ser explorado para melhoria da capacidade produtiva. Isto permite a mensuração do impacto de cada ação promovida no sistema com o uso de indicadores de efetividade (WU et al., 2006).

De acordo com Wu et al. (2006), a utilidade do DBR diminui quando o sistema é bem balanceado, uma vez que dificulta a execução da primeira etapa. Além disso, o método DBR não se mostra muito efetivo em ambientes de produção repetitiva.

2.4.2. Lógica de Funcionamento

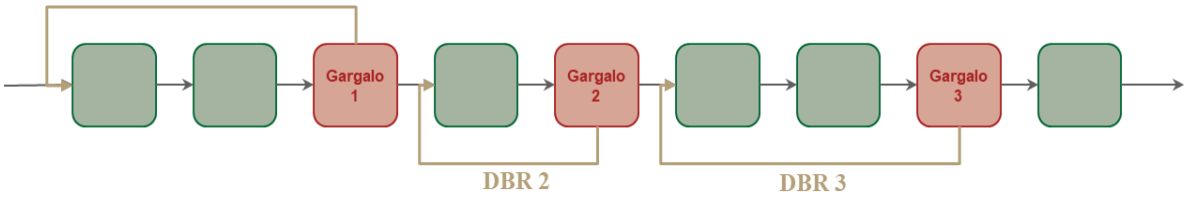
O método DBR pode ser dividido em três componentes, cada um diretamente relacionado com as três primeiras etapas do método ToC. O componente *drum*, ou tambor, que se relaciona a primeira etapa da ToC; o componente *buffer*, ou pulmão, que se relaciona com a segunda etapa da ToC, e o componente *rope*, ou corda, relacionado à terceira etapa da ToC (DE SOUZA, 2005).

O ***drum*** é o recurso com maior uso dentro de um sistema de produção, ou seja, é o recurso gargalo do sistema, que efetivamente controla a produtividade do sistema de produção. É de fundamental importância que não haja ociosidade no gargalo, pois não há forma de recuperar a produção perdida com a ociosidade do gargalo, diferentemente das outras operações do sistema (GOLDRATT & COX, 1986).

O ***buffer*** está associado ao WIP entre o recurso gargalo e a entrada de *jobs* no sistema com a função de proteger o recurso gargalo de variabilidades e incertezas, evitando sua ociosidade. Diferentemente de técnicas como o *kanban* e WLC, o *buffer* no DBR é apresentado em unidades de tempo e não em número de *jobs*, ou seja, consiste em um *time buffer* (DARLINGTON et al., 2015).

Em determinado sistema de produção, o DBR pode ser implementado de forma segregada para diversos segmentos do sistema de produção, conforme destacado por Wu et al. (2006). Desta forma, um sistema pode ser formado por mais de um *drum* e um *buffer* (Figura 9).

Figura 9 - DBR em sistema com múltiplos laços



Fonte: Adaptado de Wu et al. (2006)

Além disso, dependendo do ambiente de produção, o DBR pode ter dois outros tipos diferentes de *buffer* (DE SOUZA, 2005). O segundo tipo seria o *buffer* de montagem, aplicável apenas em *assembly shops*. O *buffer* consiste em WIP de *jobs* em linhas que não contém recursos gargalos, de forma a evitar que componentes vindos de linhas que contêm o recurso gargalo tenham de esperar a chegada de componentes provindos de outras linhas.

O terceiro tipo seria o *buffer* de mercado, aplicável apenas em estratégia de produção MTS, que consiste em um estoque de produtos acabados ao final do sistema de forma a evitar a falta de produtos para a venda, mas também o excesso de estoque em caso de queda de demanda (DE SOUZA, 2005).

O ***rope*** é o último componente do DBR, sendo o mecanismo que conecta o *buffer* com a entrada de *jobs* no sistema. O *rope* mantém, assim, o nível de WIP (em unidade de tempo) entre a entrada e o gargalo em um nível pré-determinado, permitindo a entrada de um novo *job* no sistema assim que o recurso gargalo (*Drum*) conclui uma operação (DARLINGTON et al., 2015).

Desta forma, o método de controle da produção via DBR é composto por três etapas básicas (GOLDRATT & COX, 1986):

1ª Etapa – A identificação do recurso gargalo dentro do sistema de produção. De Souza (2005) destaca que, em sistemas balanceados, esta etapa pode ser um desafio e pode ter múltiplos gargalos alternantes, dependendo do apoio de um sistema computacional especializado para a efetiva implementação do controle via DBR.

2ª Etapa – Consiste no dimensionamento de um *time buffer* que seja capaz de evitar a ociosidade no recurso gargalo e otimizar o volume de produção ou qualquer outra variável relevante ao sistema. Em recursos em que o *buffer* de montagem e mercado sejam aplicáveis, o seu dimensionamento também deve ser feito nesta etapa. Ela depende diretamente da primeira etapa, uma vez que só é possível dimensionar e controlar o *buffer* se sua posição no sistema for conhecida.

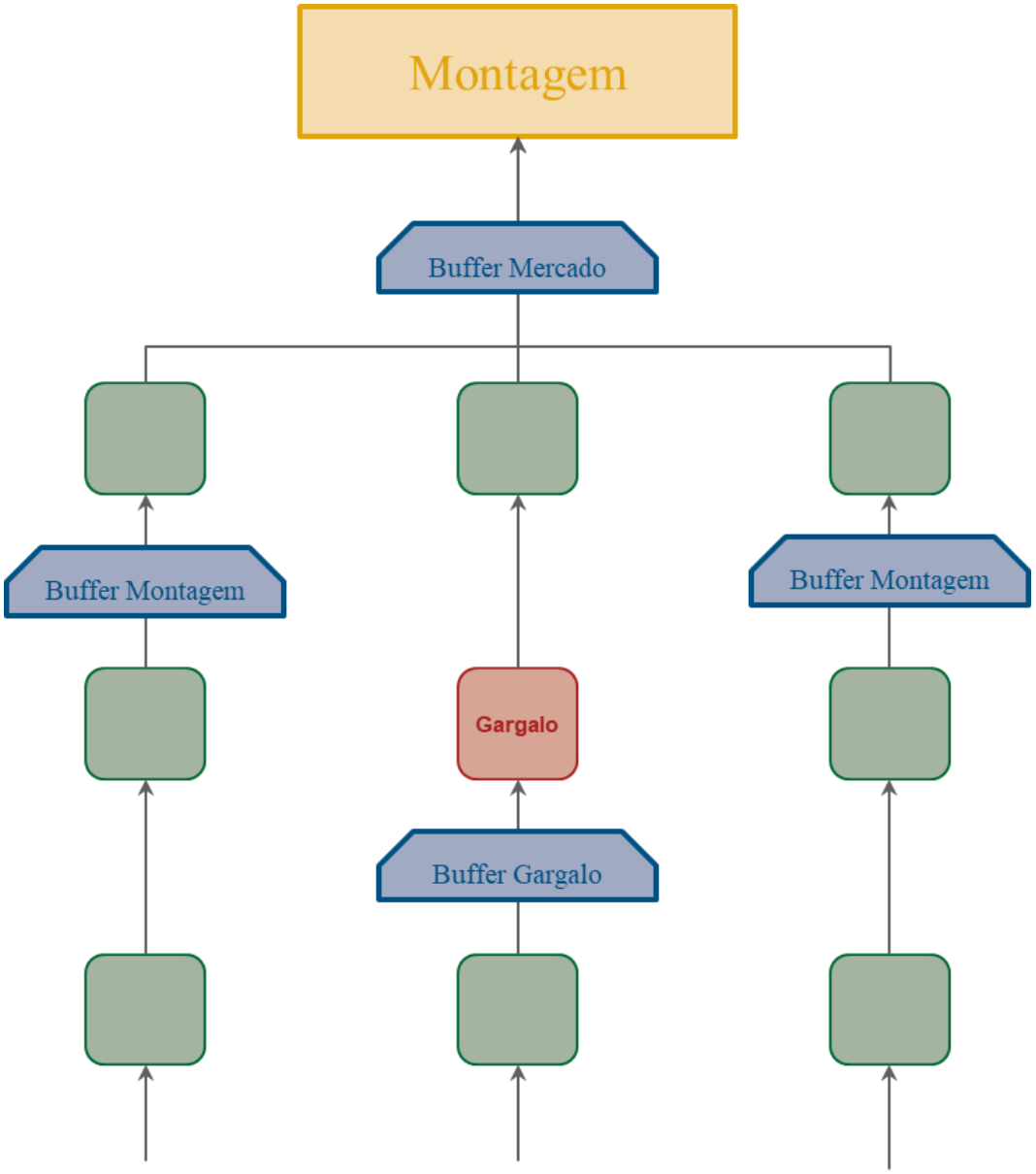
3ª Etapa – É a implementação do mecanismo que controla a entrada de *jobs* no sistema, quando o *time buffer* pré-determinado for maior que o WIP em unidade de tempo atual, entre o ponto de entrada e o gargalo.

Para sistemas MTO, ou MTS com tipos diferentes de *jobs*, na entrada de novo *job* no sistema, é importante determinar uma regra de sequenciamento que permita determinar qual *job* deve entrar no sistema, dentre de um leque de opções (DE SOUZA, 2005).

Nos últimos anos, diversos trabalhos foram publicados estudando o impacto das regras de sequenciamento em indicadores de desempenho de sistema com controle de estoque em processo. Os trabalhos mostraram que a determinação desta regra influencia consideravelmente parâmetros como atraso de pedidos e *lead time* médio do sistema (THURER et al., 2017). As regras de sequenciamento mais comuns serão apresentadas no próximo tópico.

A Figura 10 ilustra a aplicação do DBR em um ambiente de produção *assembly shop* com os três tipos de *buffer*, de acordo com Souza (2005): *buffer* de produção, *buffer* de montagem e *buffer* de mercado.

Figura 10 - Ilustração de DBR aplicado em *Assembly Shop* com produção MTS



Fonte: Adaptado de De Souza (2005)

2.4.3. Regras de Sequenciamento

Regras de sequenciamento, ou *dispatching rules*, são comumente utilizadas para ordenar a entrada de *jobs* contidos em um *pré-shop* para dentro do sistema de produção. O *pré-shop* é um conjunto de *jobs* ordenados, segundo determinada regra de sequenciamento, que aguardam o ativamento do mecanismo *rope* para liberação de entrada no sistema de produção (DA SILVA et al., 2012).

As principais regras de sequenciamento apresentadas por Silva et al. (2012), adaptadas de Gaither e Frazier (2001), Chan e Chan (2004), Suresh e Sridharan (2007), Tubino (2007) e Lustosa et al. (2008), são:

FIFO (*"First In, First Out"*) – o ordenamento de entrada de *jobs* no sistema segue a ordem de chegada dos *jobs* dentro do *pré-shop*. Logo, os *jobs* que chegam primeiro são os primeiros a sair. Esta regra é útil para minimizar o tempo entre a chegada/geração do *job* e a sua saída do sistema.

LIFO (*"Last In, First Out"*) – é a regra de sequenciamento inversa da regra FIFO, ou seja, o último *job* a ser gerado é aquele que irá adentrar primeiro no sistema de produção. Ou seja, os *jobs* são processados na ordem inversa de chegada.

SPT (*"Shortest Processing Time"*) – nesta regra, a entrada de *jobs* no sistema é ordenada pelo tempo total de processamento, do menor para o maior. A lógica desta regra é que *jobs* com menor tempo esperado de processamento tendem a sair mais rapidamente do sistema, de forma a agilizar o fluxo no sistema.

LPT (*"Longest Processing Time"*) – é a regra de sequenciamento inversa à regra SPT, ou seja, *jobs* com maior tempo estimado de processamento são os primeiros a entrar no sistema.

EDD (*"Earliest Due Date"*) – nesta regra de sequenciamento, os *jobs* entram no sistema com base no prazo de entrega (*due date*). A lógica é simples - *jobs* que precisam ser entregues primeiro devem ter prioridade na fila para entrada no sistema de produção.

LS (*"Least Slack"*) – nesta regra, o sequenciamento ocorre com base na diferença entre a data de entrega prometida e o tempo estimado de processamento, ou seja, menor folga entre a data mais cedo de conclusão estimada e a data em que o pedido deve ser entregue. A lógica também é priorizar *jobs* com menor folga para evitar atrasos.

LWQ (*"Least Work Next Queue"*) – aqui, a prioridade de entrada é de *jobs* cujo destino primário tenha a menor fila, buscando produzir um item que vai para uma máquina com fila em detrimento de outro que vai para uma máquina que corre o risco de parar por falta de material.

CR (*"Critical Ratio"*) – nesta regra, tem prioridade o *job* com menor razão crítica, que consiste na divisão do tempo de folga do *job* (tempo para entrega – tempo de estimado processamento) pelo seu tempo de processamento estimado. Logo, é uma regra que mescla os conceitos das regras SPT e EDD.

2.4.4. Indicadores de Desempenho

Thurer et al. (2017), ao estudarem o desempenho dos métodos de controle DBR e WLC em um sistema com estratégia de produção MTO em um ambiente *flow shop* flexível, sugerem uma série de indicadores de desempenho. Para fins deste trabalho, serão considerados os seguintes indicadores de desempenho utilizados por Thurer et al. (2017):

***Mean Throughtput Time* –** corresponde ao tempo médio de produção dos *jobs* do sistema. Para calcular este parâmetro, calcula-se o tempo de produção de cada *job* efetivamente terminado durante o período estudado e tira-se a média. O tempo de produção de um *job* corresponde à diferença entre o tempo de entrada do *job* no sistema e o fim da última etapa de processamento e saída do sistema, ou seja, desconsidera o tempo de espera no *pré-shop*.

***Mean Lead Time* –** corresponde ao *lead time* médio de entrega do *job*. Para calcular este parâmetro, calcula-se para cada *job* a diferença entre o tempo do pedido que originou determinado j*ob* e o tempo de término da última etapa de processamento e saída do sistema. Calcula-se a média. Portanto, este indicador de desempenho considera o tempo que o *job* ficou no *pré-shop* aguardando liberação, sendo sempre maior ou igual que o indicador *mean throughtput time*.

***Percentage Tardy* –** corresponde ao percentual de *jobs* que foram concluídos após a data combinada, ou seja, corresponde ao número de *jobs* cujo término da produção ocorreu posteriormente ao seu *due date* dividido pelo número total de *jobs* terminados.

***Mean Tardiness* –** corresponde ao tempo médio de atraso de entrega de *jobs* ao cliente final sistema. Este parâmetro é calculado pela média do atraso de cada *job* terminado no período estudado, contando que este atraso é zero para *jobs* com saída do sistema anterior ao *due date* estipulado.

2.4.5. Questões Exploradas na Literatura

Desde o final da década de 90, com a introdução da ToC e do DBR por Goldratt & Cox (1986), este método de controle da produção tem sido amplamente estudado e tratado em artigos e outras publicações científicas.

Diversos foram os temas estudados, desde estudos de caso até o estudo de pontos chave na implantação do DBR em um sistema MTO: programação da fila de entrada (*pré-shop pool*), determinação do recurso gargalo do sistema (*drum)* e definição de data de entrega de pedidos (*due date setting*).

Wu et al. (1994) aplicaram o DBR em uma empresa fabricante de móveis, estudo de caso que visou medir a eficácia e a eficiência do método DBR em um caso prático, utilizando como principal indicador o *mean throughtput time*.

Krajewski et al. (1996) estudaram os resultados da implantação do método DBR no Centro Logístico de Manutenção da Marinha dos Estados Unidos da América. Segundo estes autores, o tempo médio de reparos após a implementação do DBR caiu de 167 dias para 58 dias. Por sua vez, os níveis de WIP foram reduzidos severamente em aproximadamente 25% dos custos de manutenção, causando um aumento de mais de 100% de capacidade ao mês.

Daniel & Guide (1997) elaboraram um modelo de simulação para testar a eficiência de diversas regras de sequenciamento em combinação com o algoritmo DBR em um estudo de caso em um ambiente de re-manufatura.

Corbett & Csillag (2001) analisaram sete empresas que adotaram o método de controle da produção DBR, visando medir a eficiência em termos de programação da produção.

Um experimento pautado no desenvolvimento de um modelo de simulação para medir a eficiência do método de controle DBR foi publicado por Atwater & Chakravorty (2002). Eles variaram o nível de utilização do recurso gargalo, medindo o impacto de *free goods* (*jobs* que não passam pelo recurso gargalo) na efetividade do DBR.

Uma análise comparativa da implementação dos métodos DBR e MRP, em um sistema de produção real, foi feita por Steele et al. (2005). O objetivo foi identificar similaridades e diferenças, neste caso, pontuando-se quem leva vantagem sobre o que. Os resultados mostraram performance superior do método DBR quando comparado ao MRP.

Wu et al. (2010) utilizaram um modelo para simular a implantação de um método de PCP com base no DBR em uma empresa fabricante de televisores. Russel & Taylor (2011) estudaram diversos exemplos de aplicação do método DBR combinado com a programação da produção, variando o tamanho de lotes e tempo de *set up*. Darlington et al. (2015) publicaram um artigo sobre a concepção e implantação de um modelo DBR para uma linha de painéis automotivos de uma empresa britânica.

A rotina de programação de *jobs* em ambiente *job shop* com o modelo DBR, acrescido de um modelo de otimização, foi feita por Golmohammadi (2015). Por sua vez, Thurer et al. (2017) compararam a eficiência do método DBR em relação ao método WLC em ambientes de produção *job shop* e *flow shop* sob a lógica de produção MTO, variando diversos parâmetros dos ambientes estudados.

Além destas publicações mencionadas, diversos autores também abordaram métodos modificados baseados no DBR, como Lee et al. (2010) com o *simplified-DBR;* e Sirikrai et al. (2006) com o *modified-DBR*. Vale ressaltar que a literatura sobre o método DBR é vasta. Nesta seção, foi apresentada apenas uma seleção de publicações científicas mais afeitas ao escopo da presente investigação.

2.4.6. Dimensionamento do Buffer

O dimensionamento do *time buffer* é uma das etapas mais importantes do método de controle DBR, sendo uma das decisões fundamentais a ser tomada. A decisão tem impacto direto nos indicadores de desempenho de um sistema de produção. Entretanto, após revisão de literatura, foram identificados poucos artigos científicos sobre este tema, quando comparado com outros temas fundamentais do método DBR, conforme apresentado na seção anterior.

Além disto, em relação ao componente *buffer* do DBR, a grande maioria da literatura existente aborda a comparação do método DBR com outros métodos como o WLC (THURER et al., 2017); o MRP (STEELE et al., 2005); e demais métodos de controle da produção (CORRÊA & GIANESI, 1993; DE SOUZA, 2005; GUPTA & SNYDER, 2009; MANIKAS et al., 2015).

Os únicos artigos encontrados pelo autor deste trabalho que tratam especificamente sobre o tema foram os estudos realizados por Radovilsky (1997), Ye & Han (2008) e Zhang & Du (2015), descritos a seguir.

No trabalho de Radovilsky (1997), discute-se uma abordagem quantitativa que busca encontrar o *time buffer* ótimo para maximizar os lucros operacionais do sistema de produção, em um ambiente de produção *flow shop* puro com estratégia de produção MTS. Manipulando as equações da teoria das filas, Radovilsky (1997) obteve a seguinte equação para o *time buffer* ótimo e lucro operacional ótimo, considerando taxa de chegada igual a taxa de processamento:

$$\boldsymbol{K^*} = \sqrt{\frac{2\mu \boldsymbol{C_{TH}}}{\boldsymbol{C_{OE}}}} - \mathbf{1} \tag{2}$$

$$\boldsymbol{NP^*} = \frac{\mathbf{1}}{\mathbf{2}} * (\sqrt{2\mu \boldsymbol{C_{TH}}} - \sqrt{\boldsymbol{C_{OE}}})^{\mathbf{2}} \tag{3}$$

Na fórmula apresentada em (2), K* corresponde ao time buffer ótimo. Na fórmula apresentada em (3), NP* corresponde ao lucro operacional ótimo. Nas fórmulas mencionadas, C_{th} corresponde a margem de contribuição de cada produto que sai do sistema, C_{oe} corresponde ao custo de estoque por WIP unitário, e μ representa a taxa de processamento de cada estágio no sistema, considerando distribuição exponencial.

Desta forma, o resultado apresentado por Radovilsky (1997) obteve um *time buffer* ótimo via abordagem analítica para um caso muito específico de sistema de produção, um *flow shop* puro com tempo médio de produção igual a tempo médio de chegada, distribuição exponencial/*Poisson* e margem de contribuição e custo de estoque constantes para todos *jobs*.

Portanto, o resultado de Radovilsky (1997) mostra-se bastante restrito ao caso específico, não permitindo análise comparativa com base na variação de parâmetros do sistema. Além disso, a otimização focou apenas de parâmetros financeiros e não operacionais.

Uma abordagem analítica foi apresentada por Ye & Han (2008) para a determinação do *time buffer* em um sistema de produção *assembly shop*, considerando não apenas para o gargalo de produção, como também o processo de montagem (*buffer* de montagem). Os autores revisaram estudos anteriores sobre o método DBR e a pouca atenção dada à questão de como dimensionar o *time buffer*, com pouquíssimos artigos tendo como foco principal este tema.

Além disso, em muitas publicações, o dimensionamento do *buffer* se dá de forma empírica, ou seja, sem nenhum critério mais elaborado para a escolha, como um valor arbitrário (THURER et al., 2017) ou como um múltiplo do *lead time* estimado para os *jobs* (SCHRAGENHEIM & RONEN, 1990; DANIEL & GUIDE, 1997).

Apesar de Ye & Han (2008) terem estabelecido um modelo heurístico para a determinação do *time buffer* ótimo, o modelo proposto é extremamente complexo para aplicações práticas e restrito à *assembly shops*. Além disso, os autores não estudaram em profundidade o impacto que variáveis do ambiente de produção possuem sobre o *time buffer* ótimo. Por sua complexidade e aplicação somente para *assembly shops*, o método de Ye & Han (2008) não será detalhado no presente trabalho.

No trabalho de Zhang & Du (2015), propõe-se um modelo de simulação com a aplicação do método DBR, ou seja, capaz de identificar o recurso gargalo, otimizar o *time buffer* e implementar o mecanismo do componente *rope*, conectando a entrada de *jobs* no sistema com o *buffer*.

Em relação ao dimensionamento do *buffer*, a otimização é feita por meio de um modelo de programação linear que visa minimizar o custo de produção. Desta forma, assim como o trabalho de Radovilsky (1997), o trabalho de Zhang & Du (2015) é um modelo de otimização pautado em indicadores financeiros e não operacionais, não considerando a sensibilidade do *time buffer* ótimo com a variação de parâmetros do sistema.

Por fim, Negahban & Smith (2014) mostram em seu trabalho que o tema controle da produção foi identificado por como uma das áreas de destaque na aplicação de ferramentas e modelos de simulação, com número significante e crescente de artigos relativos a este tema que utilizam o conceito de simulação. A simulação por eventos discretos se mostra uma boa forma de estudar o dimensionamento do *time buffer* no método DBR.

2.5. Simulação

A simulação é uma técnica de pesquisa operacional largamente empregada para a resolução de problemas de natureza variada. É uma tentativa de representar um sistema real a partir da construção de um modelo de simulação, podendo ter maior ou menor nível de generalidade. Devido a sua grande utilização, diversos autores têm revisado a literatura sobre a utilização de modelos de simulação (p. ex., JAHANGIRIAN et al., 2010; NEGAHBAN & SMITH, 2014). Segundo Jahangirian et al. (2010), entre 1999 e 2007, pelo menos onze artigos foram publicados com este objetivo.

Desde a sua concepção, o ato de simular vem sendo mais e mais utilizado, para as mais diversas finalidades em diversos setores da economia, como manufatura, setor de saúde, serviços públicos, logística, entre outros (JAHANGIRIAN et al., 2010).

Especificamente, quando se trata do estudo de sistemas de produção, o uso de simulação tem sido uma das ferramentas de maior importância na definição do *layout* e na promoção de melhorias operacionais. Na última década, houve uma tendência de mudança da aplicação de modelos de simulação, que deixaram de ser utilizados apenas para *design*, sendo cada vez mais presentes em projetos de melhoria da performance operacional (NEGAHBAN & SMITH, 2014).

Jahangirian et al. (2010) dividem a simulação em três classes diferentes: (i) classe A, formada por artigos para resolução de problemas reais; (ii) classe B, formada por artigos para resolução de problemas hipotéticos; (iii) classe C, formada por artigos voltados ao desenvolvimento de metodologias.

Na classe A, o estudo objetiva a resolução de um problema real, concreto e específico, e para isto, utiliza dados reais. Na classe B, a simulação objetiva a resolução de problemas reais, porém, comuns a mais de uma situação. Portanto, a busca é por uma solução genérica, e para isto utilizam-se dados artificiais. Na classe C, o objetivo é desenvolver e validar uma metodologia de simulação, independentemente de sua aplicação prática (JAHANGIRIAN et al., 2010).

A revisão de literatura feita por Jahangirian et al. (2010) retrata a importância crescente que modelos de simulação têm tido em termos de produção científica. A produção de artigos com pesquisas baseadas em modelos de simulação relativos às classes A e B cresceu cerca de 92% entre 2000 e 2010. Entre 1970 e 2000, Shafer & Smunt (2004) verificaram que a produção de artigos com pesquisa baseada em modelos de simulação cresceu apenas 14%.

Resultados similares foram obtidos por Negahban & Smith (2014), que também detectaram o crescimento produção de artigos com pesquisa baseada em modelos de simulação quanto o tema é referente à sistemas de produção.

Entrando no âmbito deste trabalho, os resultados são ainda mais expressivos. A revisão realizada por Jahangirian et al. (2010) mostra que a técnica de simulação por eventos discretos é a mais utilizada na penúltima década (40% de todos os artigos levantados) e é apoiada pelo crescimento contínuo do mercado de *softwares* de simulação.

3. METODOLOGIA

Para o desenvolvimento do presente trabalho, adotou-se a metodologia de trabalho descrita a seguir, em amplo escopo. A descrição fornecerá uma ideia geral da metodologia de trabalho adotada na execução deste estudo.

Portanto, detalhes específicos que não estão aqui descritos serão, posteriormente, detalhados nas próximas seções deste trabalho, de acordo com o tema abordado em cada seção.

Modelagem conceitual

Nesta etapa, definiu-se a estrutura do modelo de simulação a ser elaborado. Primeiramente, foram definidos os *inputs* do sistema, ou seja, os parâmetros e variáveis de entrada utilizadas no processo de simulação.

Depois, foram definidos os *outputs* do sistema, ou seja, as variáveis de saída resultantes da simulação, sendo utilizadas para análise dos resultados e discussão.

Por fim, definiu-se o ambiente de desenvolvimento do modelo de simulação, ou seja, a ferramenta computacional utilizada de suporte para o desenvolvimento do modelo de simulação.

Implantação do Modelo Computacional

Neste procedimento, houve criação efetiva do modelo de simulação, com base no ambiente/ferramenta escolhido, permitindo a simulação com base nos parâmetros de entrada (*inputs*) definidos e obtendo-se os parâmetros de saída (*outputs*) escolhidos na etapa anterior. Esta etapa gerou indicadores de performance para verificação do modelo e das instâncias utilizadas.

Verificação do Modelo de Simulação

A verificação incluiu testes iniciais sobre cada parte do modelo para garantir que erros ou inconsistências que prejudicassem a obtenção de resultados consistentes não fossem considerados ou propagados. Para tal finalidade, foi necessário testar cenários em que os resultados fossem conhecidos.

Para a validação do modelo, foi necessário atender as condições de contorno descritas a seguir durante os testes de validação:

- **Inexistência de erros durante processo de simulação –** durante o período em que o modelo estava efetivamente simulando o ambiente de teste, o programa não deveria apresentar mensagens de erros que pudessem resultar no término do processo de simulação.
- **Verificação e validação do processamento de *jobs*–** o ambiente de simulação deveria estar funcionando corretamente, independentemente do método de controle DBR. Para isto, foi utilizada a teoria das filas, envolvendo casos conhecidos para validar o correto funcionamento do ambiente de produção.
- **Correto funcionamento dos mecanismos do DBR –** uma condição importante é que o modelo deveria ser capaz de realizar corretamente as três etapas básicas do DBR: identificar o recurso gargalo do sistema com base nos *inputs* fornecidos; ser capaz de monitorar o WIP entre a entrada do sistema e o recurso gargalo; utilizar o monitoramento de WIP e *time buffer* fornecido como *input* para determinar o correto momento de entrada de *jobs* no sistema.
- **Correto funcionamento das regras de sequenciamento –** o funcionamento e a ordem de entrada de *jobs* no sistema de produção deveria ser compatível com a regra de equacionamento escolhida para a simulação.

Definição das instâncias

A quarta etapa deste trabalho foi definir as instâncias de simulação, ou seja, especificar os valores a serem utilizados para simulação dentre os parâmetros de entrada previamente definidos para a geração de *jobs*, estágios, *due dates*, entre outros.

Na literatura, é possível obter diversas instâncias teóricas utilizadas por outros autores para simulação, bem como há a possibilidade de realização de estudo de caso baseado em sistema de produção existente. As instâncias definidas serão posteriormente detalhadas neste trabalho, na seção correspondente.

Experimentos

Esta etapa consistiu da execução dos experimentos de simulação das instâncias definidas na etapa anterior. Para pautar a etapa de análise e discussão, gráficos e tabelas foram elaboradas com base nos resultados obtidos pela saída do modelo de simulação para cada cenário rodado.

Análise e Discussão

Esta foi a etapa final do trabalho, consistindo em analisar os resultados obtidos durante a etapa de experimento para fornecer subsídios para as conclusões do estudo. Nesta etapa, discutiu-se também as limitações do modelo de simulação, recomendando-se estudos futuros para dar sequência ao tema estudado.

4. MODELO DE SIMULAÇÃO

Este trabalho propõe o desenvolvimento de um modelo de simulação por eventos discretos para o estudo do dimensionamento do *time buffer* em alguns cenários. O objetivo foi simular um ambiente de produção *flow shop* genérico cuja estratégia de produção segue o padrão MTO, onde os *jobs* seriam gerados a partir do pedido ou efetivação da venda, já com *due date* estabelecido. A este ambiente de simulação, foi acrescido o método de controle DBR.

O presente capítulo discorre sobre o modelo de simulação, desde seu desenho conceitual, passando pelo sua implantação computacional, até sua verificação e validação. Por fim, foram definidas as instâncias de simulação que serão analisadas no próximo capítulo.

4.1. Modelagem Conceitual

Qualquer modelo de simulação pode ser dividido em três partes essenciais: parâmetros de entrada ou *inputs*, ambiente de simulação (onde se desenvolve o modelo) e parâmetros de saída ou *outputs*. Os parâmetros de entrada são variáveis escolhidas para alimentarem o modelo de simulação, de forma a reger o comportamento da simulação.

O ambiente de simulação executa a simulação de acordo com a lógica de simulação e libera como resultado os parâmetros de saída ou *outputs* (Figura 11). Os parâmetros de saída são, geralmente, indicadores diretamente utilizados para a análise dos resultados da simulação e conclusão do experimento. Porém, podem ser também dados utilizados para montar estes indicadores, sendo, portanto, usados indiretamente para a análise de resultados da simulação e conclusão do experimento.

Figura 11 - Componentes principais para desenvolvimento de um modelo de simulação

Fonte: Elaborado pelo autor

Desta forma, esta seção busca definir conceitualmente as variáveis usadas como *inputs*, o ambiente de simulação escolhido para o desenvolvimento do modelo de simulação, os *outputs* do modelo e a lógica de simulação.

4.1.1. *Inputs* do Modelo de Simulação

Para simular um sistema de produção *flow shop* genérico, com estratégia de produção MTO com método de controle DBR, uma série de parâmetros precisam ser fornecidos para definir o ambiente que efetivamente se quer simular. Estes serão os *inputs* do sistema.

Número de simulações – O número de simulações realizadas cada vez que o modelo é rodado será um parâmetro de entrada. O valor estabelecido será o número de vezes em que o ambiente simulará de forma repetida o sistema escolhido, ou seja, não se muda os outros *inputs* de simulação. A única grandeza que efetivamente muda entre simulações é a semente para geração de números aleatórios.

O objetivo de definir esta grandeza como parâmetro de entrada é permitir maior grau de confiabilidade dos indicadores de desempenho, de forma a simular o mesmo sistema um número significante de vezes para eliminar a potencial existência de *outliers*.

Tempo de simulação – É o parâmetro de entrada que delimitará até quando uma simulação irá rodar. O presente modelo de simulação terá como gatilho de parada uma grandeza temporal genérica, que pode ser segundos, minutos, horas, dias, semanas ou qualquer outra grandeza escolhida na hora de fornecer os dados temporais dos experimentos.

Tipo de distribuição para geração da chegada de *jobs* – A geração de *jobs* no sistema será fixo ou variável (probabilístico), com os tempos entre gerações definidos a partir de um tipo de distribuição fornecida como parâmetro de entrada. No caso de geração de *jobs* com distribuição fixa, *jobs* chegam após o mesmo intervalo temporal no sistema. Logo, o modelo não trabalhará com tempos de geração diferentes entre *jobs*.

As distribuições escolhidas para a elaboração do modelo foram as mais comumente apresentadas na literatura existente e que melhor representam ambientes de simulação reais. São elas: distribuição exponencial, distribuição normal, distribuição uniforme e distribuição fixa.

Parâmetros para a geração das chegadas de *jobs* – Após definir qual o tipo de distribuição a ser considerado para a geração de *jobs* no sistema, é necessário definir parâmetros temporais para determinar o tempo entre gerações. Dependendo do caso, quatro parâmetros de simulação serão necessários: tempo médio entre gerações, tempo mínimo entre gerações, tempo máximo entre gerações e desvio padrão do tempo médio entre gerações.

Para distribuição exponencial, será necessário fornecer o tempo médio entre gerações que consiste no inverso da taxa média utilizada para gerar tempos de chegada de *jobs*. Para distribuição normal, será necessário fornecer tempo médio e desvio padrão dos tempos entre gerações. Para distribuição uniforme, será necessário fornecer os tempos máximos e mínimos, para determinar um intervalo de valores com a mesma probabilidade para ser o tempo entre a chegada de *jobs* no sistema. Por fim, na distribuição fixa, o tempo médio será o tempo entre a geração de *jobs* no sistema.

Tipo de distribuição para geração de *due date*– Como o presente modelo considerará apenas a estratégia de produção MTO, quando um *job* é gerado, ele já deve possuir um determinado *due date*. Para a geração de *due date* para cada *job* gerado na simulação, também se deve determinar um tipo de distribuição, de forma similar ao tipo de distribuição determinado para a geração de tempos de chegada, porém, de forma independente.

Os tipos de distribuição que podem ser escolhidas para a geração de *due dates* são as mesmas disponíveis para a geração de tempos de chegada, ou seja: distribuição exponencial, distribuição normal, distribuição uniforme e distribuição fixa.

Parâmetros para a geração de *due date*– Após definir qual o tipo de distribuição a ser considerado para a geração de *jobs* no sistema, é necessário definir os parâmetros para a definição dos tempos. Os parâmetros que o sistema considera para geração de *due date* devem ser os mesmos que para a geração de tempos de chegada, ou seja: tempo médio, tempo mínimo, tempo máximo e desvio padrão. Os parâmetros só serão necessários dependendo do tipo de distribuição, da mesma forma que para a geração de tempos de chegada.

Número de estágios – O número de estágios deve ser um *input* do modelo de simulação. O modelo proposto simulará as passagens do número de *jobs* gerados pelo número de estágios gerados, considerando as características de cada estágio, que devem ser customizáveis de estágio para estágio.

Número de máquinas por estágio – Cada estágio pode ter número de máquinas diferentes, como definição do ambiente *flow shop* genérico. O número de máquinas para cada estágio é um *input* do modelo de simulação. As máquinas existentes dentro de um estágio de produção devem ser idênticas entre si.

Tipo de distribuição para geração de tempos de processamento – Da mesma forma que a geração de tempos de chegada de *jobs* no sistema e *due date*, a geração de tempos de processamento podem ser probabilísticas ou determinísticas e seguem uma distribuição previamente estabelecida.

As distribuições para a geração de tempos de processamento podem ser: exponencial, normal, uniforme, fixa (determinística), triangular ou Erlang. Vale ressaltar que o tipo de distribuição, bem como os parâmetros para a geração de tempos de processamento, são características do estágio e não dos *jobs.*

Parâmetros para a geração de tempos de processamento – Após definir qual o tipo de distribuição a ser considerado para a geração de tempos de processamento, é necessário definir parâmetros temporais para determinar o tempo entre gerações. Dependendo do caso, cinco parâmetros de simulação serão necessários: tempo médio, tempo mínimo, tempo máximo, desvio padrão e número de exponenciais para formação da distribuição Erlang.

Para distribuição exponencial, será necessário fornecer o tempo médio, que consiste no inverso da taxa média utilizada para gerar tempos de processamento. Para distribuição normal, será necessário fornecer tempo médio e desvio padrão dos tempos de processamento. Para distribuição uniforme, será necessário fornecer os tempos máximos e mínimos, para determinar um intervalo de valores com a mesma probabilidade para ser o tempo de processamento.

Na distribuição fixa, o tempo médio será o tempo de processamento. Na distribuição triangular, os tempos mínimos e máximos funcionam como valores limites e o tempo médio será a moda.

A distribuição Erlang é uma distribuição de probabilidade contínua com uma ampla aplicabilidade, principalmente devido à sua relação com a distribuição exponencial e a distribuição gama. Atualmente, esta distribuição é utilizada em várias áreas que aplicam processos estocásticos, consistindo em uma combinação de distribuições exponenciais. Para a distribuição Erlang, será necessário fornecer o tempo médio de processamento e o número de exponenciais combinados, denominado *beta* de Erlang.

Probabilidade de um *job* qualquer passar por cada estágio – Cada estágio terá um atributo de entrada que consiste na probabilidade de um *job* qualquer passar ou não por este estágio. No processo de geração do *job*, seu caminho será determinado com base nestas probabilidades, que são atributos do estágio e não do *job* gerado. Desta forma, conseguiu-se adaptar o modelo para a criação de um *flow shop* genérico. Caso estas probabilidades sejam todas 100%, temos um *flow shop* flexível.

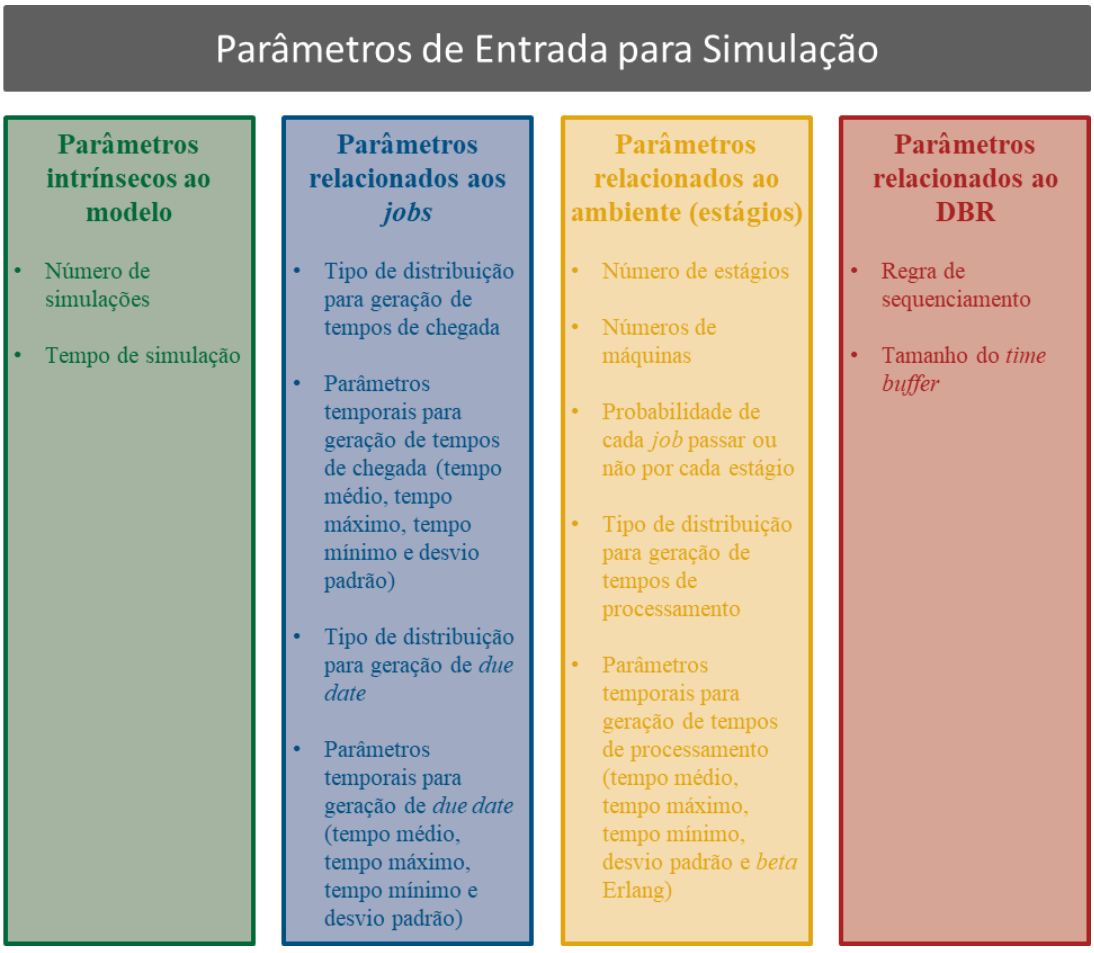
Regra de sequenciamento – A regra de sequenciamento utilizada para ordenar *jobs* contidos no pré-shop devido ao método de controle DBR é um *input* do sistema. Vale ressaltar que este ordenamento só ocorre no *pré-shop*, sendo que quando o *job* está dentro do sistema, seguirá a regra de sequenciamento FIFO.

As regras escolhidas para serem retratadas no modelo de simulação foram as mais comumente utilizadas e que apresentam maior aplicação prática: FIFO (*First In, First Out*), SPT (*Shortest Processing Time*), EDD (*Earliest Due Date*) e LS (*Least Slack*).

Tamanho do *time buffer* limite para a entrada de *jobs* no sistema – O tamanho do *time buffer* limite, parâmetro fundamental para os objetivos do presente trabalho, será um *input* do sistema. Isto nos permitirá criar diversos cenários com diferentes tamanhos de *time buffer* para como se comportam os indicadores de desempenho em função do *time buffer* para diferentes *layouts* de ambientes de produção.

A Figura 12 ilustra todos os *inputs* que o modelo de simulação requer para efetivamente conduzir a simulação pelo modelo de simulação elaborado.

Figura 12 - Parâmetros de entrada do modelo de simulação (*Inputs*)



Fonte: Elaborado pelo autor

4.1.2. Ambiente de Simulação

O ambiente de simulação escolhido para o desenvolvimento do modelo de simulação foi a linguagem de programação *Python*, com auxílio da biblioteca *Simpy*. *Simpy* © é um módulo da linguagem *Python*© *open source* voltado à criação de modelos de simulação por eventos discretos, permitindo a criação de entidades, como *jobs* e estágios. Esta biblioteca permite realizar simulações com processos que não interagem uns com os outros, mas que possuem recursos partilhados, fator fundamental para o caso deste trabalho.

Este foi o ambiente escolhido, pois permite maior grau de customização do ambiente escolhido, ou seja, pode-se montar um modelo genérico que receba dados e gere uma das possíveis versões de sistemas simulados. Isto não é possível em *softwares* especializados clássicos de simulação de sistemas de produção por eventos discretos, como *Simul8*©, *Arena*© e *ProModel*©.

4.1.3. *Outputs* do Modelo de Simulação

O output do modelo de simulação deste trabalho foi a simulação em si, ao invés de indicadores de desempenho. Em outras palavras, ao invés de calcular os indicadores de desempenho que serão posteriormente analisados com base nos eventos da simulação, o modelo de simulação exportará os eventos relevantes da simulação em si, de forma a se ter o retrato do que ocorreu na simulação.

Isto foi importante porque, ao exportar os eventos da simulação ao invés de exportar apenas os indicadores de desempenho, foi possível entender o funcionamento do modelo e analisar pontos específicos da simulação. Isto foi essencial para a identificação de erros do modelo e sua correção, de forma a facilitar a validação do modelo.

Além disso, isto não prejudicou a análise de indicadores de desempenho. Estes indicadores continuaram a ser calculados, porém, fora da simulação. Os indicadores de desempenho considerados no presente trabalho foram: *mean lead time*, *mean throughtput time*, *mean tardiness* e *percentage tardy*. A descrição destes indicadores foi feita no Capítulo 2 (seção 2.4.4).

Os eventos de simulação exportados foram: geração do *job* dentro do sistema; entrada do *job* no sistema de produção; entrada do *job* na fila de cada uma das operações em seu caminho; início da operação do *job* em cada uma das operações em seu caminho; fim da operação do *job* em cada uma das operações em seu caminho; saída do *job* do sistema de produção, representando sua entrega para o cliente. Eventualmente, se o job for encaminhado para o *pré-shop* após a sua geração, os eventos de entrada e saída do *pré-shop* também devem ser exportados como eventos da simulação.

4.1.4. Lógica de Simulação

A lógica é função do ambiente de simulação ser capaz de incluir os *inputs* fornecidos e, por meio da lógica de simulação, fornecer os *outputs* determinados, no caso, exportar os eventos de simulação.

Para isto, a lógica do modelo de simulação deve seguir a de um sistema de produção *flow shop* genérico com estratégia MTO controlada pelo método de controle DBR. Para tal finalidade, controlou-se as seguintes diretrizes.

- A simulação deve ser repetida com base no número de simulações determinadas e, em cada vez, o tempo de simulação deve ser igual ao tempo determinado como parâmetro de entrada.
- Os *jobs* devem ser gerados em intervalos de tempos determinados a partir do tipo de distribuição escolhida e parâmetros temporais fornecidos como *inputs* do modelo. A geração de novos *jobs* é livre, ou seja, não é restringida a não ser pelo tempo de simulação.
- No mesmo instante em que o *job* é gerado no sistema, seu *due date* e caminho dentro do sistema (estágios em que passa) devem ser gerados. Para isso, usa-se o tipo de distribuição escolhida e parâmetros determinados para a geração de *due date* bem como a probabilidade de cada máquina estar no caminho de um *job*.
- Após a geração de um *job* qualquer, checa-se o volume de WIP na forma de *time buffer* e compara-se este valor com o *time buffer* limite definido como *input*. Se o valor atual for menor que o limite, o *job* gerado entra no sistema de produção. Caso contrário, entrará no *pré-shop* e aguardará seu momento de entrar no sistema.

- O método de controle DBR é ativado a cada vez que um novo *job* é gerado ou alguma máquina de alguma estação termina seu processamento. Caso o DBR indique que um novo *job* contido no *pré-shop* deve entrar no sistema, o *pré-shop* é reordenado conforme regra de sequenciamento definida como parâmetro de entrada e aquele *job* mais prioritário é o escolhido para entrar no sistema de produção.
- No sistema, cada *job* deve seguir seu caminho no sentido do fluxo do sistema, não necessitando ficar em filas de estágios em que não precisam passar.
- A regra de sequenciamento de *jobs* dentro do sistema de produção é FIFO, ou seja, sempre o primeiro a entrar terá prioridade.
- Caso um *job* necessite operar em um estágio e todas as máquinas estiverem em operação, ele deve aguardar na fila do estágio.
- Para seguir corretamente o método de controle DBR, o modelo de simulação deve identificar o estágio gargalo do sistema, antes do início da simulação.
- O modelo de simulação deve ser capaz de calcular o tamanho do *time buffer* atual do sistema de produção simulado a cada instante que for necessário durante a simulação.
- O modelo de simulação deve ser capaz de sequenciar o *pré-shop* de acordo com as possibilidades existentes de regras de prioridade e deve ser capaz de liberar um novo *job* para o sistema dentro desta ordem toda vez em que a checagem identificar que o sistema está abaixo do *time buffer* limite.

4.2. Modelagem Computacional

Busca-se resumir nesta seção como a modelagem conceitual foi efetivamente implantada em *Python*© com auxílio da biblioteca *Simpy*©. Desta maneira, o presente item será dividido da mesma maneira que o modelo de simulação foi dividido: em funções.

Vale ressaltar que funções menos importantes e mais simples, cujo escopo para sua criação foi meramente o de suporte, serão descritas de forma conjunta e menos detalhada como “funções de suporte”.

O modelo completo e comentado pode ser visto no ANEXO A, bem como fotos da planilha Excel e as macros de suporte utilizadas para importar inputs e analisar os indicadores de desempenho (ANEXOS B, C e D).

4.2.1. Função “Inputs”

Esta é a função do modelo de simulação responsável por importar da planilha Excel de suporte os *inputs* necessários para a simulação. Como se pode observar no ANEXO A, as variáveis são importadas por meio de suas respectivas coordenadas da planilha de Excel que podem ser vistas no ANEXO B.

Além de importar todos os *inputs* mencionados no item 4.1.1 do presente trabalho, esta função gera uma lista (“ListaEstagios”) que possuirá tamanho equivalente ao número de estágios, em que cada item da lista corresponde a um estágio do ambiente de simulação a ser gerado.

Cada item da lista, correspondente a um estágio, terá oito elementos ou componentes: números de máquinas do estágio, tipo de distribuição para geração de tempos de processamento, tempo médio, desvio padrão, tempo mínimo, tempo máximo, *beta* Erlang e probabilidade de um *job* passar ou não pelo estágio.

A lista criada terá todos os atributos dos estágios, auxiliando na criação do ambiente de simulação. Para criar esta lista, um *loop* é gerado pela importação do número de estágios, de forma a ler os atributos de cada estágio. O número de estágios corresponde a uma fórmula CONT.NÚM do Excel, de forma a contar o número de máquinas atribuídas como parâmetro de entrada. Por fim, a função retorna todos os parâmetros de entrada importados, inclusive a lista de estágios.

4.2.2. Funções de Suporte

Com o objetivo de suporte a operações muito utilizadas durante o modelo de simulação, duas funções de suporte foram criadas: a função ‘NumAleatorio”, responsável por gerar números aleatórios e a função “TempoMed”, responsável por retornar o tempo médio esperado de acorda com a distribuição escolhida e parâmetros de entrada.

A função responsável por gerar números aleatórios possui como parâmetro de entrada o tipo de distribuição escolhida e parâmetros auxiliares para geração de tempos: tempo médio, desvio padrão, tempo mínimo, tempo máximo e *beta* Erlang.

A função é formada por um conjunto de *ifs* que realizam o teste condicional de forma a identificar o tipo de distribuição escolhida. A partir da distribuição escolhida, gera-se um número aleatório por meio de uma função da biblioteca *random* (ex: random.expovariate). Por fim, a função retorna o número aleatório gerado. Caso a distribuição seja fixa, retorna-se o tempo médio de parâmetro de entrada.

A função responsável por retornar o tempo médio recebe os mesmos parâmetros que a função responsável por gerar números aleatórios. Também é formada por um conjunto de *ifs* que realizam o teste condicional de forma a identificar o tipo de distribuição escolhida. Caso a distribuição seja exponencial, normal, fixa ou Erlang, retorna o tempo médio. Caso seja uniforme, retorna a média aritmética entre o tempo mínimo e o tempo máximo. Caso seja triangular, retorna a média aritmética entre o tempo mínimo, tempo máximo e tempo médio.

4.2.3. Função “GeracaoJobs”

A função “GeracaoJobs” é responsável por gerar os *jobs*, desta forma, dando início a simulação, sendo que a geração de *jobs* só é cessada quando o tempo de simulação é ultrapassado. Para isto, um elemento lógico *while* é utilizado.

Cada *job*, ao ser criado, recebe uma numeração, um nome, um *due date* e um caminho, ou seja, no momento de geração já é definido os estágios em que o *job* passará. O nome do *job* será formado pelo prefixo “Job” acrescido à sua numeração. Desta forma, o quinto *job* gerado terá nomenclatura “Job5”. A numeração de *jobs* não é independente entre simulações, de forma que uma variável global foi definida para estocar o número do último *job* gerado na simulação anterior para dar sequência nesta ordem.

O caminho será uma lista binária com o mesmo número de itens em relação ao número de estágios. Caso o elemento *n* da lista seja 1, o estágio *n* do sistema de produção, na ordem do fluxo, estará no caminho do *job* gerado. Caso contrário (elemento 0), o *job* ultrapassa o estágio correspondente.

Para a geração desta lista caminho, uma lista auxiliar denominada “SimouNao” foi criada com a proporcionalidade de uns e zeros igual a relação de probabilidade do estágio *n* estar ou não no caminho do *job*. Para cada estágio *n*, é gerada uma lista “SimouNao” e um elemento é aleatoriamente escolhido desta lista para compor a lista caminho.

Por fim, após a geração efetiva do *jobs*, a presente função define a próxima etapa do *job*, que será seu acréscimo ao *pré-shop*, um recurso *Store* do *Simpy*©, ou o início de seu processamento, ou seja, a chamada da função "Processamento". Para isto, a função "Buffer" é chamada para comparar o *time buffer* atual do sistema com o *time buffer* limite determinado. Vale ressaltar que os tempos de geração são acrescidos ao tempo atual da simulação no momento da geração.

4.2.4. Função "Processamento"

A função "Processamento" é a responsável por simular a passagem de *jobs* pelo sistema de produção, ou seja, pelos estágios do sistema que estão no caminho de cada *job*. Para isto, um *loop* é gerado para cada *job*, de forma que ele passe por cada estágio de produção, sendo que é realizado um teste na forma do operador lógico *if* para identificar se o estágio está ou não no caminho do *job*. Se ele não estiver, o estágio é pulado. Se ele estiver no caminho, o *job* entrará na fila do estágio, realizará a operação e será liberado para o próximo estágio.

Cada estágio do modelo de simulação recebe um *Resource* da biblioteca *Simpy*©, que o permite ter as características de um recurso com *jobs* na fila de entrada e *jobs* em operação, facilitando o processo de simulação. Os estágios do modelo de simulação foram definidos como "classe", de forma a facilitar ao modelo de simulação a criação de um número finito de estágios, determinado como parâmetro de entrada. Sem o uso de "classes", isso seria inviável.

Os operadores logísticos "*request*" e "*release*" foram utilizados respectivamente para simular o tempo de fila de cada *job* e sua remoção do estágio. Após a saída de qualquer *job* de qualquer estágio, a função "Rope" é chamada, de forma a ativar o mecanismo DBR, caso o *time buffer* permita.

Vale ressaltar que todos os eventos de chegada de Jobs na fila, bem como o início e término de suas operações, em qualquer estágio, foram exportados como *output* da simulação.

Ao final da função de processamento, o *job* termina a sua simulação e sai do sistema, evento que também é exportado da simulação como *output*. Por fim, os tempos de espera em fila e de processamento foram sempre acrescidos ao tempo de simulação. Os tempos de processamento foram definidos com o auxílio da função "NumAleatorio".

4.2.5. Função “Drum”

A função “Drum” é a responsável por simular a primeira etapa do método de controle DBR, ou seja, a identificação do recurso gargalo. Esta função é chamada no início do bloco principal, antes do início da simulação temporal. No modelo de simulação desenvolvido para fins deste trabalho, o recurso gargalo do sistema é então identificado com base nos parâmetros definidos para os estágios e não com base em parâmetros obtidos durante a simulação temporal.

Além disso, o recurso gargalo é único e definido uma única vez antes da simulação, não sendo, portanto, um processo iterativo durante a simulação. A definição do recurso gargalo se dá pelo recurso cuja expectativa para processamento de um lote *n* de *jobs* fosse a maior. Para isto, a função “TempoMed”, que calcula a expectativa médio de processamento de certo *job* em cada estágio, é utilizada para achar o tempo médio esperado que uma máquina em cada estágio processa certo *job*. Além do tempo médio, o número de máquinas no estágio e a probabilidade de certo *job* passar pelo estágio são considerados para cálculo do tempo de processamento de um lote de *jobs*.

Sumarizando, o recurso gargalo será aquele que possuir o maior valor de TL (tempo de processamento de um lote de *jobs* unitário por número de *jobs* no lote), de acordo com a equação (4). Se dois estágios possuírem o mesmo TL, o gargalo será aquele que vier primeiro no sistema. Isto foi determinado de forma arbitrária, de forma a possibilitar a implementação do DBR.

$$TL_{estágio\ n} = \frac{tempo\ médio\ de\ proces.de\ 1\ job_n * \ Prob.de\ job\ ser\ processado_n}{Número\ de\ máquinas_n} \quad (4)$$

Por fim, a função “Drum” retornará a posição do estágio gargalo no sistema, parâmetro que será utilizado para o cálculo do time buffer durante a simulação.

4.2.6. Função “Buffer”

A função “Buffer” é a responsável por realizar a checagem do tamanho atual do *time buffer* toda vez que for chamada e compará-la ao *time buffer* limite. A função retorna *True*, caso haja a possibilidade de liberar certo *job* para o sistema, ou seja, o *time buffer* limite seja maior que o atual, no momento de checagem. Caso contrário, a função retorna *False*, caso o sistema não permita a liberação de novo *job*.

Para isto, a função é composta basicamente de duas etapas: cálculo do *time buffer* no momento que a função é chamada; comparação do *time buffer* calculado com o *time buffer* limite, parâmetro de entrada do sistema.

Para realizar a primeira etapa, a função necessita receber como parâmetro de entrada a lista de estágios, que contém todas as informações de processamento de cada estágio; e a posição do gargalo, pois, conceitualmente, o *time buffer* no modelo DBR consiste no tempo total de processamento esperado dos *jobs* entre a entrada do sistema e o recurso gargalo, como visto no Capítulo 2.

A partir daí, a função “Buffer” calcula o tempo médio esperado de processamento de um *job* qualquer para cada estágio entre o início do sistema de produção e o estágio gargalo e guarda estes tempos em uma lista de suporte “ListaTempos”. Para isto, utiliza a função “TempoMed” de suporte.

Posteriormente, passa contando o número de *jobs* na fila de cada estágio (considerando aqueles em operação) e o tempo médio de processamento esperado até o gargalo. Para cálculo do tempo médio de processamento até o gargalo, soma-se o tempo médio esperado de processamento entre o estágio em que os *jobs* estão na fila até o tempo médio do estágio gargalo.

Com estes dois valores determinados para cada estágio, estes são multiplicados e registrados em uma variável que somará esta multiplicação para cada um dos estágios até o gargalo, e, portanto, terá ao final do processo, o tamanho do *time buffer* atual.

A segunda etapa, que consiste em comparar o *time buffer* calculado com o *time buffer* limite, é bem simples. Ela consiste em comparar os dois atributos com o uso do operador logístico *if*, retornando *True*, caso o limite seja maior que o atual, e *False*, caso contrário.

4.2.7. Função “Rope”

A função “Rope” é a responsável pela liberação de *jobs* dentro do *pré-shop* para o sistema de produção. Ela é chamada toda vez que uma operação de processamento qualquer dentro do sistema de produção é terminada. A função “Rope” não retorna nenhum valor, apenas inicia o processamento de um novo *job* caso a função “Buffer”, que é chamada dentro da função “Rope”, retornar *True*, isto é, o *time buffer* limite for maior que o *time buffer* atual.

Caso a função “Buffer” retorne *True*, a função “Rope” realizará a seguinte rotina: chamará a função “RegrasPrioridade”, que reordenará o *pré-shop* de acordo com a ordem de prioridade escolhida como parâmetro de entrada; removerá o *job* mais prioritário do *pré-shop* por meio do comando “*get*”, ou seja, removerá o primeiro *job* no recurso do *Simpy*© equivalente ao *pré-shop*; chamará a função “Processamento”, dando início ao processamento do *job* removido do *pré-shop* dentro dos estágios do sistema de produção.

Caso a função “Buffer” retorne *False*, a função “Rope” somente acionará a função “RegrasPrioridade”, que reordenará o *pré-shop* de acordo com a ordem de prioridade escolhida como parâmetro de entrada.

4.2.8. Funções de Sequenciamento

Em relação a regras de sequenciamento ou regras de prioridade, o modelo de simulações possui quatro funções que tratam sobre este aspecto: a função “RegrasPrioridade”, a função “EDD”, a função “SPT” e a função “LS”.

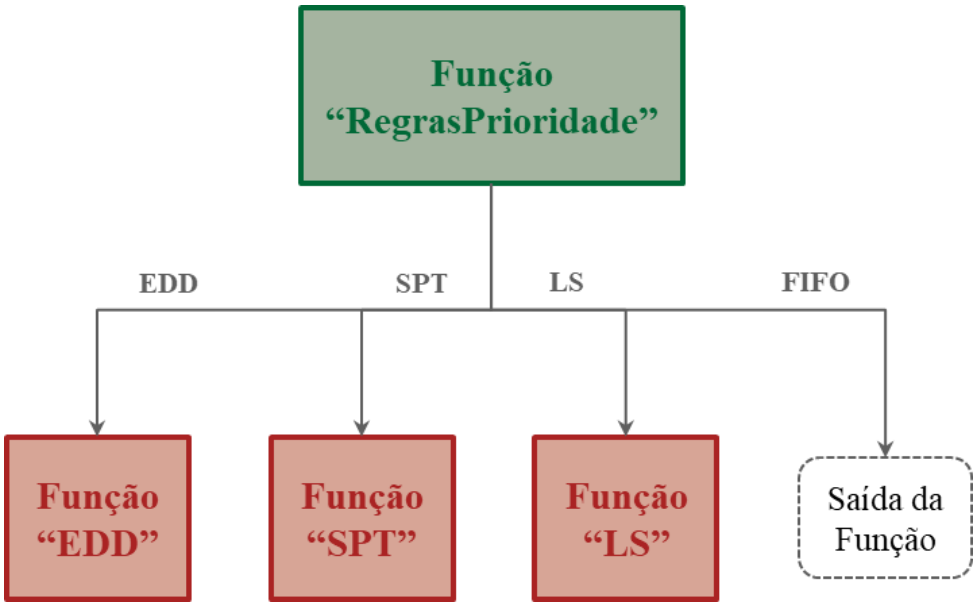
A função “RegrasPrioridade” consiste na função que efetivamente realiza o sequenciamento de *jobs* no *pré-shop*. A lógica desta função é bem simples: é composta por um conjunto de operadores lógicos *ifs* que checam se o parâmetro de entrada relativo a regra de sequenciamento escolhida é compatível com as regras de sequenciamento FIFO, EDD, SPT ou LS.

Caso corresponda a regra de sequenciamento EDD, a função “EDD”, que efetivamente realizada a ordenação do *pré-shop* é acionada. Caso corresponda a regra de sequenciamento SPT, a função “SPT”, que efetivamente realizada a ordenação do *pré-shop* é ativada. Caso corresponda a regra de sequenciamento LS, a função “LS”, que efetivamente realizada a ordenação do *pré-shop* é chamada.

A exceção se dá pela regra FIFO. Caso o parâmetro de entrada corresponda a regra de sequenciamento FIFO, a função “RegrasPrioridade” não realiza nenhuma operação e simplesmente termina sua operação. Isto ocorre, pois a mecânica de adição de novos *jobs* no *pré-shop* ocorre pela ordem de geração do *job* no sistema. Logo, o *pré-shop* já possui o seu sequenciamento de acordo com a regra FIFO.

Vale ressaltar que a função “RegrasPrioridade” não retorna nenhum valor ou variável, ou seja, apenas ordena o *pré-shop* de acordo com a regra de sequenciamento escolhido para que a saída de *jobs* do *pré-shop* siga esta mesma ordem. A Figura 13 ilustra o fluxograma de relacionamentos de funções de sequenciamento.

Figura 13 - Relação entre funções de sequenciamento do modelo de simulação



Fonte: Elaborado pelo autor

As funções “EDD”, ”SPT” e “LS” possuem a mesma finalidade: se forem acionadas, ordenarão o *pré-shop* de acordo com a regra de prioridade da qual herdam seu nome. Portanto, a função “EDD” ordenará os *jobs* dentro do *pré-shop* de acordo com *due date*, a função “SPT” ordenará os *jobs* do *pré-shop* de acordo com seu tempo global de processamento e a função “LS” ordenará os *jobs* do *pré-shop* de acordo com sua folga (tempo esperado para finalização do *job* menos seu *due date*).

Da mesma forma que a utilidade destas funções é a mesma, o seu funcionamento é bem similar entre si. Pode-se descrever este funcionamento em sete etapas:

- **Primeira etapa –** consiste na criação de uma lista auxiliar idêntica a lista do *pré-shop*, por meio de um *loop* que adiciona todos os elementos da lista do *pré-shop* na mesma ordem na lista auxiliar.
- **Segunda etapa –** consiste em esvaziar a lista do *pré-shop*, de forma que possibilita adicionar novamente os *jobs* de acordo com a nova ordem, numa lista vazia.
- **Terceira etapa –** consiste em definir o primeiro item da lista como parâmetro de comparação inicial para achar o *job* mais prioritário dentro da lista auxiliar, de acordo com a regra escolhida.
- **Quarta etapa –** comparar um a um os *jobs* da lista auxiliar de forma a identificar aquele mais prioritário. Ou seja, começamos comparando o primeiro *job* da lista com o segundo e identificando o mais prioritário. Em seguida, comparamos o terceiro item da lista com o mais prioritário entre o segundo e o primeiro juntos. Isto é feito desta maneira em diante até o final da lista auxiliar, de forma que teremos identificado o *job* mais prioritário ao final desta etapa.
- **Quinta etapa –** consiste em adicionar este *job* mais prioritário a lista do *pré-shop*, por meio do elemento lógico do *Simpy*© denominado *put*.
- **Sexta etapa –** consiste em remover da lista auxiliar o *job* já adicionado na lista do *pré-shop*, com o auxiio do elemento lógico *pop*.
- **Sétima etapa –** consiste em repetir as seis primeiras etapas na mesma ordem até a remoção de todos os *jobs* da lista auxiliar, de forma a esvaziá-la. Assim, teremos no final deste processo a lista do *pré-shop* devidamente ordenada de acordo com a regra escolhida.

Vale ressaltar que os *jobs* na lista do *pré-shop* são representados por uma lista composta de cinco itens: número, nome, número de estágios por onde é processado, *due date* e a lista caminho. Estas variáveis permitem o cálculo das variáveis utilizadas como parâmetro de comparação para as regras EDD, SPT e LS. Para a regra EDD, a variável a ser comparada é retirada da lista. Para as regras SPT e LS, é necessária uma série de processos de manipulação de dados que podem ser vistos em detalhes e com comentários no ANEXO A.

Portanto, a única diferença efetiva entre as funções EDD, SPT e LS é a forma de cálculo da variável a ser comparada.

4.2.9. Bloco Principal e *Output*

O bloco principal consiste na parte principal do modelo de simulação, que começará a simulação e acionará as funções auxiliares, apresentadas anteriormente neste capítulo. O código de programação é bem simples, comparativamente as funções do modelo, podendo ser dividido em três partes: importação de parâmetros de entrada e bibliotecas; execução do número escolhido de simulações por meio de um *loop*, de acordo com o tempo de simulação determinado; importação do *output* da simulação para planilha excel e análise de dados via *macro* do Excel.

A primeira etapa começa com a importação das bibliotecas utilizadas como suporte para o desenvolvimento do modelo de simulação. São três bibliotecas importadas: a já mencionada *Simpy*©, que permite a simulação por eventos discretos; a biblioteca *random*, que possibilita a geração de número aleatórios, com base nas distribuições de probabilidade previamente retratadas; a biblioteca *xlwings*, que permite a interface do código *python* com o Excel.

Após importar estas bibliotecas, o código define a planilha na qual a interface entre *Python*© e Excel será gerada. Fotos desta planilha podem ser vistos nos ANEXOS B e C. Para acabar esta primeira fase, o bloco principal do código do modelo de simulação ativa duas funções: a função "Inputs", responsável por ler os inputs do arquivo Excel e retornar os parâmetros de entrada para geração de *jobs*, *due dates* e estágios; a função "Drum", que identifica a posição do recurso gargalo, parte do método de controle DBR que será fundamental para a simulação do modelo.

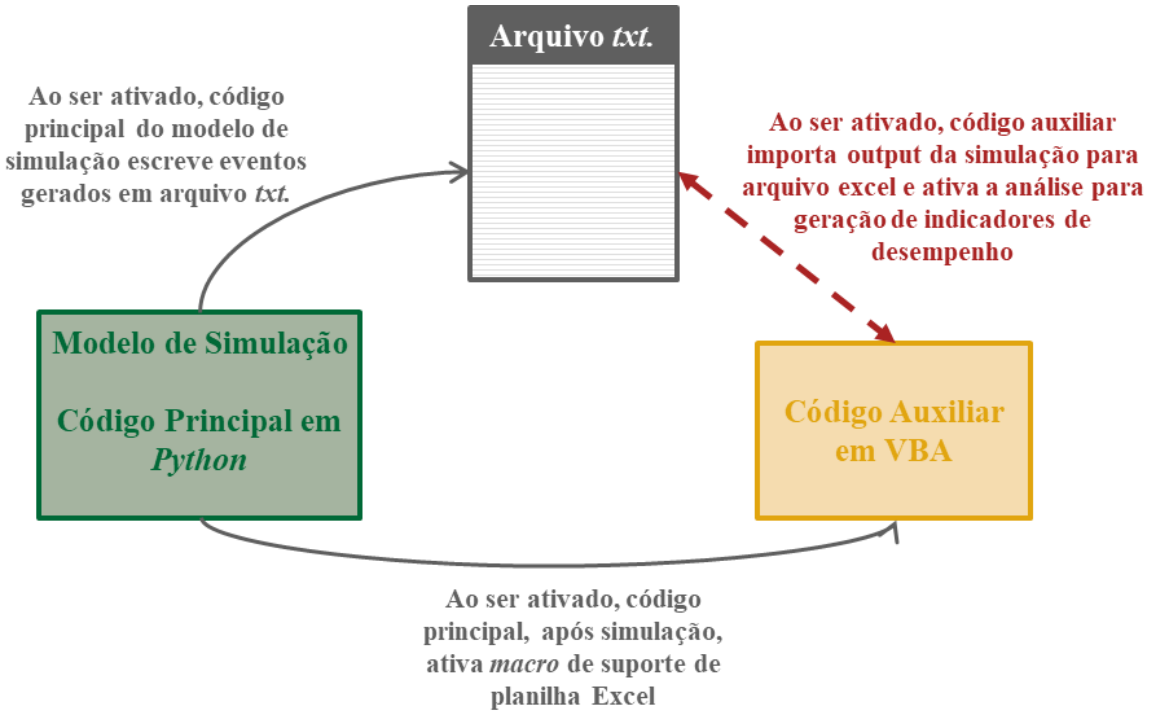
Por fim, a primeira etapa ainda engloba a geração de uma entidade *Store* para simular o *pré-shop* e *n* entidades *Resource*, para simular os estágios do sistema de produção, com o auxílio do *Simpy*©. Os recursos gerados estão sobre a classe "Máquinas" e são englobados em uma lista denominada "Estágio".

A segunda etapa do bloco principal consiste em chamar a função "GeracaoJobs", que dá início ao processo de simulação e geração de *jobs* no sistema, durante a simulação. Isto só é permitido, pois, após chamar esta função, o bloco principal aciona o comando *"env.run(until=TempoSimulacao)"*, que corresponde ao gatilho de início da simulação, terminando-a quando se estoura o tempo de simulação previamente definido como parâmetro de entrada. As chamadas desta função e do comando gatilho são feitas no mesmo número de vezes que o número de simulações definido como *input*.

A terceira etapa consiste em ativar a macro “Automatico” escrita em linguagem de programação VBA, atrelada à planilha Excel dos ANEXOS B e C. Esta *macro* importa os eventos gerados durante a simulação para planilha Excel e ativa as fórmulas que a analisam para gerar os indicadores de desempenho definidos na modelagem conceitual: *mean lead time*, *mean throughtput time*, *mean tardiness* e *percentage tardy*. Esta *macro* pode ser observada no ANEXO D do presente trabalho.

Vale ressaltar que o modelo de simulação, durante a execução da simulação, registra os eventos de simulação ocorridos, em ordem temporal, em arquivo *txt.*, que ao rodar a *macro* “Automatica”, é importada para o arquivo Excel (Figura 14).

Figura 14 - Arquivos pertencentes ao modelo de simulação e interfaces



Fonte: Elaborado pelo autor

Em relação ao *output* do sistema, já foi previamente afirmado que ocorre na forma da impressão de eventos em um arquivo txt., que é exportado para o Excel. Porém, ainda não foi mencionado a forma em que estes eventos são retratados. A Figura 15 mostra um exemplo de *output* da simulação, elaborado para fins de ilustração.

Figura 15 - Exemplo de *output* do modelo de simulação

Número da Simulação	Número do Job	Evento	Local	Tempo de Simulação	Due Date
1.0	Job1	Geração	Sistema	2.0	23.6
1.0	Job1	Entrada	Sistema	2.0	23.6
1.0	Job1	ChegadaOp	Operação1	2.0	-
1.0	Job1	InícioOp	Operação1	2.0	-
1.0	Job1	FimOp	Operação1	3.9	-
1.0	Job1	ChegadaOp	Operação2	3.9	-
1.0	Job1	InícioOp	Operação2	3.9	-
1.0	Job2	Geração	Sistema	4.4	28.1
1.0	Job2	Entrada	Sistema	4.4	28.1
1.0	Job2	ChegadaOp	Operação1	4.4	-
1.0	Job2	InícioOp	Operação1	4.4	-
1.0	Job3	Geração	Sistema	6.2	27.2
1.0	Job3	Entrada	Sistema	6.2	27.2
1.0	Job3	ChegadaOp	Operação1	6.2	-
1.0	Job3	InícioOp	Operação1	6.2	-
1.0	Job2	FimOp	Operação1	6.9	-
1.0	Job2	ChegadaOp	Operação2	6.9	-
1.0	Job2	InícioOp	Operação2	6.9	-
1.0	Job1	FimOp	Operação2	7.2	-
1.0	Job1	ChegadaOp	Operação3	7.2	-
1.0	Job1	InícioOp	Operação3	7.2	-
1.0	Job4	Geração	Sistema	7.9	25.4
1.0	Job4	Entrada	Sistema	7.9	25.4
1.0	Job4	ChegadaOp	Operação1	7.9	-
1.0	Job4	InícioOp	Operação1	7.9	-
1.0	Job3	FimOp	Operação1	9.0	-
1.0	Job3	ChegadaOp	Operação2	9.0	-
1.0	Job3	InícioOp	Operação2	9.0	-
1.0	Job4	FimOp	Operação1	9.3	-
1.0	Job4	ChegadaOp	Operação2	9.3	-
1.0	Job5	Geração	Sistema	9.9	25.9
1.0	Job5	Entrada	PréShop	9.9	25.9
1.0	Job2	FimOp	Operação2	10.1	-
1.0	Job2	ChegadaOp	Operação3	10.1	-
1.0	Job2	InícioOp	Operação3	10.1	-
1.0	Job4	InícioOp	Operação2	10.1	-
1.0	Job1	FimOp	Operação3	10.5	-
1.0	Job1	ChegadaOp	Operação4	10.5	-
1.0	Job1	InícioOp	Operação4	10.5	-
1.0	Job6	Geração	Sistema	11.8	26.6
1.0	Job6	Entrada	Sistema	11.8	26.6
1.0	Job6	ChegadaOp	Operação1	11.8	-
1.0	Job6	InícioOp	Operação1	11.8	-
1.0	Job3	FimOp	Operação2	12.5	-
1.0	Job3	ChegadaOp	Operação3	12.5	-

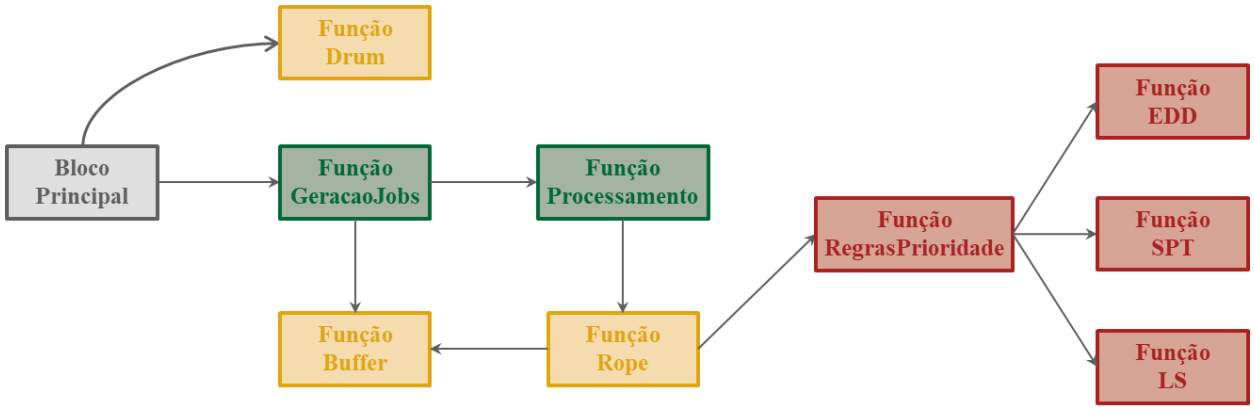
Fonte: Elaborado pelo autor

Como pode-se observar, a impressão de um evento da simulação ocorre com o apoio de seis colunas: número da simulação, nome do *job*, evento, local do evento, tempo de simulação e *due date*. Vale ressaltar que o *due date* só é impresso no momento de geração do *job*, de forma a simplificar o *output*, evitando a repetição constante deste número.

Os eventos a serem impressos durante uma simulação são: a geração de um *job*; sua entrada e saída do *pré-shop*; entrada, início da operação e fim da operação de um *job* em todos os estágios em seu caminho; a saída do *job* do sistema.

Por fim, a Figura 16 retrata a relação existente entre as principais funções do código elaborado que corresponde ao modelo de simulação. Para fins de esclarecimento, quando uma função aciona a outra, ela está conectada à esta por uma linha, sendo que aquela função que recebe a seta é a acionada. Funções de suporte são amplamente utilizadas no modelo e, para melhor representação gráfica, não foram consideradas.

Figura 16 - Funções do modelo computacional e suas interfaces



Fonte: Elaborado pelo autor

4.3. Verificação e Validação

O modelo de simulação passou por um rigoroso processo de verificação, que foi conduzido de forma modular, validando parte por parte do modelo. Todas as funções foram verificadas individualmente. No final, o modelo foi verificado como um todo por meio de uma série de testes. A seguir, será retratado como foi conduzido o processo de verificação das principais partes do modelo.

4.3.1. Verificação da Geração de *jobs* e *due date*

A etapa de verificação do processo de geração de *jobs* e *due dates* focou em garantir dois fatores principais: que a geração de *jobs* seguisse efetivamente a distribuição e parâmetros escolhidos; que o processo de geração de *jobs* fosse conduzido até o tempo de simulação determinado como parâmetro de entrada.

Para garantir que a geração de *jobs* seguisse a distribuição escolhida, uma série de simulações foram conduzidas para cada tipo de distribuição e os tempos entre gerações foram plotados em gráficos. Calculou-se então a média, desvio padrão, tempo mínimo e tempo máximo, conforme o tipo de distribuição. O mesmo processo foi conduzido para a geração de *due dates*.

O que se observou foi que o modelo de simulação está aderente a qualquer um dos tipos de distribuição habilitados dentro do modelo de simulação: distribuição exponencial, normal, uniforme e fixa. Esta observação é válida tanto para a geração de *jobs* quanto para a determinação de *due dates*.

4.3.2. Verificação do Processamento nos Estágios

A etapa de verificação da operação de processamento dos *jobs* dentro dos estágios foi feita de modo independente do método de controle DBR. Para isto, um *time buffer* limite inatingível foi estipulado, de forma a evitar seu acionamento e a entrada de *jobs* no *pré-shop*.

Desta forma, o sistema de produção simulado funcionou sem nenhum mecanismo de controle de *jobs* em processo. Vale ressaltar que o método de controle DBR também foi verificado de forma separada seguindo metodologia específica, tema a ser discutido adiante.

A etapa de verificação da função “Processamento” foi dividida em duas partes: parte conceitual, para garantir que o funcionamento da passagem de *jobs* dentro do modelo de simulação é coerente com o de um *flow shop* genérico; parte quantitativa, para assegurar que os tempos de processamento estão compatíveis com o tipo de distribuição e parâmetros temporais determinados como parâmetros de entrada.

Na parte conceitual, buscou-se garantir os seguintes mecanismos: (i) o fluxo dentro do sistema de produção simulado é sempre o mesmo; (ii) o número de máquinas em cada estágio corresponde ao valor determinado como parâmetro de entrada; (iii) caso certo *job* chegue em determinado estágio e todas as máquinas estejam ocupadas, ele deve aguardar na fila deste estágio; (iv) o ordenamento de *jobs* na fila dos estágios deve ser sempre FIFO; (v) a probabilidade de certo *job* passar pela máquina deve seguir o parâmetro de entrada correspondente a esta probabilidade, determinado individualmente para cada máquina.

Para verificar (i), simulou-se cerca de 20 vezes o modelo de simulação, com cenários diferentes de estágios, em termos de número de estágios, número de máquinas e probabilidade de estágio estar no caminho de *jobs*. Observando os eventos exportados da simulação, em todos os cenários testados, observou-se sempre o mesmo sentido de fluxo.

Para verificar (ii), simulou-se cerca de 20 vezes o modelo de simulação, variando o número de máquinas no estágio. Estas simulações foram conduzidas com taxa de geração grande o suficiente (não a ponto de acionar o DBR) para lotar as máquinas dos primeiros estágios, estágios efetivamente usados para fins de verificação. Em todos os estágios avaliados, em todos os cenários, o número de máquinas da simulação correspondeu ao número de máquinas determinado como *input*.

A verificação de (iii) e (iv) ocorreu em paralelo à verificação de (ii). Em todos os estágios avaliados, em todos os cenários, após o atingimento do número de máquinas determinado, os *jobs* excedentes aguardaram na fila de entrada. Nenhum tipo de reordenação foi observado. Logo, a ordenação das filas de entrada é compatível com a regra FIFO.

A verificação de (v) ocorreu da seguinte forma: foram estabelecidos cinco cenários com 10 estágios cada, com tempos de processamento dos estágios extremamente rápidos e tempos entre chegada de *jobs* no sistema extremamente pequenos. A cada estágio, foi atribuído a mesma probabilidade de passagem de *jobs* pelo estágio. Os cinco cenários testaram diferentes probabilidades de passagem. Os cenários avaliados e os resultados do teste podem ser vistos na Figura 17.

Figura 17 - Teste de verificação de probabilidade de passagem

Cenário Testado	Probabilidade de Passagem	*Jobs* gerados e concluídos durante simulação	Média entre estágios do número de jobs processados e concluídos na simulação	Parâmetro Comparativo
1	100%	488	488	100,0%
2	75%	511	384	75,1%
3	50%	494	251	50,8%
4	25%	498	121	24,3%
5	0%	503	0	0,0%

Fonte: Elaborado pelo autor

Como os resultados ficaram extremamente próximos e dentro de um intervalo de confiança de 95% de probabilidade, comparativamente ao parâmetro de entrada estabelecido para os cinco cenários de teste, o modelo de simulação elaborado se mostrou aderente ao modelo conceitual desenhado, em termos de processamento, dando fim a verificação da parte conceitual.

Na parte quantitativa, buscou-se garantir que os tempos de processamento dos estágios fossem compatíveis com o tipo de distribuição escolhida e os parâmetros escolhidos como *input*. Para isto, o teste conduzido foi similar ao teste conduzido para verificar o processo de geração de *jobs*.

Uma série de simulações foram conduzidas para cada tipo de distribuição, com cada estágio seguindo a mesma distribuição. Os tempos entre gerações foram plotados em gráficos e calculados a média, desvio padrão, tempo mínimo e tempo máximo, conforme aplicabilidade à distribuição.

O que se observou, com o auxílio de um intervalo de confiança, foi que o modelo de simulação está aderente a qualquer um dos tipos de distribuição habilitados dentro do modelo de simulação para processamento de *jobs*: distribuição exponencial, normal, uniforme, fixa, triangular e Erlang.

Desta forma, a etapa de verificação da função “Processamento” foi encerrada, mostrando que o modelo de simulação desenvolvido está de acordo com o modelo conceitual concebido.

4.3.3. Verificação do DBR

A etapa de verificação do método de controle DBR foi dividida em três etapas de forma a verificar separadamente as três funções relacionadas aos três componentes principais do método DBR: *drum*, *buffer* e *rope*.

A checagem do *drum* foi feita por meio da criação de uma impressão intermediária da posição do estágio gargalo, de forma a identificar qual estágio o modelo identifica como gargalo. A impressão gerada foi comparada com o estágio gargalo real, calculado pelo autor deste trabalho com base nos *inputs* determinados anteriormente à execução da simulação.

Este teste foi realizado em pelo menos 10 cenários diferentes para cada tipo de distribuição possível em termos de tempo de processamento: exponencial, normal, uniforme, fixa, triangular e Erlang. Em outras palavras, o teste da função "Drum" foi realizado em mais de 60 cenários diferentes, retornando sempre o valor esperado.

A checagem do *buffer* foi feita de forma similar a checagem do *drum*. O valor do *time buffer* a cada etapa de checagem foi impresso juntamente a simulação. O valor real do time buffer foi calculado manualmente, com base no retrato dos estágios no momento da checagem, ou seja, número de *jobs* na fila de cada estágio entre a entrada do sistema e o estágio gargalo.

Este processo foi repetido por cerca de 30 vezes. Em todas as checagens, os valores foram compatíveis, concluindo a verificação da função "Buffer". A posição do recurso gargalo foi variada propositalmente durante as checagens, porém, o modelo se mostrou aderente ao método DBR.

Por fim, a realização da verificação do componente *rope* foi realizada de forma similar a verificação do *buffer*, pela impressão do *time buffer* durante a simulação. A cada checagem foi possível visualizar se o mecanismo da função "Rope" funcionava corretamente, ou seja, se o tamanho do *time buffer* limite efetivamente limita e controla a entrada de *jobs* no sistema.

Portanto, checou-se a aderência a dois eventos: (i) se o *time buffer* atual for maior ou igual que o *time buffer* limite, nenhum *job* é permitido de entrar no sistema; (ii) se o *time buffer* atual for menor que o *time buffer* limite, o *job* mais prioritário do *pré-shop* é liberado para o sistema.

Cerca de 15 simulações foram realizadas e a checagem do mecanismo da função "Rope" foi realizada durante todo os instantes destas simulações. O mecanismo implementado no modelo de simulação se mostrou compatível ao conceito apresentado durante a etapa de modelo conceitual, mostrando que o modelo de simulação está compatível com o método de controle DBR.

A partir da verificação destas três funções separadamente, finalizou-se a etapa de verificação do mecanismo de controle da produção DBR. Vale ressaltar que as regras de sequenciamento não foram testadas durante a verificação do método DBR, de forma que a sua verificação ocorreu de forma apartada, como será apresentado a seguir.

4.3.4. Verificação das Regras de Sequenciamento

A etapa de verificação das regras de sequenciamento aceitas pelo modelo de simulação foi feita por meio da impressão do *pré-shop* durante a simulação. A lista contendo as informações de cada *job* na fila foram impressas sempre que a ordenação do *pré-shop* foi realizada, de forma a possibilitar a comparação entre a ordem antes e depois da ordenação.

Para a função “EDD”, “SPT” e “LS”, o processo de checagem foi o mesmo: a cada simulação, no instante de cada reordenação do *pré-shop*, calculou-se os parâmetros responsáveis pela ordenação do *pré-shop* para cada *job*, ou seja, calculou-se o *due date*, tempo esperado de processamento e folga, respectivamente para funções “EDD”, “SPT” e “LS”.

Em seguida, realizou-se a ordenação ideal por meio destes parâmetros e comparou-se esta ordenação com a ordenação resultante da simulação. Este processo foi realizado em 10 simulações para cada uma das funções. A ordenação não apresentou diferenças em relação a ordenação resultante da simulação.

Por fim, a função “RegrasPrioridade” foi testada em 10 checagens. Para testar esta função, comparou-se a ordenação resultante da simulação com a ordenação ideal de acordo com a regra de prioridade escolhida (da mesma maneira que o procedimento utilizado para checar as outras funções de ordenamento).

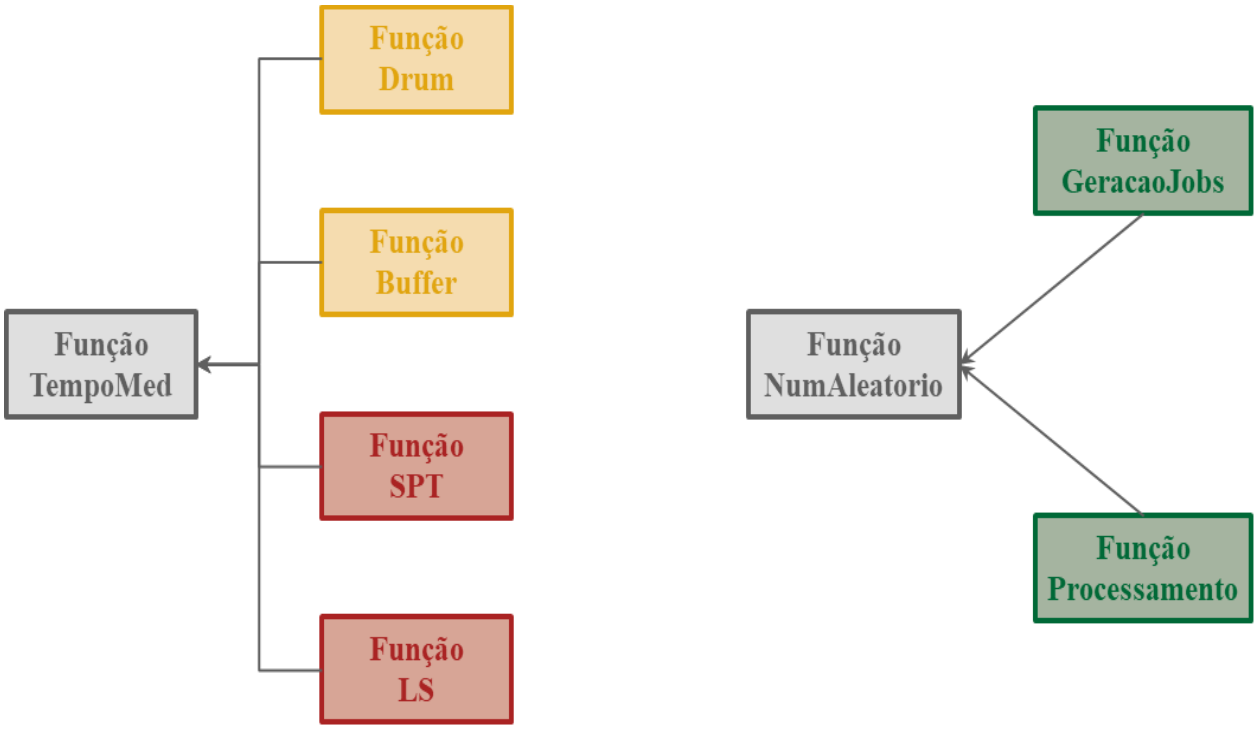
A ordenação ideal se mostrou compatível com a ordenação resultante da simulação em todos os casos. Desta maneira, concluiu-se a etapa de verificação das regras de sequenciamento.

4.3.5. Verificação das Funções de Suporte

Nenhum procedimento específico foi realizado para validar as funções de suporte. Porém, pode-se afirmar que estas funções estão corretas em sua função, pois são extensamente utilizadas no modelo dentro de outras funções que foram validadas. Este foi o motivo da não elaboração de um método para verificação destas funções.

A Figura 18 ilustra a aplicação das duas funções de suporte do modelo, mostrando sua ampla presença no código do modelo de simulação. Para fins de esclarecimento, quando uma função aciona a outra, ela está conectada à está por uma linha, sendo que aquela função que recebe a seta é a acionada.

Figura 18 - Funções de suporte do modelo de simulação



Fonte: Elaborado pelo autor

4.3.6. Validação

Enquanto a etapa de verificação consiste em verificar que o modelo de simulação está isento de erros e compatível com o modelo conceitual concebido, a validação do modelo consiste em garantir sua utilidade. Desta forma, o modelo pode ser considerado válido, pois possui as seguintes potenciais aplicações:

- **Aplicação científica –** o modelo de simulação pode ser aplicado na geração de produção científica relacionada a controle de sistemas de produção via DBR. Como visto durante a revisão bibliográfica, o método DBR tem sido um tema discutido e estudado de forma extensa nos últimos anos. O presente trabalho retrata esta aplicação do modelo de simulação elaborado.
- **Aplicação prática –** o modelo de simulação pode ser aplicado no estudo de sistemas de produção com características de produção *flow shop* genérico reais, cujas simplificações necessárias para adequação do sistema para o modelo não impactem de forma relevante nos resultados gerados. Portanto, o modelo de simulação gerado pode ser muito útil no estudo de partes de sistemas de produção, como, por exemplo, no estudo de implementação do método de controle DBR para produção de uma peça dentro de uma linha de produção.

- **Aplicação acadêmica –** o modelo de simulação pode ser aplicado para fins acadêmico, ou seja, de ensino e aprendizado. O modelo de simulação desenvolvido apresenta-se de forma bem documentada, de forma a facilitar o entendimento da lógica de funcionamento e construção por trás do modelo. Desta forma, pode ser utilizado para: (i) ensino sobre o método de controle DBR; (ii) ensino sobre o desenvolvimento de modelos de simulação em *Python*© com o auxílio do *Simpy*©.

5. INSTÂNCIAS DE SIMULAÇÃO

O modelo de simulação foi utilizado para testar o efeito do *time buffer* no desempenho do sistema, sob diferentes configurações (cenários), com a finalidade de atingir os objetivos específicos propostos neste trabalho. Esta seção possui o intuito de apresentar as instâncias de simulação, que consistem nos cenários que serão fornecidos de parâmetro de entrada para a simulação, cujos resultados serão posteriormente analisados no próximo capítulo.

Para a determinação das instâncias de simulação, partiu-se de um cenário padrão para a criação dos demais cenários de simulação. Foram criados uma série de cenários que variam apenas um tipo de grandeza. Diferentes simulações foram rodadas com diferentes tamanhos diferentes de *time buffer* para cada cenário. O objetivo é entender como que o impacto do tamanho do *time buffer* nos indicadores de desempenho se comporta conforme se variam isoladamente certos parâmetros de entrada, bem como estes parâmetros de entrada influenciam os indicadores de desempenho do sistema.

O cenário padrão para a criação de cenários alternativos usados na análise foi adaptado de Thurer et al. (2017) e pode ser visto nas Tabelas 2 e 3. Consiste em um sistema de produção com sete estágios, sendo seis idênticos e um com tempo maior de processamento, sendo este o estágio gargalo do sistema. A posição escolhida para o estágio gargalo no cenário padrão foi a de número quatro dentro do sistema de produção.

Tabela 2 - *Inputs* gerais e de geração para cenário padrão

Parâmetro de Entrada	Valor
Número de Simulações	5
Tempo de Simulação	100
Regra de Sequencionamento	FIFO
Distribuição – tempo entre gerações	Exponencial
Tempo médio entre gerações	0,9
Tipo de distribuição - due date	Uniforme
Tempo mínimo para due date	15,0
Tempo máximo para due date	25,0

Fonte: Elaborado pelo autor

Tabela 3 - *Inputs* de estágios para cenário padrão

Estágios	Número de máquinas	Tipo de distribuição	Tempo médio	Desvio padrão	Estágio Gargalo	Probabilidade de job estar no caminho
Estágio 1	1	Normal	1,0	0,2		100%
Estágio 2	1	Normal	1,0	0,2		100%
Estágio 3	1	Normal	1,0	0,2		100%
Estágio 4	1	Normal	1,3	0,2	Sim	100%
Estágio 5	1	Normal	1,0	0,2		100%
Estágio 6	1	Normal	1,0	0,2		100%
Estágio 7	1	Normal	1,0	0,2		100%

Fonte: Elaborado pelo autor

Os fatores objetivo dos testes realizados foram:

- **Razão entre tempo médio de entrada e tempo médio de processamento –** este parâmetro está diretamente relacionado ao nível de utilização do sistema de produção simulado e a capacidade deste de atender a demanda. A razão do cenário padrão é de 0,9x, sendo que para testar esta grandeza, outros dois valores foram simulados, um maior e outro menor. As razões e parâmetros de entrada para os três cenários a serem avaliados são apresentados na Tabela 4. Os outros parâmetros são mantidos iguais ao do cenário padrão.

Tabela 4 - Razão entre tempos médios de geração e processamentos testados

Cenários	Razão entre tempo médio de geração e de processamento	Tempo médio entre gerações	Tempo médio de processamento (ex gargalo)	Tempo médio de processamento do gargalo	Desvio padrão
Cenário 1	0,9	0,9	1,0	1,3	0,2
Cenário 2	0,7	0,7	1,0	1,3	0,2
Cenário 3	1,1	1,1	1,0	1,3	0,2

Fonte: Elaborado pelo autor

- **Severidade do gargalo –** esta grandeza mede o quanto o estágio leva a mais de tempo para processar determinado *job*. Segundo Thurer et al. (2017), a severidade de um estágio gargalo pode ser determinado pelo percentual a mais de tempo médio de processamento do estágio gargalo, comparativamente aos demais estágios. O cenário padrão possui severidade do gargalo de 30%. Para a criação de cenários de simulação para testar esta grandeza, outros dois valores foram determinados - um menor e outro maior. Estes podem ser vistos na Tabela 5. Os demais parâmetros foram mantidos iguais aos do cenário padrão.

Tabela 5 - Cenários testados para severidade do gargalo

Cenários	Severidade do gargalo	Tempo médio de processamento (ex gargalo)	Tempo médio de processamento do gargalo	Desvio padrão
Cenário 4	30%	1,0	1,3	0,2
Cenário 5	15%	1,0	1,15	0,2
Cenário 6	60%	1,0	1,6	0,2

Fonte: Elaborado pelo autor

- **Regras de Sequenciamento –** as quatros regras de sequenciamento serão testadas, dando origem a quatro cenários distintos. A Tabela 6 ilustra os cenários gerados pelas regras de sequenciamento. Os demais parâmetros foram mantidos iguais aos do cenário padrão.

Tabela 6 - Cenários com base nas regras de sequenciamento

Cenários	Regra de sequenciamento
Cenário 7	FIFO
Cenário 8	SPT
Cenário 9	EDD
Cenário 10	LS

Fonte: Elaborado pelo autor

- **Posição do recurso gargalo –** a posição do recurso gargalo dentro do sistema de produção também será alvo de testes. Três cenários diferentes serão testados, como demonstra a Tabela 7. Os demais parâmetros serão mantidos iguais aos do cenário padrão, inclusive tempos de processamento.

Tabela 7 - Cenários com base na posição do estágio gargalo

Cenários	Posição do estágio gargalo
Cenário 11	Estágio 4
Cenário 12	Estágio 2
Cenário 13	Estágio 6

Fonte: Elaborado pelo autor

- **Variância da distribuição normal –** o desvio padrão dos tempos de processamento também será alvo de testes. Dois cenários diferentes serão testados (Tabela 8). Os demais parâmetros serão mantidos iguais aos valores do cenário padrão, inclusive tempos de processamento.

Tabela 8 - Cenários com base no desvio padrão do tempo de processamento

Cenários	Desvio padrão do tempo de processamento
Cenário 14	0,2
Cenário 15	0,4

Fonte: Elaborado pelo autor

6. ANÁLISE DE EXPERIMENTOS

Para a análise de cada cenário definido no capítulo 5, utilizou-se sete níveis diferentes de *time buffer* limite para método de controle DBR. Os níveis de *time buffer* foram definidos com base em experimentos preliminares de forma a atingir níveis entre 2,5 vezes a 20 vezes o tempo de processamento do sistema do cenário padrão, em intervalos constantes e de forma aproximada. Os valores de *time buffer* simulados para cada cenário podem ser vistos na Tabela 9. Valores temporais serão apresentados sem uma unidade de tempo definida, mas na mesma base de valores.

Tabela 9 - Níveis de *time buffer* para teste de cada cenário

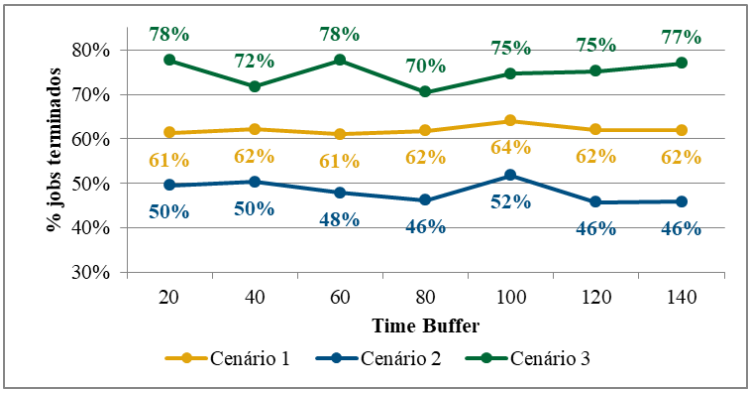
Teste por cenário	Tamanho do time buffer
Teste 1	20
Teste 2	40
Teste 3	60
Teste 4	80
Teste 5	100
Teste 6	120
Teste 7	140

Fonte: Elaborado pelo autor

6.1. Análise da Razão entre Tempos de Geração e Processamento

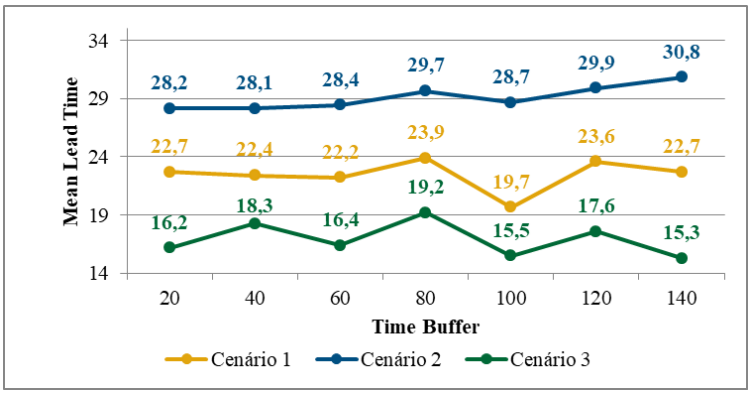
Os resultados das simulações com base na variação da razão entre tempos de geração e processamentos, para os cenários 1 (padrão), 2 e 3 definidos no capítulo 5, podem ser vistos nos Gráficos 1, 2, 3, 4 e 5.

Gráfico 1 - Percentual de *jobs* terminados para cenários 1, 2 e 3



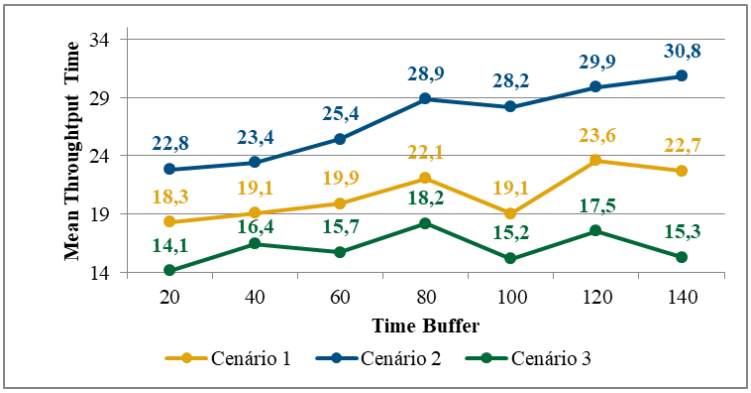
Fonte: Elaborado pelo autor

Gráfico 2 - *Mean lead time* para cenários 1, 2 e 3



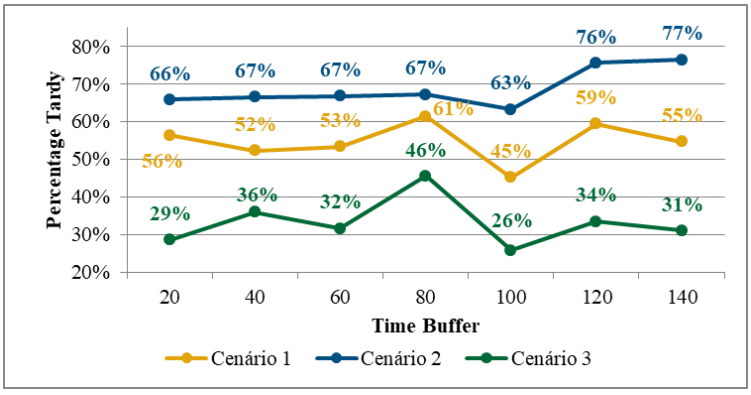
Fonte: Elaborado pelo autor

Gráfico 3 - *Mean throughtput time* para cenários 1, 2 e 3



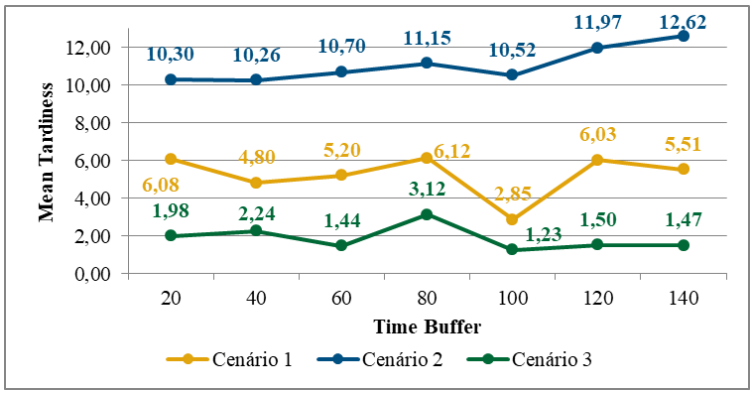
Fonte: Elaborado pelo autor

Gráfico 4 - *Percentage tardy* para cenários 1, 2 e 3



Fonte: Elaborado pelo autor

Gráfico 5 - *Mean tardiness* para cenários 1, 2 e 3



Fonte: Elaborado pelo autor

No Gráfico 1, pode-se perceber como o percentual de *jobs* terminados entre *jobs* gerados é diretamente influenciado pela razão temporal aqui tratada. O percentual de jobs terminados consiste na média da razão entre números de *jobs* gerados e número de *jobs* terminados durante as simulações, sendo un indicador da produtividade do cenário simulado.

Isto ocorre, pois, apesar da média do número de *jobs* terminados nos três cenários ser similar (345, 350 e 346, para os cenários 1, 2 e 3), o número de *jobs* gerados no sistema é diretamente proporcional a razão temporal (556, 727 e 462, para os cenários 1, 2 e 3). Isto demonstra o principal teorema da ToC: o estágio gargalo dita o ritmo de produção de todo o sistema de produção.

Em termos de indicadores de desempenho, claramente se percebe que quanto maior a razão entre o intervalo médio entre chegadas e tempos de processamento, pior é o desempenho dos quatro indicadores de desempenho escolhidos. Ou seja, o cenário 2, com razão de 1,1, possui maiores *mean lead time, mean throughtput time, percentage tardy e mean tardiness*.

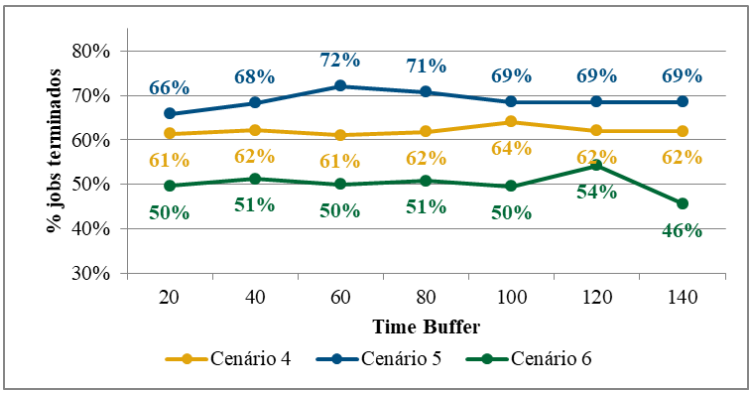
Em termos de *time buffer*, é possível notar o impacto deste fator, principalmente para os indicadores de desempenho *percentage tardy* e *mean tardiness*. Nos três cenários, o valor do *time buffer* influencia a curva dos indicadores de desempenho de forma similar, com o melhor cenário sendo um *time buffer* perto de 100 unidades de tempo.

Porém, percebe-se que o impacto do *time buffer* nos indicadores de desempenho não produz variações claras entre cenários. Em outras palavras, o comportamento das curvas é similar, com o melhor cenário de simulação sendo igual para todos. Além disso, para os três cenários, o *time buffer* limite para a ativação do DBR alcança cerca de 120 unidades de tempo. Isto pode ser notado pela diferença relativamente pequena entre *mean lead time* e *mean throughtput time* para *time buffer* limite de 120 e 140 unidades de tempo.

6.2. Análise da Severidade do Gargalo

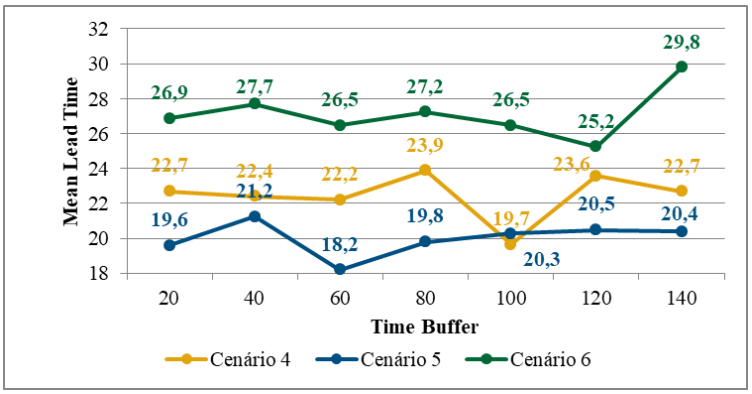
Os resultados das simulações da variação da severidade do gargalo, para os cenários 4 (padrão), 5 e 6 definidos no capítulo 5, podem ser vistos nos Gráficos 6, 7, 8, 9 e 10.

Gráfico 6 - Percentual de *jobs* terminados para cenários 4, 5 e 6



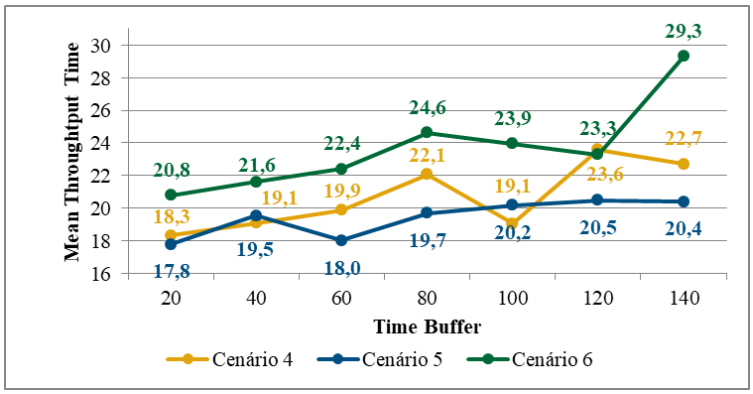
Fonte: Elaborado pelo autor

Gráfico 7 - *Mean lead time* para cenários 4, 5 e 6



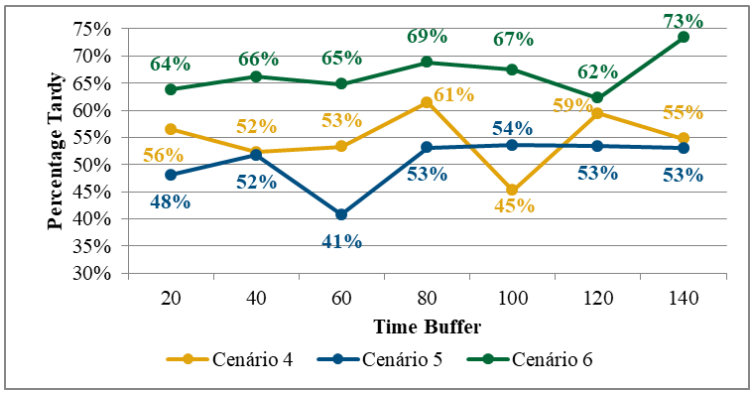
Fonte: Elaborado pelo autor

Gráfico 8 - *Mean throughtput time* para cenários 4, 5 e 6



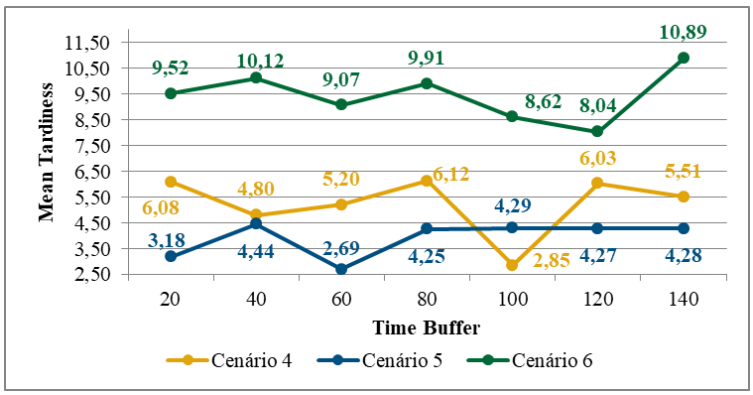
Fonte: Elaborado pelo autor

Gráfico 9 - *Percentage tardy* para cenários 4, 5 e 6



Fonte: Elaborado pelo autor

Gráfico 10 - *Mean tardiness* para cenários 4, 5 e 6



Fonte: Elaborado pelo autor

No Gráfico 6, pode-se perceber como o percentual de *jobs* terminados entre *jobs* gerados é diretamente influenciado pela severidade do gargalo. O cenário 5, com gargalo com menor severidade, apresentou, considerando a média, 20% a mais percentualmente de *jobs* concluídos do que o cenário 6, com severidade quatro vezes maior do que a do cenário 5.

Diferentemente do teste realizado na seção 5.1, a variável que efetivamente mudou entre cenários não foi o número de *jobs* gerados, mas sim o número de *jobs* concluídos. Enquanto as médias de *jobs* gerados nos testes foram de 556, 567 e 564, respectivamente, para os cenários 4 (padrão), 5 e 6, a média de *jobs* concluídos foi de 345, 391 e 283, respectivamente, para a mesma sequência de cenários. Mais uma vez, pode-se observar o princípio central da ToC: o estágio gargalo dita o ritmo de produção de todo o sistema de produção.

Além disto, observando os Gráficos de 6 a 10, percebemos que não é somente no quesito número de *jobs* concluídos que a severidade do gargalo influencia diretamente nos resultados obtidos. Conforme mostra o presente experimento, quanto maior é a severidade do gargalo, maior é a tendência de se obter piores indicadores de desempenho, considerando que o resto dos parâmetros de entrada produzem valores equivalentes.

Em termos de impacto do *time buffer* nos indicadores de desempenho, observou-se dois eventos importantes. O primeiro evento consiste no deslocamento do tamanho do *time buffer* limite que melhora os indicadores de desempenho. Em outras palavras, quanto maior é a severidade do *time buffer*, maior é o *time buffer* limite que trouxe melhores resultados nos testes realizados, em termos de indicadores de desempenho.

Por exemplo, observando os Gráficos de 6 a 10, é possível notar que, para o cenário 4 (padrão), um *time buffer* de 100 unidades de tempo apresentou melhores resultados. Para o cenário 5, um *time buffer* de 60 unidades de tempo resultou em melhor desempenho. A variação do tempo total de processamento possui impacto neste evento. Porém, em testes adicionais realizados no modelo, testando-se cenários com aumento equivalente de tempo de processamento, mas sem alteração na severidade do gargalo, os resultados não se mostraram tão relevantes como os resultados apresentados nos Gráficos de 6 a 10.

O segundo evento observado foi o fato do método de controle DBR possuir um impacto maior em cenários de maior severidade do que em cenários de menor severidade. Nos Gráficos 6 a 10, é possível observar que, conforme o mecanismo de controle deixa de exercer controle efetivo sobre a entrada de *jobs* no sistema, no cenário 6, o impacto negativo sobre os indicadores de desempenho é consideravelmente maior do que nos outros cenários.

Por exemplo, no cenário 6, um *time buffer* de 140, que possui pouco controle sobre a entrada de *jobs*, uma vez que o *mean lead time* está muito próximo ao *mean throughtput time* (indicando que poucos *jobs* entram no *pré-shop*), resulta em piora significativa nos indicadores de desempenho *percentage tardy*. O *percentage tardy* aumenta mais de 10% em relação ao *time buffer* anterior testado, enquanto o *mean tardiness* apresenta um aumento percentual de cerca de 35% em relação ao *time buffer* anterior.

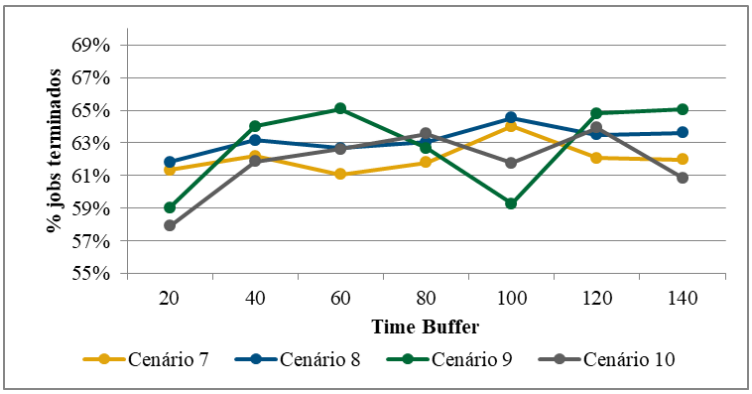
Esta grande diferença não é observada nos cenários 4 (padrão) e 5. Por exemplo, no cenário 5, a partir do time buffer de 100 unidades temporais, em que o *mean lead time* quase se iguala ao *mean throughtput time*, os indicadores de desempenho não se mostram tão sensíveis.

Esta observação está em linha com o estudo realizado por Thurer et al. (2017). Eles concluíram que concluiu que o método DBR é mais eficaz do que sistemas sem mecanismos de controle e o método WLC em *flow shops* flexíveis e *job shops* flexíveis, com base nos mesmos indicadores de desempenho analisados neste trabalho.

6.3. Análise das Regras de Sequenciamento

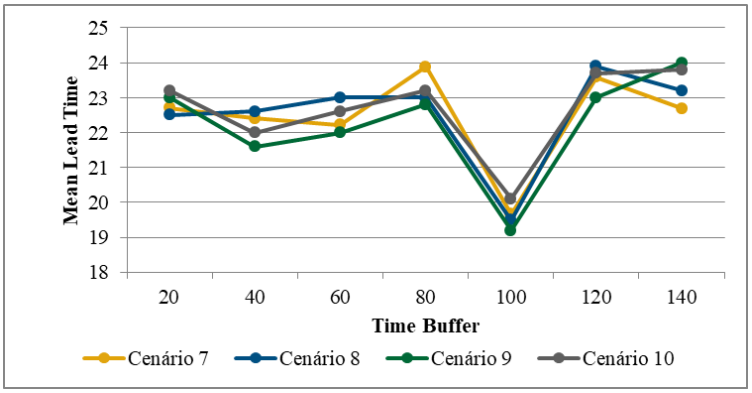
Os resultados das simulações com as regras de sequenciamento, para os cenários 7 (padrão), 8, 9 e 10 definidos no capítulo 5, podem ser vistos nos Gráficos 11, 12, 13, 14 e 15.

Gráfico 11 - Percentual de *jobs* terminados para cenários 7, 8, 9 e 10



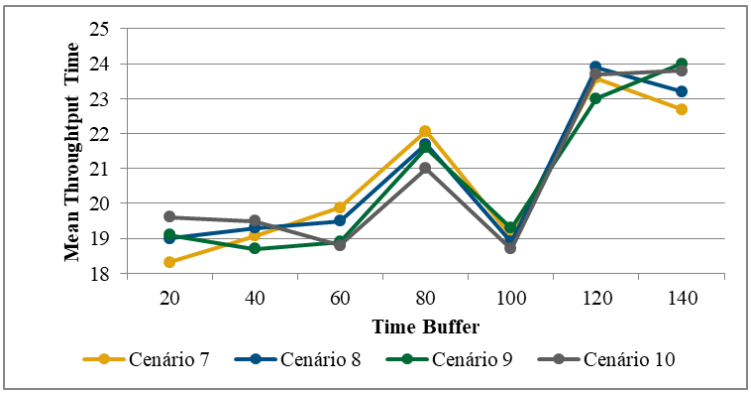
Fonte: Elaborado pelo autor

Gráfico 12 - *Mean lead time* para cenários 7, 8, 9 e 10



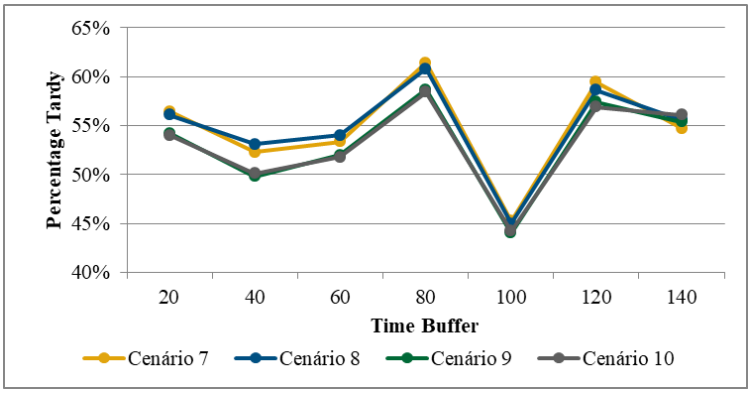
Fonte: Elaborado pelo autor

Gráfico 13 - *Mean throughtput time* para cenários 7, 8, 9 e 10



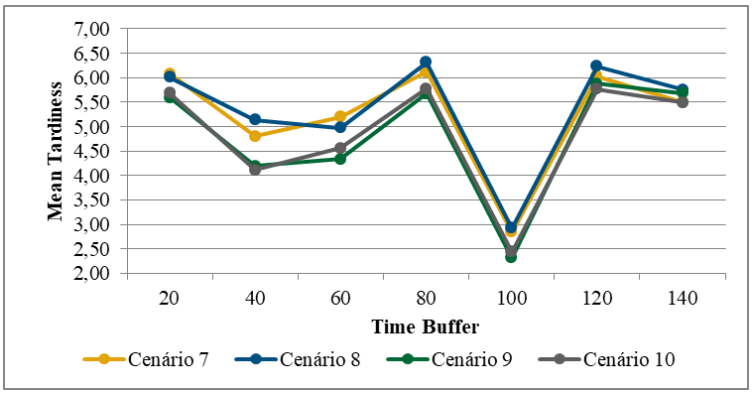
Fonte: Elaborado pelo autor

Gráfico 14 - *Percentage tardy* para cenários 7, 8, 9 e 10



Fonte: Elaborado pelo autor

Gráfico 15 - *Mean tardiness* para cenários 7, 8, 9 e 10



Fonte: Elaborado pelo au tor

Observando-se os Gráficos 11, 12 e 13, não se nota nenhuma diferença causada pelas diferentes regras de sequenciamento testadas, em termos de percentual de *jobs* gerados que são concluídos, *mean lead time* e *mean throughtput time*. Isto pode ser explicado pelo fato das regras de sequenciamento não interferirem no fluxo de *jobs* dentro do sistema de produção simulado, mas somente controlar a ordem de entrada dos *jobs* no sistema.

Como todos os *jobs* passam por todas as máquinas nos cenários simulados, e o tempo de processamento é característica das máquinas, todos os *jobs* possuem a mesma expectativa de tempo de processamento. Este fator não só explica a ausência de diferenças significativas com base nas informações dos Gráficos 11, 12 e 13, como também explica a ausência de diferenças relevantes entre as regras de sequenciamento FIFO e SPT. O mesmo ocorre também para a ausência de diferenças relevantes entre as regras de sequenciamento EDD e LS (Gráficos de 11 e 15).

A única diferença perceptível entre os cenários simulados reside nos indicadores de desempenho *percentage tardy* e *mean tardiness* (Gráficos 14 e 15). Nestes indicadores, as regras de sequenciamento EDD e LS tiveram desempenho superior em *time buffers* limites baixos. Esta diferença se explica pela priorização que as regras de sequenciamento EDD e LS fornecem a *jobs* mais prioritários.

Este resultado é compatível com o estudo realizado por Daniel & Guide (1997), em que a regra de sequenciamento EDD se mostrou significantemente superior do que outras regra s analisadas, em termos de *percentage tardy* e *lateness*.

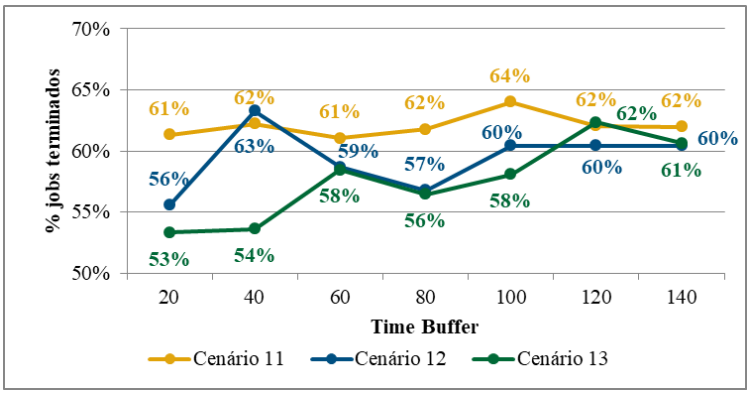
Embora exista, a diferença não é significativa. Isto ocorre, pois *jobs* gerados posteriormente possuem a tendência de possuir *due date* maior, em função do método de geração de *due dates*, abordado no capítulo 4 deste trabalho.

Além disso, conforme o tamanho do *time buffer* limite simulado aumenta, esta diferença diminui, uma vez que diminui a influência de mecanismos de controle e regras de sequenciamento para a entrada de *jobs* no sistema. Por fim, a influência do tamanho do *time buffer* para o valor obtido nos indicadores de desempenho não muda entre regras de sequenciamento. O seu comportamento é similar ao do cenário padrão.

6.4. Análise da Posição do Estágio Gargalo

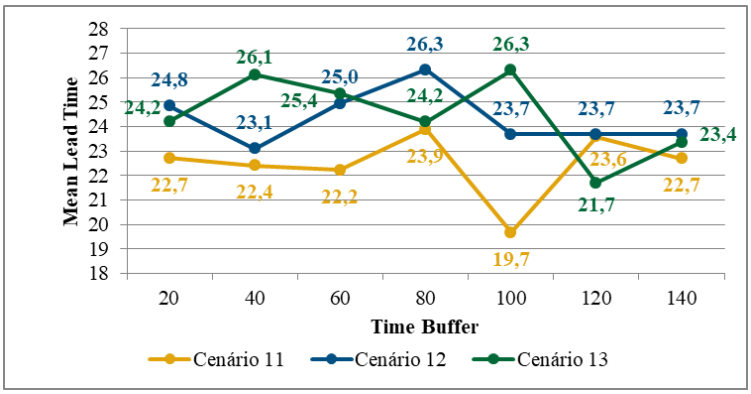
Os resultados das simulações que variam a posição do estágio gargalo, ou seja, da simulação dos cenários 11 (padrão), 12 e 13 definidos no capítulo 5, podem ser vistos nos Gráficos 16 a 20.

Gráfico 16 - Percentual de *jobs* terminados para cenários 11, 12 e 13



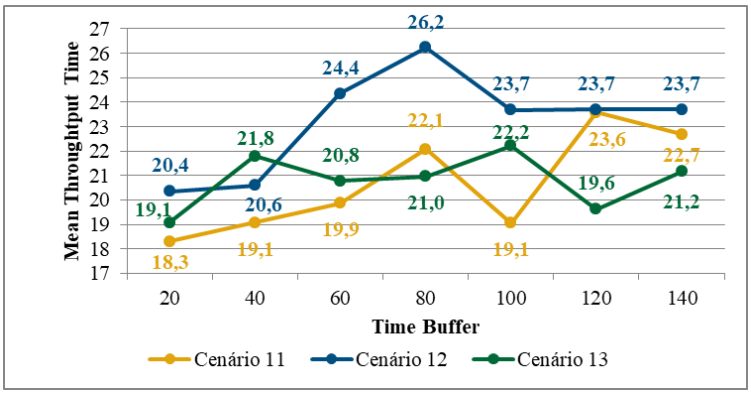
Fonte: Elaborado pelo autor

Gráfico 17 - *Mean lead time* para cenários 11, 12 e 13



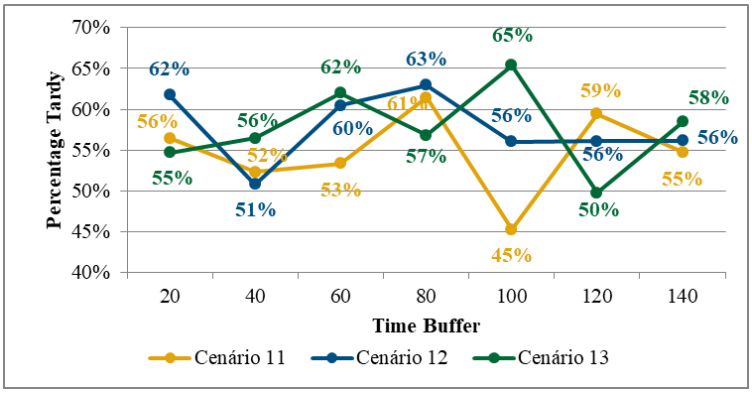
Fonte: Elaborado pelo autor

Gráfico 18 - *Mean throughtput time* para cenários 11, 12 e 13



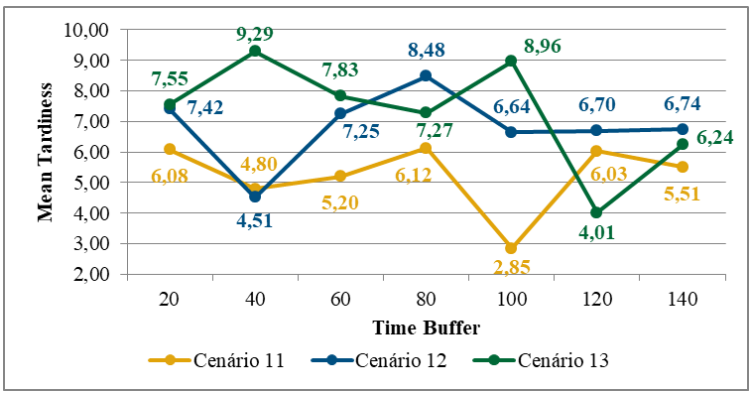
Fonte: Elaborado pelo autor

Gráfico 19 - *Percentage tardy* para cenários 11, 12 e 13



Fonte: Elaborado pelo autor

Gráfico 20 - *Mean tardiness* para cenários 11, 12 e 13



Fonte: Elaborado pelo autor

Primeiramente, em termos gerais, não há diferença significativa em nenhum indicador de desempenho analisado dentre os cenários aqui testados. Considerando todos os testes realizados para os cenários 11 (padrão), 12 e 13, embora o cenário 11 (padrão) apresente resultados melhores, as médias dos indicadores de desempenho obtidas estão muito próximas, não se percebendo nenhuma tendência aparente nos gráficos plotados.

Desta maneira, não se pode afirmar que a posição do estágio gargalo dentro de um sistema de produção é relevante para a produtividade do sistema como um todo, caso os demais parâmetros sejam mantidos. As médias dos indicadores de desempenho analisados para tais cenários podem ser vistas na Tabela 10.

Tabela 10 - Média dos indicadores de desempenho obtidos para cenários 11, 12 e 13

Cenário	% jobs terminados	Mean lead time	Mean throughtput time	Percentage tardy	Mean tardiness
Cenário 11	62%	22,4	20,7	54,7%	5,2
Cenário 12	59%	24,3	23,2	57,7%	6,8
Cenário 13	58%	24,5	20,8	57,6%	7,3

Fonte: Elaborado pelo autor

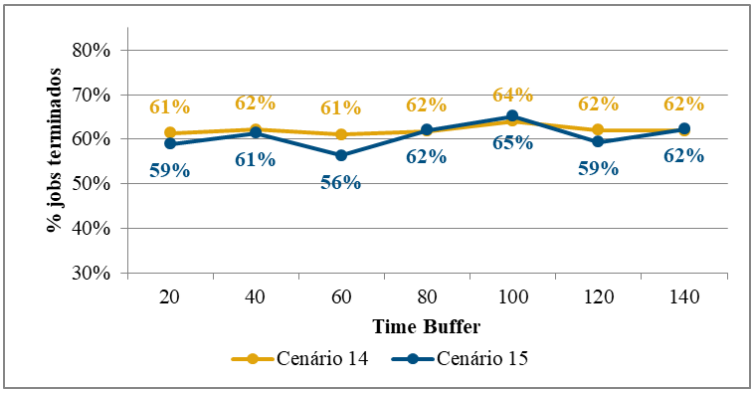
A única diferença perceptível entre os cenários testados está na influência do *time buffer* nos indicadores de desempenho analisados. Observando os Gráficos 19 e 20, é possível concluir que os três cenários apresentam um comportamento similar. Dentre os sete valores de *time buffer* testados, há variações, porém, há um único vale, cuja abcissa corresponde ao *time buffer* limite que minimiza o total de *jobs* atrasados e tempo médio de atraso.

Entre os cenários testados é perceptível a relação do *time buffer* correspondente ao vale com a posição do gargalo. Quanto mais para o fim do sistema simulado está o gargalo, maior o *time buffer* correspondente ao vale. A explicação para este evento é simples: para controlar um menor número de *jobs* entre a entrada do sistema e o estágio gargalo, é necessário um *time buffer* maior para estágios gargalos localizados mais ao final da linha. Caso tratássemos os 3 cenários analisados com o mesmo *time buffer* limite, aquele com estágio gargalo mais ao final (cenário 13) teria um *buffer* menor em números de *jobs*.

6.5. Análise da Variância no Tempo de Processamento

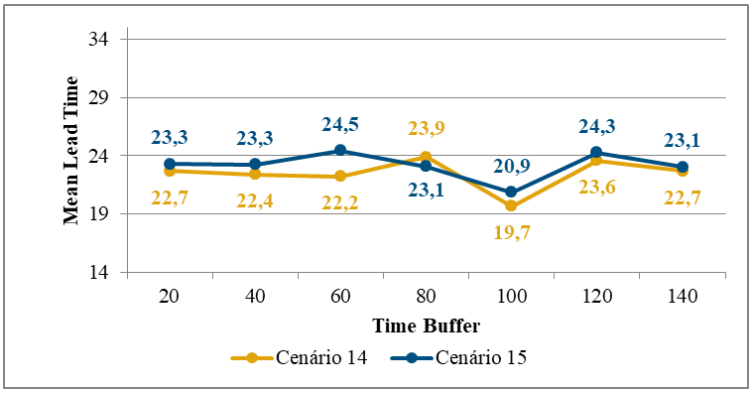
Os resultados das simulações com base na variação do desvio padrão dos tempos de processamento, correspondentes aos cenários 14 (padrão) e 15 definidos no capítulo 5, podem ser vistos nos Gráficos 21, 22, 23, 24 e 25.

Gráfico 21 - Percentual de *jobs* terminados para cenários 14 e 15



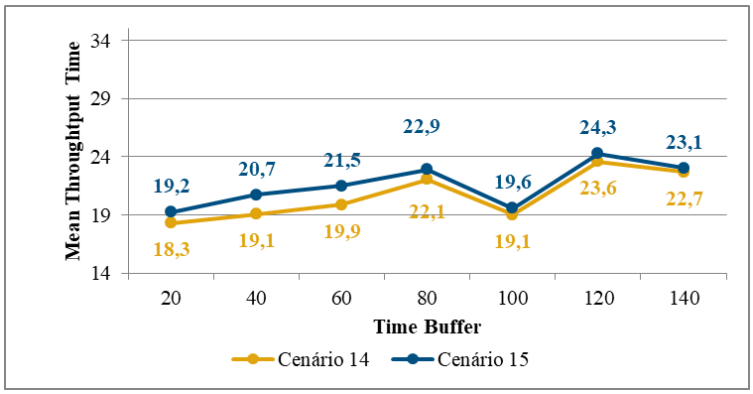
Fonte: Elaborado pelo autor

Gráfico 22 - *Mean lead time* para cenários 14 e 15



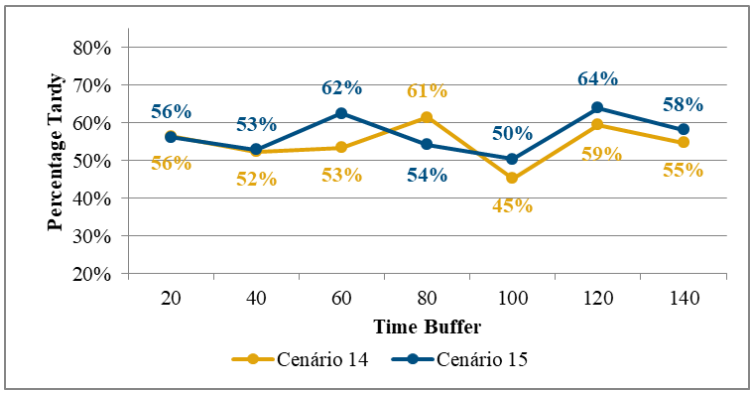
Fonte: Elaborado pelo autor

Gráfico 23 - *Mean throughtput* time para cenários 14 e 15



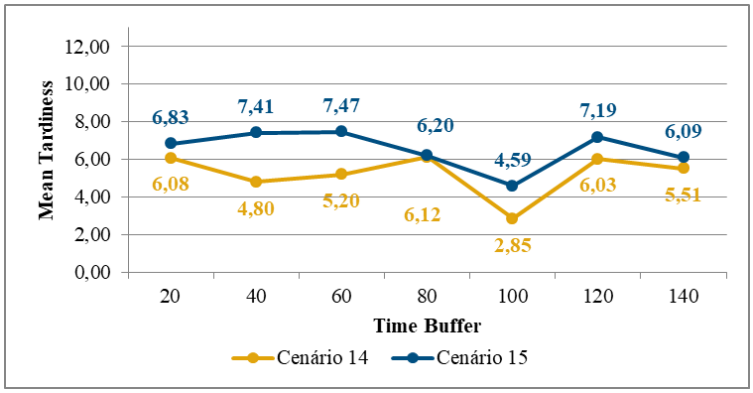
Fonte: Elaborado pelo autor

Gráfico 24 - *Percentage tardy* para cenários 14 e 15



Fonte: Elaborado pelo autor

Gráfico 25 - *Mean tardiness* para cenários 14 e 15



Fonte: Elaborado pelo autor

Observando os Gráficos de 21 a 25, nota-se que a variação do desvio padrão dos tempos de processamento em uma distribuição normal pouco influencia os resultados de indicadores de desempenho analisados. As simulações realizadas para o cenário 15, em que o desvio padrão é o dobro do cenário 14 (padrão), se mostraram ligeiramente inferiores, em termos de desempenho, comparativamente ao cenário 14 (padrão), em termos de percentual de *jobs* terminados, *mean lead time*, *mean throughtput time*, *percentage tardy* e *mean tardiness*.

Em termos de influência do *time buffer* nos indicadores de desempenho, poucos pontos de teste divergiram. De forma geral, as curvas para os quatro indicadores de desempenho analisados se mostraram aderentes entre os cenários 14 (padrão) e 15.

Não se observou nenhum grau maior de variação de nível de indicadores de desempenho analisados entre testes, mostrando que a maior variabilidade de tempos de simulação só é observável em eventos individuais da simulação. Observou-se a tendência do elevado número de *jobs* processados remover tal grau de variabilidade.

7. CONCLUSÕES

7.1. Síntese

O primeiro objetivo específico deste trabalho foi o desenvolvimento de um modelo de simulação que conseguisse replicar a lógica do método de controle DBR em um sistema *flow shop* genérico com estratégia de produção MTO. O capítulo 4 descreve em detalhe o modelo elaborado, incluindo suas interfaces de entrada e saída

Além disto, o modelo desenvolvido permite testar diferentes configurações de sistemas de produção operando no modelo DBR,, aceitando número praticamente ilimitado de máquinas e estágios e diversos tipos de distribuições de intervalos entre chegadas e tempos de processamento e alguns dos mais utilizados indicadores de desempenho.

O modelo de simulação mostrou-se compatível com seu desenho conceitual concebido, fornecendo resultados consistentes durante a etapa de verificação e validação retratada na seção 4.3. Por fim, forneceu uma gama de possibilidades de aplicação nos contextos de ensino, pesquisa e prática, de forma a constituir ferramenta para a análise de sistemas de produção que se enquadrem nas simplificações adotadas. Desta maneira, o presente trabalho cumpre com o primeiro objetivo específico proposto.

Os demais objetivos propostos deste trabalho consistem no uso do modelo de simulação para o estudo combinado de uma série de fatores e diferentes tamanho de *time buffers*, de forma a analisar o efeito do *time buffer* em diferentes indicadores de desempenho, considerando diferentes configurações do sistema (máquinas, posição do gargalo, regra etc).

Com base nos experimentos apresentados e analisados no capítulo 6 deste trabalho, uma série de conclusões podem ser obtidas. Primeiramente, nos experimentos sobre a relação entre tempos de geração e tempos de processamento e severidade do gargalo, foi possível notar o princípio central da teoria das restrições do trabalho de Goldratt & Cox (1986): a produtividade de um sistema de produção é ditada pelo ritmo de produção do estágio gargalo.

O fato do aumento da presença de *jobs* no sistema não ter causado aumento do número de *jobs* terminados, assim como do aumento da severidade ter afetado de forma diretamente proporcional ao número de *jobs* terminados, ilustra a teoria de Goldratt & Cox (1986). Isto, de certa forma, originou a criação do método de controle DBR.

Ainda no âmbito da relação tempos entre gerações e tempos de processamento, é possível concluir que esta relação impacta diretamente os indicadores de desempenho *mean lead time*, *mean throughtput time*, *percentage tardy* e *mean tardiness* em sistemas sobrecarregados, ou seja, com atraso de grande parte dos *jobs* gerados devido ao aumento do nível de utilização e estoque em processo.

No estudo do fator severidade do gargalo, duas observações podem ser feitas com base na análise de resultados. A primeira consiste no fato de que, quanto maior a severidade do gargalo, maior o *time buffer* responsável pelo melhor desempenho dentre as simulações feitas. Ou seja, quanto maior a severidade, maior a proteção requerida em números de *jobs*.

A segunda observação consiste no fato de que, quanto maior a severidade do gargalo, maior a performance do método de controle DBR, uma vez que *time buffers* elevados que causam a diminuição da efetividade do mecanismo de controle produzem mudanças abruptas nos dos indicadores de desempenho analisados, o que não é observado em cenários menos severos.

No estudo do faotr regras de sequenciamento, pode-se observar que a regra de sequenciamento EDD apresenta um ligeiro melhor desempenho do que a regra de sequenciamento FIFO sob a ótica dos indicadores de desempenho *percentage tardy* e *mean tardiness*.

Isto ocorre, pois, esta regra prioriza o processamento de *jobs* que possuem prazo de entrega mais apertado. É provável que a diferença de desempenho seria mais significante caso a geração de *due dates* tivesse maior variância e/ou menor correlação com a data de geração. Pelo cenário padrão estudado (cada *job* deve passar por todos os estágios) e o fato do tempo de processamento ser intrínseco aos estágios, não houve discriminação de tempos de processamento entre *jobs*. Desta maneira, não se pode fazer nenhuma observação sobre o desempenho das regras SPT ou LS, pois elas se comportaram de forma similar às regras FIFO e EDD, respectivamente.

Em termos de posicionamento do estágio gargalo, não foi possível observar diferenças absolutas de desempenho (considerando a média das simulações). A única diferença notável foi o deslocamento das curvas dos indicadores de desempenho dos Gráficos 16 a 20. Ou seja, é essencial conhecer o estágio gargalo e a posição dentro do sistema de produção para a etapa de dimensionamento do time buffer e, consequentemente, adoção do DBR para maximizar o desempenho do sistema.

Sumarizando, o presente trabalho permitiu a criação de uma ferramenta adequada para o estudo do método DBR dentro de múltilos cenários que possam ser enquadrados no ambiente *flow shop* genérico, permitindo o estudo do dimensionamento do *buffer*. Adicionalmente, o presente trabalho preenche uma lacuna na lliteratura sobre o tema, uma vez que não foi identificado nenhum trabalho anterior que estudou o impacto conjunto do tamanho do *time buffer* limite no DBR com diversos parâmetros de entrada na performance do sistema de produção.

O que se concluiu é que o desempenho de um sistema de produção *flow shop* genérico está diretamente atrelado a um correto dimensionamento do *time buffer*. Para realizar este dimensionamento de forma correta, é necessário considerar diversas características do sistema de produção que impactam severamente no tamanho de *time buffer* ótimo. Desta maneira, considerar todas estas características pode ser um trabalho árduo. A simulação por eventos discretos se mostra uma ferramenta poderosa para conduzir este dimensionamento.

7.2. Limitações

O presente trabalho possui uma série de limitações. Primeiramente,o modelo de simulação desenvolvido somente permite a análise em ambientes *flow shop*, não podendoser aplicado para ambiente jobshop ou assembly shop, por exemplo.

Além disse, toda a análse de resultados foi feita sobre os indicadores de desempenho *mean throughtput time*, *mean lead time*, *percentage tardy* e *mean tardiness*. Desta maneira,indicadores de desemepnho relacionados a aspectos como nível de estoque em processo e nível de utilização não foram utilizados e analisados. Desta maneira, além de não podermos influir nada sobre estes indicadores nos cenários simulados, o modelo de simulação se mostra mais adequado a estratégias de produção MTO.

Sob o ponto de vista dos resultados retirados dos experimentos analisados, a primeira limitação consiste no fato dos cenários gerados não nos permitirem uma análise comparativa das regras de sequenciamento SPT e LS.

Além disto, o estudo foi focado na geração de cenários em que apenas uma variável é alterada. Ou seja, a ausência de uma análise correlativa entre variáveis limita a geração de literatura uma vez que não identifica correlações.

7.3. Desdobramentos

O presente trabalho estudou o impacto de diversos parâmetros de entrada no *time buffer* ótimo do método de controle DBR e o impacto de *time buffers* diferentes para diversos cenários avaliados. O trabalho visou preencher uma lacuna existente atualmente sobre o estudo do dimensionamento do *time buffer* no método de controle DBR.

Durante a execução do presente trabalho, uma ferramenta muito poderosa e com diversas possibilidades de aplicação foi elaborada: o modelo de simulação. Como recomendações para trabalhos futuros, relacionados ao estudo do *time buffer* no DBR ou ao modelo de simulação, é importante destacar alguns aspectos.

Em relação ao estudo do *time buffer* no método de controle DBR, o presente trabalho abordou o estudo deste tema em sistemas *flow shop* genérico com estratégia de produção MTO. Logo, ainda há muito que ser estudado sob a ótica de outras estratégias de produção e ambientes de simulação.

O estudo deste tema em ambientes de produção *job shop* flexível e *assembly* shop éum potencial desdobramento destetrabalho, sendo que no último cabe o estudo não somente do *time buffer* do gargalo, como também do *time buffer* de montagem.

Outro potencial desdobramento é o estudo do *time buffer* em sistemas com estratégias de produção MTS. Nestes sistemas, a importância do ordenamento de *jobs* no método de controle DBR diminui e o foco passa a ser evitar a ociosidade no gargalo. Neste caso, recomenda-se o estudo do *time buffer* relacionado à restrição de mercado, formado para evitar perdas com demanda perdida.

O modelo de simulação possui um leque de aplicações muito grande, em termos científicos, práticos e acadêmicos. Sob a ótica da produção científica, constitui uma oportunidade para aplicação do método DBR sob ambientes de produção *flow shop* genérico.

O presente trabalho focalizou o estudo do comportamento dos indicadores de desempenho escolhidos com base em cenários formados por variedades de fatores escolhidos, como severidade do gargalo e posição do gargalo, em combinação com cenários de *time buffer* limite.

Porém, o trabalho não exauriu o universo de testes e estudos que podem ser realizados com base no modelo elaborado, como o teste de cenários com maior número de máquinas por estágio, números diferentes de estágios ou com possibilidades diferentes de 100% de certo *job* passar ou não em cada estágio. Além disso, testar cenários combinados, variando mais de uma grandeza, é uma possibilidade de estudo utilizando o modelo de simulação. Esta possibilidade não fez parte do escopo deste trabalho. Todas estas oportunidades citadas podem ser então alvos de estudos futuros.

Por fim, o modelo de simulação pode ser modificado para tornar-se ainda mais genérico de forma a comportar outros ambientes de produção e métodos de controle de produção. Em outras palavras, pode se tornar uma ferramenta ainda mais poderosa.

Finalmente, permitir a leitura de uma carteira de pedidos também é um potencial desdobramento de melhoria do modelo elaborado. Isto permitiria uma outra vertente de estudo, menos focada na parte de controle da produção e mais focada na programação da produção, ou até mesmo combinando estas areas.

REFERÊNCIA BIBLIOGRÁFICA

ARNOLD, JR Tony; RIMOLI, Celso; ESTEVES, Lenita R. **Administração de materiais: uma indrodução**. Atlas, 1999.

ATWATER, J. Brian; CHAKRAVORTY, Satya S. A Study of the Utilization of Capacity Constrained Resources in Drum-Buffer-Rope Systems. **Production and Operations Management**, v. 11, n. 2, p. 259-273, 2002.

DA SILVA, Edna et al. Avaliação de regras de sequenciamento da produção em ambientes Job shop e Flow shop por meio de simulação computacional. **Exacta**, v. 10, n. 1, 2012.

CHAN, Felix TS; CHAN, H. K. Analysis of dynamic control strategies of an FMS under different scenarios. **Robotics and Computer-Integrated Manufacturing**, v. 20, n. 5, p. 423-437, 2004.

CORBETT, Thomas; CSILLAG, Joao Mario. Analysis of the effects of seven drum-buffer-rope implementations. **Production and Inventory Management Journal**, v. 42, n. 3/4, p. 17, 2001.

CORRÊA, H. L.; GIANESI, IGN. **Just-in-Time, MRPII e OPT: um enfoque estratégico.** São Paulo: Atlas, 1993.

DANIEL, V.; GUIDE, R. Scheduling with priority dispatching rules and drum-buffer-rope in a recoverable manufacturing system. **International Journal of Production Economics**, v. 53, n. 1, p. 101-116, 1997.

DARLINGTON, John et al. Design and implementation of a Drum-Buffer-Rope pull-system. **Production Planning & Control**, v. 26, n. 6, p. 489-504, 2015.

DE SOUZA, FERNANDO BERNARDI. Do OPT à Teoria das Restrições: avanços e mitos. **Production**, v. 15, n. 2, p. 184-197, 2005.

ENNS, S. T. An integrated system for controlling shop loading and work flow. **International Journal of Production Research**, v. 33, n. 10, p. 2801-2820, 1995.

FERNANDES, Flávio Cesar Faria; GODINHO FILHO, Moacir. Sistemas de coordenação de ordens: revisão, classificação, funcionamento e aplicabilidade. **Revista Gestão & Produção, São Carlos**, v. 14, n. 2, 2007.

GAITHER, Norman; FRAZIER, Greg. **Administração da produção e operações**. Pioneira Thomson Learning, 2001.

GARDINER, S.C., BLACKSTONE, J.H., and GARDINER, L.R. The evolution of the theory of constraints. **Industrial Management**, 1994.

GOLDRATT, E.M.; COX, J. **A meta: um processo de aprimoramento contínuo.** São Paulo: IMAM, 1986.

GOLMOHAMMADI, Davood. A study of scheduling under the theory of constraints. **International Journal of Production Economics**, v. 165, p. 38-50, 2015.

GUPTA, Mahesh; SNYDER, Doug. Comparing ToC with MRP and JIT: a literature review. **International Journal of Production Research**, v. 47, n. 13, p. 3705-3739, 2009.

JAHANGIRIAN, Mohsen et al. Simulation in manufacturing and business: A review. **European Journal of Operational Research**, v. 203, n. 1, p. 1-13, 2010.

KRAJEWSKI, Lee J.; RITZMAN, Larry P. Process Management. **Operations Management—Strategy and Analysis,** Addison—Wesley Publishing Company, p. 93-138, 1996.

LAMBRECHT, Marc R.; DECALUWE, Lieve. Jit And Constraint Theory: The Issue Of Bottleneck Manageme. **Production and Inventory Management Journal**, v. 29, n. 3, p. 61, 1988.

LEE, Jun-Huei et al. Research on enhancement of ToC Simplified Drum-Buffer-Rope system using novel generic procedures. **Expert Systems with Applications**, v. 37, n. 5, p. 3747-3754, 2010.

MANIKAS, Andrew; GUPTA, Mahesh; BOYD, Lynn. Experiential exercises with four production planning and control systems. **International Journal of Production Research**, v. 53, n. 14, p. 4206-4217, 2015.

MESQUITA, M. et al. Programação detalhada da produção. In: LUSTOSA, L. J. et al. **Planejamento e Controle da Produção**. Rio de Janeiro: Elsevier, 2008.

NEGAHBAN, Ashkan; SMITH, Jeffrey S. Simulation for manufacturing system design and operation: Literature review and analysis. **Journal of Manufacturing Systems**, v. 33, n. 2, p. 241-261, 2014.

PINEDO, Michael. **Scheduling: Theory, algorithms, and systems**. New Jersey: Prentice Hall, 2002.

PIRES, Silvio RI. **Gestão da cadeia de suprimentos: conceitos, estratégias, práticas e casos**. São Paulo: Atlas, 2004.

PRICE, Wilson; GRAVEL, Marc; NSAKANDA, Aaron Luntala. A review of optimisation models of Kanban-based production systems. **European Journal of Operational Research**, v. 75, n. 1, p. 1-12, 1994.

RADOVILSKY, Zinovy D. A quantitative approach to estimate the size of the time buffer in the theory of constraints. **International Journal of Production Economics**, v. 55, n. 2, p. 113-119, 1998.

S RUSSELL, Roberta; W TAYLOR, Bernard. **Operations Management Creating Value Along the Supply Chain**. 2011.

SCHRAGENHEIM, Eli; RONEN, Boaz. Drum-buffer-rope shop floor control. **Production and Inventory Management Journal**, v. 31, n. 3, p. 18-22, 1990.

SHAFER, Scott M.; SMUNT, Timothy L. Empirical simulation studies in operations management: context, trends, and research opportunities. **Journal of Operations Management**, v. 22, n. 4, p. 345-354, 2004.

SIRIKRAI, Vimalin; YENRADEE, Pisal. Modified drum–buffer–rope scheduling mechanism for a non-identical parallel machine flow shop with processing-time variation. **International Journal of Production Research**, v. 44, n. 17, p. 3509-3531, 2006.

SPEARMAN, Mark L.; WOODRUFF, David L.; HOPP, Wallace J. CONWIP: a pull alternative to kanban. **The International Journal of Production Research**, v. 28, n. 5, p. 879-894, 1990.

STEELE, Daniel C. et al. Comparisons between drum–buffer–rope and material requirements planning: a case study. **International Journal of Production Research**, v. 43, n. 15, p. 3181-3208, 2005.

SURESH, K.N.; SRIDHARAN, R. Simulation modeling and analysis of tool sharing and part Scheduling decisions in single-stage multimachine flexible manufacturing systems. **Robotics and Computer-Integrated Manufacturing**, n. 23, p. 361-370, 2007.

THÜRER, Matthias et al. Drum-buffer-rope and workload control in High-variety flow and job shops with bottlenecks: An assessment by simulation. **International Journal of Production Economics**, v. 188, p. 116-127, 2017.

TUBINO, D. F. **Planejamento e controle da produção: teoria e prática**. São Paulo: Atlas, 2007.

UMBLE, E. J.; UMBLE, M. Integrating the Theory of Constraints into Supply Chain Management. **Proceedings of the 33rd annual Decision Sciences Conference**, San Diego, CA, p. 479-484, 2002.

WU, Shih-Yun; MORRIS, John S.; GORDON, Thomas M. A simulation analysis of the effectiveness of drum-buffer-rope scheduling in furniture manufacturing. **Computers & Industrial Engineering**, v. 26, n. 4, p. 757-764, 1994.

WU, H.-H.; YEH, M.-L. A DBR scheduling method for manufacturing environments with bottleneck re-entrant flows. **International journal of production research**, v. 44, n. 5, p. 883-902, 2006.

WU, Horng-Huei et al. Simulation and scheduling implementation study of TFT-LCD Cell plants using Drum–Buffer–Rope system. **Expert Systems with Applications**, v. 37, n. 12, p. 8127-8133, 2010.

YE, T.; HAN, W. Determination of buffer sizes for drum–buffer–rope (DBR)-controlled production systems. **International Journal of Production Research**, v. 46, n. 10, p. 2827-2844, 2008.

ZHANG, X. M.; DU, Y. L. Research of Production Scheduling Based on Theory of Constraints. In: **2015 International Conference on Electrical, Automation and Mechanical Engineering**. Atlantis Press, 2015.

ANEXO A – MODELO DE SIMULAÇÃO

```
""" x-----------Bibliotecas-------------------------------------------------------"""

""" Importa Bibliotecas do Python e se conecta a planilha de simulação,que fornecerá os inp
uts para a simulação deste programa bem como receberá a simulação como output para análise
da simulação. Adicionalmente, cria variável global JobAcumulados, que serve de suporte para
 numeração dos jobs gerados de forma a manter sequência de numeração mesmmo após simulações
 diferentes"""

import random

import simpy

import xlwings as xw

wb = xw.Book('Modelo_DBR - Input&Output.xlsm')

shtInput = wb.sheets['Input']

file = open ("OutputSimulacao.txt","w")

  global JobAcumulado

JobAcumulado = 0

  """ -----------Inputs-------------------------------------------------------"""

  def Inputs ():

      """ Lê da planilha excel os inputs a serem utilizados para fins do modelo de simulaçã
o. Estes inputs são todos os dados usados para fins de simulação. Estes estão destacados ab
aixo """

    NumSimulacoes = int (shtInput.range('C7').value)  # Número de Simulações diferentes rod
adas com o mesmo cenário

    TempoSimulacao = float(shtInput.range('C9').value) # Tempo limite para cada simulação


    TimeBuffer = int (shtInput.range('C11').value) # Tamanho do Time Buffer utilizado para
implementação do DBR

    Sequenciamento = str (shtInput.range('C13').value) # Regra de Sequenciamento escolhida,
 podendo ser FIFO, EDD, SPT ou LS

    # Parâmetros para geração de Jobs no sistema. Podem ou não ser úteis durante a simulaçã
o, dependendo da distribuição escolhida

    DistrGeracao = int (shtInput.range('C17').value)  #Distribuição escolhida, podendo ser
exponencial(1), normal(2), uniforme(3), ou fixa(4)

    TMGeracao = float (shtInput.range('C24').value)  # Tempo médio entre gerações. Aplicáve
l para distr. exponencial, normal e fixa (neste caso é o tempo entre gerações)
```

```
    DVGeracao = float (shtInput.range('C25').value)  # Desvio Padrão para distribuição norm
al


    TMINGeracao = float (shtInput.range('C26').value)  # Tempo mínimo para geração da distr
ibuição uniforme


    TMAXGeracao = float (shtInput.range('C27').value)  # Tempo máximo para geração da distr
ibuição uniforme


    # Parâmetros para geração de Due Date para Jobs gerados. Podem ou não ser úteis durante
 a simulação, dependendo da distribuição escolhida. Vale ressaltar que o número gerado por
estas variáveis é acrescido ao tempo atual da simulação na geração


    DistrDueDate = int (shtInput.range('C32').value)  #Distribuição escolhida, podendo ser
exponencial(1), normal(2), uniforme(3), ou fixa(4)


    TMDueDate = float (shtInput.range('C40').value) # Tempo médio. Aplicável para distr. ex
ponencial, normal e fixa (neste caso é o tempo entre gerações)


    DVDueDate = float (shtInput.range('C41').value)  # Desvio Padrão para distribuição norm
al


    TMINDueDate = float (shtInput.range('C42').value)  # Tempo mínimo para distribuição uni
forme


    TMAXDueDate = float (shtInput.range('C43').value)  # Tempo máximo para distribuição uni
forme


    # Parâmetros do Ambiente - Estágios e Máquinas


    NumEstagios = int (shtInput.range('C55').value) # Lê o valor do número de estágios da p
lanilha excel


    ListaEstagios = []


    for i in range (NumEstagios):  # Loop para geração de lista de estágios com cada estági
o sendo um item da lista. Em cada item da lista correspondente a um estágio, temos:


        i = i + 1


        Line = str (58 + i)


        Estg = []


        Estg.append (shtInput.range('C'+Line).value)  # Número de Máquinas


        Estg.append (shtInput.range('D'+Line).value)  # Distribuição escolhida para geração
 de tempos de processamento, podendo ser: exponencial(1), normal(2), uniforme(3), fixa(4),
triangular(5), erlang(5)


        Estg.append (shtInput.range('E'+Line).value)  # Tempo médio para fins de distribuiç
ão


        Estg.append (shtInput.range('F'+Line).value)  # Desvio padrão para fins de distribu
ição
```

```
        Estg.append (shtInput.range('G'+Line).value)  # Tempo mínimo para fins de distribui
ção

        Estg.append (shtInput.range('H'+Line).value)  # Tempo máximo para fins de distribui
ção

        Estg.append (shtInput.range('I'+Line).value)  # Número de exponenciais (beta) para
distr. erlang

        Estg.append (shtInput.range('J'+Line).value)  # Probabilidade de cada job de passar
 pelo estágio i

        ListaEstagios.append (Estg)

    return (NumSimulacoes, TempoSimulacao, TimeBuffer, Sequenciamento,DistrGeracao,TMGeraca
o,DVGeracao,TMINGeracao,TMAXGeracao,DistrDueDate,TMDueDate,DVDueDate,TMINDueDate,TMAXDueDat
e,NumEstagios,ListaEstagios)  # Retorna valores para simulação

""" -----------Funções de Suporte-----------------------------------------------------"""

def NumAleatorio (Distr,TM,DV,TMIN,TMAX,ERLANG):

    """ Cria números aleatórios a partir da distribuição escolhida e parâmetros importados.
 Serve para gerar números aleatórios para determinar tempos de geração, tempos de processam
ento e due dates"""

    if Distr == 1:  # Distr. exponencial

        Taxa = 1/TM

        NumAleatorio = random.expovariate (1/Taxa)

    elif Distr == 2:  # Distr. normal

        NumAleatorio = random.normalvariate (TM,DV)

    elif Distr == 3:  # Distr. uniforme

        NumAleatorio = random.uniform (TMIN,TMAX)

    elif Distr == 4:  # Distr. fixa

        NumAleatorio = TM

    elif Distr == 5:  # Distr. triangular

        NumAleatorio = random.triangular (TMIN,TMAX, TM)

    elif Distr == 6:  # Distr. erlang

        NumAleatorio = random.gammavariate (ERLANG, TM/ERLANG)

    return (NumAleatorio) # Retorna número aleatório de acordo com a distribuição escolhida
```

```
def TempoMed (Distr,TM,DV,TMIN,TMAX,ERLANG):

    """ Calcula o tempo médio esperado para a geração de número aleatórios de acordo com a
distribuição escolhida """

    if Distr == 1:  # Distr. exponencial

        TempoMed = TM

    elif Distr == 2:  # Distr. normal

        TempoMed = TM

    elif Distr == 3:  # Distr. uniforme

        TempoMed = (TMIN + TMAX)/2

    elif Distr == 4:  # Distr. fixa

        TempoMed = TM

    elif Distr == 5:  # Distr. triangular

        TempoMed = (TM + TMIN + TMAX)/3

    elif Distr == 6:  # Distr. erlang

        TempoMed = TM

    return (TempoMed)  # Retorna tempo médio de acordo com a distribuição escolhida

""" -----------Classe de Máquinas-----------------------------------------"""

class Maquinas(object):

    def __init__(self, env, cod, ListaEstagios):

        """Função que define classe máquinas. Quando chama-se a classe Máquinas, gera-
se um novo Estágio com as características de Tempo de Processamento, Probabilidade de estar
 no caminho e número de máquinas correspondente ao input fornecido na lista ListaEstagios""
"


        self.env = env  # Parte do Ambiente de Simulação do Simpy

        NumRecursos = int (ListaEstagios [cod-1] [0])

        self.maq = simpy.Resource(env, NumRecursos)  # Cada Estágio consiste em um Resource
 da biblioteca simpy com seu respectivo número de maquinas


        self.cod = cod

        self.tp = [ ListaEstagios [cod-1] [1], ListaEstagios [cod-
1] [2], ListaEstagios [cod-1] [3], ListaEstagios [cod-1] [4], ListaEstagios [cod-
1] [5], ListaEstagios [cod-1] [6]]
```

```
        # Cria lista de variáveis que serão posteriormente usadas para criar tempos aleatór
ios

""" -----------Entrada no Sistema-----------------------------------------"""

def GeracaoJobs (env,NumAtualSimulacao,DistrGeracao,TMGeracao,DVGeracao,TMINGeracao,TMAXGer
acao,DistrDueDate,TMDueDate,DVDueDate,TMINDueDate,TMAXDueDate,ListaEstagios,NumEstagios,Pos
Buffer):

    """ Função que simula a geração de jobs no sistema, gerando seu tempo de entrada na sim
ulação, seu due date e decidindo se o job entra no ambiente de simulação ou no pré-
shop segundo método DBR """

    Job=0

    global JobAcumulado  # Importa variável global para função para manter numeração do núm
ero de jobs

    Job = JobAcumulado

    while True: # Este loop faz com que a geração de jobs ocorra até o tempo de simulação d
eterminado na planilha

        TGeracao = NumAleatorio (DistrGeracao,TMGeracao,DVGeracao,TMINGeracao,TMAXGeracao,0
) # Tempo de geração é criado conforme distr. escolhida pela função NumAleatorio

        yield env.timeout(max (TGeracao,0)) # Adiciona o tempo gerado ao tempo atual da sim
ulação, rodando a simulação

        Job=Job+1  # Próximo Job será numerado com número que vem na sequencia, mesmo que s
imulação mude

        JobAcumulado = Job

        JobName = "Job%d" % Job  # Cria variável com nome do job, com base em sua numeração


        TDueDate = NumAleatorio (DistrDueDate,TMDueDate,DVDueDate,TMINDueDate,TMAXDueDate,0
)  # Gera número aleatório com base na distribuição escolhida para determinar due date do j
ob

        DueDate = env.now + TDueDate  # Due Date consiste na soma do número aleatório mais
o tempo de simulação na hora que o job é gerado

        file.write('%d\t%s\tGeração\tSistema\t%7.4f\t%7.4f\n' % (NumAtualSimulacao,JobName,
env.now,DueDate))  # Escreve em arquivo txt o evento Geração

        Caminho = []  # Lista Caminho guiará o job no sistema de
produção. Possuirá tamanho igual o número de estágios, com valores de o ou 1. 0, se não pas
sar pelo estágio correspondendte a ordem na lista e 1 se passar por este estágio

        NumEstagioJob = 0 # Variável que conterá o número de estágios em que o job gerado p
assará
```

```
        for i in range (NumEstagios):  # Loop para gerar as variáveis Caminho e NumEstagioJ
ob, conforme a probabilidade de cada máestágio de um job passar ou não por ela

            SimouNao= []  # Lista contendo 10000 elementos na proporção de 1/(0+1) equivale
nte a probabilidade de certo job passar pelo estágio i

            ProbCaminho = (ListaEstagios [i][7])*10000  #ProbCaminho é variável decisiva no
 cálculo

            ProbCaminho = int (ProbCaminho)

            for j in range (ProbCaminho):

                SimouNao.append (1)

            for j in range (10000 - ProbCaminho):

                SimouNao.append (0)

            k= random.choice (SimouNao)  # Escolha de número aleatório na lista SimouNão si
mula geração de número aleatório entre o e 1 de acordo com a variável ProbCaminho

            NumEstagioJob = NumEstagioJob + k

            Caminho.append (k)  # Lista caminho é composta pelo mesmo número de itens do nú
mero de estágios, com itens binários (0:job não é processado no estágio / 1:job é processad
o no estágio)

        RopeEntrada = Buffer (env, PosBuffer, TimeBuffer, NumEstagios,ListaEstagios)  # Var
iável que checa se sistema esta lotado (com base no time buffer)

        if RopeEntrada == True :  # Se sistema não tiver lotado, recebe True e começa proce
ssamento

            env.process(Processamento(env, NumAtualSimulacao,Job, JobName, Caminho, PosBuff
er,NumEstagios,ListaEstagios))  #Função que inicia processamento

            file.write('%d\t%s\tEntrada\tSistema\t%7.4f\t%7.4f\n' % (NumAtualSimulacao,JobN
ame,env.now,DueDate))  # Escreve em arquivo txt o evento de entrada do job no sistema

        else :  # Se sistema estiver lotado, recebe False e adiciona job ao pré-
shop, onde será devidamente ordenado e aguardará sua hora de entrar

            file.write('%d\t%s\tEntrada\tPréShop\t%7.4f\t%7.4f\n' % (NumAtualSimulacao,JobN
ame,env.now,DueDate))  # Escreve em arquivo txt a entrada do job no pré-shop

            yield PreShop.put ([Job,JobName,DueDate,NumEstagioJob,Caminho])  # Adiciona job
 na o pré-shop



""" -----------Funções de Processamento-------------------------------------"""

def Processamento (env, NumAtualSimulacao, Job, JobName, Caminho, PosBuffer,NumEstagios, Li
staEstagios):
```

```
    """ Função que simula o processamento de cada job dentro do flow shop flexível.Utiliza
o caminho de cada job e importa recursos do simpy (classse máquinas) para simular cada está
gio """

    for i in range (NumEstagios):  # Este Loop faz o job rodar por cada um dos estágios da
simulação

        NumOp = i+1  # Número do estágio, ou operação

        NomeOp = 'Operação'+ str(NumOp)  # Nome do estágio, ou operação

        if Caminho [i] == 1:  # Verifica se o estágio está no caminho do job, se sim, simul
a a operação ou adiciona-o a fila de espera. Se não, pula para o próximo estágio

            file.write('%d\t%s\tChegadaOp\t%s\t%7.4f\t-
\n' % (NumAtualSimulacao,JobName,NomeOp, env.now))  # Escreve em arquivo txt o evento de ch
egada do job na fila do estágio i

            atendReq1 = Estagios[i].maq.request()  # Gera um request e tempo até o job come
çar efetivamente o processamento

            yield atendReq1  # Adiciona o tempo de fila à simulação

            file.write('%d\t%s\tInícioOp\t%s\t%7.4f\t-
\n' % (NumAtualSimulacao,JobName,NomeOp, env.now))  # Escreve em arquivo txt o início do pr
ocessamento jo job i no estágio j

            DistrProcesso = int (Estagios[i].tp [0])

            TMProcesso = float (Estagios[i].tp [1])

            DVProcesso = float (Estagios[i].tp [2])

            TMINProcesso = float (Estagios[i].tp [3])

            TMAXProcesso = float (Estagios[i].tp [4])

            BetaErlang = int (Estagios[i].tp [5])

            TProcesso = max (NumAleatorio (DistrProcesso,TMProcesso,DVProcesso,TMINProcesso
,TMAXProcesso,BetaErlang),0)  # Com base na distribuição para tempo de simulação e parâmetr
os de cada estágio, gera-se o tempo de processamento

            yield env.timeout (TProcesso)  # Adiciona ao tempo de simulação o tempo de proc
essamento do job j na máquina i, calculado com base na distribuição escolhida

            Estagios[i].maq.release(atendReq1)  # Solta o job do estágio em que estava oper
ando, caracterizando fim de seu processamento neste estágio

            env.process(Rope (env,NumAtualSimulacao,NumEstagios,ListaEstagios,PosBuffer,Tim
eBuffer)) # Saída do job de um estágio ativa a função Rope do método DBR, que checa o buffe
r e compara com o time buffer máximo para liberar ou não novo job que está no pré-
shop no sistema
```

```
            file.write('%d\t%s\tFimOp\t%s\t%7.4f\t-
\n' % (NumAtualSimulacao,JobName,NomeOp, env.now))  # Escreve em arquivo txt fim do evento
de processammento do job i no estágio j

    file.write('%d\t%s\tSaídaSist\t%s\t%7.4f\t-
\n' % (NumAtualSimulacao,JobName,NomeOp, env.now))  # Escreve em arquivo txt fim do evento
de processammento do job i e sua saída do ambiente de simulação

""" -----------Drum-Buffer-Rope-----------------------------------------"""

def Drum(ListaEstagios):

    """ Função do método de controle DBR que exerce a 1ª etapa do método: Identifica o recu
rso gergalo do sistema. Por isto, está diretamente relacionado ao componente Drum """

    # Chama parâmetros do primeiro estágio do sistema de
produção para usar como ponto de partida para achar o gargalo. Estágio 1 parte como sendo o
 gargalo, no começo da função

    DistrGargalo = ListaEstagios [0][1]

    TMGargalo = ListaEstagios [0][2]

    DVGargalo = ListaEstagios [0][3]

    TMINGargalo = ListaEstagios [0][4]

    TMAXGargalo = ListaEstagios [0][5]

    ERLANGGargalo = ListaEstagios [0][6]

    MAQGargalo = ListaEstagios [0][0]

    PROBGargalo = ListaEstagios [0][7]

    TMaqGargalo = TempoMed (DistrGargalo,TMGargalo,DVGargalo,TMINGargalo,TMAXGargalo,ERLANG
Gargalo)  # Gera o tempo medio de operação de uma máquina dentro do estágio

    TMGargalo = TMaqGargalo * PROBGargalo / MAQGargalo  # Multiplica-
se o tempo médio estimado pela probabilidade de um job estar no caminho do estágio e divide
-
se este valor pelo número de máquinas no estágio. O estágio gargalo será aquele com o númer
o

    PosGargalo = 1

    for i in range (len(ListaEstagios)-
1):  # Loop que compara 1 a 1 os estágios até identificar aquele que e o gargalo

         # Chama parâmetros dos outros estágios do sistema de
produção para fazer comparação 1 a 1

        DistrComp = ListaEstagios [i+1][1]

        TMComp = ListaEstagios [i+1][2]
```

```
        DVComp = ListaEstagios [i+1][3]

        TMINComp = ListaEstagios [i+1][4]

        TMAXComp = ListaEstagios [i+1][5]

        ERLANGComp = ListaEstagios [i+1][6]

        MAQComp = ListaEstagios [i+1][0]

        PROBComp = ListaEstagios [i+1][7]

        TMaqComp = TempoMed (DistrComp,TMComp,DVComp,TMINComp,TMAXComp,ERLANGComp)

        TMComp = TMaqComp * PROBComp / MAQComp

        if TMComp > TMGargalo :  # Se estágio i é mais lento que o mais lento até o momento
, passa a ser momentaneamente o gargalo e, portanto, ponto de comparação com os próximos es
tágios

            TMGargalo = TMComp

            PosGargalo = i + 2

    return (PosGargalo)  # Esta função retorna onde está o recurso gargalo

def Buffer (env,PosBuffer,TimeBuffer, NumEstagios,ListaEstagios):

    """ Função do método de controle DBR que exerce a checagem do buffer atual do sistema e
 faz a comparação com o time buffer limite escolhido.

    Retorna True se pode-
se lançar um novo job no sistema e retorna False caso contrário """

    BufferAtual = 0

    ListaTempos = []  # Lista contendo todos as expectativas de tempos médios para cálculo
do buffer atual

    for i in range (PosBuffer):  # Loop para criar ListaTempos

        # Pega parâmetros que serão utilizados

        Distr = ListaEstagios [i][1]

        TM = ListaEstagios [i][2]

        DV = ListaEstagios [i][3]

        TMIN = ListaEstagios [i][4]

        TMAX = ListaEstagios [i][5]
```

```
        ERLANG = ListaEstagios [i][6]

        MAQ = ListaEstagios [i][0]

        PROB = ListaEstagios [i][7]

        TMaq = TempoMed (Distr,TM,DV,TMIN,TMAX,ERLANG)  # Gera tempo médio de processamento
 esperado

        TEstagio = TMaq * PROB / MAQ  # Tempo utilizado para fins de comparação será o temp
o esperado de processamento (tempo médio) x Probabilidade dos jobs passarem pelo estágio /
número de máquinas no estágio

        ListaTempos.append (TEstagio)

    for j in range (PosBuffer):  # Loop para calcular o buffer atual com o auxílio da Lista
Tempos

        TamFila = len (Estagios[i].maq.queue)  # Tamanho da fila do estágio i

        TempoUnitFila = 0  # Variável que irá calcular o tempo esperado para os jobs da fil
a i concluírem o processamento

        for k in range (PosBuffer - j):  # Calcula TempoUnitFila

            TempoUnitFila = TempoUnitFila + ListaTempos [PosBuffer-1-k]

        BufferAtual = BufferAtual + TamFila * TempoUnitFila  # Variável TempoUnitFila para
as filas de todos os estágios são somadas para achar o tamanho atual do buffer

    if TimeBuffer > BufferAtual:  # Teste comparativo entre time buffer limite e tamanho at
ual

        return True  # Libera job para o sistema

    else:

        return False  # Não libera job

def Rope(env,NumAtualSimulacao,NumEstagios,ListaEstagios,PosBuffer,TimeBuffer):

    """ Função do método de controle DBR que, caso a função buffer retorne True, exerce a e
ntrada de um novo job no sistema e sua remoção do pré-shop.

    Função Rope, sempre quando acionada, realiza o Sequenciamento do pré-shop """



    if len (PreShop.items) != 0:  # Se o pré-
shop não tiver nenhum job, não há o que fazer. Caso haja, realiza a checagem do buffer e se
quenciamento

        RopeDBR = Buffer (env, PosBuffer, TimeBuffer, NumEstagios,ListaEstagios)  # Checa s
e irá liberar job para o sistema ou não
```

```
        if RopeDBR == True:  # Libera job para o sistema

            RegrasPrioridade (env,Sequenciamento,ListaEstagios)  # Realiza o sequenciamento
 antes de liberar o primeiro job

            Job = PreShop.items[0][0]

            JobName = PreShop.items[0][1]

            DueDate = PreShop.items[0][2]

            NumEstagioJob = PreShop.items[0][3]

            Caminho = PreShop.items[0][4]

            yield PreShop.get()  # Remove o job mais prioritário do pré-shop

            env.process(Processamento(env, NumAtualSimulacao, Job, JobName, Caminho, PosBuf
fer,NumEstagios,ListaEstagios))  # Job mais prioritário entra no sistema de produção

            file.write('%d\t%s\tSaída\tPréShop\t%7.4f\t%7.4f\n' % (NumAtualSimulacao,JobNam
e,env.now,DueDate))  # Escreve em arquivo txt o evento de saída do job mais priritário do p
ré-shop

            file.write('%d\t%s\tEntrada\tSistema\t%7.4f\t%7.4f\n' % (NumAtualSimulacao,JobN
ame,env.now,DueDate))  # Escreve em arquivo txt o evento de entrada do job mais prioritário
 no sistema de produção

        else:  # Não libera job, mais realiza o sequenciamento

            RegrasPrioridade (env,Sequenciamento,ListaEstagios)

""" ----------Regras de Sequenciamento-----------------------------------"""

def RegrasPrioridade (env,Sequenciamento, ListaEstagios):

    """ Função que ativa o sequenciamento do pré-
shop de acordo com a regra de prioridade escolhida e importada da planilha excel """

    if Sequenciamento == 'EDD' :  # Mecanismo EDD (early due date) - ordena jobs no sistema
 pela menor due date até maior due date

       env.process (EDD (env))

    elif Sequenciamento == 'SPT':  # Mecanismo SPT (shortest processing time) - ordena jobs
 no sistema pela menor expectativa de Tempo de Processamento até a maior

        env.process (SPT (env,ListaEstagios))

    elif Sequenciamento == 'LS':  # Mecanismo SPT (shortest processing time) - ordena jobs
no sistema pela menor folga até a maior folga, sendo que folga corresponde a diferença entr
e a data de due date e o tempo estimado de conclusão

        env.process (LS (env,ListaEstagios))
```

```
    # Caso default consiste em sequenciamento FIFO (first in first out), em que o primeiro
job na fila é o que efetivamente sai

def EDD (env):

    """ Ordena pré-shop segundo regra de sequenciamento EDD """

    k = len(PreShop.items)

    Auxiliar = []

    for i in range (k) : # Cria PreShop auxiliar com mesma ordem de PreShop e Esvazia PreSh
op

        Job=PreShop.items[0][0]

        JobName=PreShop.items[0][1]

        DueDate=PreShop.items[0][2]

        NumEstagioJob=PreShop.items[0][3]

        Caminho=PreShop.items [0][4]

        Auxiliar.append ([Job,JobName,DueDate,NumEstagioJob,Caminho])

        yield PreShop.get()

        n = k

for i in range (k) : # Loop do novo ordenamento de jobs no sistema com sequenciamento EDD


        Job = Auxiliar[0][0]

        JobName = Auxiliar[0][1]

        DueDate = Auxiliar[0][2]

        NumEstagios = Auxiliar[0][3]

        Caminho = Auxiliar [0][4]

        Position = 0


        for j in range (n-
1) : # Loop comparativo - compara jobs no sistema com base no Due Date e seleciona o menor


            ParEDD =  Auxiliar[j+1][2]
```

```
            if ParEDD < DueDate:
                Job = Auxiliar[j+1][0]
                JobName = Auxiliar[j+1][1]
                DueDate = Auxiliar[j+1][2]
                NumEstagios = Auxiliar[j+1][3]
                Caminho = Auxiliar [j+1][4]
                Position = j+1
        n = n -1
        yield PreShop.put ([Job,JobName,DueDate,NumEstagios,Caminho]) # Adiciona novamente
na lista elemento de menor due date
        Auxiliar.pop (Position)# Elimina elemento da lista de suporte
def SPT (env,ListaEstagios):
    """ Ordena pré-shop segundo regra de sequenciamento SPT """
    k = len(PreShop.items)
    Auxiliar = []
    for i in range (k) : # Cria PreShop auxiliar com mesma ordem de PreShop e Esvazia PreSh
op
        Job = PreShop.items[0][0]
        JobName = PreShop.items[0][1]
        DueDate = PreShop.items[0][2]
        NumEstagioJob = PreShop.items[0][3]
        Caminho = PreShop.items [0][4]
        Auxiliar.append ([Job,JobName,DueDate,NumEstagioJob,Caminho])
        yield PreShop.get()
        q = k


    for i in range (k) : # Loop do novo ordenamento de jobs no sistema com sequenciamento S
PT
```

```
        Job = Auxiliar[0][0]

        JobName = Auxiliar[0][1]

        DueDate = Auxiliar[0][2]

        NumEstagioJob = Auxiliar[0][3]

        Caminho = Auxiliar [0][4]

        Position = 0

        SPT = 0

        for m in range (len(Caminho)):

            AdSPT = Caminho [m]* TempoMed (ListaEstagios [m][1],ListaEstagios[m][2],ListaEs
tagios[m][3],ListaEstagios[m][4],ListaEstagios[m][5],ListaEstagios[m][6])

            SPT = SPT + AdSPT

        for j in range (q-
1) : # Loop comparativo - compara jobs no sistema com base no Due Date e seleciona o menor


            ParCaminho = Auxiliar [j+1][4]

            ParSPT = 0

            for n in range (len(ParCaminho)):

                AdParSPT = ParCaminho[n] * TempoMed (ListaEstagios [m][1],ListaEstagios[m]
[2],ListaEstagios[m][3],ListaEstagios[m][4],ListaEstagios[m][5],ListaEstagios[m][6])


                ParSPT = ParSPT +AdParSPT

            if ParSPT < SPT:

                Job = Auxiliar[j+1][0]

                JobName = Auxiliar[j+1][1]

                DueDate = Auxiliar[j+1][2]

                NumEstagioJob = Auxiliar[j+1][3]

                Caminho = Auxiliar [j+1][4]

                Position = j+1

                SPT = ParSPT
```

```
            q = q - 1

        yield PreShop.put ([Job,JobName,DueDate,NumEstagioJob,Caminho]) # Adiciona novament
e na lista elemento de menor tempo de processamento

        Auxiliar.pop (Position)# Elimina elemento da lista de suporte

def LS (env,ListaEstagios):

    """ Ordena pré-shop segundo regra de sequenciamento LS """

    k = len(PreShop.items)

    Auxiliar = []

    for i in range (k) : # Cria PreShop auxiliar com mesma ordem de PreShop e Esvazia PreSh
op

        Job = PreShop.items[0][0]

        JobName = PreShop.items[0][1]

        DueDate = PreShop.items[0][2]

        NumEstagioJob = PreShop.items[0][3]

        Caminho = PreShop.items [0][4]

        Auxiliar.append ([Job,JobName,DueDate,NumEstagioJob,Caminho])

        yield PreShop.get()

    for i in range (k) : # Loop do novo ordenamento de jobs no sistema com sequenciamento S
PT

        Job = Auxiliar[0][0]

        JobName = Auxiliar[0][1]

        DueDate = Auxiliar[0][2]

        NumEstagioJob = Auxiliar[0][3]

        Caminho = Auxiliar [0][4]

        Position = 0

        SPT = 0

        q = k
```

```
        for m in range (len(Caminho)):

            AdSPT = Caminho [m]* TempoMed (ListaEstagios [m][1],ListaEstagios[m][2],ListaEs
tagios[m][3],ListaEstagios[m][4],ListaEstagios[m][5],ListaEstagios[m][6])

            SPT = SPT + AdSPT

        LS = DueDate - SPT

        for j in range (q-
1) : # Loop comparativo - compara jobs no sistema com base no Due Date e seleciona o menor


            ParCaminho = Auxiliar [j+1][4]

            ParSPT = 0

            for n in range (len(ParCaminho)):

                AdParSPT = ParCaminho[n] * TempoMed (ListaEstagios [m][1],ListaEstagios[m][
2],ListaEstagios[m][3],ListaEstagios[m][4],ListaEstagios[m][5],ListaEstagios[m][6])


                ParSPT = ParSPT +AdParSPT

            ParDueDate = Auxiliar[j+1][2]

            ParLS = ParDueDate - ParSPT

            if ParLS < LS:

                Job = Auxiliar[j+1][0]

                JobName = Auxiliar[j+1][1]

                DueDate = Auxiliar[j+1][2]

                NumEstagioJob = Auxiliar[j+1][3]

                Caminho = Auxiliar [j+1][4]

                Position = j+1

                SPT = ParSPT

                LS = ParLS

        q = q -1

        yield PreShop.put ([Job,JobName,DueDate,NumEstagioJob,Caminho]) # Adiciona novament
e na lista elemento de menor tempo de processamento

        Auxiliar.pop (Position)# Elimina elemento da lista de suporte
```

```
""" -----------Bloco Principal----------------------------------------------"""

""" Bloco Principal, que ativa a simulação chamando as funções deste programa"""

random.seed()  # Semente para gerar números aleatórios

NumSimulacoes, TempoSimulacao, TimeBuffer, Sequenciamento,DistrGeracao,TMGeracao,DVGeracao,
TMINGeracao,TMAXGeracao,DistrDueDate,TMDueDate,DVDueDate,TMINDueDate,TMAXDueDate,NumEstagio
s,ListaEstagios = Inputs ()  # Recebe os inputs do sistema

NumAtualSimulacao = 0

PosBuffer = Drum (ListaEstagios)  # Identifica a posição do recurso gargalo no sistema, que
 será um input da simulação

while NumAtualSimulacao < NumSimulacoes :  # Gera n simulações distintas com o mesmo cenári
o de inputs retirados do excel

    NumAtualSimulacao = NumAtualSimulacao + 1

    env = simpy.Environment()  # Importa ambiente de simulação do simpy

    PreShop = simpy.Store (env)  # Cria pr-e-shop como objeto de estoque do simpy

    Estagios =  []

    for i in range(NumEstagios):  # Loop que cria os estágios onde ocorre a simulação

        Estagios.append(Maquinas(env, i+1 , ListaEstagios))

    env.process(GeracaoJobs(env, NumAtualSimulacao,DistrGeracao,TMGeracao,DVGeracao,TMINGer
acao,TMAXGeracao,DistrDueDate,TMDueDate,DVDueDate,TMINDueDate,TMAXDueDate,ListaEstagios,Num
Estagios, PosBuffer))  # Ativa a funçao de Processamento

    env.run(until=TempoSimulacao) # Ativa a simulação

""" -----------Ler Macros do Arquivo Excel--------------------------------"""

""" Ativa macro da planilha excel, que importa o arquivo txt para dentro da aba Output da p
lanilha excel e ativa a análise, puxando as fórmulas necessárias para obter os Indicadores
de Desempenho na aba Indicadores de Desempenho """

ImportSimulacao = wb.macro('Automatico')

ImportSimulacao ()

file.close()
```

ANEXO B – PLANILHA DE ENTRADA (INPUT)

Modelagem DBR

Planilha de Input

Parâmetros de Simulação

Número de simulações	1
Tempo de simulação	100.0
Time buffer	3.0
Regra de sequenciaonamento de jobs para Pré-Shop	EDD

Parâmetros para Geração de Jobs

Tipo de distribuição para geração de jobs	**2**
Exponencial	1
Normal	2
Uniforme	3
Fixa	4

Parâmetros

Tempo Médio	2.0
Desvio Padrão	0.2
Tempo Mínimo	0.0
Tempo Máximo	0.0

Parâmetros para Determinação de Due Date

Tipo de distribuição para determinação de due date	**3**
Exponencial	1
Normal	2
Uniforme	3
Fixa	4

Parâmetros

Tempo Médio	0.0
Desvio Padrão	0.0
Tempo Mínimo	10.0
Tempo Máximo	30.0

Parâmetros de Processo

Tipo de distribuição para determinação de taxa de processamento	
Exponencial	1
Normal	2
Uniforme	3
Fixa	4
Triangular	5
Erlang	6

Número de Estágios	10

Estágios	Quant. Máquinas	Distribuição Processamento	Tempo Médio	Desvio Padrão	Tempo Mínimo	Tempo Máximo	Erlang Beta	Probabilidade de estágio estar no caminho do job
1	10	2	2.0	0.5	0.0	0.0	0.0	100%
2	2	2	3.0	0.5	0.0	0.0	0.0	100%
3	3	2	4.0	0.5	0.0	0.0	0.0	100%
4	4	2	2.0	0.5	0.0	0.0	0.0	100%
5	5	2	1.0	0.5	0.0	0.0	0.0	100%
6	6	2	5.0	0.5	0.0	0.0	0.0	100%
7	6	2	4.0	0.5	0.0	0.0	0.0	100%
8	6	2	3.0	0.5	0.0	0.0	0.0	80%
9	6	2	2.0	0.5	0.0	0.0	0.0	50%
10	6	2	1.0	0.5	0.0	0.0	0.0	100%

ANEXO C – PLANILHA DE SAÍDAS (OUTPUT)

Modelagem DBR

Output

Número da Simulação	Número do Job	Evento	Local	Tempo de Simulação	Due Date	Geração de jobs	Saída de jobs
1.0	Job1	Geração	Sistema	2.0	23.6	1.0	
1.0	Job1	Entrada	Sistema	2.0	23.6		
1.0	Job1	ChegadaOp	Operação1	2.0	-		
1.0	Job1	InicioOp	Operação1	2.0	-		
1.0	Job1	FimOp	Operação1	3.9	-		
1.0	Job1	ChegadaOp	Operação2	3.9	-		
1.0	Job1	InicioOp	Operação2	3.9	-		
1.0	Job2	Geração	Sistema	4.4	28.1	1.0	
1.0	Job2	Entrada	Sistema	4.4	28.1		
1.0	Job2	ChegadaOp	Operação1	4.4	-		
1.0	Job2	InicioOp	Operação1	4.4	-		
1.0	Job3	Geração	Sistema	6.2	27.2	1.0	
1.0	Job3	Entrada	Sistema	6.2	27.2		
1.0	Job3	ChegadaOp	Operação1	6.2	-		
1.0	Job3	InicioOp	Operação1	6.2	-		
1.0	Job2	FimOp	Operação1	6.9	-		
1.0	Job2	ChegadaOp	Operação2	6.9	-		
1.0	Job2	InicioOp	Operação2	6.9	-		
1.0	Job1	FimOp	Operação2	7.2	-		
1.0	Job1	ChegadaOp	Operação3	7.2	-		
1.0	Job1	InicioOp	Operação3	7.2	-		
1.0	Job4	Geração	Sistema	7.9	25.4	1.0	
1.0	Job4	Entrada	Sistema	7.9	25.4		
1.0	Job4	ChegadaOp	Operação1	7.9	-		
1.0	Job4	InicioOp	Operação1	7.9	-		
1.0	Job3	FimOp	Operação1	9.0	-		
1.0	Job3	ChegadaOp	Operação2	9.0	-		
1.0	Job3	InicioOp	Operação2	9.0	-		
1.0	Job4	FimOp	Operação1	9.3	-		
1.0	Job4	ChegadaOp	Operação2	9.3	-		
1.0	Job5	Geração	Sistema	9.9	25.9	1.0	
1.0	Job5	Entrada	PréShop	9.9	25.9		
1.0	Job2	FimOp	Operação2	10.1	-		
1.0	Job2	ChegadaOp	Operação3	10.1	-		
1.0	Job2	InicioOp	Operação3	10.1	-		
1.0	Job4	InicioOp	Operação2	10.1	-		
1.0	Job1	FimOp	Operação3	10.5	-		
1.0	Job1	ChegadaOp	Operação4	10.5	-		
1.0	Job1	InicioOp	Operação4	10.5	-		
1.0	Job6	Geração	Sistema	11.8	26.6	1.0	
1.0	Job6	Entrada	Sistema	11.8	26.6		
1.0	Job6	ChegadaOp	Operação1	11.8	-		
1.0	Job6	InicioOp	Operação1	11.8	-		
1.0	Job3	FimOp	Operação2	12.5	-		
1.0	Job3	ChegadaOp	Operação3	12.5	-		
1.0	Job3	InicioOp	Operação3	12.5	-		
1.0	Job4	FimOp	Operação2	13.2	-		
1.0	Job4	ChegadaOp	Operação3	13.2	-		
1.0	Job4	InicioOp	Operação3	13.2	-		
1.0	Job2	FimOp	Operação3	13.4	-		
1.0	Job2	ChegadaOp	Operação4	13.4	-		
1.0	Job2	InicioOp	Operação4	13.4	-		
1.0	Job1	FimOp	Operação4	13.5	-		
1.0	Job1	ChegadaOp	Operação5	13.5	-		
1.0	Job1	InicioOp	Operação5	13.5	-		
1.0	Job6	FimOp	Operação1	13.5	-		
1.0	Job6	ChegadaOp	Operação2	13.5	-		
1.0	Job6	InicioOp	Operação2	13.5	-		
1.0	Job1	FimOp	Operação5	13.7	-		
1.0	Job1	ChegadaOp	Operação6	13.7	-		
1.0	Job1	InicioOp	Operação6	13.7	-		
1.0	Job7	Geração	Sistema	13.8	30.4	1.0	
1.0	Job7	Entrada	Sistema	13.8	30.4		
1.0	Job7	ChegadaOp	Operação1	13.8	-		
1.0	Job7	InicioOp	Operação1	13.8	-		

Número do Job	Data de Geração	Data de Entrada	Data de Saída (0= Não saiu)	Due Date
Job1	2.0	2.0	25.3	23.6
Job2	4.4	4.4	25.5	28.1
Job3	6.2	6.2	33.8	27.2
Job4	7.9	7.9	30.6	25.4
Job5	9.9	0.0	0.0	25.9
Job6	11.8	11.8	37.4	26.6
Job7	13.8	13.8	33.1	30.4
Job8	15.3	15.3	42.8	32.6
Job9	17.4	17.4	41.0	43.1
Job10	19.6	19.6	41.0	41.5
Job11	21.7	21.7	44.6	49.4
Job12	23.9	23.9	47.9	43.0
Job13	26.4	26.4	50.9	41.1
Job14	28.4	28.4	55.3	43.3
Job15	30.7	30.7	57.7	42.5
Job16	32.5	32.5	58.9	52.6
Job17	34.5	34.5	62.1	57.7
Job18	36.5	36.5	64.8	48.3
Job19	38.4	38.4	62.3	52.8
Job20	40.2	40.2	67.3	64.3
Job21	42.3	42.3	68.6	54.4
Job22	44.4	44.4	71.2	65.6
Job23	46.7	46.7	76.3	64.7
Job24	48.6	48.6	77.5	66.6
Job25	50.5	50.5	81.0	80.1
Job26	52.2	52.2	76.0	79.2
Job27	53.9	53.9	78.9	69.4
Job28	55.9	0.0	0.0	73.7
Job29	58.1	58.1	79.9	71.6
Job30	59.6	59.6	79.8	77.1
Job31	61.5	61.5	86.4	82.4
Job32	63.7	63.7	0.0	90.7
Job33	65.9	65.9	90.6	86.3
Job34	68.0	68.0	0.0	95.0
Job35	70.0	0.0	0.0	94.2
Job36	72.1	72.1	0.0	88.1
Job37	74.0	74.0	0.0	88.5
Job38	75.9	75.9	0.0	86.9
Job39	78.1	78.1	0.0	95.6
Job40	80.2	80.2	0.0	97.7
Job41	82.1	82.1	0.0	99.0
Job42	84.6	84.6	0.0	95.4
Job43	86.7	86.7	0.0	108.5
Job44	88.9	88.9	0.0	105.0

Lead Time	Throughput Time	Concluído?	Atraso?	Tardiness
23.3	23.3	Sim	Sim	1.7
21.2	21.2	Sim		0.0
27.6	27.6	Sim	Sim	6.6
22.7	22.7	Sim	Sim	5.2
25.6	25.6	Sim	Sim	10.8
19.3	19.3	Sim	Sim	2.7
27.5	27.5	Sim	Sim	10.3
23.6	23.6	Sim		0.0
21.4	21.4	Sim		0.0
23.0	23.0	Sim		0.0
24.1	24.1	Sim	Sim	4.9
24.6	24.6	Sim	Sim	9.9
27.0	27.0	Sim	Sim	12.0
27.1	27.1	Sim	Sim	15.2
26.5	26.5	Sim	Sim	6.3
27.7	27.7	Sim	Sim	4.4
28.3	28.3	Sim	Sim	16.6
23.9	23.9	Sim	Sim	9.5
27.1	27.1	Sim	Sim	3.0
26.3	26.3	Sim	Sim	14.2
26.8	26.8	Sim	Sim	5.6
29.6	29.6	Sim	Sim	11.6
29.0	29.0	Sim	Sim	10.9
30.5	30.5	Sim	Sim	0.9
23.8	23.8	Sim		0.0
25.0	25.0	Sim	Sim	9.6
21.8	21.8	Sim	Sim	8.3
20.2	20.2	Sim	Sim	2.7
24.9	24.9	Sim	Sim	4.0
24.7	24.7	Sim	Sim	4.3

Modelagem DBR

Indicadores de Desempenho

Indicadores de Desempenho	
Jobs gerados	92
Jobs terminados	64
Mean Throughtput Time	27.0
Mean Lead Time	27.0
Percentage Tardy	39%
Mean Tardiness	0.6

ANEXO D – MACROS DE SUPORTE

```
Sub Automatico()

  ' Macro que une importar simulação de arquivo txt e executa a análise

    Sheets("Output").Select

    Range("B6").Select

    With ActiveSheet.QueryTables.Add(Connection:= _

        "TEXT; OutputSimulacao.txt" _

        , Destination:=Range("$B$6"))

        .Name = "OutputSimulacao"

        .FieldNames = True

        .RowNumbers = False

        .FillAdjacentFormulas = False

        .PreserveFormatting = True

        .RefreshOnFileOpen = False

        .RefreshStyle = xlInsertDeleteCells

        .SavePassword = False

        .SaveData = True

        .AdjustColumnWidth = True

        .RefreshPeriod = 0

        .TextFilePromptOnRefresh = False

        .TextFilePlatform = 1252

        .TextFileStartRow = 1

        .TextFileParseType = xlDelimited

        .TextFileTextQualifier = xlTextQualifierDoubleQuote

        .TextFileConsecutiveDelimiter = False

        .TextFileTabDelimiter = True
```

```
        .TextFileSemicolonDelimiter = False
        .TextFileCommaDelimiter = False
        .TextFileSpaceDelimiter = False
        .TextFileColumnDataTypes = Array(1, 1, 1, 1, 1, 1)
        .TextFileTrailingMinusNumbers = True
        .Refresh BackgroundQuery:=False
    End With
    Range("B5").Select
    Range(Selection, Selection.End(xlToRight)).Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.Columns.AutoFit
    Range("A1").Select
    Sheets("Indicadores de Desempenho").Select
    Range("A1").Select
    Dim NumJobs As Integer
    Dim i As Integer
    NumJobs = Sheets("Indicadores de Desempenho").Cells(7, 3).Value
    i = 0
    Sheets("Output").Select
    Do While i < NumJobs
        i = i + 1
        Cells(i + 5, 12) = "Job" & i
    Loop
    Cells(i + 5, 13) = "x"
    Cells(i + 5, 18) = "x"
    Range("M6:P6").Select
    Range(Selection, Selection.End(xlDown)).Select
```

```
    Selection.FillDown
    Range("R6:V6").Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.FillDown
    Sheets("Indicadores de Desempenho").Select
    Range("A1").Select
End Sub
Sub Limpa_Automatico()
'Macro que limpa análise da planilha e limpa simulação do excel e arquivo txt
    Sheets("Output").Select
    Range("L7").Select
    Range(Selection, Selection.End(xlDown)).Select
    Range(Selection, Selection.End(xlToRight)).Select
    Selection.ClearContents
    With Selection.Interior
        .Pattern = xlNone
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    Range("R7").Select
    Range(Selection, Selection.End(xlDown)).Select
    Range(Selection, Selection.End(xlToRight)).Select
    Selection.ClearContents
    With Selection.Interior
        .Pattern = xlNone
        .TintAndShade = 0
        .PatternTintAndShade = 0
```

```
    End With

    Sheets("Indicadores de Desempenho").Select

    Range("A1").Select

    Sheets("Output").Select

    Range("B6:G6").Select

    Range(Selection, Selection.End(xlDown)).Select

    Selection.ClearContents

    Sheets("Indicadores de Desempenho").Select

    Range("A1").Selec

    Open "OutputSimulacao.txt" For Output As #1: Close #1 '

End Sub
```