unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

LUIS ALBERTO OSÉS RODRIGUEZ

MÉTODOS DE SOLUÇÃO PARA UM PROBLEMA DE SEQUENCIAMENTO DA PRODUÇÃO COM SINCRONISMO DE EXECUÇÃO DE TAREFAS

Guaratinguetá
2013

LUIS ALBERTO OSÉS RODRIGUEZ

MÉTODOS DE SOLUÇÃO PARA UM PROBLEMA DE SEQUENCIAMENTO DA PRODUÇÃO COM SINCRONISMO DE EXECUÇÃO DE TAREFAS

Tese apresentada à Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, para a obtenção do título de Doutor em Engenharia Mecânica na área de Gestão e Otimização.

Orientador: Prof. Dr. Edson Luiz França Senne

Guaratinguetá
2013

Rodriguez, Luis Alberto Osés
R696m Métodos de solução para um problema de sequenciamento da produção com sincronismo de execução de tarefas / Luis Alberto Osés Rodriguez – Guaratinguetá : [s.n], 2013.
136 f : il.
Bibliografia: f. 127-136

Tese (doutorado) – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2013.
Orientador: Prof. Dr. Edson Luiz França Senne

1. Planejamento da produção 2. Programação linear 3. Programação heurística I. Título

CDU 658.5(043)

unesp

UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE GUARATINGUETÁ

LUÍS ALBERTO OSÉS RODRIGUEZ

ESTA TESE FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE
"DOUTOR EM ENGENHARIA MECÂNICA"

PROGRAMA: ENGENHARIA MECÂNICA
ÁREA: GESTÃO E OTIMIZAÇÃO

APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO

Prof. Dr. Edson Cocchieri Botelho
Coordenador

B A N C A E X A M I N A D O R A:

Prof. Dr. EDSON LUIZ FRANÇA SENNE
Orientador/UNESP

Prof. Dr. FERNANDO AUGUSTO SILVA MARINS
UNESP-FEG

Prof. Dr. JOSÉ ROBERTO DALE LUCHE
UNESP-FEG

Prof. Dr. HORACIO HIDEKI YANASSE
UNIFESP-São José dos Campos

Prof. Dr. ANTÔNIO AUGUSTO CHAVES
UNIFESP-São José dos Campos

Novembro de 2013

DADOS CURRICULARES

LUIS ALBERTO OSÉS RODRIGUEZ

NASCIMENTO	27.10.1985 – SÃO BERNARDO DO CAMPO – SP
FILIAÇÃO	Delmiro Rodriguez Álvarez Maria Del Pilar Osés Lassa
2003/2007	Curso de Graduação em Engenharia Mecânica, na Faculdade de Engenharia do Campus de Guaratinguetá da Universidade Estadual Paulista Júlio de Mesquita Filho.
2008/2009	Curso de Pós-Graduação em Engenharia Mecânica, nível de Mestrado, na Faculdade de Engenharia do Campus de Guaratinguetá da Universidade Estadual Paulista Júlio de Mesquita Filho.
2012/2013	Curso de Pós-Graduação em Engenharia Mecânica, nível de Doutorado, na Faculdade de Engenharia do Campus de Guaratinguetá da Universidade Estadual Paulista Júlio de Mesquita Filho.

RODRIGUEZ, L. A. O. Métodos de solução para um problema de sequenciamento da produção com sincronismo de execução de tarefas. 2013. 136 p. Tese (Doutorado em Engenharia Mecânica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2013.

RESUMO

Nesta tese é apresentado o problema de sequenciamento da produção em máquinas paralelas, sem interrupções, sem *buffers*, com tempos de processamento dependentes da sequência, restrições de capacidade, e sincronismo de execução das tarefas, com o objetivo de minimizar o *makespan*. Este problema, encontrado no mundo real em processos de fabricação de cilindros de laminação fundidos, possui a particularidade de que pares de tarefas devem ser concluídos ao mesmo tempo, o que torna a solução do problema ainda mais complexa. Inicialmente é proposta uma formulação de programação linear inteira para o problema. A seguir, são desenvolvidos vários métodos de solução que combinam as heurísticas *relax-and-fix*, *iterated local search*, *variable neighborhood search*, e *large neighborhood search*. Resultados computacionais obtidos a partir de problemas reais mostram que as soluções obtidas pelos métodos propostos superam aqueles obtidos por um *software* de programação inteira mista padrão executado durante uma hora e meia, sendo que no melhor deles, o *gap* entre a solução gerada e o melhor limitante inferior conhecido é, em média, de 6%.

PALAVRAS-CHAVE: Problemas de sequenciamento. *Relax-and-fix*. *Iterated local search*. *Variable neighborhood search*. *Large neighborhood search*.

RODRIGUEZ, L. A. O. Solution methods for a production scheduling problem with task execution synchronization. 2013. 136 p. Tese (Doutorado em Engenharia Mecânica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2013.

ABSTRACT

This paper presents the no-preemptive parallel scheduling problem without buffers, with sequence dependent set-up times, machine eligibility restrictions, and task execution synchronization, with the aim of minimizing the makespan. This problem, found in the real world in manufacturing processes of cast rolling mill rolls, has the particularity that pairs of tasks must be completed at the same time, which makes the problem solution more complex. Initially, a mixed-integer programming of the problem is proposed. Following that, several methods that combine heuristics relax-and-fix, iterated local search, variable neighborhood search, e large neighborhood search are developed. Computational results obtained from real problems show that the solutions obtained by the proposed methods outperform those returned by a standard MIP (Mixed Integer Programming) solver after one and a half hours. In the best method, the gap between the solution and the best lower bound known is on average 6%.

KEYWORDS: Scheduling problems. Relax-and-fix. Iterated local search. Variable neighborhood search. Large neighborhood search.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 NATUREZA DO PROBLEMA

O trabalho desta tese insere-se dentro da classe de problemas de sequenciamento da produção, e trata de uma variação do problema ainda não descrita na literatura científica.

Os problemas de sequenciamento da produção (*scheduling*) estão largamente difundidos em diversas atividades industriais e têm adquirido grande importância, uma vez que a utilização eficiente dos recursos disponíveis e o cumprimento dos prazos acordados com os clientes tornam-se fatores fundamentais para a sobrevivência das empresas no atual mundo globalizado. Sua ampla utilização em problemas práticos justifica o grande número de pesquisas já realizadas e mantém o interesse em desenvolver técnicas de resolução que consigam boas soluções em um tempo computacional adequado (GUIMARÃES, 2007).

De acordo com Pinedo (2008), os problemas de *scheduling* referem-se aos procedimentos de alocação, num determinado período de tempo, de recursos e equipamentos limitados para executar o processamento de tarefas. O *scheduling* possui três elementos fundamentais: a designação dos recursos, que envolve a seleção de um conjunto de recursos apropriados para uma atividade conhecida; o sequenciamento de atividades, que define a ordem de execução das atividades designadas para os recursos; e a determinação do tempo de utilização dos recursos pelas respectivas atividades, ou seja, a determinação dos tempos de início e término de cada atividade.

Apesar da grande variedade de métodos de resolução e do avanço computacional, dependendo de suas dimensões, os problemas de sequenciamento são considerados de difícil solução (NP – difíceis), em função da sua natureza combinatória. Além disso, a introdução de características extras ao problema, tais como a possibilidade de uma operação ser realizada em mais de um recurso, ou a necessidade de considerar tempos de operação dependentes da execução da tarefa anterior, tornam a resolução do problema ainda mais difícil.

Nesta tese é estudado o problema de sequenciamento da produção sem interrupções, sem *buffers*, com máquinas paralelas, tempos de processamento dependentes da sequência, restrições de capacidade, e com sincronismo de execução das tarefas.

No processo de fabricação de cilindros de laminação fundidos, cada peça tem sua produção iniciada em fornos de fusão por indução. Assim, torna-se necessário determinar em que equipamento cada cilindro será processado, e qual será a sequência de processamento das

peças em cada uma das máquinas. Por causa de sua natureza combinatória e das restrições envolvidas neste sistema, encontrar a solução ótima para este problema torna-se extremamente difícil a medida que suas dimensões crescem (GUIMARÃES, 2007). No sistema que será analisado, cada forno possui um tamanho diferente. Dessa forma, nem todos os cilindros podem ser produzidos em qualquer forno. Além disso, os equipamentos possuem taxas de fusão diferentes, o que significa que o tempo de processamento de cada peça depende do forno em que esta será produzida. Yu *et al*. (2002) afirmam que o sequenciamento de máquinas em que o tempo de execução da tarefa é dependente do equipamento torna a solução dos problemas ainda mais complexa.

Em função do princípio de funcionamento dos fornos e dos diferentes tipos de materiais que são produzidos, um cilindro pode aproveitar uma parcela do residual deixado num forno pelo cilindro produzido imediatamente anterior a ele. Com isso, o tempo de processamento de cada peça depende, também, da peça que foi feita anteriormente a ele. De acordo com Gupta e Smith (2006), isso complica consideravelmente o problema.

Finalmente, o problema que será estudado apresenta uma particularidade que o torna diferente dos demais problemas de sequenciamento já analisados. Há cilindros fundidos que são constituídos por dois materiais diferentes: um material resistente ao impacto para a parte interior do cilindro, e outro resistente ao desgaste para a parte exterior do cilindro. Esses materiais não podem ser processados num mesmo forno ao mesmo tempo, e, normalmente, possuem tempos de processamento diferentes. Por se tratar de metal líquido, não é possível produzir um deles primeiro, retirá-lo do forno e deixá-lo aguardando até que a outra parte seja concluída. As duas partes devem ficar prontas ao mesmo tempo. Se uma delas ficar pronta antes, ela deverá permanecer no forno em que estiver até que a outra parte fique pronta. Esta restrição torna o problema ainda mais complexo e adiciona a possibilidade de se chegar a soluções inviáveis.

1.2 OBJETIVOS

Este trabalho tem como objetivo resolver um problema de sequenciamento da produção de cilindros de laminação em fornos de indução para casos reais. Para isso, sugere-se:

(a) Elaborar um modelo matemático de otimização para o problema citado;

(b) Propor um ou mais métodos de solução para o mesmo;

(c) Aplicar os métodos propostos em problemas reais.

Dessa forma, o presente trabalho trará como contribuição principal o desenvolvimento de um modelo de otimização e de um método de solução para o problema de sequenciamento da produção sem interrupções, sem *buffers*, com máquinas paralelas, tempos de processamento dependentes da sequência, restrições de capacidade, e sincronismo de execução das tarefas.

Estes modelos poderão ser utilizados como ferramentas para auxiliar a programação da produção de empresas do setor siderúrgico.

1.3 METODOLOGIA

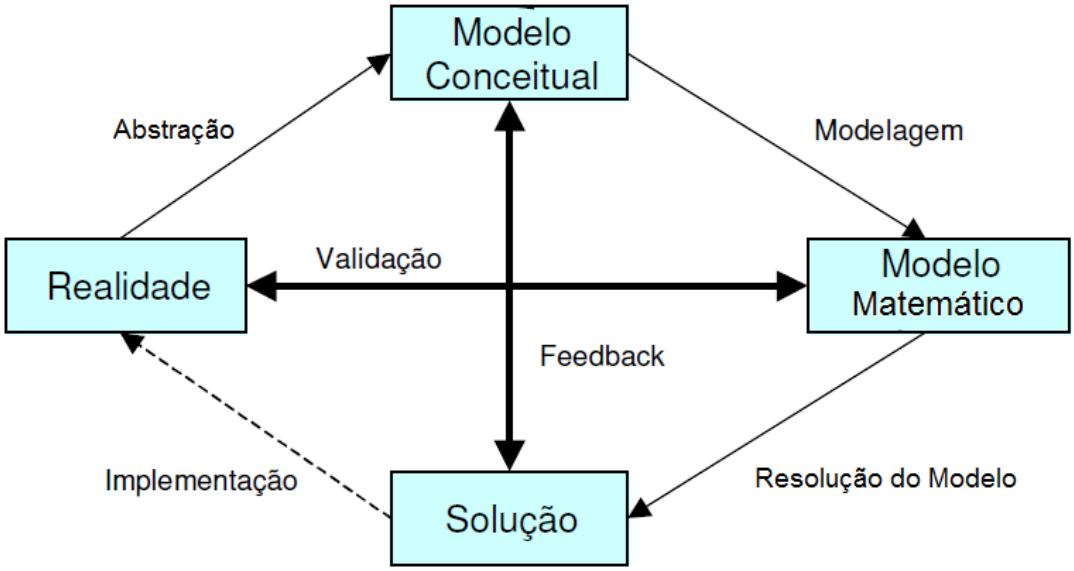
A elaboração de um projeto de pesquisa e o desenvolvimento da própria pesquisa seja ela uma dissertação ou uma tese, necessitam, para que seus resultados sejam satisfatórios, estar baseados num planejamento cuidadoso, em reflexões conceituais sólidas e alicerçados em conhecimentos já existentes (MIGUEL *et al.*, 2012).

Lakatos e Marconi (2005) definem método como um conjunto de atividades ordenadas e racionais que determina o trajeto a ser seguido para atingir os objetivos propostos, detectando erros e auxiliando o pesquisador a tomar decisões.

De acordo com Bertrand e Fransoo (2002), a Pesquisa Operacional utiliza como método um conjunto de etapas inter-relacionadas para atingir seus objetivos, como pode ser observado na Figura 1.

Na etapa de abstração, o problema real é transformado num modelo conceitual, no qual o pesquisador deve definir a abrangência do problema, as variáveis que devem ser incorporadas e as questões que devem ser respondidas. Na fase seguinte, o pesquisador deve elaborar um modelo quantitativo, considerando as relações entre as variáveis, de forma a preservar as características e o comportamento do sistema real. A seguir, tem início a etapa de resolução do modelo, em que deve ser escolhido um método já conhecido, ou ser desenvolvido um novo método de solução. Finalmente, os resultados do modelo devem ser implementados no sistema real.

Figura 1 – Metodologia para solução de problemas em Pesquisa Operacional



Fonte: (BERTRAND; FRANSOO, 2002)

Neste trabalho, após a apresentação detalhada do problema e de suas particularidades, o sistema é modelado por meio da Programação Linear Inteira Mista (ARENALES *et al.*, 2007). A seguir, métodos de solução heurística baseados na formulação matemática do problema são desenvolvidos e aplicados. Tais métodos são baseados nas heurísticas e metaheurísticas *relax-and-fix, iterated local search* (ILS), *variable neighborhood search* (VNS), e *large neighborhood search* (LNS).

1.4 ORGANIZAÇÃO DO TRABALHO

O Capítulo 1 contém a introdução do trabalho.

O Capítulo 2 contempla a revisão da literatura, na qual são apresentados os diversos tipos de problemas de sequenciamento da produção existentes, assim como os métodos de resolução utilizados neste trabalho: *Relax-and-Fix*, ILS, VNS, e LNS.

O Capítulo 3 apresenta o problema de sequenciamento da produção com sincronismo de execução de tarefas.

O Capítulo 4 propõe a formulação matemática do problema.

O Capítulo 5 traz os métodos desenvolvidos nesta tese para resolver estes problemas.

Finalmente, nos Capítulos 6 e 7, são apresentados os resultados, as conclusões do trabalho, e sugestões para futuras pesquisas. A seguir, apresenta-se a bibliografia consultada.

2 REVISÃO DA LITERATURA

2.1 PROBLEMAS DE SEQUENCIAMENTO DA PRODUÇÃO (*SCHEDULING*)

O planejamento da produção em ambientes de manufatura é, em geral, uma tarefa complexa que compreende, dentre outras atividades, o sequenciamento (*scheduling*) da produção (DOMINGOS *et al*., 2008).

Os problemas de *scheduling* referem-se aos procedimentos de alocação de recursos e equipamentos, num determinado período de tempo, para executar o processamento de tarefas, visando à otimização de um ou mais objetivos, sendo requeridos quando existe competição entre atividades por recursos limitados dentro de um horizonte de tempo definido. De acordo com Stebel *et al*. (2003), muitos deles podem ser modelados como problemas de Programação Linear Inteira Mista.

Tais problemas estão amplamente difundidos em diversas atividades industriais ou tecnológicas, tais como sistemas de produção e manufatura, sistemas de processamento de informações, logística, comunicações, etc; e têm adquirido um caráter notório, em virtude das exigências no aumento da produção de bens (GUIMARÃES, 2007).

O *scheduling* possui três elementos fundamentais: a designação dos recursos (máquinas, em um chão de fábrica, pistas em um aeroporto, processadores em um sistema de computação, caminhões de distribuição), que envolve a seleção de um conjunto de recursos apropriados para uma atividade conhecida; o seqüenciamento de atividades (operações em um processo de produção, pousos e decolagens em um aeroporto, execução de softwares em um sistema de computação, encomendas a serem entregues), que define a ordem de execução das atividades designadas para os recursos; e a determinação do tempo de utilização dos recursos pelas respectivas atividades, ou seja, a determinação dos tempos de início e término de cada atividade.

Guimarães (2007) destaca que, apesar da diversidade dos métodos de solução criados e da evolução dos computadores, muitos destes problemas ainda são considerados de difícil solução, devido a sua natureza combinatória. Num ambiente de produção do tipo *job shop* com m máquinas e n tarefas, por exemplo, existem $(n!)^m$ possibilidades de se executar as tarefas. Além disso, ao longo do tempo, os modelos criados passaram a incorporar novas características e restrições presentes nos ambientes de produção, o que dificulta ainda mais a sua resolução.

Este fato somado a sua grande aplicação em problemas do mundo real, faz com que se mantenha o interesse em estudar este tipo de problema, e em desenvolver técnicas de resolução que gerem boas soluções, num tempo computacional aceitável.

Desde os primeiros trabalhos apresentados na década de 50 que abordam este tema até os dias de hoje, inúmeras pesquisas foram realizadas, diversos tipos de problemas foram abordados e várias características e restrições foram incorporadas aos modelos (POTTS; STRUSEVICH, 2009).

Devido ao grande número de aspectos a serem considerados entre os diversos tipos de problemas de sequenciamento existentes, vários autores, baseados no trabalho de Graham *et al*. (1979) e Błazewicz *et al*. (1983), apresentam uma notação sistemática a fim de proporcionar uma visão conjunta dos mesmos, compreender com maior clareza suas semelhanças e diferenças, oferecer uma visão geral das principais abordagens propostas, antever direções e perspectivas de pesquisa, além de facilitar sua apresentação e discussão (ARAÚJO JÚNIOR, 2006; BRUCKER, 2010; CANTIERE; BOIKO, 2011; GOMES, 2008; GUIMARÃES, 2007; KAZAMA, 2011; LEUNG, 2004; LOPES, 2004; PACHECO; SANTORO, 1999; PINEDO, 2012).

Esta notação é composta por três termos $\alpha/\beta/\gamma$, onde cada um se refere a um componente do problema, conforme será visto a seguir.

2.1.1 Termo α

O termo α descreve o tipo de problema e o número de máquinas nele presentes. Geralmente ele é apresentado através de um único símbolo. Os possíveis tipos de problemas são:

a) Máquina única (1) – ambiente de produção que possui apenas uma máquina. Pode ser considerado como um caso particular de todos os outros ambientes mais complexos.

b) Máquinas paralelas idênticas (P_m) – ambiente de produção onde existem m máquinas idênticas. Cada tarefa pode ser processada em qualquer uma das máquinas com a mesma velocidade.

c) Máquinas paralelas uniformes (Q_m) – várias máquinas podem executar as mesmas tarefas com velocidades de processamento diferentes, mas que seguem uma relação conhecida entre si.

d) Máquinas paralelas não-relacionadas (R_m) – generalização do ambiente anterior, no qual as velocidades de processamento são diferentes em cada máquina, mas não seguem uma relação conhecida entre elas. Dessa forma, a velocidade de processamento depende da tarefa a ser executada.

e) *Flow shop* (F_m) – ambiente de produção que possui m máquinas especializadas em série, onde cada uma das tarefas deve ser processada em todas as máquinas, seguindo um mesmo roteiro, isto é, deve ser executada primeiro na máquina 1, depois na máquina 2, e assim sucessivamente. Em geral, depois de concluir a operação em uma máquina, as tarefas entram numa fila antes de iniciarem seu processamento na máquina seguinte.

f) *Flexible flow shop* (FF_s) – este ambiente é uma generalização do *flow shop* e do ambiente com máquinas paralelas, em que existem s estágios de processamento em série, com um número de máquinas em paralelo em cada um. Todas as tarefas seguem o mesmo roteiro e devem passar por uma das máquinas de todos os estágios, primeiramente no estágio 1, a seguir no estágio 2, e assim por diante.

g) *Job shop* (J_m) – neste ambiente, as operações de cada tarefa são executadas numa sequência específica de máquinas especializadas, havendo uma rota própria através das máquinas para cada tarefa. Desse modo, o *job shop* é caracterizado por permitir diferentes fluxos de tarefas entre as máquinas, e diferentes números de operações por tarefa.

h) *Flexible job shop* (FJ_m) – este ambiente de produção é uma extensão do *job shop*, sendo constituído por vários centros de trabalho. Em cada centro de trabalho existem várias máquinas capazes de executar a mesma operação. Dessa forma, há uma rota própria para cada tarefa ao longo dos centros de trabalho, mas as operações são realizadas em apenas uma das máquinas de cada centro.

i) *Open shop* (O_m) – neste ambiente, cada tarefa deve ser processada em várias máquinas, mas não necessariamente em todas. Além disso, não existem roteiros de

processamento pré-estabelecidos para as tarefas, isto é, não existe nenhuma sequência obrigatória de processamento das tarefas ao longo das máquinas.

2.1.2 Termo β

O termo β descreve as características de processamento, e as restrições das tarefas e dos recursos. Este termo pode ser apresentado através de um único símbolo, múltiplos símbolos, e até mesmo sem nenhum símbolo. Os possíveis símbolos de β são:

a) Data de liberação (*release date* - r_j) – este termo indica que as tarefas possuem uma data mínima a partir da qual sua execução pode ser iniciada. Dessa forma, nenhuma tarefa pode entrar em processo antes de sua data de liberação.

b) Data de entrega de uma tarefa (*due date* - d_j) – é o momento em que uma tarefa deveria ser concluída. Normalmente existem penalidades quando este prazo não é cumprido.

c) Data de limite de entrega (*deadline* - $\overline{d_j}$) – este campo indica que existe um limite máximo de tempo em que uma tarefa deve ser impreterivelmente concluída.

d) Tempos de preparação (*set-up*) dependentes das máquinas e tarefas (s) – nestes problemas, existe um tempo de preparação entre duas tarefas, o qual depende da máquina onde as tarefas serão processadas, e da própria tarefa. Esse tempo inclui o tempo para obtenção das ferramentas, posicionamento dos materiais a serem usados no trabalho, processos de limpeza, preparação e ajuste das ferramentas, inspeção de materiais, etc. Durante esse tempo, as máquinas não podem executar operações específicas de nenhuma tarefa.

e) Tempo de preparação (*set-up*) dependente da sequência ($s_{i,j}$) – este símbolo indica que o tempo de preparação das máquinas depende tanto da tarefa a ser processada, quanto daquela que foi processada imediatamente antes numa mesma máquina.

f) Tempo de preparação dependente das famílias de produtos (*batch set-up problem*): neste tipo de problema, as tarefas são agrupadas em famílias de produtos antes de serem processadas. A transição de uma família para outra requer um grande tempo de preparação,

enquanto que a transição entre tarefas de uma mesma família necessita de um tempo de preparação muito menor. Maiores detalhes sobre problemas que envolvem tempos de preparação podem ser vistos nos trabalhos de Allahverdi, Gupta e Aldowaisan (1999) e de Allahverdi *et al*. (2008). Já para problemas de sequenciamento com agrupamentos em famílias de produtos, recomenda-se o trabalho de Méndez *et al*. (2006).

g) Interrupção (*preemptions* – prmp) – neste tipo de problema, existe a possibilidade de uma tarefa ter sua execução interrompida antes de sua conclusão. O tempo já processado não é perdido; e, após o seu retorno, a tarefa deve permanecer na máquina apenas o tempo restante necessário para concluir sua operação. Se existirem máquinas paralelas, a tarefa não é obrigada a terminar sua execução na mesma máquina em que começou. Durante o período de interrupção, pode-se executar a operação de outra tarefa na máquina em que estava sendo processada.

h) Recirculação (recrc) – em sistemas com esta característica, uma tarefa pode visitar uma mesma máquina mais de uma vez.

i) Restrições de precedência (prec) – este parâmetro indica que existe uma ordem de execução entre as tarefas. Dessa forma, uma determinada tarefa só pode iniciar seu processamento após o término de outra específica.

j) Restrições de dependência entre operações – em alguns ambientes, depois de se executar uma determinada operação de uma dada tarefa, deve-se obrigatoriamente executar a próxima operação dentro de uma janela de tempo. Um caso particular deste tipo de restrição (*no-wait* – nwt) ocorre em ambientes onde não existem filas intermediárias entre duas máquinas. Neste caso, o tempo de espera entre as operações deve ser nulo, e o início da primeira operação deve ser postergado o bastante para garantir que a segunda máquina esteja disponível no instante em que a primeira finalize seu trabalho.

k) Bloqueio (*block*) – esta condição pode ocorrer em ambientes *flow shop* quando existe um local de armazenamento de tamanho limitado entre duas máquinas consecutivas. Dessa forma, quando este local estiver lotado, a máquina imediatamente anterior a ele não pode liberar a tarefa que acabou de ser executada, o que impede temporariamente que uma nova tarefa seja processada.

l) Permutação (prmu) – em alguns ambientes *flow shop*, as tarefas devem ser executadas obrigatoriamente de acordo com a regra FIFO (*First In First Out*). Isto implica que a ordem de execução das tarefas na primeira máquina é mantida a mesma durante todo o sistema.

m) Tempo de comunicação/transporte entre máquinas – neste tipo de problema, uma tarefa deve aguardar um tempo mínimo entre duas operações consecutivas, mesmo que a máquina da frente esteja disponível no instante de término da operação da máquina anterior.

n) Quebra de máquinas (brkdwn) – neste ambiente, as máquinas não estão disponíveis continuamente. Dependendo do problema, esta indisponibilidade pode ser modelada como um tempo fixo, ou através de distribuições de probabilidade.

o) Restrições de escolha de máquinas – em alguns problemas que possuem máquinas paralelas, algumas tarefas não podem ser executadas em todas as máquinas de um estágio.

p) Restrição do número de tarefas (nbr) – neste ambiente, existe um número máximo de tarefas que podem ser processadas, dentre todas as tarefas disponíveis.

2.1.3 Termo γ

O termo γ se refere à função objetivo e define os critérios de otimização do problema. Antes de apresentar os possíveis símbolos deste campo, convém apresentar algumas definições:

a) Tempo de término de uma tarefa (*completion time* - C_j) – corresponde ao instante de término de processamento da tarefa j.

b) Tempo de fluxo de uma tarefa (*flowtime* - F_j) – é a soma dos tempos de espera e de processamento de uma tarefa.

c) Desvio da data de entrega (*lateness*) de uma tarefa (L_j) – é a diferença entre a data de término de uma tarefa e a sua data de entrega, podendo assumir um l positivo quando a tarefa estiver atrasada, ou negativo, quando está adiantada.

d) *Tardiness* (T_j) – é semelhante ao *lateness*, mas não considera as tarefas que estiverem adiantadas. Pode ser interpretado como o tempo de atraso das tarefas.

e) *Earliness* (E_j) – é semelhante ao *lateness*, mas não considera as tarefas que estiverem atrasadas. Pode ser interpretado como o tempo de antecipação das tarefas.

Em geral, o termo γ está relacionado com os prazos de entrega das tarefas ou com os tempos de término de processamento.

Os critérios de otimização baseados no tempo de conclusão das tarefas conduzem a um menor tempo médio de permanência das mesmas no sistema, o que resulta num menor tempo médio de resposta aos clientes, e menor custo médio de estoques em processo. Os principais critérios de otimização deste tipo são:

a) *Makespan* ($C_{máx}$) - é o instante de término da última tarefa a deixar o sistema. A sua minimização normalmente conduz a níveis elevados de utilização das máquinas.

b) Tempo total de término (*total completion time*) – é a soma dos instantes de conclusão de cada tarefa. Este critério vem ganhando cada vez mais importância nas indústrias com a busca por uma produção mais enxuta, uma vez que, por meio dele, consegue-se uma redução do estoque em processo (*work in process* – WIP), e uma utilização mais estável dos equipamentos.

c) Tempo total de término ponderado (*total weighted completion time*) – é a soma ponderada dos instantes de término de cada tarefa, de acordo com um peso atribuído a cada uma delas.

Os principais critérios de otimização baseados no tempo de fluxo são:

d) Tempo de fluxo total (*total flowtime*)

e) Soma ponderada dos tempos de fluxo (*total weighted flowtime*)

Os critérios de otimização baseados nas datas de entrega das tarefas são de grande importância em ambientes de produção *make-to-order*. Sua minimização resulta num melhor serviço prestado aos clientes. Dessa forma, estes critérios geralmente têm um forte impacto na preferência dos clientes, e podem constituir uma vantagem competitiva. Os principais critérios de otimização deste tipo são:

f) Desvio máximo com relação à data de entrega (*maximum lateness* - $L_{máx}$) – mede a maior violação de tempo de entrega entre todas as tarefas.

g) Soma dos atrasos (*total tardiness*).

h) Soma ponderada dos atrasos (*total weighted tardiness*) – é definido como a soma ponderada da função *tardiness* de cada tarefa.

i) Número total de tarefas em atraso (*number of tardy jobs*).

j) Soma dos atrasos e antecipações (*total earliness-tardiness penalty*).

l) Soma ponderada dos atrasos e antecipações (*total weighted earliness-tardiness penalty*).

Além disso, há outros objetivos não tão frequentes:

m) Minimização do tempo de preparação das máquinas.

n) Minimização do custo de preparação das máquinas.

2.1.4 Outros aspectos

Além dos três aspectos principais apresentados até aqui ($\alpha/\beta/\gamma$), existem outros que diferenciam os problemas de sequenciamento entre si (PACHECO; SANTORO, 1999).

Os tempos de processamento das tarefas podem ser determinísticos, quando são fixos e conhecidos; ou estocásticos, quando estão submetidos a leis do acaso.

Com relação ao processo de chegada das ordens, os sistemas podem ser divididos em estáticos, quando as tarefas a serem programadas estão todas disponíveis no início da programação; ou dinâmicos, quando as tarefas chegam ao longo do tempo. Chegadas dinâmicas podem ser divididas, ainda, em determinísticas, quando os instantes de chegada são previamente conhecidos, ou aleatórias, quando os instantes seguem, por exemplo, uma distribuição de probabilidade.

2.1.5 Problemas com operações simultâneas e/ou algum tipo de sincronismo

Esta seção apresenta classes de problemas de sequenciamento encontrados na literatura que possuem operações simultâneas e/ou algum tipo de restrição de sincronismo.

Uma dessas classes abrange os problemas de programação de rotação de culturas (GOMES; ARENALES, 2010; HARI; YANG; SURYANI, 2012; SANTOS *et al*., 2010). Nestes problemas, culturas diferentes são plantadas periodicamente e de maneira sucessiva em uma mesma área, de forma a aumentar a sustentabilidade ecológica, econômica e social da agricultura. De uma forma geral, o objetivo destes problemas é encontrar a ordem de plantio das diversas culturas, atendendo às restrições de base ecológica, tais como: períodos em que a vegetação espontânea cresce livremente (o que contribui para o controle biológico de diversos insetos e para recuperar a estrutura física, química e biológica do solo); cultivo de espécies para adubação verde (importante para a fixação de nitrogênio no solo); e proibição do plantio de duas culturas de mesma família botânica em sequência (já que muitas pragas e doenças atacam culturas de mesma família).

Uma das variações deste problema é o problema de programação da produção com restrições de adjacência (SANTOS, 2009; SANTOS *et al*., 2011), no qual culturas de uma mesma família são impedidas de serem cultivadas simultaneamente em áreas vizinhas. Esta restrição, que tem como objetivo dificultar a propagação de pragas e doenças, é uma equivalente espacial da restrição temporal anterior, e pode ser vista como uma restrição de sincronismo. Entretanto, este sincronismo difere do sincronismo que será tratado neste trabalho.

A primeira diferença aparece na interação entre as tarefas: enquanto neste trabalho há tarefas que são obrigadas a caminhar juntas, no problema de rotação de culturas, algumas

tarefas são impedidas de serem processadas lado a lado. Além disso, enquanto aqui há apenas a exigência que o término das tarefas seja o mesmo, lá o sincronismo possui um momento de início e outro de término. Finalmente, há uma diferença relacionada com a quantidade de tarefas envolvidas durante o sincronismo: no problema apresentado nesta tese, o sincronismo envolve sempre pares de tarefas; já no outro problema, o sincronismo pode envolver diversas tarefas.

Outro tipo de problema aparece em ambientes que possuem máquinas capazes de processar várias tarefas simultaneamente (GENTNER *et al*., 2004; POTTS; KOVALYOV, 2000). Como exemplos, podem-se citar a formação de cargas de processamento em fornos de tratamento térmico, alguns processos químicos que ocorrem em tanques ou estufas, dentre outros. Em geral, nesses sistemas, cada equipamento possui um limite máximo de tarefas que podem ser processadas simultaneamente; além disso, todas as tarefas que compõem uma mesma carga iniciam seu processamento no mesmo instante, e permanecem no mesmo equipamento até que a tarefa com maior tempo de processamento seja concluída. Dessa forma, todas as tarefas possuem, também, o mesmo horário de término.

Este problema também apresenta diferenças em relação ao deste trabalho. No problema desta tese, pares de tarefas devem terminar seu processamento, obrigatoriamente, em duas máquinas diferentes, ao mesmo tempo, podendo ter inícios em momentos diferentes. Além disso, os pares não podem ser processados numa mesma máquina. Já em problemas com processamento simultâneo de tarefas, diversas tarefas podem ser processadas ao mesmo tempo em uma única máquina, e, quando isto acontece, todas têm os mesmos instantes de início e fim.

Existem também problemas, como o RASDST (*Resource-assignable Sequence Dependent Set-up Time*), proposto por Ruiz e Andrés (2007) que podem utilizar mais de um recurso ao mesmo tempo. Nesse problema, deve-se determinar a melhor sequência de *n* tarefas em *m* máquinas paralelas distintas, sabendo-se que o tempo de *set-up* varia com a quantidade de recursos designados para sua execução; isto é, quanto mais recursos forem utilizados, menor será o tempo de *set-up*, e vice-versa (BICALHO; SANTOS; ARROYO, 2011; KAMPKE, 2010). Um dos objetivos do problema pode ser, por exemplo, minimizar simultaneamente, o tempo total de produção e a quantidade de recursos empregados para preparar as máquinas.

Este problema ocorre, por exemplo, na fabricação de cerâmicas, onde existem diferentes máquinas de polimento. Ao final do polimento de cada tipo de cerâmica, a máquina precisa ser limpa e preparada para realizar o polimento de outro tipo de cerâmica. Este tempo varia de

acordo com a máquina, com o tipo da cerâmica polida anteriormente, com o tipo de cerâmica que será polida em seguida, e com o número de recursos utilizados, que neste caso, pode ser o número de pessoas que realizarão a limpeza e o ajuste da máquina.

Neste problema, há um sincronismo entre os recursos, e não entre as tarefas. Mesmo assim, este sincronismo não é obrigatório: ainda que leve mais tempo, um único recurso consegue realizar o *set-up*. Também não há restrições em relação aos momentos de início e fim de processamento de cada tarefa, e não há impedimento de que certas tarefas não possam ser processadas numa mesma máquina em que outras tarefas foram produzidas.

Há ainda problemas em que as tarefas podem ser divididas em partes menores para serem processadas simultaneamente, ou não, em máquinas diferentes (BERALDI *et al*., 2008; SERAFINI, 1996; TAHAR *et al*., 2006; XING; ZHANG, 2000). Estes problemas são comuns em sistemas de computação com multiprocessadores, e na indústria têxtil, onde uma tarefa pode representar, por exemplo, um conjunto de 1000 pares de meias; o qual pode ser fragmentado em subconjuntos menores para serem processados em diversas máquinas, a fim de reduzir o tempo total de execução.

Enquanto nesses problemas uma tarefa pode ser desmembrada num número qualquer de subtarefas e ocupar diversas máquinas, nesta tese há sempre pares de tarefas que são processados em duas máquinas. Além disso, como o desmembramento não é obrigatório nos problemas com divisão de tarefas, uma tarefa pode ser processada inteiramente numa única máquina, o que não ocorre neste trabalho, já que os pares de tarefas não podem ser processados numa mesma máquina. Finalmente, nos problemas com divisão de tarefas, as subtarefas podem terminar seu processamento em momentos diferentes, o que não ocorre no problema apresentado aqui.

2.2 MÉTODOS DE RESOLUÇÃO

Ao longo do tempo, diversos métodos exatos têm sido desenvolvidos para encontrar a solução ótima dos problemas de *scheduling*, merecendo destaque os métodos de enumeração implícita do tipo *branch-and-bound*.

Entretanto, encontrar uma solução ótima para este tipo de problemas continua sendo uma tarefa difícil. Em geral, estes métodos conseguem encontrar a solução ótima num tempo aceitável somente em problemas que envolvem um pequeno número de máquinas e tarefas, não sendo eficientes para problemas de médio e grande porte, comuns em sistemas reais.

Além disso, como em muitas situações práticas não se exige a obtenção de uma solução ótima, diversos métodos heurísticos e metaheurísticos têm sido desenvolvidos e aplicados com bons resultados.

Nesta tese são apresentados os métodos que serão utilizados neste trabalho: *relax-and-fix*, busca local iterativa (*iterated local search* – ILS), busca em vizinhança variável (*variable neighborhood search* – VNS), e busca em grande vizinhança (*large neighborhood search* - LNS).

2.2.1 Método *relax-and-fix*

A heurística *relax-and-fix* é um método iterativo que utiliza a relaxação linear e que decompõe um problema de programação inteira mista de difícil solução em subproblemas menores, que podem ser resolvidos rapidamente (FURLAN, 2011; KAWAMURA, 2012; TOFFOLO, 2009).

Neste método, as variáveis inteiras são divididas em vários conjuntos disjuntos. Cada um desses conjuntos pode ser visto como uma partição do universo de variáveis. A cada iteração, apenas as variáveis de um dos conjuntos são definidas como inteiras, enquanto todas as outras são relaxadas; reduzindo, assim, o número de variáveis inteiras, e tornando mais fácil a resolução exata do problema. Em seguida, resolve-se o submodelo gerado, e o valor encontrado para as variáveis inteiras é fixado para as próximas iterações. O processo se repete até que todas as variáveis sejam fixadas, ou até que uma solução inviável seja encontrada. O número de conjuntos determina o número teórico de iterações. A Figura 2 apresenta o pseudocódigo deste método.

Figura 2 – Pseudocódigo *relax-and-fix*

```
01: Dividir as variáveis inteiras em N conjuntos disjuntos Q_i
02: Relaxar todas as variáveis inteiras
03: i = 1;
04: Repita
05:    Retornar variáveis do conjunto Q_i ao seu estado original;
06:    Resolver modelo resultante;
07:    Fixar os valores das variáveis do conjunto Q_i com os valores obtidos;
08: Até i = N
09: Retornar solução encontrada
```

Fonte: (TOFFOLO, 2009)

Caso seja encontrada uma solução inviável em alguma iteração, pode-se retirar os valores fixados na iteração anterior e repetir a iteração em que a inviabilidade foi detectada.

Uma característica importante desta heurística, é que ela fornece um limitante inferior para o problema, uma vez que sua primeira iteração é realizada sobre uma relaxação do problema original.

Estas heurísticas vêm sendo amplamente utilizadas em problemas de planejamento da produção (FURLAN, 2011).

Araujo, Arenales e Clark (2008) empregaram o *relax-and-fix* em problemas de dimensionamento e sequenciamento de lotes de produção em uma máquina, com o objetivo de minimizar atrasos de entrega.

Para resolver os problemas, os autores separaram as variáveis em períodos de tempo, e resolveram os subproblemas resultantes em ordem cronológica crescente. À medida que cada período era resolvido, as variáveis iam sendo fixadas definitivamente com os valores encontrados em cada iteração.

Foram gerados problemas com 10, 50 e 100 itens a serem produzidos a partir de 2, 10 e 20 materiais diferentes, respectivamente; sendo que um material podia gerar mais de um item, e que um item deveria ser gerado a partir de um único material.

O tempo total de execução para os problemas foi de 3, 6 ou 12 minutos, de acordo com o tamanho do problema, e os resultados obtidos ficaram, em média, 8,6% distantes do limitante inferior obtido pelo *software* CPLEX 7.1 para o problema original, após uma hora de execução. Para efeito de comparação, os resultados obtidos pelo CPLEX ficaram, em média, 22,3% distantes do limitante inferior.

Beraldi *et al*. (2008) utilizaram o *relax-and-fix* para resolver problemas de dimensionamento e sequenciamento da produção com máquinas paralelas idênticas, custos de *set-up* dependentes da sequência e tarefas que podiam ser desmembradas para serem processadas em mais de uma máquina, ao mesmo tempo.

Os autores desenvolveram quatro estratégias para agrupar as variáveis.

Na primeira delas, o horizonte de planejamento foi dividido em intervalos de tempo, gerando subproblemas que eram resolvidos a partir daquele que estivesse mais próximo do instante zero.

A segunda estratégia dividiu as variáveis de acordo com os produtos, e deu-se preferência em resolver os subproblemas que tivessem uma maior demanda.

Na terceira estratégia, o horizonte de planejamento foi dividido em intervalos de tempo, e, dentro de cada um deles, as variáveis foram separadas em produtos e de acordo com sua demanda.

Na última, foi utilizado um raciocínio inverso: inicialmente as variáveis foram separadas por produtos, e, a seguir, por períodos de tempo.

Foram resolvidos 60 problemas com 100 máquinas, 24 produtos e 30 intervalos de tempo, e os resultados foram comparados com os limitantes inferiores gerados pelo *software* CPLEX 7.0 após uma hora de execução.

Todas as abordagens encontraram soluções até 6% distantes do limitante inferior gerado pelo CPLEX, e foi observado que os melhores resultados foram obtidos pela segunda estratégia.

Ferreira, Morabito e Rangel (2008, 2010), utilizaram o método *relax-and-fix* para resolver problemas de dimensionamento e sequenciamento de lotes de produção com tempos e custos dependentes da sequência.

Numa primeira abordagem os autores agruparam as variáveis por períodos de tempo.

Na segunda estratégia, além de dividir o problema em períodos, foram fixadas também as variáveis que determinavam se haveria trocas de produtos em cada período.

O terceiro modelo era semelhante ao segundo, mas fixava também a quantidade de cada produto que deveria ser produzida em cada período.

A quarta abordagem assemelhava-se à terceira, mas permitia reajustar o tamanho dos lotes de produção.

A última permitia, também, reavaliar os períodos que ficassem ociosos ao longo das iterações.

Foram resolvidos problemas com 27 produtos distintos, e os resultados das heurísticas foram comparados com os obtidos pela resolução do problema original pelo *software* CPLEX.

Apenas a terceira abordagem encontrou soluções piores que a do CPLEX, executado durante três horas. A melhor estratégia foi a segunda, que encontrou soluções 14% melhores que o CPLEX, em média.

Mohammadi e Ghomi (2010) estudaram o problema de dimensionamento e sequenciamento da produção com custos de *set-up* dependentes da sequência em um ambiente *flow shop* permutacional, que tinha como objetivo, minimizar os custos de produção, armazenamento e de *set-up*.

Os autores desenvolveram uma heurística *relax-and-fix*, onde as variáveis relacionadas com o *set-up* entre dois produtos eram relaxadas de acordo com o período de tempo que

estava sendo avaliado. Dessa forma, a cada iteração, apenas as variáveis de um dado intervalo de tempo eram consideradas para otimização.

Foram resolvidos diversos problemas, variando-se o número de produtos, máquinas e intervalos de tempo de 3 a 15, cada um.

James e Almada-Lobo (2011) criaram um método que combinava programação inteira mista com metaheurísticas para resolver o problema de dimensionamento e sequenciamento de lotes de produção com máquinas paralelas com capacidades diferentes, e tempos de *set-up* dependentes da sequência.

Este método era composto de uma fase de construção, onde era gerada uma solução inicial, e de uma fase de refino desta solução, baseada no método VNS.

Na fase de construção, foi utilizado o método *relax-and-fix*, no qual as variáveis foram agrupadas em intervalos de tempo. Para isso, o horizonte de planejamento foi dividido em diversos períodos sem sobreposição. A cada iteração, apenas as variáveis inteiras de um dos períodos eram consideradas para otimização. Os períodos eram resolvidos seguindo-se uma ordem cronológica crescente, até que todos fossem solucionados.

Foram avaliados problemas com 10 a 25 produtos e com 5 a 15 períodos de tempo. Os resultados obtidos com o método proposto superaram aqueles obtidos pelo software CPLEX e por outras heurísticas e metaheurísticas implementadas, como a *fix-and-optimize*, Busca Tabu e VNS.

Kawamura (2012) aplicou o método *relax-and-fix* em problemas de dimensionamento e sequenciamento de lotes de produção em um sistema com máquinas distintas em paralelo, restrição de armazenagem e penalidades por atraso de entrega.

Foram propostas diversas estratégias de divisão das variáveis, as quais foram classificadas em três grupos. No primeiro, as variáveis foram divididas em intervalos de tempo de execução; no segundo, de acordo com a velocidade e eficiência das máquinas; e no terceiro, de acordo com os tipos de produtos.

Para o primeiro grupo, havia duas variações do problema. Numa delas, a sequência de resolução dos subproblemas seguia uma ordem cronológica crescente; na outra, uma ordem decrescente. Os melhores resultados foram obtidos com a primeira opção.

No segundo grupo, foram feitas duas abordagens. Na primeira, a resolução dos subproblemas tinha início com o sequenciamento das máquinas melhores, e terminava com o sequenciamento das piores. Na segunda, o sequenciamento era feito de acordo com a criticidade das máquinas. Assim, as máquinas que possuíam produtos com menos alternativas de produção eram priorizadas. A primeira estratégia foi a que gerou os melhores resultados.

Com relação aos tipos de produtos, dava-se prioridade aos que tivessem maiores demandas. Numa segunda abordagem, foram priorizados os produtos com menor demanda. Finalmente, a última estratégia se baseava na criticidade dos produtos; dessa forma, os produtos com menos alternativas de produção eram fixados primeiro. Esta última foi a que gerou os melhores resultados para este grupo.

Foram resolvidos problemas com 2 a 10 máquinas distintas em paralelo e com 12 a 80 produtos. Entretanto, os *gaps* de otimalidade dos resultados obtidos foram elevados para todas as abordagens.

O Quadro 1 apresenta um resumo com as principais estratégias de partição das variáveis dos trabalhos citados nesta seção.

Quadro 1 – Estratégias de partição de variáveis

Ano	Autores	Partição
2008	Araujo, Arenales e Clark	- em intervalos de tempo.
2008	Beraldi *et al*.	- em intervalos de tempo; - em função dos tipos de produtos.
2008	Ferreira, Morabito e Rangel	- em intervalos de tempo; - de acordo com as variáveis que determinavam se haveria trocas de produtos.
2010	Mohammadi e Ghomi	- de acordo com as variáveis relacionadas com o *set-up*; - em intervalos de tempo.
2011	James e Almada-Lobo	- em intervalos de tempo.
2012	Kawamura	- em intervalos de tempo; - em função da velocidade e da eficiência das máquinas; - em função dos tipos de produtos.

Como pode ser visto no Quadro 1, as estratégias mais comuns nos trabalhos que utilizaram o *relax-and-fix* foram: dividir as variáveis em intervalos de tempo ou em função dos tipos de produtos.

2.2.2 Método ILS

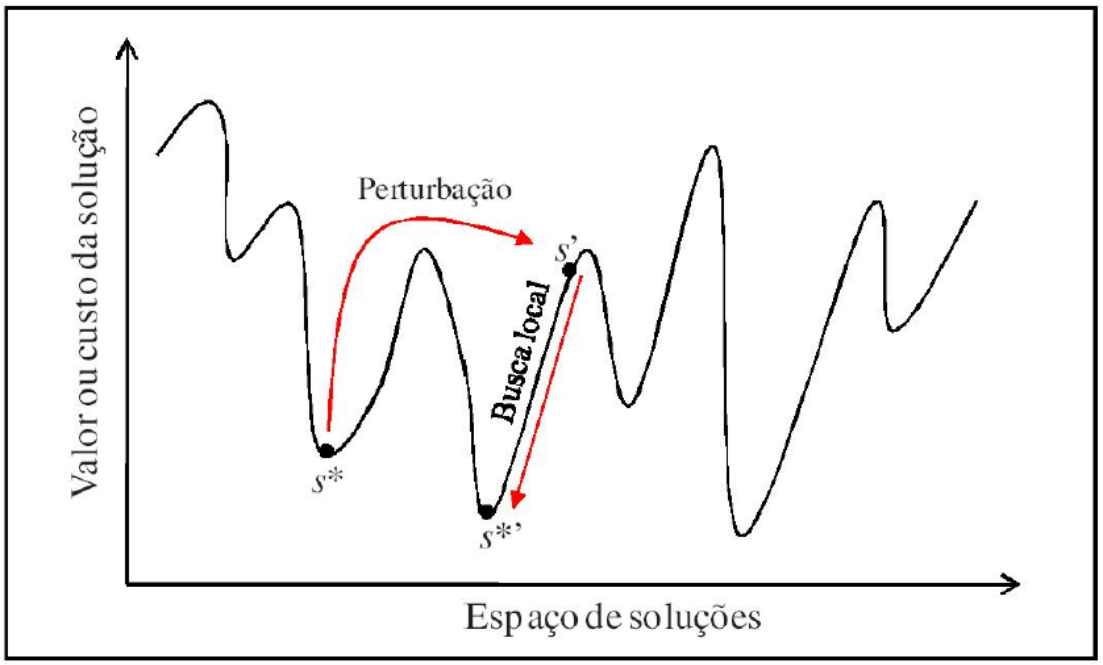
O método ILS (*Iterated Local Search*) é um método de busca local que permite a exploração de pequenos espaços de vizinhança, definidos por ótimos locais de um

determinado procedimento de otimização, ao invés de explorar todo o universo de soluções possíveis para o problema em estudo (GOMES JUNIOR, 2007; KAMPKE, 2010).

Considerada como uma das metaheurísticas mais generalizadas, simples, robustas, eficazes e de fácil implementação, baseia-se na ideia de que uma busca local pode ser melhorada através da geração de novas soluções de partida, produzidas por meio de perturbações na solução ótima local corrente. Tais perturbações precisam ser suficientemente fortes para fugir do ótimo local atual, de forma a explorar novas regiões no espaço de soluções, mas ao mesmo tempo fracas para guardar características do ótimo local corrente e evitar um reinício aleatório (BLUM; ROLI, 2003; LOURENCO; MARTINS; STUTZLE, 2001, 2003).

A Figura 3 representa tal ideia: a partir de uma solução ótima local S*, realiza-se uma perturbação, de forma a se obter uma solução S', que será utilizada como ponto de partida por um processo de busca local, que tem como objetivo tentar encontrar soluções S*' melhores que a atual.

Figura 3 – Método ILS



Fonte: (KAMPKE, 2010).

Se a solução S*' for aprovada no critério de aceitação escolhido, ela se tornará o próximo elemento a sofrer a perturbação; caso contrário, realizar-se-á outra perturbação na solução S* atual.

Dessa forma, Lourenço, Martin e Stutzle (2001, 2003) afirmam que há quatro elementos que devem ser considerados durante a criação de um algoritmo ILS: geração de uma solução

inicial, que pode ser, por exemplo, obtida aleatoriamente ou por meio de algum algoritmo guloso; execução de um procedimento de busca local, que retornará uma solução melhorada (em relação à solução inicial); execução de um procedimento de perturbação, que alterará a solução corrente de maneira a escapar do mínimo local atual, permitindo a exploração de novas soluções; e determinação de um critério de aceitação que indicará a partir de qual solução a próxima perturbação será realizada.

Assim sendo, pode-se afirmar, ainda segundo esses autores, que a busca local, juntamente com o mecanismo de perturbação definem as possíveis transições entre a solução atual e a solução vizinha; enquanto que o critério de aceitação, em conjunto com o mecanismo de perturbação, controla a relação entre intensificação e diversificação da busca.

A Figura 4 apresenta o pseudocódigo deste método.

Figura 4 – Pseudocódigo ILS

```
01:  Entrada: Solução ótima local S
02:  S* = S;
03:  Repita
04:     S' <= Perturbação (S);
05:     S*' <= Busca Local (S');
06:     se c (S*') < c (S*) então
07:        S* = S*';
08:     fim se
09:     S <= Critério de Aceitação (S, S*');
10:  Até atingir critério de parada
11:  Retornar S*
```

Fonte: (KAMPKE, 2010)

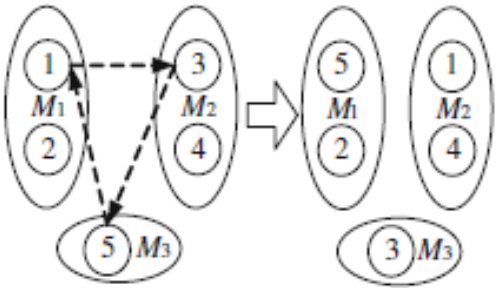
O método ILS vem sendo aplicado satisfatoriamente em diversos problemas de otimização combinatória. Vale a pena destacar seu emprego em problemas do caixeiro viajante e em problemas de sequenciamento da produção (KAMPKE, 2010; LOURENCO; MARTINS; STUTZLE, 2001).

Tang e Luo (2006) desenvolveram um método para minimizar o tempo total de conclusão das tarefas em problemas de sequenciamento em máquinas paralelas idênticas.

A solução inicial era gerada a partir da heurística LPT (*longest processing time*), na qual a tarefa com maior tempo de processamento é selecionada, a cada iteração, para ocupar a primeira máquina que ficar livre.

A seguir, era executado um algoritmo de busca local, que realizava trocas cíclicas entre tarefas de máquinas distintas, como pode ser visto na Figura 5.

Figura 5 – Trocas cíclicas de tarefas



Fonte: (TANG; LUO, 2006)

Inicialmente, as máquinas eram colocadas em ordem crescente, de acordo com o tempo total de conclusão das tarefas de cada uma; a seguir, o algoritmo tentava encontrar um ciclo entre as máquinas com o maior e o menor tempo que melhorasse a solução atual. Se tal ciclo não fosse encontrado, procurava-se um novo ciclo entre a máquina com maior tempo de conclusão e outra máquina selecionada aleatoriamente. Se ainda assim não houvesse melhoria, realizava-se uma nova busca entre a máquina com maior tempo de conclusão e as duas menores, e assim sucessivamente, até que todas as máquinas fizessem parte do ciclo. A melhor solução encontrada neste procedimento correspondia ao ótimo local.

Para realizar a perturbação, eram geradas duas trocas cíclicas entre as tarefas aleatoriamente.

Como critério de parada, foi estabelecido um número máximo de iterações sem melhoria na solução atual.

O método foi aplicado em problemas com 3 a 40 máquinas e 10 a 1000 tarefas. As soluções encontradas pelo ILS foram até 7,21% superiores às encontradas pela heurística LPT. Em 83% dos casos, ele encontrou a solução ótima do problema.

Gomes Júnior (2007) propôs um método heurístico baseado no ILS para resolver problemas de sequenciamento de 6 a 75 *jobs* em uma máquina, com penalidades por antecipação e atraso da produção, tempo de preparação dependente da sequência e janelas de entrega.

Para gerar a solução inicial, o autor utilizou o método de construção da metaheurística GRASP, no qual os elementos são inseridos um a um na solução de acordo com uma função gulosa.

A seguir, foram utilizados três métodos de busca local: método randômico de descida (MRD); método de descida; e o método de descida em vizinhança variável (VND). Todos eles

utilizaram dois tipos de movimentos para explorar o espaço de soluções: troca da ordem de processamento de dois *jobs* da sequência de produção, o que permite explorar $n(n\text{-}1)/2$ vizinhos (onde n é o número de *jobs*; e realocação do *job* para outra posição na sequência de produção, o que permite explorar outros $(n\text{-}1)^2$ vizinhos.

As perturbações foram feitas por meio de trocas aleatórias entre dois *jobs* quaisquer da solução. O nível de perturbação era aumentado em uma unidade sempre que pelo menos 10% da vizinhança da solução atual fosse explorada sem melhoria da solução corrente, e retornava para seu valor original quando houvesse melhora na solução. Este processo era interrompido após um número máximo de iterações sem melhora no valor da melhor solução encontrada até o momento.

O método se deslocava de uma solução para outra somente se o ótimo local encontrado fosse melhor do que o ótimo local corrente.

Para problemas de pequenas dimensões (até 12 *jobs*), o método encontrou soluções até 0,5% distantes das soluções ótimas.

Nunes e Arroyo (2008) utilizaram o ILS como processo de pós-otimização de um método GRASP aplicado em problemas de sequenciamento de tarefas em uma máquina com tempos de preparação dependentes da sequência, com o objetivo de minimizar o atraso total em relação às datas de entrega.

O método tinha início com a geração de uma solução inicial, a partir de um método GRASP que utilizava uma estratégia gulosa para minimizar o atraso das tarefas (GUPTA; SMITH, 2006). Nesta estratégia, a sequência era gerada com a inserção das tarefas, uma a uma, de acordo com suas datas de entrega, e seus tempos de processamento, e de preparação.

Depois de gerar a sequência inicial, era realizado um processo de busca local que utilizava movimentos de inserção, seguidos de movimentos de trocas entre as tarefas, em conjunto com o critério primeiro de melhora, sendo escolhida a primeira solução que fosse melhor que a solução atual.

O ILS era, então, aplicado sobre a melhor solução encontrada pelo método GRASP, a qual era perturbada por duas trocas: a primeira entre duas tarefas adjacentes; e a segunda entre tarefas que se encontravam a certo número de posições distantes entre si, definido previamente.

A seguir, era aplicado o mesmo processo de busca local visto anteriormente. O ótimo local era aceito sempre que fosse melhor que a melhor solução corrente.

O método era interrompido após um número máximo de iterações, no caso, 1000.

Os autores resolveram 64 problemas, que possuíam de 15 a 85 tarefas. A utilização do ILS melhorou as soluções, em 6,5%, em média, com relação ao método GRASP básico. Comparando com o algoritmo proposto por Gupta e Smith (2006), dos 64 problemas resolvidos, os dois algoritmos encontraram a mesma solução em 37 problemas, mas o GRASP-ILS obteve uma solução melhor em 23 casos.

Chen (2008) aplicou o ILS em problemas de sequenciamento com máquinas paralelas não relacionadas, tempos de *set-up* dependentes da sequência de operações, e diferentes datas de entrega, com o objetivo de minimizar o peso total em atraso.

O autor utilizou, separadamente, quatro heurísticas para obter uma solução inicial. A primeira gerava a sequência aleatoriamente. A segunda utilizava a heurística *shortest weighted processing* time (SWPT), onde as tarefas eram sequenciadas de acordo com a relação entre seus tempos de processamento e seus pesos. A terceira utilizava a heurística *earliest weighted due date* (EWDD), na qual, as tarefas eram sequenciadas de acordo com a relação entre a data acordada de entrega e seu respectivo peso. A última utilizava a heurística *shortest ready time* (SRT), onde as tarefas eram sequenciadas de acordo com seus tempos de prontidão.

Foi utilizada a busca tabu como processo de busca local.

A estrutura de vizinhança utilizada consistia em movimentos de inserção das tarefas em atraso em diferentes posições, e em movimentos de trocas aos pares entre todas as tarefas.

A perturbação era gerada pela troca aleatória de $n/10$ pares, sendo n o número de tarefas.

Foram utilizados como critérios de parada o número máximo de iterações e o número máximo de iterações sem melhoria na solução atual.

Foram gerados 20 problemas pequenos, com 10 tarefas e duas máquinas e 180 problemas maiores, com 30, 40 e 50 tarefas, e 3, 4 ou 5 máquinas. O método proposto encontrou a solução ótima em 19 dos problemas pequenos. Para os problemas maiores o método encontrou soluções melhores que as heurísticas iniciais e que um método de busca tabu convencional.

Wang, Li e Wang (2008) utilizaram o ILS em problemas do tipo *no-wait flow shop*, com o objetivo de minimizar o makespan.

Para obter a solução inicial, foi utilizada a heurística NEH (*Nawaz-Enscore-Ham*). Inicialmente são escolhidas as duas tarefas com maior tempo total de execução e colocadas na sequência que minimize o *makespan*. A seguir, adiciona-se a tarefa com maior tempo total na posição que gerar o menor *makespan*, até que todas as tarefas sejam selecionadas.

A busca local era realizada pela inserção, uma a uma, das tarefas em diferentes posições, e pela troca de posição de pares de tarefas entre si.

No mecanismo de perturbação utilizado, um determinado número de tarefas era retirado da sequência aleatoriamente e reinserido, um a um e na mesma ordem, na posição que gerasse o menor *makespan*.

O algoritmo finalizava depois de um determinado número máximo de iterações.

O método proposto foi utilizado em 23 problemas com 20 a 100 tarefas e com 5 a 20 máquinas, e encontrou soluções melhores que as obtidas por um algoritmo híbrido baseado nas técnicas VND e *Particle Swarm Optimization* (PAN; TASGETIREN, LIANG, 2008).

Naderi, Ruiz e Zandieh (2010) empregaram o ILS em problemas do tipo *flow shop* com máquinas paralelas e tempos de *set-up* dependentes da sequência de operações, com o objetivo de minimizar o *makespan.*

A solução inicial era gerada através de um método NEH adaptado, o qual era, segundo os autores, bem adequado para o tipo de problema estudado.

O mecanismo de busca local tinha início com a inserção da tarefa da primeira posição em outra posição escolhida aleatoriamente. Se não houvesse melhora na solução, repetia-se o procedimento com a tarefa da segunda posição, e assim sucessivamente.

A perturbação consistia em selecionar um determinado número de tarefas e inseri-las em diferentes posições, aleatoriamente. Entretanto, ao invés de gerar a perturbação uma única vez, este procedimento era repetido diversas vezes, e a melhor solução gerada era selecionada para a próxima perturbação, mesmo não sendo a melhor solução corrente.

O critério de parada adotado foi o de tempo máximo de execução.

O método foi empregado em problemas com 20 a 150 tarefas, 2 a 8 estágios e 1 a 4 máquinas por estágio; chegando a resultados melhores que os obtidos por outras sete heurísticas e metaheurísticas.

Kampke (2010) utilizou o ILS para minimizar o tempo total de conclusão das tarefas e o número de recursos empregados em problemas de sequenciamento com máquinas paralelas não relacionadas com tempos de preparação dependentes da máquina, da sequência de operação, e do número de recursos utilizados.

Para obter a solução inicial, o autor utilizou o método heurístico construtivo guloso DJASA (*Dynamic Job Assignment with Set-ups Resource Assignment)*, proposto por Ruiz e Andrés (2007). Este método tem início com uma sequência vazia, e, iterativamente, adiciona-se uma tarefa à sequência parcial. Para determinar qual tarefa deve ser adicionada, são feitas simulações de inclusão de todas as tarefas não sequenciadas em cada uma das máquinas. É

selecionada aquela que causar o menor acréscimo no valor da função objetivo. Este processo se repete até que todas as tarefas tenham sido adicionadas à sequência. Para finalizar, aplica-se um procedimento de busca local ao resultado anterior, gerando a solução inicial para o ILS.

O procedimento de busca local utilizado foi o de realocação dos *jobs* para outras posições na sequência de produção em conjunto com o critério "melhor de todos", onde todos os vizinhos da solução atual são testados para definir qual deles é o melhor.

Depois de atingir um ótimo local, o autor aplicou a técnica de reconexão de caminhos (*Path Relinking*) para refinar a solução encontrada. Esta técnica integra estratégias de intensificação, ao proporcionar a geração de novas soluções através da exploração de trajetórias que conectam soluções de alta qualidade; e de diversificação, ao tentar encontrar soluções que estão em diferentes regiões da vizinhança e que não foram exploradas pelo procedimento de busca local.

Seu objetivo é construir iterativamente um caminho de soluções entre a solução ótima local e outra solução, denominada solução guia, por meio de movimentos de troca entre as tarefas da solução de origem, de forma a incluir atributos da solução guia, a cada iteração. Por exemplo, partindo-se do ótimo local 1-2-3-6-4-5, e desejando-se chegar à solução guia 3-2-4-6-1-5, pode-se trocar a tarefa 1 com a 3, e, a seguir, a 1 com a 4: 1-2-3-6-4-5 => 3-2-1-6-4-5 => 3-2-4-6-1-5.

Como solução guia, o autor utilizou as 10 melhores soluções encontradas até o momento da execução.

O procedimento de perturbação utilizado pelo autor consistia de duas fases: uma de destruição, no qual um determinado número de tarefas era removido da sequência original aleatoriamente; e uma de reconstrução, na qual os elementos removidos eram testados, um a um, em todas as posições da sequência atual, e inseridos na posição que gerasse o menor valor para a função objetivo. Foi definido, em seu trabalho, que a solução não poderia sofrer perturbação em mais de 50% de seus elementos.

Para aceitar a nova solução, utilizou-se um critério semelhante ao utilizado pelo método *Simulated Annealing*, mas com temperatura constante (RUIZ; STUTZLE, 2007).

O critério de parada utilizado foi o de tempo limite de execução, baseado no tamanho do problema e dado pela metade do produto do número de máquinas pelo número de tarefas.

O autor resolveu os 720 problemas propostos por Ruiz e Andrés (2007), divididos em 360 problemas pequenos, que possuíam de 6 a 10 tarefas e de 3 a 5 máquinas; e 360 problemas grandes, que continham de 50 a 100 tarefas e de 10 a 20 máquinas.

Para os problemas de pequeno porte, onde a solução ótima era conhecida, o método encontrou o mesmo valor em muitos deles. Para os problemas maiores, em que a solução ótima não era conhecida, o método encontrou soluções superiores ou iguais em relação à melhor solução conhecida.

Com relação ao *Path Relinking*, para os problemas pequenos, sua utilização não gerou soluções melhores do que as obtidas sem ele; já para os problemas maiores, houve uma pequena melhora.

Sidhoum e Sourd (2010) desenvolveram diferentes modelos de ILS para resolver problemas de sequenciamento em uma máquina com datas de entrega e penalizações por antecipação e atraso.

Os autores dividiram a busca local em três fases. Na primeira, havia trocas de posições entre tarefas adjacentes até que não houvesse mais melhoria na solução. Quando as trocas terminavam, tinha início a segunda fase, que consistia em inserir as tarefas, uma a uma em diferentes posições. Na fase final, combinava-se a inserção de uma tarefa com uma ou duas trocas de posição. Sempre que uma nova solução fosse encontrada, reiniciava-se a busca local a partir da primeira fase.

Para realizar a perturbação foram desenvolvidos três modelos. O primeiro consistia em realizar um determinado número de trocas aleatórias entre pares de tarefas. O segundo consistia em modificar a relação entre os custos por atraso e antecipação e realizar uma busca local considerando estes novos valores. O ótimo local desta busca seria a solução perturbada. O último modelo era uma combinação dos dois anteriores, onde a perturbação do primeiro modelo era realizada com uma probabilidade p e a do segundo com uma probabilidade 1-p.

A solução inicial era gerada aleatoriamente para os dois primeiros modelos e em ordem crescente de datas de entrega para o terceiro.

Os modelos foram testados em problemas com 50 a 500 tarefas. Os autores observaram que o primeiro modelo gerava grandes perturbações, enquanto o segundo, pequenas. O terceiro, ao combinar os dois, gerou resultados melhores. Para problemas com 50 tarefas, ele encontrou soluções 0,01% distantes da solução ótima; para os demais, até 5,5%.

Kalili (2012) utilizou o ILS para minimizar o *total weighted completion time* em um *flow shop* com máquinas paralelas e tempos de *set-up* dependentes da sequência.

Para gerar a solução inicial, o autor utilizou a heurística NEH.

A busca local consistia em selecionar e inserir as tarefas, uma a uma, da primeira até a última posição, até que uma solução melhor fosse encontrada.

Para gerar a perturbação, um determinado número de tarefas era retirado da sequência atual e recolocado na posição que resultasse no menor valor da função-objetivo.

O critério de aceitação da solução era baseado naquele utilizado por um método S*imulated Annealing* com temperatura constante. Um tempo máximo de execução foi escolhido como critério de parada.

Foram resolvidos problemas com 20 a 120 tarefas com até 8 estágios. Os resultados obtidos pelo ILS foram superiores aos encontrados por outras cinco heurísticas e metaheurísticas (*shortest processing time, flow time multiple insertion heuristic*, (g/2, g/2) *Johnson's rule*, NEH, e algoritmos genéticos).

Pan e Ruiz (2012) aplicaram o ILS em problemas de *flow shop* para minimizar o tempo de permanência das tarefas no sistema (tempo de fluxo).

Para gerar a solução inicial, foi utilizada a heurística LR(x), proposta por Liu e Reeves (2001), e considerada adequada para este tipo de problema.

O procedimento de busca local consistia em testar cada uma das tarefas em todas as posições possíveis e inseri-la naquela que retornasse o melhor resultado. Este procedimento era repetido até que um mínimo local fosse encontrado.

A perturbação consistia em selecionar uma tarefa aleatoriamente e reinseri-la em outra posição, também escolhida aleatoriamente. Este processo era repetido algumas vezes antes de realizar a busca local novamente.

O critério de aceitação da solução era similar ao utilizado por um método *simulated annealing* com temperatura constante.

O algoritmo era executado até que um tempo máximo fosse atingido.

Foram analisados 120 problemas com 20 a 500 tarefas e com 5 a 20 máquinas. O método proposto encontrou soluções melhores que as encontradas por outras 12 metaheurísticas (baseadas em algoritmos evolutivos, algoritmos genéticos, *iterated local search, variable neighborhood search*, colônia de abelhas) desenvolvidas para este tipo de problema, nos últimos anos.

Lin, Lin e Fang (2013) tentaram minimizar a soma do tempo de conclusão das tarefas em um *flow shop* com duas máquinas, onde os produtos da segunda máquina não poderiam ter início enquanto todos os seus componentes, produzidos na primeira máquina, não estivessem concluídos.

Para resolver o problema, os autores propuseram um método que utilizava uma heurística construtiva, seguida da aplicação do ILS.

A heurística construtiva selecionava, iterativamente, as tarefas que gerassem o menor *makespan* ao serem adicionadas na sequência de execução.

Após a inclusão da última tarefa, obtinha-se a solução inicial para o ILS.

O método de busca local utilizado consistia em inserir uma a uma, todas as tarefas em todas as posições possíveis. Durante este processo, se fosse encontrada uma solução melhor que a atual, a busca local era reiniciada a partir desta solução.

O mecanismo de perturbação utilizado consistia em trocar a posição de duas tarefas escolhidas aleatoriamente.

Após a perturbação, era realizada uma nova busca local, até que um mínimo local fosse encontrado. Esta solução era aceita somente se fosse melhor que a melhor solução corrente.

O método terminava depois de 200 iterações.

Foram resolvidos problemas com combinações de 10 a 40 tarefas (n) realizadas na segunda máquina e com 0,25*n*, 0,5*n* e *n* tarefas realizadas na primeira máquina, e alguns problemas com até 200 tarefas. Para os problemas menores, o ILS encontrou a solução ótima em alguns casos, e soluções próximas à ótima nos demais (0,1% em média); para os problemas maiores, o método encontrou soluções melhores e em menos tempo do que aquelas obtidas por um algoritmo *branch-and-bound* executado por 30 minutos.

O Quadro 2 apresenta um resumo com os principais mecanismos utilizados nos trabalhos citados nesta seção para gerar uma solução inicial, para realizar uma busca local, e uma perturbação.

Como pode ser visto no Quadro 2, foram utilizadas diversas heurísticas para gerar as soluções iniciais dos algoritmos ILS, havendo uma maior ocorrência da heurística NEH. Em alguns trabalhos optou-se por utilizar a metaheurística GRASP.

Com relação à busca local, houve uma predominância de movimentos simples de trocas e inserções. Como alternativas, apareceram alguns mecanismos de trocas cíclicas entre máquinas e a técnica de *Path Relinking.*

Já para realizar as perturbações, optou-se, em geral, por realizar diversas trocas e inserções de tarefas. Foram usadas, também, algumas técnicas de destruição e reconstrução das soluções vigentes. Deve-se observar também que alguns autores utilizaram uma estrutura de vizinhança diferente daquelas utilizadas durante a busca local para realizar as perturbações, enquanto outros utilizaram um dos movimentos da busca local repetidas vezes.

Quadro 2 – Metaheurística ILS: solução inicial, busca local e perturbação

Ano	Autores	Solução inicial	Busca local	Perturbação
2006	Tang e Luo	- heurística LPT.	- trocas cíclicas entre tarefas de máquinas distintas.	- duas trocas cíclicas entre tarefas aleatoriamente.
2007	Gomes Júnior	- GRASP.	- trocas de posições entre tarefas; - inserção de tarefas em diferentes posições.	- trocas aleatórias entre duas tarefas quaisquer.
2008	Nunes e Arroyo	- GRASP.	- trocas de posições entre tarefas; - inserção de tarefas em diferentes posições.	- duas trocas entre tarefas.
2008	Chen	- sequência aleatória; - heurística SWPT; - heurística EWDD; - heurística SRT.	- trocas de tarefas aos pares; - inserção de tarefas em diferentes posições.	- troca aleatória de n/10 pares.
2008	Wang, Li e Wang	- heurística NEH.	- inserção de tarefas em diferentes posições; - trocas de posições entre tarefas.	- destruição e reconstrução.
2010	Naderi, Ruiz e Zandieh	- heurística NEH.	- inserção de tarefas em diferentes posições.	- inserção de tarefas repetidas vezes.
2010	Kampke	- heurística DJASA.	- inserção de tarefas em diferentes posições, seguido de *Path Relinking*.	- destruição e reconstrução.
2010	Sidhoum e Sourd	- aleatoria; - ordem crescente de datas de entrega.	- trocas de posições entre tarefas; - inserção de tarefas em diferentes posições.	- trocas aleatórias entre pares de tarefas; - modificação da relação entre os custos por atraso e antecipação.
2012	Kalili	- heurística NEH.	- inserção de tarefas em diferentes posições.	- destruição e reconstrução.
2012	Pan e Ruiz	- heurística LR(x).	- inserção de tarefas em diferentes posições.	- inserção de tarefas repetidas vezes.
2013	Lin, Lin e Fang	- heurística construtiva orientada pelo menor *makespan*.	- inserção de tarefas em diferentes posições.	- trocas aleatórias entre duas tarefas.

2.2.3 Método VNS

A busca em vizinhança variável (*Variable Neighborhood Search* - VNS) é uma metaheurística caracterizada por uma mudança sistemática de estruturas de vizinhança, que tem como intuito escapar dos ótimos locais (MENDES, 2007).

Este método parte do princípio de que um ótimo local para uma estrutura de vizinhança não é, necessariamente, um ótimo local para outra estrutura de vizinhança. Desta forma, ao se utilizarem vários tipos de vizinhanças simultaneamente, aumenta-se a chance de se encontrar o ótimo global (HANSEN; MLADENOVIC, 2003; MARTINS, 2009).

Melo (2010) adverte, entretanto, que se as estruturas de vizinhança escolhidas não forem apropriadas para o problema, a qualidade da solução encontrada pode não ser satisfatória.

Existem diversas formas de se implementar o VNS.

No método VND (*Variable Neighborhood Descent*), deve-se, inicialmente, determinar as estruturas de vizinhança que serão utilizadas, e encontrar uma solução inicial para o problema. A seguir, escolhe-se a primeira estrutura de vizinhança e realiza-se uma busca local, até que um ótimo local seja encontrado.

Se a solução encontrada for melhor que a solução atual, realiza-se uma nova busca local, partindo-se desta nova solução, e utilizando a primeira estrutura de vizinhança. Caso contrário, seleciona-se uma nova vizinhança e repete-se a busca local.

Este processo é repetido até que não haja mais melhora na solução para nenhuma das vizinhanças. A solução final encontrada por este método é um ótimo local comum a todas as vizinhanças utilizadas. A Figura 6 apresenta o pseudocódigo deste método.

Figura 6 – Pseudocódigo VND

```
01: Definir k estruturas de vizinhança
02: Entrada: Solução viável S
03: S* = S;
04: Repita k = 1
05:    S*' <= Busca Local (S*);
06:    se c (S*') < c (S*) então
07:       S* = S*';
08:       k = 1;
09:    fim se
10:    senão k = k + 1
11: Até k_máx
12: Retornar S*
```

Fonte: (HANSEN; MLADENOVIC, 2003)

Para problemas de grande porte, onde a busca local pode ser custosa, Hansen e Mladenovic (2003) sugerem a utilização do método RVNS (*Reduced Variable Neighborhood Search*).

Neste método, depois de se determinar as estruturas de vizinhança que serão utilizadas, e de se encontrar uma solução inicial para o problema, gera-se, aleatoriamente, um vizinho para esta solução. Se este ponto for melhor que a solução atual, ele a substitui, e repete-se o procedimento, a partir da primeira estrutura de vizinhança. Caso contrário, gera-se outro vizinho aleatoriamente, utilizando outra estrutura de vizinhança.

O método termina depois de atingir algum critério de parada, como, por exemplo, o número máximo de iterações sem melhora na solução.

A Figura 7 apresenta o pseudocódigo deste método.

Figura 7 – Pseudocódigo RVNS

```
01: Definir k estruturas de vizinhança
02: Entrada: Solução viável S
03: S* = S;
04: Repita
05:    Repita k = 1;
06:       S' <= Vizinho (S*);
07:       se c (S') < c (S*) então
08:          S* = S';
09:          k = 1;
10:       fim se
11:       senão k = k + 1
12:    Até k_máx
13: Até atingir critério de parada
14: Retornar S*
```

Fonte: (HANSEN; MLADENOVIC, 2003)

Um terceiro método apresentado por esses autores é o *General VNS*, que é uma combinação dos dois anteriores. De acordo com Martins (2009), o GVNS utiliza o VND para encontrar ótimos locais de qualidade, e o RVNS para encontrar novas regiões promissoras a partir de um ótimo local, o que resulta num método de excelente desempenho.

Inicialmente, encontra-se uma solução inicial para o problema, determinam-se as estruturas de vizinhança que serão utilizadas, e um critério de parada, como tempo máximo de execução, número máximo de iterações ou número máximo de iterações sem melhora na solução.

A seguir, gera-se, aleatoriamente, um vizinho para esta solução, utilizando a primeira estrutura de vizinhança, e realiza-se uma busca local, a partir deste vizinho, até que um ótimo local seja encontrado. Se este ótimo local for melhor que a solução atual, ele a substitui, e repete-se o procedimento, a partir da primeira estrutura de vizinhança. Caso contrário, gera-se outro vizinho aleatoriamente, utilizando outra estrutura de vizinhança, e realiza-se uma nova busca local a partir deste ponto.

O processo é repetido, até que o critério de parada seja atingido. A Figura 8 apresenta o pseudocódigo deste método.

Figura 8 – Pseudocódigo GVNS

```
01: Definir k estruturas de vizinhança
02: Entrada: Solução viável S
03: S* = S;
04: Repita
05:    Repita k = 1;
06:       S' <= Vizinho (S*);
07:       S*' <= Busca Local - VND(S');
07:       se c (S*') < c (S*) então
08:          S* = S*';
09:          k = 1;
10:       fim se
11:       senão k = k + 1
12:    Até k_máx
13: Até atingir critério de parada
14: Retornar S*
```

Fonte: (HANSEN; MLADENOVIC, 2003)

O VNS vem sendo aplicado com sucesso em diversas áreas, tais como: problemas de sequenciamento da produção, caixeiro viajante, localização de facilidades, roteamento de veículos, redes, etc.

Paula *et al*. (2007) e Paula, Ravetti e Pardalos (2006) criaram alguns algoritmos VNS para resolver problemas de sequenciamento de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência e da máquina, com o objetivo de minimizar a soma do tempo de conclusão e dos atrasos ponderados.

Os autores utilizaram três vizinhanças: trocas de tarefas numa mesma máquina, trocas de tarefas entre duas máquinas diferentes, e transferências de tarefas de uma máquina para outra.

Para gerar a solução inicial, foi criado um algoritmo NEH.

No VNS desenvolvido, gerava-se um vizinho aleatoriamente e, a seguir, realizava-se uma busca local. Se a solução encontrada fosse melhor que a atual, esta era substituída, e gerava-se um novo vizinho utilizando a primeira vizinhança. Caso contrário, gerava-se um novo vizinho utilizando outra vizinhança. O processo terminava após um determinado número de iterações.

Havia três processos para gerar os vizinhos da solução atual. No primeiro, escolhia-se, aleatoriamente, uma máquina e duas tarefas desta máquina, e realizava-se a troca de posição destas duas entre si. No segundo, escolhiam-se duas máquinas e uma tarefa de cada máquina, aleatoriamente, e trocava-se uma tarefa pela outra. No terceiro, escolhia-se uma tarefa de uma máquina e uma posição de outra máquina, e transferia-se essa tarefa para esta posição.

Para as buscas locais, realizavam-se todas as trocas entre pares de tarefas para cada máquina, para a primeira vizinhança; para a segunda, para cada par de máquinas, eram avaliadas todas as trocas possíveis entre as tarefas, duas a duas; e para a terceira, todas as transferências possíveis entre as máquinas com maior e menor *makespan*.

Foram resolvidos problemas com 10 a 150 tarefas, e adotou-se como critério de parada, um número máximo de 10.000 iterações.

Os resultados foram comparados com os obtidos por alguns modelos GRASP. O VNS encontrou soluções equivalentes para problemas com até 60 tarefas, e melhores para problemas maiores.

Paula (2008) utilizou o VNS para minimizar o atraso em problemas de sequenciamento de máquinas paralelas com tempos de preparação dependentes da sequência.

A solução inicial era gerada através de uma heurística NEH.

A seguir, gerava-se um vizinho aleatoriamente, através da inserção de uma tarefa numa posição qualquer; e realizava-se uma busca local.

Se a solução encontrada fosse melhor que a atual, retornava-se para a primeira vizinhança. Caso contrário, repetia-se o processo com uma nova estrutura de vizinhança.

As buscas locais e as estruturas de vizinhança utilizadas empregavam movimentos de troca entre duas tarefas, movimentos de inserção de uma tarefa em diferentes posições, e os dois movimentos anteriores simultaneamente.

O critério de parada escolhido foi o de número máximo de iterações sem melhora na solução.

De acordo com o autor, o modelo proposto se mostrou competitivo tanto em termos de qualidade da solução, quanto em termos de tempo de processamento para problemas com até 300 tarefas e 6 máquinas.

Koshanaei et al. (2009) utilizaram o VNS para resolver problemas de sequenciamento em *job shops* com tempos de *set-up* dependentes da sequência, com o objetivo de minimizar o *makespan*.

Para gerar a solução inicial, foi utilizada a heurística SPT (*Shortest Processing Time*).

Os autores criaram um algoritmo VND com três estruturas de vizinhança. Na primeira delas, uma operação era removida aleatoriamente e inserida numa posição também aleatoriamente. Todas as operações eram testadas, uma a uma. A busca era reiniciada sempre que uma solução melhor fosse encontrada. Analogamente, a segunda estrutura de vizinhança retirava e reinseria duas operações, e a terceira estrutura, três.

Sempre que fosse encontrada uma solução melhor, retornava-se para a primeira estrutura. Além disso, mesmo que não houvesse melhoria após o término da terceira estrutura, aceitava-se a melhor solução encontrada por essa vizinhança e reiniciava-se a busca com a primeira vizinhança.

O critério de parada escolhido foi o tempo máximo de execução.

Foram analisados 320 problemas com 15 tarefas e 15 estágios até 100 tarefas e 20 estágios e comparados com três algoritmos genéticos e uma heurística SPT. O VNS superou todos os algoritmos anteriores em todos os problemas analisados.

Liang, Chen e Tien (2009) utilizaram o VNS em problemas de sequenciamento com máquinas paralelas idênticas, com o objetivo de minimizar o *makespan* e o atraso total.

Os autores utilizaram duas estruturas de vizinhança: inserção de tarefas em posições diferentes e troca de tarefas entre máquinas.

Depois de se obter uma solução inicial, escolhia-se uma estrutura de vizinhança aleatoriamente, gerava-se um vizinho e realizava-se uma busca local utilizando a vizinhança escolhida. Este processo era repetido até que um tempo máximo de execução fosse atingido.

Foram analisados problemas com 35, 50 e 65 tarefas e 7, 10 e 13 máquinas, respectivamente. Os resultados foram comparados com duas metaheurísticas baseadas em algoritmos genéticos. O VNS proposto encontrou soluções melhores e em menos tempo na maioria dos casos analisados.

Driessel e Mönch (2009; 2011) aplicaram o VNS em problemas de seqüenciamento em *job shops* com máquinas paralelas, tempos de *set-up* dependentes da sequência e restrições de precedência, com o objetivo de minimizar o atraso total.

Para gerar a solução inicial, os autores utilizaram a heurística ATCSR (*Apparent Tardiness Cost with Set-ups and Ready times* – PFUND *et al*., 2008).

Os autores definiram quatro estruturas de vizinhança básicas: transferência de tarefa entre máquinas, onde uma tarefa de uma máquina era escolhida e transferida para todas as demais máquinas; troca de tarefas entre máquinas, em que uma tarefa de uma máquina era selecionada e trocada pelas tarefas das outras máquinas; transferência de tarefas em uma mesma máquina, na qual uma tarefa de uma máquina era reinserida em outras posições da mesma máquina; e trocas entre tarefas numa mesma máquina, onde uma tarefa trocava de posição com outra, numa mesma máquina.

Além dessas quatro estruturas, os autores criaram outras oito, combinando as anteriores entre si.

No algoritmo proposto, após encontrar uma solução inicial, gerava-se um vizinho aleatoriamente, de acordo com a estrutura de vizinhança selecionada, e realizava-se uma busca local, utilizando esta vizinhança, até que o melhor vizinho fosse encontrado. Se a solução fosse melhor que a atual, o processo era repetido a partir da primeira vizinhança; caso contrário, utilizava-se uma nova vizinhança. O processo era repetido até que um tempo máximo fosse atingido.

Foram analisados problemas com 5 máquinas e 55, 95 e 135 tarefas. Os resultados foram comparados com outros obtidos pelas heurísticas ATCSR e FIFO (*First In, First Out*). Os resultados encontrados pelo VNS foram superiores.

O Quadro 3 resume as formas utilizadas nos trabalhos citados nesta seção para gerar uma solução inicial, para gerar um novo vizinho, e para realizar uma busca local.

Quadro 3 – Metaheurística VNS: solução inicial, geração de vizinhos e busca local

Ano	Autores	Solução inicial	Geração de vizinhos	Busca local
2006	Paula, Ravetti e Pardalos	- heurística NEH	- trocas de tarefas; - transferências de tarefas para outras máquinas.	- trocas de tarefas; - transferências de tarefas para outras máquinas.
2007	Paula	- heurística NEH	- trocas de tarefas; - transferências de tarefas para outras máquinas.	- trocas de tarefas; - transferências de tarefas para outras máquinas.
2008	Paula	- heurística NEH	- inserção de uma tarefa numa posição qualquer.	- trocas de tarefas; - inserção de tarefas numa nova posição.
2009	Koshanaei et al.	- heurística SPT	- inserção de uma tarefa numa posição qualquer.	- inserção de tarefas numa posição qualquer.

Quadro 3 – Metaheurística VNS: solução inicial, geração de vizinhos e busca local

Ano	Autores	Solução inicial	Geração de vizinhos	Busca local
2009	Liang, Chen e Tien	- não informado	- inserção de tarefas em posições diferentes; - troca de tarefas entre máquinas.	- inserção de tarefas em posições diferentes; - troca de tarefas entre máquinas.
2009	Driessel e Mönch	- heurística ATCSR	- inserção de tarefas em posições diferentes; - troca de tarefas entre máquinas.	- inserção de tarefas em posições diferentes - troca de tarefas entre máquinas
2011	Driessel e Mönch	- heurística ATCSR	- inserção de tarefas em posições diferentes; - troca de tarefas entre máquinas.	- inserção de tarefas em posições diferentes - troca de tarefas entre máquinas

A partir do Quadro 3, pode-se ver que os autores optaram por usar diferentes heurísticas para gerar as soluções iniciais dos algoritmos VNS, sendo que as heurísticas NEH e ATCSR foram as mais utilizadas. Com relação à busca local e à geração de vizinhos, houve uma predominância de movimentos de trocas e inserções.

2.2.4 Método LNS

A metaheurística LNS (*Large Neighborhood Search*) é um método iterativo constituído por uma fase de destruição e por outra de reconstrução da solução corrente (PISINGER; ROPKE, 2010).

O método tem início a partir de uma solução viável qualquer.

Na fase de destruição, parte dos elementos da solução atual é removida, normalmente de forma aleatória. O principal parâmetro a ser definido nesta fase é o grau de destruição da solução. Se apenas uma pequena parte da solução for destruída, o algoritmo pode ter dificuldades para explorar o espaço de soluções. Por outro lado, se grande parte da solução for destruída, serão geradas muitas soluções de baixa qualidade e haverá um grande consumo de tempo.

Na fase de reconstrução, os elementos removidos na etapa anterior são reinseridos em novas posições, até que uma nova solução seja obtida. Nesta fase, pode ser utilizada uma heurística gulosa, ou até mesmo um algoritmo de otimização. Neste último caso, o LNS pode ser visto, segundo Ropke e Pisinger (2006), como um método *fix-optimize*.

Desta maneira, a vizinhança de uma solução pode ser definida, neste método, como o conjunto de soluções que pode ser obtido depois de se destruir e reconstruir uma dada solução.

A função de aceitação pode seguir diversos critérios. Em geral, uma solução é aceita somente se for melhor que a melhor conhecida até o momento.

O método termina depois de atingir algum critério de parada, como, por exemplo, um número máximo de iterações ou um tempo máximo de execução.

O pseudocódigo deste método é exibido na Figura 9, onde x é a solução atual, x_m é a melhor solução obtida, x_n é a nova solução, d( ) é a fase de destruição, r( ) a de reconstrução, c( ) a função-objetivo.

Figura 9 – Pseudocódigo LNS

```
01:  Entrada: Solução viável x
02:  x_m = x;
03:  Repita
04:     x_n = r(d(x));
05:     se aceitação(x_n , x) então
06:        x = x_n;
07:     fim se
08:     se c(x_n) < c(x_m) então
09:        x_m = x_n;
10:     fim se
11:  Até atingir critério de parada
12:  Retornar x_m
```

Fonte: (PISINGER; ROPKE, 2010)

É importante observar que o LNS não avalia toda a vizinhança, mas apenas uma fração dela, o que permite que o método possa navegar pelo universo de soluções facilmente.

Uma extensão do LNS é o ALNS (*Adaptive large neighborhood search*), que permite o uso de diferentes métodos de destruição e reconstrução a cada iteração.

Cada um desses métodos de destruição e reconstrução tem uma probabilidade de ser escolhido a cada iteração, a qual é atualizada dinamicamente, com base no desempenho alcançado por cada uma ao longo do tempo.

O pseudocódigo deste método é exibido na Figura 10.

Figura 10 – Pseudocódigo ALNS

```
01:  Entrada: Solução viável x
02:  x_m = x;
03:  Repita
04:     Selecionar método de destruição e de reconstrução
05:     x_n = r(d(x));
06:     se aceitação(x_n , x) então
07:        x = x_n;
08:     fim se
09:     se c(x_n) < c(x_m) então
10:        x_m = x_n;
11:     fim se
12:     Atualizar probabilidades de escolha de cada método
13:  Até atingir critério de parada
14:  Retornar x_m
```

Fonte: (PISINGER; ROPKE, 2010)

De acordo com Pisinger e Ropke (2010), o LNS e o ALNS têm sido aplicados com sucesso em diversos problemas de transporte e de sequenciamento.

Sourd (2001) utilizou o LNS em problemas de sequenciamento com máquinas paralelas não-relacionadas, com o objetivo de minimizar o *makespan*.

Na fase de destruição, duas máquinas eram selecionadas aleatoriamente e todas as tarefas que estavam nessas máquinas eram removidas.

Na fase de construção, as tarefas excluídas eram direcionadas para as máquinas com menor tempo de processamento e menor tempo de conclusão das tarefas nela alocadas.

Foram resolvidos problemas com 100 tarefas e 5 a 20 máquinas, e com 500 tarefas e 10 a 100 máquinas, gerando, segundo o autor, bons resultados.

Carchrae e Beck (2009) apresentaram um método LNS para resolver problemas de sequenciamento em ambientes *job shop* com o objetivo de minimizar o *makespan*.

Na fase de destruição foram apresentados três tipos distintos de vizinhança. No primeiro foi utilizada uma janela de tempo para selecionar as tarefas que seriam removidas da solução corrente. No segundo, eram selecionadas todas as tarefas que estivessem nas máquinas selecionadas de acordo com um critério que levava em consideração a ocupação de cada máquina. No terceiro, as tarefas eram selecionadas aleatoriamente.

Na fase de reconstrução, foi utilizada uma heurística construtiva direcionada pelas restrições do problema.

Os autores também combinaram as vizinhanças anteriores, gerando algoritmos ALNS e aplicaram os modelos criados em problemas com 20 tarefas e 20 máquinas e 40 tarefas e 40

máquinas. Todos os métodos geraram bons resultados. Comparando os algoritmos do tipo LNS, nenhum deles predominou sobre os demais. O desempenho de cada um variou de acordo com o tempo de execução e as dimensões do problema. Os algoritmos ALNS geraram resultados tão bons ou até melhores que os obtidos pelos algoritmos LNS.

Pacino e Hentenryck (2011) aplicaram o LNS num ambiente *job shop* flexível, com o objetivo de minimizar o *makespan*.

Em seu trabalho, um conjunto de variáveis era escolhido para ser relaxado, enquanto as demais variáveis permaneciam fixas. A seguir, era realizada uma otimização.

Os autores utilizaram três estratégias para escolher as tarefas que seriam relaxadas. Na primeira, uma seleção aleatória; na segunda, escolhia-se aleatoriamente uma janela de tempo e todas as tarefas que estivessem nesse intervalo seriam relaxadas; na terceira, escolhiam-se algumas máquinas, e todas as tarefas que estivessem nessas máquinas seriam relaxadas.

O modelo foi testado em 20 problemas encontrados na literatura (*eData set of flexible shop instances* – HURINK; JURISCH; THOLE, 1994). Após uma hora de execução, ele foi capaz de alcançar todos os melhores resultados conhecidos, e de encontrar soluções melhores em mais de 50% dos problemas.

O Quadro 4 apresenta as técnicas utilizadas durante as fases de destruição e reconstrução utilizadas por estes autores.

Quadro 4 – Metaheurística LNS: destruição e reconstrução

Ano	Autores	Destruição	Reconstrução
2001	Sourd	- remoção de todas as tarefas de duas máquinas	- tarefas excluídas eram direcionadas para as máquinas com menor tempo de processamento e menor tempo de conclusão das tarefas nela alocadas.
2009	Carchrae e Beck	- janelas de tempo para selecionar as tarefas que seriam removidas; - todas as tarefas que estivessem nas máquinas selecionadas de acordo com a ocupação de cada uma; - seleção aleatória.	- heurística construtiva direcionada pelas restrições do problema.
2011	Pacino e Hentenryck	- seleção aleatória; - todas as tarefas que estivessem numa janela de tempo aleatória; - todas as tarefas que estivessem em máquinas escolhidas aleatoriamente.	- algoritmo de otimização.

O Quadro 4 mostra que as técnicas utilizadas na fase de destruição do LNS se basearam numa escolha aleatória de tarefas, numa janela de tempo, ou na escolha de alguma máquina. Já na fase de destruição, houve o emprego de heurísticas e de algoritmos de otimização.

2.2.5 Método *fix-and-optimize*

A heurística *fix-and-optimize* é um método iterativo de melhoria que utiliza a fixação dos conjuntos de variáveis em seus valores inteiros para gerar subproblemas resultantes menores, e mais simples de serem resolvidos (FURLAN, 2011).

Neste método, as variáveis inteiras são divididas em vários conjuntos disjuntos. A cada iteração, apenas as variáveis de um dos conjuntos são escolhidas para serem otimizadas, enquanto todas as outras são fixadas; dessa forma, reduz-se consideravelmente o número de variáveis inteiras, tornando mais fácil a resolução exata do problema. Em seguida, resolve-se o submodelo gerado, e se a solução encontrada for melhor que a solução corrente, esta é substituída. Na iteração seguinte, outro conjunto de variáveis (partição) é escolhido para ficar livre para otimização, e todas as demais variáveis são fixadas.

O processo é repetido até que a solução de todos os subproblemas não gere nenhuma solução melhor que a atual.

Deve-se ressaltar que para utilizar este método é necessário conhecer uma solução viável.

Helber e Sahling (2010) destacam a obtenção de uma solução viável a cada iteração como a grande vantagem deste método.

Estas heurísticas vêm sendo amplamente utilizada em problemas de planejamento da produção (FURLAN, 2011).

Assim como no método relax-and-fix, costuma-se dividir as variáveis de acordo com os tipos ou características dos produtos, e das máquinas envolvidas nos problemas ou em horizontes de tempo. Outra estratégia comum consiste em escolher aleatoriamente as variáveis e/ou a quantidade de variáveis que devem ser otimizadas a cada iteração (FURLAN, 2011; HELBER; SAHLING, 2010; LANG, SHEN, 2011).

3. APRESENTAÇÃO DO PROBLEMA

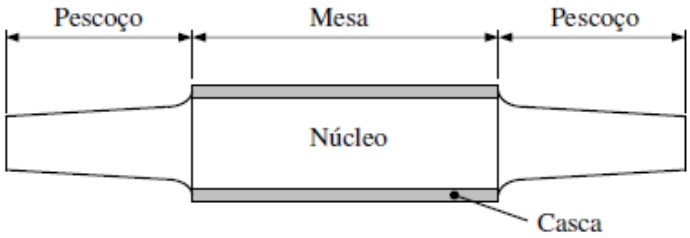
Na Seção 2.1 foram apresentados diversos tipos de problemas de sequenciamento encontrados na literatura: problemas com tempos de preparação, interrupções, bloqueios, quebras de máquinas, restrições de precedência, etc. Entretanto, há uma característica presente em certos ambientes de produção que ainda não foi estudada: o sincronismo de execução de tarefas.

3.1 PROCESSO DE FABRICAÇÃO DE CILINDROS DE LAMINAÇÃO FUNDIDOS

Cilindros de laminação são ferramentas utilizadas no processo de conformação mecânica denominada laminação, que consiste na passagem de um material entre dois cilindros que giram sentidos opostos com a mesma velocidade periférica. (XAVIER, 2010).

Os cilindros de laminação destinados à produção de placas em laminadores de tiras a quente são, em geral, produtos bi-metálicos constituídos de uma casca externa, resistente ao desgaste; e de um núcleo interno de ferro fundido nodular, resistente ao impacto (Figura 11).

Figura 11 - Esquema de um cilindro de laminação fundido



Fonte: (CORNÉLIO, 2006).

O processo de fabricação destes produtos tem início em fornos de fusão por indução. Em geral, neste ambiente de produção, existem vários fornos em paralelo, cada um com um tamanho, e com uma capacidade específica, o que faz com que nem todos os cilindros possam ser produzidos em todos os fornos.

Em função do princípio de funcionamento destes equipamentos, e dos diferentes tipos de materiais que são produzidos, um cilindro pode aproveitar uma parcela do residual metálico deixado no forno pelo cilindro produzido imediatamente anterior a ele. Este fato,

somado às diferentes taxas de fusão de cada máquina, faz com que o tempo de processamento de cada produto dependa do equipamento em que será produzido e de sua sequência de processamento.

Vale ressaltar ainda que, por serem constituídos de materiais com diferentes composições químicas, a casca e o núcleo de um cilindro não podem ser processados, ao mesmo tempo, num mesmo equipamento.

Depois de fundidos, os dois metais são transferidos para um molde de areia.

Por se tratar de um processo que envolve metal líquido, não é possível fundir um dos elementos que constituem um cilindro (casca ou núcleo) primeiro, retirá-lo do forno e deixá-lo aguardando até que o outro elemento fique pronto. As duas partes devem ficar prontas ao mesmo tempo. Se uma delas ficar pronta antes, ela deverá permanecer no forno em que estiver até que a outra parte fique pronta também. Esta é a particularidade que torna este problema diferente dos demais problemas de sequenciamento, e que dificulta ainda mais a obtenção de uma solução ótima.

Para finalizar a apresentação do processo de fabricação de cilindros de laminação fundidos, deve-se mencionar ainda que, além dos cilindros constituídos por dois materiais, existem também cilindros fundidos destinados a outras aplicações, que são constituídos por apenas um metal (diferente dos materiais das cascas e dos núcleos), e que utilizam os mesmos fornos de indução dos cilindros bi-metálicos.

3.2 REPRESENTAÇÃO DE UMA SOLUÇÃO

A geração de uma solução para este problema consiste em se determinar o equipamento em que cada tarefa (casca, núcleo ou cilindro mono-metálico) será processada, e a sequência de processamento de cada uma delas ao longo das máquinas.

Uma forma possível de se representar esta solução é por meio de uma matriz, onde cada coluna representa um forno, e cada linha, a ordem de processamento das tarefas. Dessa forma, a tarefa que estiver na primeira linha (Posição 1) da primeira coluna será a primeira a ser processada no forno de número 1; a tarefa que estiver na segunda linha (Posição 2) da primeira coluna será a segunda a ser processada no forno de número 1; e assim sucessivamente.

A Figura 12 mostra um exemplo, onde as tarefas 7 e 5 serão processadas, nesta ordem, no forno 1; as tarefas 1 e 4, no forno 2; e as tarefas 2, 8, 3 e 6, no forno 3.

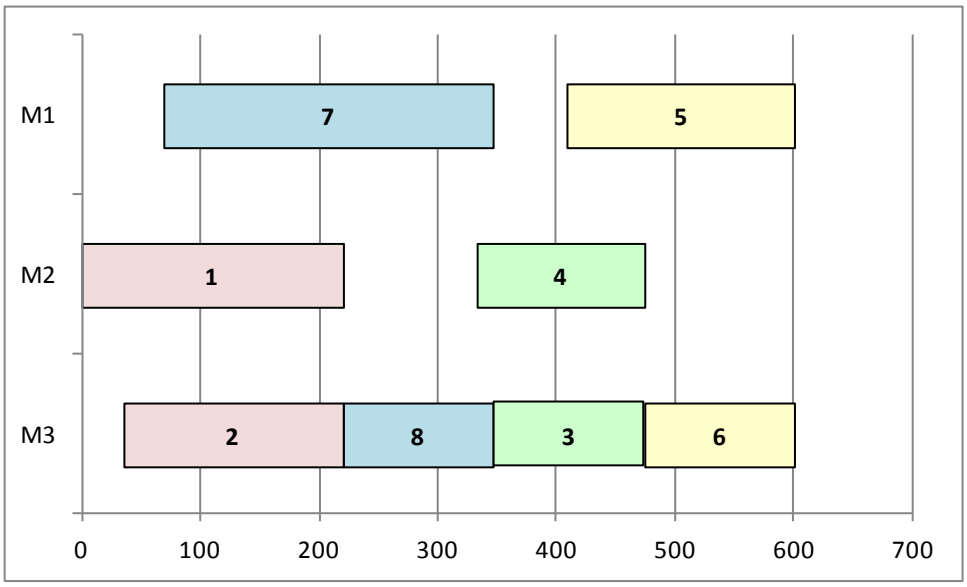
Figura 12 – Representação de uma solução

	Forno 1	Forno 2	Forno 3
Posição 1	7	1	2
Posição 2	5	4	8
Posição 3			3
Posição 4			6

Fonte: (Autor)

Outra forma de se representar uma solução é por meio de um gráfico de Gantt (Figura 13).

Figura 13 – Representação de uma solução – Gráfico de Gantt



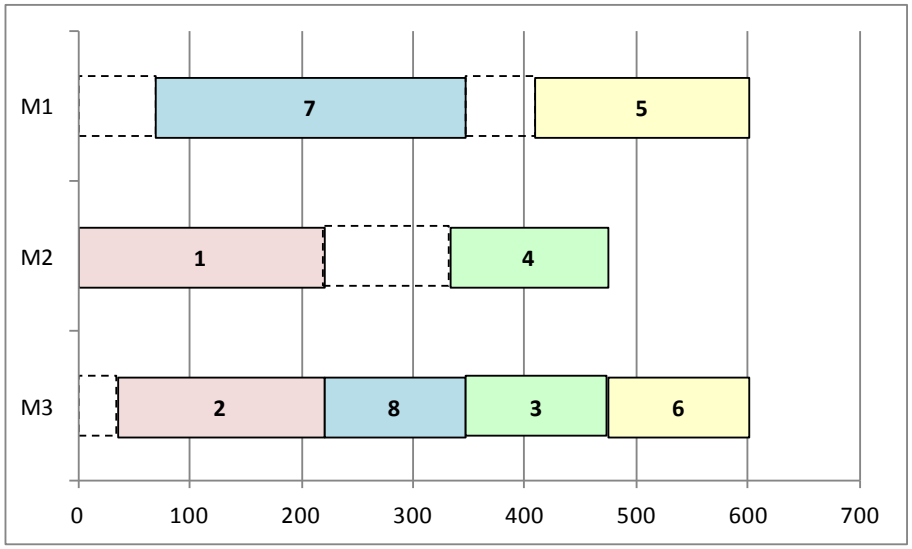
Fonte: (Autor)

3.3 CONSIDERAÇÕES SOBRE OS PROBLEMAS DE SEQUENCIAMENTO COM SINCRONISMO DE EXECUÇÃO DE TAREFAS

O problema de sequenciamento com sincronismo de execução de tarefas ocorre somente em sistemas que possuem máquinas paralelas, e ao menos um par de tarefas que deve ser concluído no mesmo instante.

O sincronismo de execução de tarefas pode levar à aparição de tempos mortos nas máquinas, os quais estão representados na Figura 14, por meio de linhas tracejadas.

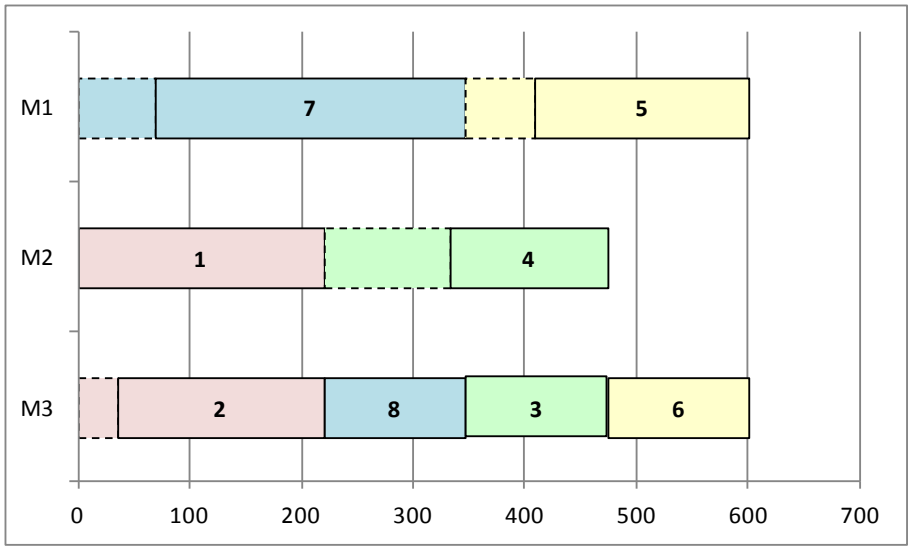
Figura 14 – Tempos mortos



Fonte: (Autor)

Analisando a solução ótima representada no exemplo da Figura 14, onde os pares são formados pelas tarefas 1-2, 3-4, 5-6 e 7-8, percebe-se que o tempo de processamento da tarefa 1 é maior que o da tarefa 2 e que há um espaço livre, na máquina 3, entre o instante zero e o início da operação da tarefa 2. Como não é possível encaixar nenhuma outra tarefa neste vão, a máquina 3 ficará ociosa neste período. O único aspecto positivo desta constatação é que este tempo morto pode servir como uma margem de segurança para o processamento da tarefa 2, pois o início de seu processamento pode ser antecipado, e o seu tempo de processamento alongado, conforme Figura 15. O mesmo ocorre com as tarefas 4, 5 e 7.

Figura 15 – Margem de segurança das tarefas 2, 4, 5 e 7



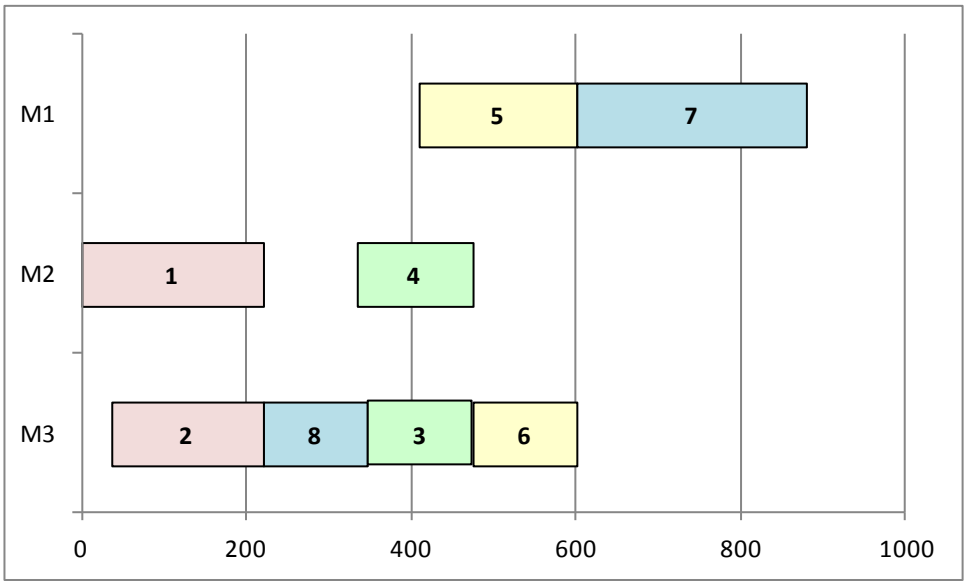
Fonte: (Autor)

Deve-se ressaltar, entretanto, que a tarefa 8 não teria seu início antecipado, caso a tarefa 2 iniciasse seu processamento durante o período em que a máquina 3 está ociosa, pois o sincronismo exige que a tarefa 2 termine seu processamento no mesmo instante que a tarefa 1.

Outra observação importante a ser feita é que, embora a tarefa 1 tenha provocado o surgimento de um tempo ocioso na máquina 3, ela não a bloqueou. Se houvesse uma tarefa que possuísse um tempo de processamento menor ou igual ao vão entre o instante zero e o início da tarefa 2, esta poderia ser alocada neste intervalo.

Além disso, a presença do sincronismo de execução de tarefas faz com que surja a possibilidade de se obterem soluções inviáveis. Como exemplo, bastaria inverter a ordem de processamento das tarefas 5 e 7, na máquina 1, na Figura 15. Invertendo estas duas tarefas, não seria mais possível fazer com que a tarefa 5 acabasse junto com a 6 e a tarefa 7 junto com a 8, ao mesmo tempo, como pode ser visto na Figura 16.

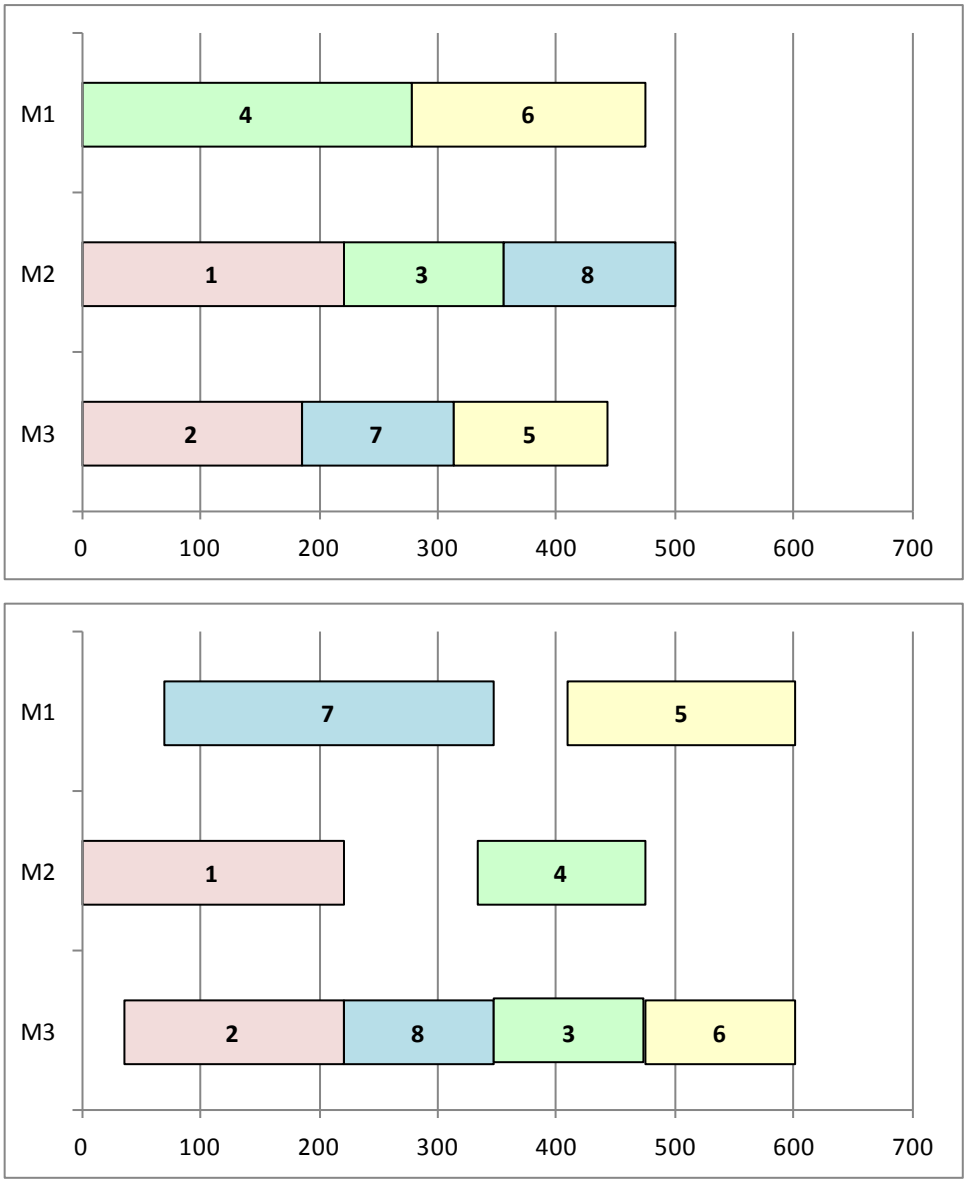
Figura 16 – Solução Inviável



Fonte: (Autor)

Finalmente, a inclusão deste tipo de restrição pode alterar drasticamente a solução ótima de um problema. A Figura 17 apresenta a solução ótima de um problema que não considera o sincronismo de execução de tarefas, e a solução ótima do mesmo problema com a inclusão desta restrição.

Figura 17 – Soluções ótimas antes e após a inclusão da restrição de sincronismo



Fonte: (Autor)

4 FORMULAÇÃO MATEMÁTICA

Nesta seção será apresentado o modelo matemático para o problema em estudo.

Nesta formulação, cada máquina corresponde a um forno de fusão, e cada tarefa corresponde à fusão de um metal para a produção de cilindros. Um cilindro que requer dois metais para ser produzido corresponde a um par de tarefas, enquanto que um cilindro produzido com um único metal corresponde a apenas uma tarefa.

4.1 PREMISSAS

Antes de iniciar a formulação matemática, é importante destacar algumas premissas:

a) cada máquina está disponível continuamente, sem interrupções;

b) cada máquina possui uma capacidade máxima (em peso) diferente. Dessa forma, nem todas as tarefas podem ser alocadas em todas as máquinas.

c) cada máquina pode processar apenas uma tarefa de cada vez;

d) cada tarefa deve ser alocada em apenas uma das diversas máquinas em que pode ser processada;

e) todas as tarefas têm a mesma data de liberação, a partir da qual, qualquer uma pode ser programada e executada;

f) os tempos de processamento das tarefas são determinísticos e fixos;

g) o tempo de execução de cada tarefa depende da máquina em que foi alocada;

h) os tempos de preparação das operações nas diversas máquinas estão incluídos nos tempos de processamento e dependem da sequência de operações em cada máquina;

i) as operações nas diversas máquinas, uma vez iniciadas não podem ser interrompidas.

j) uma máquina só pode iniciar o processamento de uma nova tarefa se a tarefa que estava sendo processada nela já tiver sido concluída. No caso de cilindros que forem constituídos de duas partes, a máquina só estará disponível para processar uma nova tarefa depois que ambas estiverem concluídas. Se uma delas terminar primeiro, ficará aguardando dentro da máquina em que estava sendo processada até que a outra parte seja finalizada em outra máquina. Esta é a premissa que diferencia este problema dos problemas tradicionais de sequenciamento.

4.2 FORMULAÇÃO INICIAL

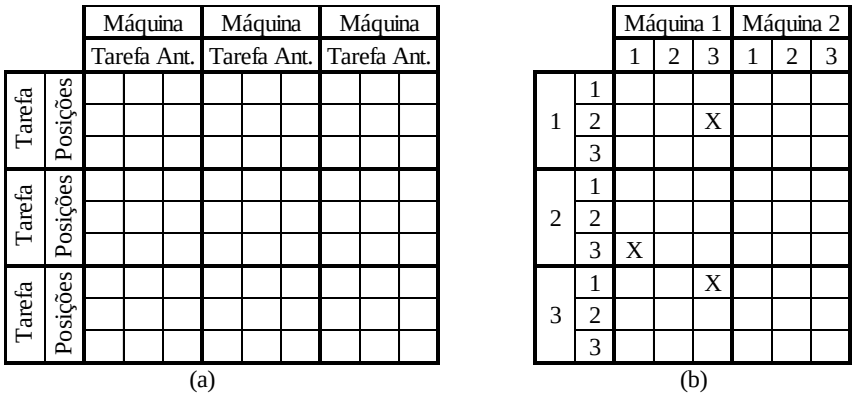
As variáveis de decisão adotadas são:

$X_{i,j,k,l}$: variável binária que indica que a tarefa *i* ocupará a posição *j* da máquina *k* e será feita imediatamente depois da tarefa *l*, quando $X_{i,j,k,l} = 1$. Ela assumirá o valor zero, caso contrário. Por convenção, quando a tarefa *i* ocupar a primeira posição de uma máquina, *l* assumirá o mesmo valor de *i*.

A Figura 18(a) mostra uma representação esquemática destas variáveis. Nesta representação as linhas representam as tarefas (índice *i*) e as colunas representam as máquinas (índice *k*). Para cada tarefa, representa-se a posição (índice *j*) que esta tarefa ocupa na sequência de produção da respectiva máquina. Para cada máquina, representa-se qual é a tarefa anterior (índice *l*) à correspondente tarefa, na sequência de produção. Esta representação será utilizada ainda nesta seção para ajudar a apresentar a formulação do sistema.

No exemplo da Figura 18(b), todas as tarefas foram produzidas na Máquina 1, na sequência 3-1-2. Cada "X" representa as variáveis $X_{3,1,1,3} = 1$; $X_{1,2,1,3} = 1$; e $X_{2,3,1,1} = 1$.

Figura 18 – Representação esquemática das variáveis de decisão
(a) Modelo; (b) Exemplo: $X_{1,2,1,3} = 1$; $X_{2,3,1,1} = 1$; $X_{3,1,1,3} = 1$



Fonte: (Autor)

$H_{i,j,k}$: variável real que determina o instante de término da tarefa *i* na posição *j* da máquina *k*. Quando uma tarefa não estiver ocupando uma determinada posição de uma máquina, esta variável assumirá o valor zero;

F_i: variável real que indica o instante de conclusão da tarefa *i*;

W: variável real que define o instante de conclusão da última tarefa.

Por convenção, as tarefas devem ser sequenciadas de tal forma que as $2n_p$ primeiras tarefas sejam componentes de cilindros constituídos por duas partes. A seguir, deverão vir as tarefas referentes aos cilindros constituídos por apenas um elemento. Além disso, as partes que formam um par devem receber números consecutivos. Dessa forma, as variáveis que tiverem i = 1 e 2; 3 e 4; 5 e 6,... $2n_p - 1$ e $2n_p$ serão as tarefas que formam um mesmo cilindro; já as variáveis que tiverem $i = 2n_p + 1$, ... n são as tarefas referentes aos cilindros constituídos de apenas um elemento.

A função-objetivo considerada neste trabalho é a minimização do *makespan*: Min W

Além das variáveis de decisão, os seguintes parâmetros devem ser utilizados:

n: número total de tarefas;

n_p: número total de pares do problema;

m_k: número total de posições na sequência de produção de cada máquina;

b: número total de máquinas;

P_i: Peso da tarefa i (quantidade de metal necessária para produzir cada componente de um cilindro);

C_k: Capacidade máxima, em peso, do forno k.

$T_{i,k,l}$: tempo de processamento da tarefa i ao ser feita na máquina k, após a tarefa l;

A: número muito grande.

As restrições adotadas são:

- Cada tarefa deve ocupar uma única posição em uma única máquina:

$$\sum_{j=1}^{m_k}\sum_{k=1}^{b}\sum_{l=1}^{n} X_{i,j,k,l} = 1 \qquad i = 1,\ldots,n \tag{1}$$

- Como cada forno possui um tamanho diferente, uma tarefa só pode ocupar uma máquina se ela não exceder a sua capacidade:

$$P_i X_{i,j,k,l} \leq C_k \qquad \begin{matrix} i = 1,\ldots,n \\ k = 1,\ldots,b \end{matrix} \qquad \begin{matrix} j = 1,\ldots,m_k \\ l = 1,\ldots,n \end{matrix} \tag{2}$$

- A posição de uma máquina só pode ser ocupada por uma única tarefa no máximo:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j,k,l} \leq 1 \qquad \begin{matrix} j = 1,\ldots,m_k \\ k = 1,\ldots,b \end{matrix} \tag{3}$$

- A posição j (j > 1) de uma máquina só pode ser ocupada se a posição j-1 estiver ocupada:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j,k,l} \leq \sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j-1,k,l} \qquad \begin{matrix} j = 2,\ldots,m_k \\ k = 1,\ldots,b \end{matrix} \tag{4}$$

- A sequência das tarefas numa mesma máquina deve ser respeitada:

$$\mathrm{X}_{1,1,k,1} + \sum_{i=1}^{n}\sum_{l=2}^{n} \mathrm{X}_{i,2,k,l} \leq 1 \qquad k = 1,\ldots,b \tag{5}$$

$$\mathrm{X}_{\mathrm{z},1,\mathrm{k},\mathrm{z}} + \sum_{i=1}^{n}\sum_{l=1}^{z-1} X_{i,2,k,l} + \sum_{i=1}^{n}\sum_{l=z+1}^{n} X_{i,2,k,l} \leq 1 \qquad \begin{matrix} k = 1,\ldots,b \\ z = 2,\ldots,n-1 \end{matrix} \tag{6}$$

$$X_{n,1,k,n} + \sum_{i=1}^{n}\sum_{l=1}^{n-1} X_{i,2,k,l} \leq 1 \qquad k = 1,\ldots,b \tag{7}$$

$$\sum_{l=1}^{n} X_{1,j,k,l} + \sum_{i=1}^{n}\sum_{l=2}^{n} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} k = 1,\ldots,b \\ j = 2,\ldots,m_k - 1 \end{matrix} \tag{8}$$

$$\sum_{l=1}^{n} X_{z,j,k,l} + \sum_{i=1}^{n}\sum_{l=1}^{z-1} X_{i,j+1,k,l} + \sum_{i=1}^{n}\sum_{l=z+1}^{n} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} k = 1,\ldots,b \\ j = 2,\ldots,m_k - 1 \\ \mathrm{z} = 2,\ldots,\mathrm{n}-1 \end{matrix} \tag{9}$$

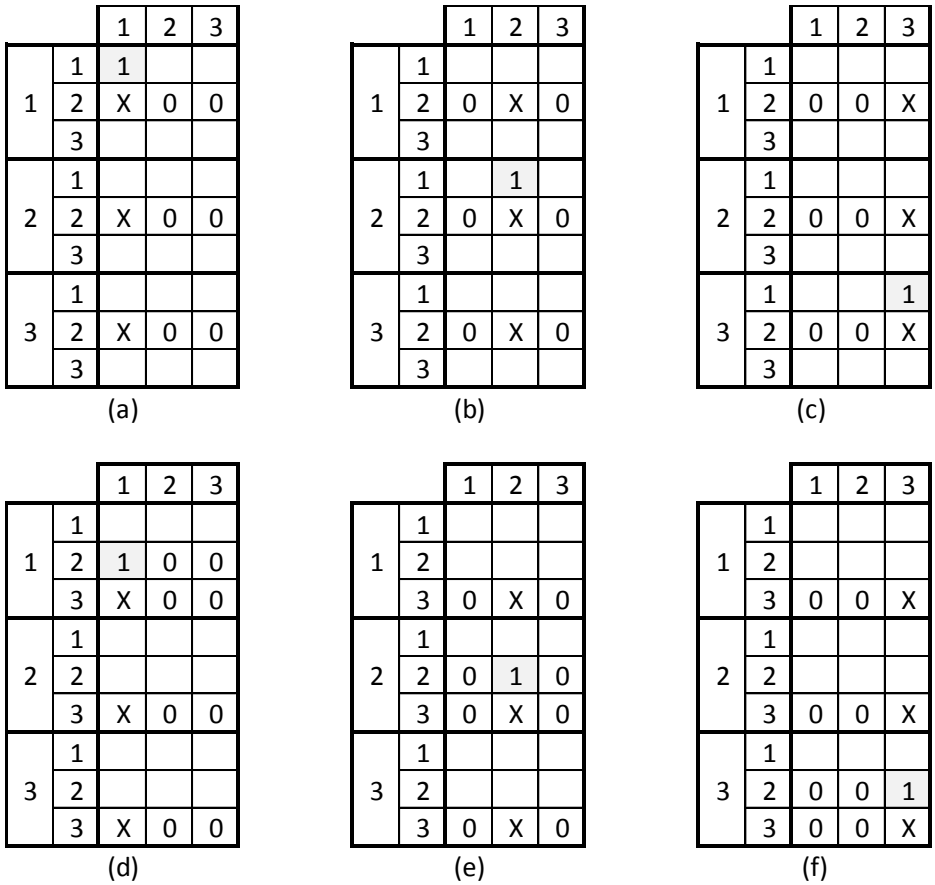
$$\sum_{l=1}^{n} X_{n,j,k,l} + \sum_{i=1}^{n}\sum_{l=1}^{n-1} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} k = 1,\ldots,b \\ j = 2,\ldots,m_k - 1 \end{matrix} \tag{10}$$

As expressões (5), (6) e (7) estão relacionadas com as primeiras posições de cada máquina, enquanto as expressões (8), (9) e (10), com as demais.

Como pode ser visto na Figura 19(a), se a tarefa 1 ocupar a primeira posição de uma máquina, diversas variáveis da segunda posição dessa máquina assumirão, obrigatoriamente, o valor zero. Isso faz com que apenas as variáveis da segunda posição que indicarem que a tarefa feita anteriormente foi a primeira tarefa (l = 1) possam assumir o valor um. Estas variáveis estão representadas por "X".

Analogamente, se a última tarefa ocupar a primeira posição de uma máquina, diversas variáveis da segunda posição dessa máquina assumirão, obrigatoriamente, o valor zero. Isso faz com que apenas as variáveis da segunda posição que indicarem que a tarefa feita anteriormente foi a última tarefa (no exemplo da Figura 19(c), l = 3) possam assumir o valor um. O mesmo raciocínio pode ser aplicado para as demais tarefas (Figura 19(b)) e para as demais posições (Figuras 19(d), 19(e) e (19(f)).

Figura 19 – Representação esquemática das restrições de sequenciamento em um forno
(a) Expressão (5); (b) Expressão (6); (c) Expressão (7); (d) Expressão (8);
(e) Expressão (9); (f) Expressão (10)



Fonte: (Autor)

- Se uma tarefa for a primeira de um forno, ela não possuirá nenhuma antecessora:

$$\sum_{l=2}^{n} X_{1,1,k,l} = 0 \qquad k = 1,\dots,b \tag{11}$$

$$\sum_{l=1}^{i-1} X_{i,1,k,l} + \sum_{l=i+1}^{n} X_{i,1,k,l} = 0 \qquad \begin{matrix} i = 2,\dots,n-1 \\ k = 1,\dots,b \end{matrix} \tag{12}$$

$$\sum_{l=1}^{n-1} X_{n,1,k,l} = 0 \qquad k = 1,\dots,b \tag{13}$$

- Uma tarefa não pode ser processada depois de si mesma:

$$\sum_{j=2}^{m_k} X_{i,j,k,i} = 0 \qquad \begin{matrix} i = 1,\dots,n \\ k = 1,\dots,b \end{matrix} \tag{14}$$

- O instante de término de uma tarefa numa determinada posição de uma máquina é maior ou igual ao instante de término da tarefa anterior somado do seu tempo de processamento:

$$H_{i,1,k} + A\left(1 - X_{i,1,k,l}\right) \geq T_{i,k,l}.X_{i,1,k,l} \qquad \begin{matrix} i = 1,\dots,n \\ k = 1,\dots,b \\ l = 1,\dots,n \end{matrix} \tag{15}$$

$$H_{i,j,k} + A\left(1 - X_{i,j,k,l}\right) \geq \sum_{z=1}^{n} H_{z,j-1,k} + T_{i,k,l}.X_{i,j,k,l} \qquad \begin{matrix} i = 1,\dots,n \\ k = 1,\dots,b \end{matrix} \qquad \begin{matrix} j = 2,\dots,m_k \\ l = 1,\dots,n \end{matrix} \tag{16}$$

$$H_{i,j,k} \leq A \sum_{l=1}^{n} X_{i,j,k,l} \qquad \begin{matrix} i = 1,\dots,n \\ j = 1,\dots,m_k \\ k = 1,\dots,b \end{matrix} \tag{17}$$

- Instante de conclusão de uma tarefa:

$$\mathrm{F_i} = \sum_{j=1}^{m_k} \sum_{\mathrm{k=1}}^{\mathrm{b}} H_{i,\mathrm{j},k} \qquad i = 1, \ldots, n \tag{18}$$

- Restrição de sincronismo de execução de tarefas - a casca e o núcleo de cilindros compostos de duas partes devem ficar prontos no mesmo instante:

$$F_i = F_{i+1} \qquad i = 1, 3, 5, \ldots 2n_p - 1 \tag{19}$$

- O instante de término da última tarefa é maior ou igual ao instante de conclusão de cada tarefa:

$$W \geq F_i \qquad i = 1, \ldots, n \tag{20}$$

- Não negatividade e variáveis binárias:

$$X_{i,j,k,l}\text{: binária} \qquad \begin{matrix} i = 1, \ldots, n & j = 1, \ldots, m_k \\ k = 1, \ldots, b & l = 1, \ldots, n \end{matrix} \tag{21}$$

$$H_{i,j,k} \geq 0 \qquad \begin{matrix} i = 1, \ldots, n \\ j = 1, \ldots, m_k \\ k = 1, \ldots, b \end{matrix} \tag{22}$$

$$F_i \geq 0 \qquad i = 1, \ldots, n \tag{23}$$

É importante fazer alguns comentários a respeito do artifício utilizado para tornar linear a restrição que diz que as partes que compõe um mesmo cilindro devem ficar prontas no mesmo instante (Expressões (15) a (19)).

Inicialmente é necessário saber o momento em que cada parte ficaria pronta individualmente. Para isso, poderiam ser utilizadas as expressões (24), quando a tarefa ocupar a primeira posição de uma máquina; e (25), caso ela ocupe qualquer outra posição.

- O instante de término de uma tarefa que ocupar a primeira posição de uma máquina é maior ou igual ao seu tempo de processamento:

$$H_{i,1,k} \geq T_{i,k,l} . X_{i,1,k,l} \qquad \begin{matrix} i = 1, \ldots, n \\ k = 1, \ldots, b \\ l = 1, \ldots, n \end{matrix} \tag{24}$$

- O instante de término de uma máquina é maior ou igual ao tempo de término da tarefa anterior somado do seu tempo de processamento:

$$H_{i,j,k} \geq H_{z,j-1,k} + T_{i,k,l}.X_{i,j,k,l} \qquad \begin{matrix} i = 1,\ldots,n \\ k = 1,\ldots,b \\ z = 1,\ldots,n \end{matrix} \qquad \begin{matrix} j = 2,\ldots,m_k \\ l = 1,\ldots,n \end{matrix} \qquad (25)$$

Os valores de $H_{i,j,k}$ obtidos pelas expressões (24) e (25) só deveriam ser considerados quando a tarefa realmente estiver ocupando a posição *j* da máquina *k*. A seguir, bastaria comparar os valores obtidos individualmente para cada uma das partes que formam um cilindro e adotar o maior deles para as duas.

Entretanto, o primeiro problema que surge com esta formulação (tal como está) é que cada tarefa possuiria um instante de término para cada uma das posições e máquinas possíveis, uma vez que não se sabe de antemão nem em que posição nem em que máquina cada uma será alocada.

Tal problema poderia ser solucionado multiplicando-se as variáveis $H_{i,j,k}$ pelas suas correspondentes $X_{i,j,k,l}$. Deste modo, a expressão (26) garantiria que as duas partes que compõe um mesmo cilindro ficariam prontas no mesmo instante:

$$\sum_{j=1}^{m_k}\sum_{k=1}^{b}\sum_{l=1}^{n} X_{i,j,k,l}.H_{i,j,k} = \sum_{j=1}^{m_k}\sum_{k=1}^{b}\sum_{l=1}^{n} X_{i+1,j,k,l}.H_{i+1,j,k} \qquad i = 1,3,5,\ldots 2n_p - 1 \quad (26)$$

O problema que surge com esta abordagem é que a restrição resultante do produto das variáveis $H_{i,j,k}$ pelas suas correspondentes $X_{i,j,k,l}$ não é linear.

A forma encontrada para eliminar as variáveis $H_{i,j,k}$ quando as variáveis $X_{i,j,k,l} = 0$, por meio de uma restrição linear, foi utilizar a expressão (17).

A seguir, a expressão (26) foi substituída pelas expressões (18) e (19).

Finalmente, as expressões (24) e (25) foram adaptadas, de tal maneira que ficassem desativadas quando $X_{i,j,k,l} = 0$. Isto foi conseguido por meio do termo $A(1 - X_{i,j,k,l})$, gerando as expressões (15) e (16).

A formulação apresentada nesta seção pode ser utilizada também em problemas de sequenciamento em máquinas paralelas sem sincronismo de tarefas. Para isso, basta eliminar a expressão (19).

Para estes problemas, embora não seja necessário, pode-se também substituir as expressões (15), (16), (17), (18) e (20), pela expressão (27):

- O instante de término da última tarefa é maior ou igual à soma dos tempos individuais de execução das tarefas em cada máquina:

$$W \geq \sum_{i=1}^{n} \sum_{j=1}^{m_k} \sum_{l=1}^{n} T_{i,k,l} \,.\, X_{i,j,k,l} \qquad k = 1, \ldots, b \tag{27}$$

4.2.1 Determinação dos valores de m_k

Um dos parâmetros que influencia nas dimensões do problema é o número total de posições na sequência de produção de cada máquina (m_k). Uma escolha adequada deste parâmetro reduz o número de variáveis e facilita a resolução do problema.

Neste trabalho utilizou-se o procedimento descrito a seguir.

Inicialmente deve-se obter uma solução viável para o problema. Esta solução pode ser conseguida através dos algoritmos que serão apresentados na Seção 5. Quanto melhor a solução, melhores serão os valores de m_k.

A seguir, deve-se selecionar o menor tempo de processamento das tarefas em cada máquina e ordená-los de forma crescente.

Finalmente, somam-se os m_k menores tempos que não ultrapassem o valor da solução viável conhecida. A Figura 20 apresenta o pseudocódigo deste procedimento.

Figura 20 – Pseudocódigo para determinação dos valores de m_k

```
01:  Entrada: Solução viável S;
02:  k = 1;
03:  Enquanto k < b
04:      Selecionar o menor tempo de processamento de cada tarefa na máquina k;
05:      Ordenar de forma crescente os tempos do passo 4 (t_i);
06:      m_k = 0;
07:      total = 0;
08:      i=1;
09:      Enquanto total + t_i < S
10:          total = total + t_i;
11:          i = i + 1;
12:          m_k = m_k + 1;
13:      Fim Enquanto;
14:      k = k + 1;
15:  Fim Enquanto
16:  Retornar m_k
```

Fonte: (Autor)

4.3 RESULTADOS COMPUTACIONAIS PRELIMINARES

O modelo matemático foi implementado no software CPLEX 12.1, e os testes computacionais foram executados em um computador com processador Intel Core 2 Duo de 3 GHz, e memória de 2 GB. O tempo máximo de processamento foi definido em 5400 segundos.

Inicialmente foram resolvidos 45 problemas de pequeno porte sem as restrições de sincronismo (caso A), divididos em dois grupos: o primeiro com 25 problemas de oito tarefas (4 pares), e o segundo com 20 problemas de 10 tarefas (5 pares), ambos com três máquinas. Vale a pena ressaltar que nenhum destes problemas possui cilindros constituídos por apenas um material, isto é, todos eles possuem casca e núcleo.

Os mesmos problemas foram resolvidos novamente, considerando as restrições de sincronismo (caso B), para avaliar o efeito desta sobre o tempo necessário para encontrar a solução ótima. Os resultados podem ser vistos na Tabela 1. Nesta tabela, Prob identifica o problema resolvido; n, o número total de tarefas do problema; n_p, o número de pares; Lim Inf, o melhor limitante inferior conhecido; Lim Sup, o melhor limitante superior; e *GAP* a diferença percentual entre os limitantes superior e inferior (*GAP* = 1 – Lim Inf/Lim Sup).

Tabela 1 – Resultados dos problemas de pequeno porte

			Lim. Inf.		Lim. Sup.		GAP		Tempo (s)	
Prob	n	n_p	A	B	A	B	A	B	A	B
1	8	4	499	423	499	601	0%	30%	0,72	5400
2	8	4	442	536	442	536	0%	0%	0,55	1016
3	8	4	464	562	464	562	0%	0%	0,58	1391
4	8	4	444	516	444	516	0%	0%	0,45	518
5	8	4	532	610	532	610	0%	0%	0,5	422
6	8	4	450	532	450	532	0%	0%	0,48	485
7	8	4	437	541	437	541	0%	0%	0,62	583
8	8	4	496	571	496	571	0%	0%	0,53	881
9	8	4	445	563	445	563	0%	0%	0,84	398
10	8	4	470	527	470	527	0%	0%	0,67	442
11	8	4	480	583	480	583	0%	0%	0,3	86
12	8	4	460	570	460	570	0%	0%	0,55	289
13	8	4	436	524	436	524	0%	0%	0,53	398

Tabela 1 – Resultados dos problemas de pequeno porte

			Lim. Inf.		Lim. Sup.		GAP		Tempo (s)	
Prob	n	n_p	A	B	A	B	A	B	A	B
14	8	4	452	564	452	564	0%	0%	0,59	473
15	8	4	454	535	454	535	0%	0%	0,73	518
16	8	4	435	536	435	536	0%	0%	0,81	803
17	8	4	485	585	485	585	0%	0%	1,26	412
18	8	4	455	555	455	555	0%	0%	0,58	872
19	8	4	444	546	444	546	0%	0%	0,66	992
20	8	4	452	552	452	552	0%	0%	0,84	601
21	8	4	496	564	496	564	0%	0%	0,7	505
22	8	4	447	552	447	552	0%	0%	0,56	311
23	8	4	433	538	433	538	0%	0%	1,03	494
24	8	4	453	558	453	558	0%	0%	0,83	188
25	8	4	434	539	434	539	0%	0%	0,86	3608
26	10	5	604	332	604	780	0%	57%	0,94	5400
27	10	5	485	206	485	648	0%	68%	3,87	5400
28	10	5	514	223	514	693	0%	68%	0,86	5400
29	10	5	602	280	602	744	0%	62%	11	5400
30	10	5	495	198	495	647	0%	69%	53	5400
31	10	5	557	227	557	725	0%	69%	21	5400
32	10	5	548	258	548	685	0%	62%	1,34	5400
33	10	5	481	216	481	660	0%	67%	1,5	5400
34	10	5	514	242	514	669	0%	64%	18	5400
35	10	5	541	245	541	679	0%	64%	13	5400
36	10	5	517	208	517	700	0%	70%	2,6	5400
37	10	5	520	212	520	642	0%	67%	83	5400
38	10	5	512	245	512	683	0%	64%	1,2	5400
39	10	5	509	244	509	668	0%	63%	6	5400
40	10	5	545	268	545	731	0%	63%	3	5400
41	10	5	565	258	565	684	0%	62%	58	5400
42	10	5	539	332	539	659	0%	50%	28	5400
43	10	5	505	198	505	686	0%	71%	16	5400
44	10	5	528	232	528	694	0%	67%	21	5400
45	10	5	511	213	511	687	0%	69%	8,7	5400

Analisando a Tabela 1, pode-se observar que foi possível encontrar a solução ótima (GAP = 0%) para todos os problemas que não possuíam a restrição de sincronismo de tarefas. Para os problemas com oito cilindros, foram necessários, em média, 0,67 segundos para atingir o ótimo. Já para os problemas com 10 cilindros, o computador gastou em média, 18 segundos.

Entretanto, não foi possível encontrar a solução ótima para todos os problemas que possuíam a restrição de sincronismo. Para os problemas com oito tarefas, foi possível chegar ao ótimo em 24 casos; já para aqueles com 10 tarefas, não se encontrou nenhuma solução ótima. Além disso, o tempo gasto nos problemas em que se encontrou a solução ótima foi, em média, de 695 segundos, mais de 1000 vezes maior do que o tempo gasto nos problemas sem esta restrição.

4.4 REFINAMENTO DA FORMULAÇÃO

Como foi visto na seção 4.3, o modelo matemático não encontrou resultados satisfatórios para os problemas com 10 tarefas.

A fim de melhorar os resultados obtidos, serão apresentadas novas restrições que não alteram o problema original, e que podem ser incluídas no modelo apresentado na seção 4.2.

O primeiro conjunto de expressões procura complementar a restrição de sincronismo de execução das tarefas. Uma vez que as duas partes que compõem um cilindro devem ficar prontas no mesmo instante, e que um forno não pode processar duas tarefas ao mesmo tempo, estas devem obrigatoriamente ser processadas em fornos diferentes. Portanto:

- As duas partes que compõem um cilindro não podem ser processadas num mesmo forno:

$$X_{i,j,k,i+1} = 0 \qquad \begin{matrix} i = 1,3,5 \ldots 2n_p - 1 \\ j = 1,\ldots,m_k \\ k = 1,\ldots,b \end{matrix} \tag{28}$$

$$X_{i+1,j,k,i} = 0 \qquad \begin{matrix} i = 1,3,5 \ldots 2n_p - 1 \\ j = 1,\ldots,m_k \\ k = 1,\ldots,b \end{matrix} \tag{29}$$

$$\sum_{j=1}^{m_k} \sum_{l=1}^{n} X_{i,j,k,l} + \sum_{j=1}^{m_k} \sum_{l=1}^{n} X_{i+1,j,k,l} \leq 1 \qquad \begin{matrix} i = 1,3,5 \ldots 2n_p - 1 \\ k = 1,\ldots,b \end{matrix} \tag{30}$$

Além das expressões (28), (29) e (30), é possível adicionar uma nova expressão à formulação da seção 4.2, desta vez com o intuito de obter um limitante inferior de melhor qualidade.

Analisando os comentários que foram feitos a respeito da formulação dos problemas que não possuem a restrição de sincronismo de execução de tarefas (Seção 4.2), verifica-se a existência da expressão (27) - o tempo final de execução da última tarefa é maior ou igual à soma dos tempos individuais de execução das tarefas em cada máquina.

Esta expressão sozinha, embora não seja suficiente para garantir o sincronismo de execução de tarefas, continua sendo perfeitamente válida nos problemas que contém esta restrição, uma vez que a presença do sincronismo poderá gerar apenas um atraso na liberação das tarefas de uma máquina, mas nunca uma antecipação.

Desta forma, o novo modelo matemático será constituído das expressões (1) a (23) e das expressões (27) a (30).

4.5 EVOLUÇÃO DOS RESULTADOS PRELIMINARES

Os mesmos problemas apresentados na seção 4.3 foram resolvidos novamente utilizando a nova formulação (caso C) e comparados com os resultados obtidos anteriormente (Tabela 2).

Tabela 2 – Resultados dos problemas de pequeno porte com a nova formulação

			Limitante Inferior			Lim. Superior			GAP			Tempo (s)		
Pro	N	Par	A	B	C	A	B	C	A	B	C	A	B	C
1	8	4	499	423	601	499	601	601	0%	30%	0%	0,72	5400	664
2	8	4	442	536	536	442	536	536	0%	0%	0%	0,55	1016	160
3	8	4	464	562	562	464	562	562	0%	0%	0%	0,58	1391	364
4	8	4	444	516	516	444	516	516	0%	0%	0%	0,45	518	50
5	8	4	532	610	610	532	610	610	0%	0%	0%	0,5	422	38
6	8	4	450	532	532	450	532	532	0%	0%	0%	0,48	485	37
7	8	4	437	541	541	437	541	541	0%	0%	0%	0,62	583	168
8	8	4	496	571	571	496	571	571	0%	0%	0%	0,53	881	171
9	8	4	445	563	563	445	563	563	0%	0%	0%	0,84	398	178
10	8	4	470	527	527	470	527	527	0%	0%	0%	0,67	442	48
11	8	4	480	583	583	480	583	583	0%	0%	0%	0,3	86	9
12	8	4	460	570	570	460	570	570	0%	0%	0%	0,55	289	40

Tabela 2 – Resultados dos problemas de pequeno porte com a nova formulação

			Limitante Inferior			Lim. Superior			GAP			Tempo (s)		
Pro	N	Par	A	B	C	A	B	C	A	B	C	A	B	C
13	8	4	436	524	524	436	524	524	0%	0%	0%	0,53	398	131
14	8	4	452	564	564	452	564	564	0%	0%	0%	0,59	473	109
15	8	4	454	535	535	454	535	535	0%	0%	0%	0,73	518	114
16	8	4	435	536	536	435	536	536	0%	0%	0%	0,81	803	341
17	8	4	485	585	585	485	585	585	0%	0%	0%	1,26	412	64
18	8	4	455	555	555	455	555	555	0%	0%	0%	0,58	872	382
19	8	4	444	546	546	444	546	546	0%	0%	0%	0,66	992	284
20	8	4	452	552	552	452	552	552	0%	0%	0%	0,84	601	102
21	8	4	496	564	564	496	564	564	0%	0%	0%	0,7	505	118
22	8	4	447	552	552	447	552	552	0%	0%	0%	0,56	311	81
23	8	4	433	538	538	433	538	538	0%	0%	0%	1,03	494	60
24	8	4	453	558	558	453	558	558	0%	0%	0%	0,83	188	27
25	8	4	434	539	539	434	539	539	0%	0%	0%	0,86	3608	747
26	10	5	604	332	667	604	780	762	0%	57%	12%	0,94	5400	5400
27	10	5	485	206	504	485	648	635	0%	68%	21%	3,87	5400	5400
28	10	5	514	223	553	514	693	648	0%	68%	15%	0,86	5400	5400
29	10	5	602	280	732	602	744	732	0%	62%	0%	11	5400	1134
30	10	5	495	198	500	495	647	629	0%	69%	21%	53	5400	5400
31	10	5	557	227	690	557	725	690	0%	69%	0%	21	5400	4852
32	10	5	548	258	601	548	685	664	0%	62%	9%	1,34	5400	5400
33	10	5	481	216	531	481	660	628	0%	67%	15%	1,5	5400	5400
34	10	5	514	242	547	514	669	644	0%	64%	15%	18	5400	5400
35	10	5	541	245	572	541	679	662	0%	64%	14%	13	5400	5400
36	10	5	517	208	528	517	700	654	0%	70%	19%	2,6	5400	5400
37	10	5	520	212	581	520	642	642	0%	67%	10%	83	5400	5400
38	10	5	512	245	569	512	683	662	0%	64%	14%	1,2	5400	5400
39	10	5	509	244	603	509	668	651	0%	63%	7%	6	5400	5400
40	10	5	545	268	700	545	731	700	0%	63%	0%	3	5400	5044
41	10	5	565	258	581	565	684	684	0%	62%	15%	58	5400	5400
42	10	5	539	332	659	539	659	659	0%	50%	0%	28	5400	2163
43	10	5	505	198	508	505	686	651	0%	71%	22%	16	5400	5400
44	10	5	528	232	558	528	694	680	0%	67%	18%	21	5400	5400
45	10	5	511	213	632	511	687	645	0%	69%	2%	8,7	5400	5400

Analisando a Tabela 2, pode-se observar que, com a nova formulação, foi possível encontrar a solução ótima para todos os problemas com oito tarefas que possuíam a restrição de sincronismo de tarefas. O tempo médio para isso foi reduzido de 695 para 179 segundos.

Para os problemas com 10 tarefas, foi possível chegar ao ótimo em quatro casos, em um tempo médio de quase 3300 segundos. Além disso, para os problemas em que não foi possível encontrar a solução ótima, a média dos *GAPs* entre os limitantes superior e inferior encontrados baixou de 66% para 14%.

4.6 RELAXAÇÃO DO PROBLEMA

Uma propriedade importante da formulação proposta que ainda não foi mencionada é que é possível relaxar a restrição de integralidade de parte das variáveis binárias e, mesmo assim, ainda obter uma solução viável (se ela existir). Dessa forma, torna-se possível propor dois teoremas.

Teorema 1: Se for possível assegurar que, na solução ótima, a segunda posição de uma máquina esteja ocupada por uma tarefa, todas as variáveis da primeira posição dessa máquina poderão ser relaxadas.

Partindo da expressão (3):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,1,k,l} \leq 1 \tag{31}$$

Utilizando a expressão (4):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,2,k,l} \leq \sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,1,k,l} \tag{32}$$

Partindo da premissa que existe uma tarefa na posição 2 de uma máquina:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,2,k,l} = 1 \tag{33}$$

Substituindo (33) em (32):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,1,k,l} \geq 1 \tag{34}$$

Comparando as expressões (31) e (34):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,1,k,l} = 1 \tag{35}$$

Pode-se combinar as expressões (11), (12) e (13), de forma a gerar uma única expressão equivalente:

$$\sum_{\substack{l=1 \\ l\neq i}}^{n} X_{i,1,k,l} = 0 \qquad i = 1,\ldots,n \tag{36}$$

Combinando-se as expressões (21) e (36):

$$X_{i,1,k,l} = 0 \qquad \begin{matrix} i = 1,\ldots,n \\ l = 1,\ldots,n \\ i \neq l \end{matrix} \tag{37}$$

Combinando as expressões (35) e (37):

$$\sum_{i=1}^{n} X_{i,1,k,i} = 1 \tag{38}$$

Pode-se combinar as expressões (5), (6) e (7), de forma a gerar uma única expressão equivalente:

$$X_{z,1,k,z} + \sum_{i=1}^{n}\sum_{\substack{l=1 \\ l\neq z}}^{n} X_{i,2,k,l} \leq 1 \qquad z = 1,\ldots,n \tag{39}$$

A expressão (33) pode ser reescrita da seguinte forma:

$$\sum_{i=1}^{n}\sum_{\substack{l=1 \\ l\neq z}}^{n} X_{i,2,k,l} + \sum_{i=1}^{n} X_{i,2,k,z} = 1 \qquad z = 1,\ldots,n \tag{40}$$

Substituindo (40) em (39):

$$X_{z,1,k,z} + \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,2,k,l} \leq \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,2,k,l} + \sum_{i=1}^{n} X_{i,2,k,z} \qquad z = 1, \ldots, n$$

$$X_{z,1,k,z} \leq \sum_{i=1}^{n} X_{i,2,k,z} \qquad z = 1, \ldots, n \tag{41}$$

Abrindo a expressão (41), tem-se:

$$X_{1,1,k,1} \leq X_{1,2,k,1} + X_{2,2,k,1} + X_{3,2,k,1} + \cdots + X_{n,2,k,1}$$
$$X_{2,1,k,2} \leq X_{1,2,k,2} + X_{2,2,k,2} + X_{3,2,k,2} + \cdots + X_{n,2,k,2}$$
$$X_{3,1,k,3} \leq X_{1,2,k,3} + X_{2,2,k,3} + X_{3,2,k,3} + \cdots + X_{n,2,k,3}$$
$$\vdots$$
$$X_{n,1,k,n} \leq X_{1,2,k,n} + X_{2,2,k,n} + X_{3,2,k,n} + \cdots + X_{n,2,k,n}$$

Como todas as variáveis $X_{i,2,k,z}$ são binárias, se todas as variáveis $X_{i,2,k,z}$ assumirem o valor zero, todas as variáveis $X_{i,1,k,z}$ também assumirão o valor zero, o que violaria a expressão (38). Portanto, uma das variáveis $X_{i,2,k,z}$ assumirá o valor um, e todas as demais variáveis $X_{i,2,k,z}$ assumirão o valor zero.

Dessa forma, devido às expressões (38) e (41), quaisquer que sejam os valores das variáveis $X_{i,2,k,z}$, haverá obrigatoriamente uma única variável $X_{z,1,k,z} = 1$.

A Figura 21 ilustra, esquematicamente, o Teorema 1.

Assumindo que, na solução ótima, a segunda posição de uma máquina esteja ocupada por uma tarefa, como todas as variáveis referentes a esta posição são binárias, haverá alguma que assumirá o valor um (Figura 21(a)).

De acordo com a expressão (3), a posição de uma máquina só pode ser ocupada por apenas uma única tarefa, no máximo. Dessa forma, todas as demais variáveis da segunda posição da máquina em questão assumirão o valor zero (Figura 21(b)).

Devido à expressão (12), que afirma que se uma tarefa for a primeira de uma máquina, ela não possuirá nenhuma antecessora, diversas variáveis da primeira posição receberão o valor zero (Figura 21(c)).

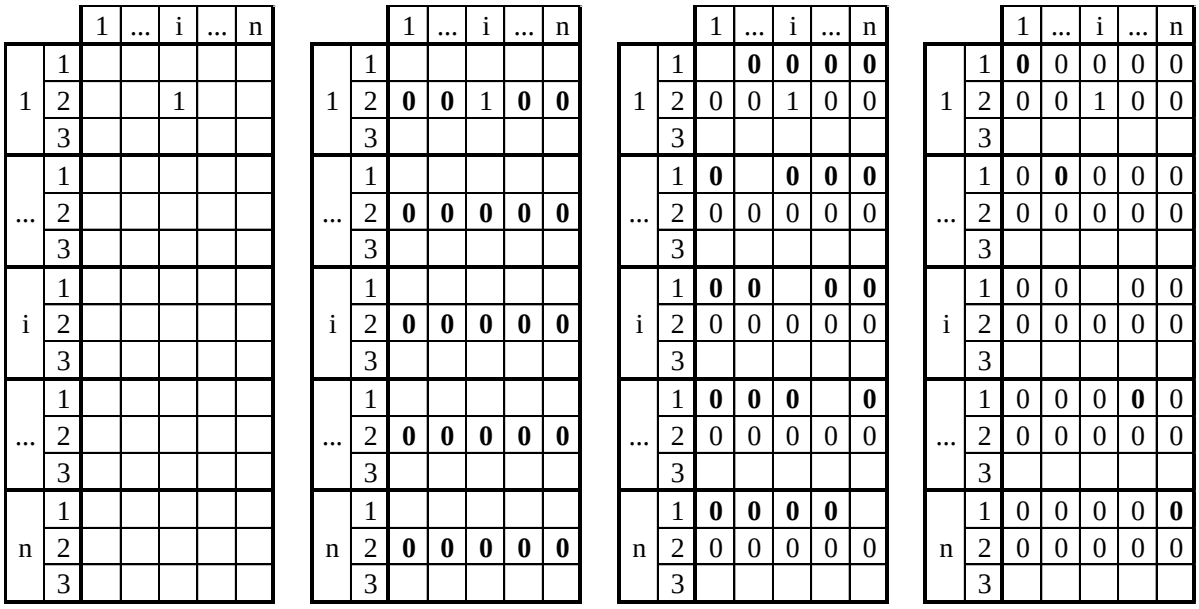
Como a segunda posição estará ocupada, a expressão (4) garante que a primeira posição também estará ocupada. Além disso, as expressões (5), (6) e (7), ao garantir que a sequência

das tarefas numa máquina seja respeitada, fazem com que todas as variáveis da primeira posição deste forno, com exceção de uma, assumam o valor zero (Figura 21(d)).

Finalmente, a expressão (3), ao ser combinada com a expressão (4), fará com que a variável restante assuma, obrigatoriamente, o valor um.

Figura 21 – Relaxação das variáveis da primeira posição de um forno

(a) Segunda posição ocupada; (b) Expressão (3); (c) Expressão (12); (d) Expressão (4) a (7)



		1	...	i	...	n
1	1					
	2			1		
	3					
...	1					
	2					
	3					
i	1					
	2					
	3					
...	1					
	2					
	3					
n	1					
	2					
	3					

(a)

		1	...	i	...	n
1	1					
	2	**0**	**0**	1	**0**	**0**
	3					
...	1					
	2	**0**	**0**	**0**	**0**	**0**
	3					
i	1					
	2	**0**	**0**	**0**	**0**	**0**
	3					
...	1					
	2	**0**	**0**	**0**	**0**	**0**
	3					
n	1					
	2	**0**	**0**	**0**	**0**	**0**
	3					

(b)

		1	...	i	...	n
1	1		**0**	**0**	**0**	**0**
	2	0	0	1	0	0
	3					
...	1	**0**		**0**	**0**	**0**
	2	0	0	0	0	0
	3					
i	1	**0**	**0**		**0**	**0**
	2	0	0	0	0	0
	3					
...	1	**0**	**0**	**0**		**0**
	2	0	0	0	0	0
	3					
n	1	**0**	**0**	**0**	**0**	
	2	0	0	0	0	0
	3					

(c)

		1	...	i	...	n
1	1	**0**	0	0	0	0
	2	0	0	1	0	0
	3					
...	1	0	**0**	0	0	0
	2	0	0	0	0	0
	3					
i	1	0	0		0	0
	2	0	0	0	0	0
	3					
...	1	0	0	0	**0**	0
	2	0	0	0	0	0
	3					
n	1	0	0	0	0	**0**
	2	0	0	0	0	0
	3					

(d)

Fonte: (Autor)

Teorema 2: Se for possível assegurar que, na solução ótima, a posição j + 2 de uma máquina esteja ocupada por uma tarefa, todas as variáveis da posição j + 1 dessa máquina poderão ser relaxadas, desde que as variáveis da posição j não sejam.

Partindo da expressão (3), obtém-se:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} j = 1,\ldots,m \\ k = 1,\ldots,b \end{matrix} \tag{42}$$

Utilizando a expressão (4):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} \leq \sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j,k,l} \tag{43}$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+2,k,l} \leq \sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} \tag{44}$$

Partindo da premissa que existe uma tarefa na posição *j* e outra na posição *j* + 2 de uma máquina:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j,k,l} = 1 \tag{45}$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+2,k,l} = 1 \tag{46}$$

Substituindo (45) em (43) e (46) em (44):

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} \leq 1 \tag{47}$$

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} \geq 1 \tag{48}$$

Portanto:

$$\sum_{i=1}^{n}\sum_{l=1}^{n} X_{i,j+1,k,l} = 1 \tag{49}$$

Pode-se combinar as expressões (8), (9) e (10) de forma a gerar uma única expressão equivalente:

$$\sum_{l=1}^{n} X_{z,j,k,l} + \sum_{i=1}^{n}\sum_{\substack{l=1 \\ l\neq z}}^{n} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} k = 1,\ldots,b \\ j = 2,\ldots,m-1 \\ \mathrm{z} = 1,\ldots,\mathrm{n} \end{matrix} \tag{50}$$

Assim:

$$\sum_{l=1}^{n} X_{z,j,k,l} + \sum_{i=1}^{n}\sum_{\substack{l=1 \\ l\neq z}}^{n} X_{i,j+1,k,l} \leq 1 \qquad \mathrm{z} = 1,\ldots,\mathrm{n} \tag{51}$$

$$\sum_{l=1}^{n} X_{z,j+1,k,l} + \sum_{i=1}^{n}\sum_{\substack{l=1 \\ l\neq z}}^{n} X_{i,j+2,k,l} \leq 1 \qquad \mathrm{z} = 1,\ldots,\mathrm{n} \tag{52}$$

Como todas as variáveis $X_{z,j,k,l}$ e todas as variáveis $X_{z,j,k,l}$ são não-negativas, as expressões (53) e (54) também são válidas:

$$X_{z,j,k,r} + \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+1,k,l} \leq 1 \qquad \begin{matrix} z = 1, \ldots, n \\ r = 1, \ldots, n \end{matrix} \tag{53}$$

$$X_{z,j+1,k,r} + \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+2,k,l} \leq 1 \qquad \begin{matrix} z = 1, \ldots, n \\ r = 1, \ldots, n \end{matrix} \tag{54}$$

As expressões (49) e (46) podem ser reescritas seguinte forma de acordo com as expressões (55) e (56), respectivamente:

$$\sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+1,k,l} + \sum_{i=1}^{n} X_{i,j+1,k,z} = 1 \qquad \mathrm{z} = 1, \ldots, \mathrm{n} \tag{55}$$

$$\sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+2,k,l} + \sum_{i=1}^{n} X_{i,j+2,k,z} = 1 \qquad \mathrm{z} = 1, \ldots, \mathrm{n} \tag{56}$$

Substituindo (55) em (53):

$$X_{z,j,k,r} + \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+1,k,l} \leq \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+1,k,l} + \sum_{i=1}^{n} X_{i,j+1,k,z}$$

$$X_{z,j,k,r} \leq \sum_{i=1}^{n} X_{i,j+1,k,z} \qquad \begin{matrix} z = 1..n \\ r = 1..n \end{matrix} \tag{57}$$

Substituindo (56) em (54):

$$X_{z,j+1,k,r} + \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+2,k,l} \leq \sum_{i=1}^{n} \sum_{\substack{l=1 \\ l \neq z}}^{n} X_{i,j+2,k,l} + \sum_{i=1}^{n} X_{i,j+1,k,z}$$

$$X_{z,j+1,k,r} \leq \sum_{i=1}^{n} X_{i,j+2,k,z} \qquad \begin{matrix} z = 1, \ldots, n \\ r = 1, \ldots, n \end{matrix} \tag{58}$$

Abrindo a expressão (58), tem-se:

$$X_{1,j+1,k,1} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$

$$X_{1,j+1,k,2} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$

$$X_{1,j+1,k,3} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$\vdots$$
$$X_{1,j+1,k,n} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$X_{2,j+1,k,1} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + X_{3,j+2,k,2} + \cdots + X_{n,j+2,k,2}$$
$$X_{2,j+1,k,2} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + X_{3,j+2,k,2} + \cdots + X_{n,j+2,k,2}$$
$$X_{2,j+1,k,3} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + X_{3,j+2,k,2} + \cdots + X_{n,j+2,k,2}$$
$$\vdots$$
$$X_{2,j+1,k,n} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + X_{3,j+2,k,2} + \cdots + X_{n,j+2,k,2}$$
$$X_{3,j+1,k,1} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$X_{3,j+1,k,2} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$X_{3,j+1,k,3} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$\vdots$$
$$X_{3,j+1,k,n} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$

Como todas as variáveis $X_{i,j+2,k,z}$ são binárias, se todas as variáveis $X_{i,j+2,k,z}$ assumirem o valor zero, todas as variáveis $X_{i,j+1,k,z}$ também assumirão o valor zero, o que violaria a expressão (43). Portanto, uma das variáveis $X_{i,j+2,k,z}$ assumirá o valor um, e todas as demais variáveis $X_{i,j+2,k,z}$ assumirão o valor zero.

Abrindo a expressão (57), tem-se:

$$X_{1,j,k,1} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1}$$
$$X_{1,j,k,2} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1}$$
$$X_{1,j,k,3} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1}$$
$$\vdots$$
$$X_{1,j,k,n} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1}$$
$$X_{2,j,k,1} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2}$$
$$X_{2,j,k,2} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2}$$
$$X_{2,j,k,3} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2}$$
$$\vdots$$
$$X_{2,j,k,n} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2}$$
$$X_{3,j,k,1} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3}$$
$$X_{3,j,k,2} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3}$$

$$X_{3,j,k,3} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3}$$
$$\vdots$$
$$X_{3,j,k,n} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3}$$

Analisando a expressão (45), como todas as variáveis $X_{i,j,k,z}$ são binárias, uma das variáveis $X_{i,j,k,z}$ assumirá o valor um, e todas as demais variáveis $X_{i,j,k,z}$ assumirão o valor zero.

Dessa forma, independentemente dos valores das variáveis $X_{i,j,k,z}$ e $X_{i,j+2,k,z}$, haverá sempre uma variável $X_{i,j+1,k,z}$ que assumirá obrigatoriamente o valor um.

Por exemplo, se $X_{3,j+2,k,2} = 1$, utilizando a expressão (46) tem-se:

$$X_{1,j+1,k,1} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$X_{1,j+1,k,2} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$X_{1,j+1,k,3} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$\vdots$$
$$X_{1,j+1,k,n} \leq X_{1,j+2,k,1} + X_{2,j+2,k,1} + X_{3,j+2,k,1} + \cdots + X_{n,j+2,k,1}$$
$$X_{2,j+1,k,1} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + \boldsymbol{X_{3,j+2,k,2}} + \cdots + X_{n,j+2,k,2}$$
$$X_{2,j+1,k,2} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + \boldsymbol{X_{3,j+2,k,2}} + \cdots + X_{n,j+2,k,2}$$
$$X_{2,j+1,k,3} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + \boldsymbol{X_{3,j+2,k,2}} + \cdots + X_{n,j+2,k,2}$$
$$\vdots$$
$$X_{2,j+1,k,n} \leq X_{1,j+2,k,2} + X_{2,j+2,k,2} + \boldsymbol{X_{3,j+2,k,2}} + \cdots + X_{n,j+2,k,2}$$
$$X_{3,j+1,k,1} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$X_{3,j+1,k,2} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$X_{3,j+1,k,3} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$
$$\vdots$$
$$X_{3,j+1,k,n} \leq X_{1,j+2,k,3} + X_{2,j+2,k,3} + X_{3,j+2,k,3} + \cdots + X_{n,j+2,k,3}$$

Portanto:

$$X_{2,j+1,k,1} \leq 1$$
$$X_{2,j+1,k,2} \leq 1$$
$$X_{2,j+1,k,3} \leq 1$$
$$\vdots$$
$$X_{2,j+1,k,n} \leq 1$$

E todas as variáveis $X_{i,j+1,k,l} = 0$, $i \neq 2$.

Se, por exemplo, $X_{1,j,k,4} = 1$, utilizando a expressão (45) tem-se:

$$
\begin{gathered}
X_{1,j,k,1} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1} \\
X_{1,j,k,2} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1} \\
X_{1,j,k,3} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1} \\
\boldsymbol{X_{1,j,k,4}} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1} \\
\vdots \\
X_{1,j,k,n} \leq X_{1,j+1,k,1} + X_{2,j+1,k,1} + X_{3,j+1,k,1} + \cdots + X_{n,j+1,k,1} \\
X_{2,j,k,1} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2} \\
X_{2,j,k,2} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2} \\
X_{2,j,k,3} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2} \\
X_{2,j,k,4} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2} \\
\vdots \\
X_{2,j,k,n} \leq X_{1,j+1,k,2} + X_{2,j+1,k,2} + X_{3,j+1,k,2} + \cdots + X_{n,j+1,k,2} \\
X_{3,j,k,1} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3} \\
X_{3,j,k,2} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3} \\
X_{3,j,k,3} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3} \\
X_{3,j,k,4} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3} \\
\vdots \\
X_{3,j,k,n} \leq X_{1,j+1,k,3} + X_{2,j+1,k,3} + X_{3,j+1,k,3} + \cdots + X_{n,j+1,k,3}
\end{gathered}
$$

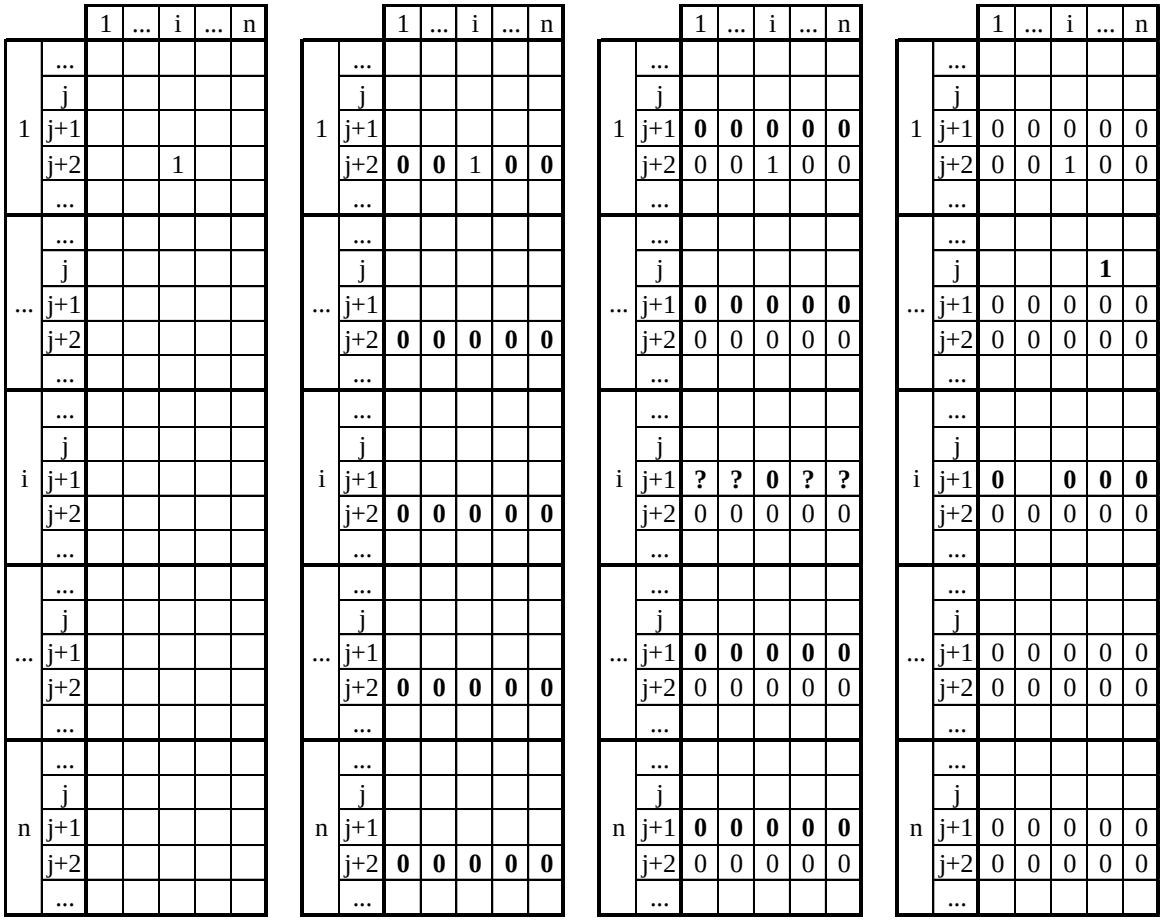
Como todas as variáveis $X_{i,j+1,k,l} = 0$, $i \neq 2$:

$$
\begin{gathered}
1 \leq X_{2,j+1,k,1} \\
0 \leq X_{2,j+1,k,2} \\
0 \leq X_{2,j+1,k,3} \\
\vdots \\
0 \leq X_{2,j+1,k,n}
\end{gathered}
$$

Combinando este resultado com aquele obtido a partir da expressão (46), a variável $X_{2,j+1,k,1}$ assumirá obrigatoriamente o valor um. Esse resultado em conjunto com a expressão (43) garante que todas as demais variáveis $X_{2,j+1,k,l}$ assumam o valor zero.

A Figura 22 ilustra, esquematicamente, o Teorema 2.

Figura 22 – Relaxação das variáveis das demais posições de um forno
(a) Posição j ocupada; (b) Expressão (3); (c) Expressão (6) e (14); (d) Posição j-2 ocupada



Fonte: (Autor)

Assumindo que, na solução ótima, a posição $j + 2$ de uma máquina esteja ocupada por uma tarefa, como todas as variáveis referentes a esta posição são binárias, haverá alguma que assumirá o valor um (Figura 22(a)).

De acordo com a expressão (3), a posição de uma máquina só pode ser ocupada por apenas uma única tarefa, no máximo. Dessa forma, todas as demais variáveis da posição $j + 2$ da máquina em questão assumirão o valor zero (Figura 22(b)).

Como a posição $j + 2$ estará ocupada, a expressão (4) garante que a posição $j + 1$ também seja ocupada. Além disso, as expressões (8), (9) e (10), ao garantir que a sequência das tarefas numa máquina seja respeitada, e a (14), ao garantir que uma tarefa não possa ser produzida depois de si mesma, fazem com que diversas variáveis da posição $j + 1$ assumam o valor zero. Entretanto, só isto não bastaria para relaxar as variáveis da posição $j + 1$, pois ainda haveria n-1 variáveis disponíveis nesta posição (Figura 22(c)).

Este problema é solucionado ao não relaxar as variáveis da posição j. Ao não relaxá-las, haverá uma variável nesta posição que assumirá o valor um. Com isso, graças à expressão (6), com exceção de uma, todas as variáveis da posição $j + 1$ que ainda não haviam assumido o valor zero, assumirão este valor (Figura 22(d)).

Por fim, a expressão (3), ao ser combinada com a expressão (4), fará com que a variável restante assuma, obrigatoriamente, o valor um.

4.7 RESULTADOS OBTIDOS APÓS RELAXAÇÃO DO PROBLEMA

Os mesmos problemas apresentados nas seções 4.3 e 4.5 foram resolvidos novamente relaxando todas as variáveis da primeira posição de cada forno (caso D), e comparados com os resultados obtidos anteriormente (Tabela 3).

Tabela 3 – Resultados dos problemas de pequeno porte utilizando a relaxação

			Limitante Inferior			Lim. Superior			GAP			Tempo (s)		
Pro	N	Par	A	C	D	A	C	D	A	C	D	A	C	D
1	8	4	499	601	601	499	601	601	0%	0%	0%	0,72	664	473
2	8	4	442	536	536	442	536	536	0%	0%	0%	0,55	160	182
3	8	4	464	562	562	464	562	562	0%	0%	0%	0,58	364	164
4	8	4	444	516	516	444	516	516	0%	0%	0%	0,45	50	40
5	8	4	532	610	610	532	610	610	0%	0%	0%	0,5	38	16
6	8	4	450	532	532	450	532	532	0%	0%	0%	0,48	37	36
7	8	4	437	541	541	437	541	541	0%	0%	0%	0,62	168	92
8	8	4	496	571	571	496	571	571	0%	0%	0%	0,53	171	85
9	8	4	445	563	563	445	563	563	0%	0%	0%	0,84	178	33
10	8	4	470	527	527	470	527	527	0%	0%	0%	0,67	48	39
11	8	4	480	583	583	480	583	583	0%	0%	0%	0,3	9	5
12	8	4	460	570	570	460	570	570	0%	0%	0%	0,55	40	21

Tabela 3 – Resultados dos problemas de pequeno porte utilizando a relaxação

			Limitante Inferior			Lim. Superior			GAP			Tempo (s)		
Pro	N	Par	A	C	D	A	C	D	A	C	D	A	C	D
13	8	4	436	524	524	436	524	524	0%	0%	0%	0,53	131	83
14	8	4	452	564	564	452	564	564	0%	0%	0%	0,59	109	47
15	8	4	454	535	535	454	535	535	0%	0%	0%	0,73	114	66
16	8	4	435	536	536	435	536	536	0%	0%	0%	0,81	341	165
17	8	4	485	585	585	485	585	585	0%	0%	0%	1,26	64	52
18	8	4	455	555	555	455	555	555	0%	0%	0%	0,58	382	263
19	8	4	444	546	546	444	546	546	0%	0%	0%	0,66	284	284
20	8	4	452	552	552	452	552	552	0%	0%	0%	0,84	102	60
21	8	4	496	564	564	496	564	564	0%	0%	0%	0,7	118	82
22	8	4	447	552	552	447	552	552	0%	0%	0%	0,56	81	37
23	8	4	433	538	538	433	538	538	0%	0%	0%	1,03	60	75
24	8	4	453	558	558	453	558	558	0%	0%	0%	0,83	27	23
25	8	4	434	539	539	434	539	539	0%	0%	0%	0,86	747	602
26	10	5	604	667	762	604	762	762	0%	12%	0%	0,94	5400	3014
27	10	5	485	504	514	485	635	635	0%	21%	19%	3,87	5400	5400
28	10	5	514	553	586	514	648	648	0%	15%	10%	0,86	5400	5400
29	10	5	602	732	732	602	732	732	0%	0%	0%	11	1134	1636
30	10	5	495	500	507	495	629	629	0%	21%	19%	53	5400	5400
31	10	5	557	690	690	557	690	690	0%	0%	0%	21	4852	3685
32	10	5	548	601	664	548	664	664	0%	9%	0%	1,34	5400	4022
33	10	5	481	531	543	481	628	628	0%	15%	14%	1,5	5400	5400
34	10	5	514	547	644	514	644	644	0%	15%	0%	18	5400	2651
35	10	5	541	572	590	541	662	662	0%	14%	11%	13	5400	5400
36	10	5	517	528	533	517	654	654	0%	19%	19%	2,6	5400	5400
37	10	5	520	581	590	520	642	642	0%	10%	8%	83	5400	5400
38	10	5	512	569	662	512	662	662	0%	14%	0%	1,2	5400	2812
39	10	5	509	603	651	509	651	651	0%	7%	0%	6	5400	4121
40	10	5	545	700	700	545	700	700	0%	0%	0%	3	5044	3052
41	10	5	565	581	684	565	684	684	0%	15%	0%	58	5400	3023
42	10	5	539	659	659	539	659	659	0%	0%	0%	28	2163	1319
43	10	5	505	508	513	505	651	651	0%	22%	21%	16	5400	5400
44	10	5	528	558	674	528	680	674	0%	18%	0%	21	5400	3225
45	10	5	511	632	645	511	645	645	0%	2%	0%	8,7	5400	2768

Analisando a Tabela 3, pode-se observar que, com a nova formulação, foi possível encontrar a solução ótima para todos os problemas com oito tarefas que possuíam a restrição de sincronismo de tarefas. O tempo médio para isso foi reduzido de 695 e 179, nas duas resoluções iniciais, para 121 segundos.

Para os problemas com 10 tarefas, foi possível chegar ao ótimo em 12 dos 20 casos, em um tempo médio de pouco mais de 2940 segundos. Além disso, para os oito problemas em que não foi possível encontrar a solução ótima, a média dos *GAPs* entre o limitante superior e o limitante inferior encontrada baixou de 68% e 17%, nas duas resoluções iniciais, para 15%.

4.8 FORMULAÇÃO ALTERNATIVA

Esta seção apresenta uma adaptação de uma formulação clássica encontrada na literatura para problemas de sequenciamento de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência (HELAL; RABADI; AL-SALEM, 2006; RUIZ; ROMANO, 2011; STEFANELLO *et al*., 2010), a fim de avaliar o desempenho do modelo matemático proposto.

A formulação original é dada a seguir.

Parâmetros

n: número de tarefas.

m: número de máquinas.

M: $\{1,...,m\}$, representa o conjunto das máquinas.

N: $\{1,...,n\}$, representa o conjunto das tarefas.

V: um número suficientemente grande.

p_{ij}: tempo de processamento da tarefa j, $j \in$ N, na máquina i, $i \in$ M.

s_{ijk}: tempo de preparação (*set-up time*) da máquina i, $i \in$ M, para execução da tarefa k após o processamento da tarefa j; $k, j \in$ N.

Variáveis

$X_{i,j,k} = 1$ se a tarefa j precede a tarefa k na máquina i; $X_{i,j,k} = 0$ caso contrário

$C_{ij} \geq 0$: Tempo de execução da tarefa j na máquina i; $i, j \in$ N.

$C_{max} \geq 0$: Tempo máximo de execução.

$$\text{Min } C_{max} \tag{59}$$

$$\sum_{i \in M} \sum_{\substack{j \in \{0\} \cup \{N\} \\ j \neq k}} X_{i,j,k} = 1 \qquad \forall k \in N \tag{60}$$

$$\sum_{i \in M} \sum_{\substack{k \in N \\ j \neq k}} X_{i,j,k} \leq 1 \qquad \forall j \in N \tag{61}$$

$$\sum_{k \in N} X_{i0k} \leq 1 \qquad \forall i \in M \tag{62}$$

$$\sum_{\substack{h \in \{0\} \cup \{N\} \\ h \neq k, h \neq j}} X_{ihj} \geq X_{ijk} \qquad \forall j, k \in N, j \neq k, \forall i \in M \tag{63}$$

$$C_{ik} + V\left(1 - x_{ijk}\right) \geq C_{ij} + s_{ijk} + p_{ik} \qquad \forall j \in \{0\} \cup \{N\}, \forall k \in N, j \neq k, \forall i \in M \tag{64}$$

$$C_{i0} = 0 \qquad \forall i \in M \tag{65}$$

$$C_{ij} \geq 0 \qquad \forall j \in N, \forall i \in M \tag{66}$$

$$C_{max} \geq C_{ij} \qquad \forall j \in N, \forall i \in M \tag{67}$$

$$X_{ijk} \in \{0,1\} \qquad \forall j \in \{0\} \cup \{N\}, \forall k \in N, j \neq k, \forall i \in M \tag{68}$$

O objetivo é minimizar o *makespan* (C_{max}).

O conjunto de restrições (60) garante que cada tarefa seja atribuída a uma única máquina e que tenha exatamente um antecessor.

As restrições (61) determinam que cada tarefa só pode ter um sucessor. É importante observar que uma tarefa artificial 0 identifica o início do sequenciamento das tarefas nas máquinas, e o conjunto de restrições (62) limita o número de sucessores da tarefa artificial em, no máximo, um para cada máquina.

O conjunto (63) garante que as tarefas são sequenciadas corretamente nas máquinas.

As restrições (64) determinam o tempo final de processamento das tarefas em cada máquina.

Os conjuntos de restrições (65) e (66) definem o tempo de conclusão como 0 para as tarefas artificiais, e valores não-negativos para as demais tarefas, respectivamente.

As restrições (67) definem o *makespan*.

Finalmente, o conjunto de restrições (68) define as variáveis binárias.

Como esta formulação não leva em consideração nem o tamanho das máquinas, nem o sincronismo de execução de tarefas, são adicionadas algumas restrições:

$$Peso_j . X_{ijk} \leq Cap_i \qquad \forall j \in N, \forall k \in N, j \neq k, \forall i \in M \tag{69}$$

$$Peso_k . X_{ijk} \leq Cap_i \qquad \forall j \in \{0\} \cup \{N\}, \forall k \in N, j \neq k, \forall i \in M \tag{70}$$

$$C_{ik} \leq V \sum_{\substack{j \in \{0\} \cup \{N\} \\ j \neq k}} X_{ijk} \qquad \forall k \in N, \forall i \in M \tag{71}$$

$$F_j = \sum_{i \in M} C_{ij} \qquad \forall j \in N \tag{72}$$

$$F_j = F_{j+1} \qquad j = 1, 3, 5, \ldots 2n_p - 1 \tag{73}$$

$$C_{max} \geq F_j \qquad \forall j \in N \tag{74}$$

As expressões (69) e (70) são equivalentes à expressão (2), e garantem que as tarefas só sejam alocadas em uma máquina se não excederem sua capacidade.

As expressões (71), (72) e (73) são equivalentes às expressões (17), (18) e (19), respectivamente, e estão relacionadas com os instantes de conclusão de cada tarefa.

Finalmente, a expressão (74) deve ser usada no lugar da expressão (67).

4.9 RESULTADOS OBTIDOS COM A FORMULAÇÃO ALTERNATIVA

Os mesmos problemas apresentados na Seção 4.3 foram resolvidos novamente com esta formulação alternativa (caso E). Os resultados obtidos são mostrados na Tabela 4 e comparados com os resultados obtidos anteriormente (Tabela 3).

Tabela 4 – Resultados dos problemas de pequeno porte – formulação alternativa

			Limitante Inferior			Lim. Superior			*GAP*			Tempo (s)		
Pro	N	Par	C	D	E	C	D	E	C	D	E	C	D	E
1	8	4	601	601	601	601	601	601	0%	0%	0%	664	473	305
2	8	4	536	536	536	536	536	536	0%	0%	0%	160	182	136
3	8	4	562	562	562	562	562	562	0%	0%	0%	364	164	281
4	8	4	516	516	516	516	516	516	0%	0%	0%	50	40	43
5	8	4	610	610	610	610	610	610	0%	0%	0%	38	16	57
6	8	4	532	532	532	532	532	532	0%	0%	0%	37	36	33
7	8	4	541	541	541	541	541	541	0%	0%	0%	168	92	32
8	8	4	571	571	571	571	571	571	0%	0%	0%	171	85	51
9	8	4	563	563	563	563	563	563	0%	0%	0%	178	33	23
10	8	4	527	527	527	527	527	527	0%	0%	0%	48	39	43
11	8	4	583	583	583	583	583	583	0%	0%	0%	9	5	14
12	8	4	570	570	570	570	570	570	0%	0%	0%	40	21	37
13	8	4	524	524	524	524	524	524	0%	0%	0%	131	83	93
14	8	4	564	564	564	564	564	564	0%	0%	0%	109	47	53
15	8	4	535	535	535	535	535	535	0%	0%	0%	114	66	82
16	8	4	536	536	536	536	536	536	0%	0%	0%	341	165	209
17	8	4	585	585	585	585	585	585	0%	0%	0%	64	52	52
18	8	4	555	555	555	555	555	555	0%	0%	0%	382	263	83
19	8	4	546	546	546	546	546	546	0%	0%	0%	284	284	268
20	8	4	552	552	552	552	552	552	0%	0%	0%	102	60	94
21	8	4	564	564	564	564	564	564	0%	0%	0%	118	82	42
22	8	4	552	552	552	552	552	552	0%	0%	0%	81	37	41
23	8	4	538	538	538	538	538	538	0%	0%	0%	60	75	97
24	8	4	558	558	558	558	558	558	0%	0%	0%	27	23	23
25	8	4	539	539	539	539	539	539	0%	0%	0%	747	602	661
26	10	5	667	762	762	762	762	762	12%	0%	0%	5400	3014	2494
27	10	5	504	514	564	635	635	639	21%	19%	12%	5400	5400	5400

Tabela 4 – Resultados dos problemas de pequeno porte – formulação alternativa

			Limitante Inferior			Lim. Superior			*GAP*			Tempo (s)		
28	10	5	553	586	648	648	648	648	15%	10%	0%	5400	5400	4832
29	10	5	732	732	732	732	732	732	0%	0%	0%	1134	1636	1183
30	10	5	500	507	424	629	629	629	21%	19%	33%	5400	5400	5400
31	10	5	690	690	551	690	690	698	0%	0%	21%	4852	3685	5400
32	10	5	601	664	664	664	664	664	9%	0%	0%	5400	4022	3312
33	10	5	531	543	628	628	628	628	15%	14%	0%	5400	5400	3600
34	10	5	547	644	644	644	644	644	15%	0%	0%	5400	2651	4116
35	10	5	572	590	662	662	662	662	14%	11%	0%	5400	5400	2234
36	10	5	528	533	429	654	654	656	19%	19%	35%	5400	5400	5400
37	10	5	581	590	406	642	642	642	10%	8%	37%	5400	5400	5400
38	10	5	569	662	662	662	662	662	14%	0%	0%	5400	2812	3427
39	10	5	603	651	651	651	651	651	7%	0%	0%	5400	4121	2856
40	10	5	700	700	700	700	700	700	0%	0%	0%	5044	3052	2952
41	10	5	581	684	684	684	684	684	15%	0%	0%	5400	3023	3326
42	10	5	659	659	659	659	659	659	0%	0%	0%	2163	1319	1344
43	10	5	508	513	536	651	651	651	22%	21%	18%	5400	5400	5400
44	10	5	558	674	674	680	674	674	18%	0%	0%	5400	3225	4336
45	10	5	632	645	556	645	645	645	2%	0%	14%	5400	2768	5400

Analisando a Tabela 4, pode-se observar que, com a formulação alternativa, também foi possível encontrar a solução ótima para todos os problemas com 8 tarefas que possuíam a restrição de sincronismo de tarefas. A nova formulação, em média, resolveu os problemas com 8 tarefas mais rapidamente (tempo médio de 114 segundos, perante 121 segundos para o caso D, e 179 segundos para o caso C). Porém, o caso E foi mais rápido que o caso D em 9 problemas (houve 2 empates), enquanto o caso D foi mais rápido que o caso E em 14 ocasiões.

Para os problemas com 10 tarefas, foi possível chegar ao ótimo com a formulação alternativa em 13 dos 20 casos, em um tempo médio de quase 3080 segundos. Além disso, para os sete problemas em que não foi possível encontrar a solução ótima, a média dos *GAPs* entre o limitante superior e o limitante inferior encontrada foi de 24%. O caso D encontrou 12 soluções ótimas em um tempo médio de pouco mais de 2940 segundos, e obteve um *GAP* médio de 15% para os 8 problemas em que a solução ótima não foi encontrada.

5 MÉTODOS DE RESOLUÇÃO

Como foi visto no Capítulo 4, o modelo matemático proposto é geralmente capaz de resolver apenas problemas de pequeno porte.

Para a resolução de problemas reais, de maiores dimensões, são propostos, nesta Seção, algoritmos baseados nas metaheurísticas apresentadas no Capítulo 2.

Durante todo este capítulo, a representação das soluções será feita sob a forma de matriz, conforme foi apresentado na Seção 3.2.

5.1 GERAÇÃO DE UMA SOLUÇÃO INICIAL

Nesta primeira etapa de resolução, é gerada uma solução inicial viável para o problema.

A presença da restrição de sincronismo de execução de tarefas dificulta a obtenção de soluções viáveis de forma aleatória.

Para ilustrar tal dificuldade, desenvolveu-se um algoritmo para gerar uma solução inicial de forma totalmente aleatória.

A cada iteração desse algoritmo, escolhia-se uma tarefa e uma máquina, aleatoriamente. Cada tarefa escolhida ocupava a primeira posição disponível da máquina selecionada.

O algoritmo terminava depois que todas as tarefas tivessem sido alocadas em algum forno.

Foram geradas 100 soluções para problemas com 25 tarefas e 11 pares, mas nenhuma delas era viável.

Vale ressaltar que esta dificuldade aumenta com o crescimento das dimensões do problema, isto é, quanto maior for a quantidade de pares do problema, menor a chance de se obter uma solução viável de forma aleatória.

Dessa forma, como existe uma grande chance de se obter uma solução inviável ao se gerar uma solução aleatoriamente, decidiu-se utilizar uma heurística que aproveitasse a formulação matemática do problema. A heurística escolhida foi a *relax-and-fix* (Seção 2.2.1).

A estratégia utilizada para criar as partições, foi separar as variáveis de acordo com as posições que as tarefas podem ocupar nos fornos (Figura 23(a)).

Dessa forma, na primeira iteração, apenas as variáveis referentes às primeiras posições de cada forno são mantidas como binárias (Figura 23(b) – R representa as variáveis relaxadas; e B, as variáveis binárias).

Na segunda iteração, as variáveis da primeira posição são fixadas nos valores obtidos na primeira iteração, e apenas as variáveis da segunda posição são mantidas como binárias (Figura 23(c) – R representa as variáveis relaxadas; B, as variáveis binárias; e F, as variáveis fixadas. O processo é repetido até que todas as variáveis sejam fixadas.

Figura 23 – Obtenção de uma solução inicial pelo método *relax-and-fix*
(a) separação das variáveis; (b) primeira iteração; (c) segunda iteração

		Fornos		
		1	2	3
Posições	1	1ª Partição		
	2	2ª Partição		
	3	3ª Partição		
	4	4ª Partição		
	5	5ª Partição		

(a)

=>

		Fornos		
		1	2	3
Posições	1	B	B	B
	2	R	R	R
	3	R	R	R
	4	R	R	R
	5	R	R	R

(b)

=>

		Fornos		
		1	2	3
Posições	1	F	F	F
	2	B	B	B
	3	R	R	R
	4	R	R	R
	5	R	R	R

(c)

Fonte: (Autor)

Para dificultar a obtenção de uma solução inviável, adiciona-se um novo conjunto de restrições à formulação original:

- Diferença máxima permitida entre posições de casca e núcleo de um mesmo cilindro:

$$\sum_{j=1}^{m}\sum_{k=1}^{b}\sum_{l=1}^{n} j.X_{i,j,k,l} - \sum_{j=1}^{m}\sum_{k=1}^{b}\sum_{l=1}^{n} j.X_{i+1,j,k,l} \geq -D \qquad i = 1,3,5,\ldots \qquad (75)$$

$$\sum_{j=1}^{m}\sum_{k=1}^{b}\sum_{l=1}^{n} j.X_{i,j,k,l} - \sum_{j=1}^{m}\sum_{k=1}^{b}\sum_{l=1}^{n} j.X_{i+1,j,k,l} \leq D \qquad i = 1,3,5,\ldots \qquad (76)$$

Como a principal causa da obtenção de soluções inviáveis é a inversão da ordem de processamento entre casca e núcleo de dois cilindros que ocupam dois fornos (Figura 24(a)), a ideia das expressões (75) e (76) é reduzir a possibilidade de que isto ocorra.

Figura 24 – Soluções: (a) inviável; (b) viável (D = 0)

		Fornos		
		1	2	3
Posições	1			4
	2	1		
	3	3		
	4			2

(a)

		Fornos		
		1	2	3
Posições	1			
	2	1		2
	3	3		4
	4			

(b)

Fonte: (Autor)

Fazendo-se D = 0, evita-se que este problema ocorra.

Entretanto, como as expressões (75) e (76) não garantem que o problema original seja preservado, deve-se tomar cuidado ao escolher o valor de D.

Quanto menor o valor de D, menores as chances de se obter uma solução inviável; contudo, maiores serão as possibilidades de se encontrar uma solução de pior qualidade. Como exemplo, ao fixar D = 0 em um problema com número ímpar de fornos, onde há apenas cilindros compostos de duas partes, um dos fornos ficará obrigatoriamente vazio.

Resultados empíricos mostram que, em problemas com um número impar de fornos e apenas cilindros compostos de duas partes, é conveniente adotar D = 1.

Já em problemas com um número par de fornos ou com pelo menos um cilindro constituído de apenas uma parte, é conveniente adotar D = 0 para a primeira posição e D = 1 para as demais. Este efeito é conseguido adotando-se D = 1 nas expressões (75) e (76) e adicionando-se as expressões (77) e (78):

$$\sum_{k=1}^{b}\sum_{l=1}^{n} X_{i,1,k,l} \leq \sum_{k=1}^{b}\sum_{l=1}^{n} X_{i+1,1,k,l} \qquad i = 1, 3, 5, \ldots \tag{77}$$

$$\sum_{k=1}^{b}\sum_{l=1}^{n} X_{i,1,k,l} \leq \sum_{k=1}^{b}\sum_{l=1}^{n} X_{i-1,1,k,l} \qquad i = 2, 4, 6, \ldots \tag{78}$$

Caso seja encontrada uma solução inviável, pode-se adotar um novo valor de D, ou adicionar uma nova restrição com base nos valores obtidos na solução inviável, e resolver o problema novamente.

5.2 ESTRUTURAS DE VIZINHANÇAS

Como será visto na Seção 5.4, os algoritmos propostos neste trabalho utilizam diversos tipos de movimentos para explorar o espaço de soluções e gerar novas soluções vizinhas de uma determinada solução. Cada um destes movimentos será apresentado nesta seção.

5.2.1 Troca de posição de duas tarefas

Nesta estrutura de vizinhança, uma tarefa pode ser trocada de posição com outra dentro de um mesmo forno, ou com outra tarefa de um forno diferente. A Figura 25 ilustra estas duas possibilidades.

Figura 25 - Troca de posição de duas tarefas: (a) solução atual;
(b) troca de tarefas no mesmo forno; (c) troca de tarefas de fornos diferentes

	F1	F2	F3
P1	1	2	3
P2	4	5	6
P3	10	8	9
P4	7		

(a)

	F1	F2	F3
P1	**4**	2	3
P2	**1**	5	6
P3	10	8	9
P4	7		

(b)

	F1	F2	F3
P1	**5**	2	3
P2	4	**1**	6
P3	10	8	9
P4	7		

(c)

Fonte: (Autor)

5.2.2 Troca de posição de dois pares

Devido à restrição de sincronismo de execução de tarefas, a estrutura de vizinhança do item 5.2.1 apresenta um inconveniente. Na Figura 26(a), supondo que os tempos de processamento não sejam muito diferentes uns dos outros, ao trocar a tarefa 6 com a tarefa 9, será encontrada uma solução pior, mesmo que haja uma melhoria dentro do forno 3. Isto ocorre porque, estando o cilindro 9 atrelado à tarefa 10, a tarefa 6 só terá início após o término de ambos. Na prática, seria como se o forno 3 passasse a processar 4 tarefas, neste exemplo.

Para contornar este problema, e incluir esta característica do problema na busca local, nesta nova estrutura de vizinhança, cada conjunto de tarefas pode ser trocado de posição com outro, de duas formas diferentes. No exemplo da Figura 26, o cilindro constituído pelo par de

tarefas 5 - 6 será trocado de posição com o cilindro formado pelo par de tarefas 9 – 10. Originalmente, a tarefa 5 ocupa a segunda posição do forno 2; a tarefa 6, a segunda posição do forno 3; a tarefa 9, a terceira posição do forno 3; e a tarefa 10, a terceira posição do forno 1 (Figura 26(a)). Na primeira troca proposta (Figura 26(b)), a tarefa ímpar do primeiro cilindros (tarefa 5) troca de posição com a tarefa par do segundo cilindro (tarefa 10), e a tarefa par do primeiro cilindro (tarefa 6) troca de posição com a tarefa ímpar do segundo cilindro (tarefa 9). Na segunda troca proposta, (Figura 26(c), a tarefa ímpar do primeiro cilindro (tarefa 5) troca de posição com a tarefa ímpar do segundo cilindro (tarefa 9), e a tarefa par do primeiro cilindro troca de posição com a tarefa par do segundo cilindro (tarefa 10).

Figura 26 - Troca de posição entre pares:
(a) solução atual; (b) 1ª Troca; (c) 2ª Troca

	F1	F2	F3
P1	1	2	3
P2	4	5	6
P3	10	8	9
P4	7		

(a)

	F1	F2	F3
P1	1	2	3
P2	4	**10**	**9**
P3	**5**	8	**6**
P4	7		

(b)

	F1	F2	F3
P1	1	2	3
P2	4	**9**	**10**
P3	**6**	8	**5**
P4	7		

(c)

Fonte: (Autor)

5.2.3 Troca de tarefas entre fornos

Como os tempos de processamento são diferentes entre os fornos, nesta estrutura de vizinhança, todas as tarefas de um forno podem ser trocadas com as de outro forno, mantendo as mesmas posições (Figura 27).

Figura 27 - Troca de posição entre fornos:
(a) solução atual; (b) troca realizada

	F1	F2	F3
P1	1	2	3
P2	4	5	6
P3	10	8	9
P4	7		

(a)

	F1	F2	F3
P1	**2**	**1**	3
P2	**5**	**4**	6
P3	**8**	**10**	9
P4		**7**	

(b)

Fonte: (Autor)

5.2.4 Inserção de uma tarefa numa nova posição

Nesta estrutura de vizinhança, uma tarefa é removida de sua posição atual e é reinserida em outra posição, podendo ser do mesmo forno em que se encontrava ou de outro diferente. As demais tarefas se rearranjam dentro de seus respectivos fornos para acomodar esta alteração (Figura 28).

Figura 28 – Inserção de uma tarefa numa nova posição: (a) solução atual;
(b) inserção no mesmo forno; (c) inserção em fornos diferentes

	F1	F2	F3
P1	1	2	3
P2	**4**	5	6
P3	10	8	9
P4	7		

(a)

	F1	F2	F3
P1	1	2	3
P2	10	5	6
P3	7	8	9
P4	**4**		

(b)

	F1	F2	F3
P1	1	2	3
P2	10	5	6
P3	7	**4**	9
P4		8	

(c)

Fonte: (Autor)

5.2.5 Destruição e reconstrução

Nesta estrutura de vizinhança, a solução atual é desmembrada iterativamente em três partes, respeitando-se a sequência das tarefas ao longo das posições e dos fornos. A seguir, gera-se uma nova solução, alterando-se a sequência das partes que foram desmembradas anteriormente, mas mantendo-se a sequência das tarefas dentro de cada parte, conforme pode ser visto no exemplo da Figura 29.

Figura 29 – Destruição e reconstrução: (a) solução inicial;
(b) desmembramento da solução; (c) nova solução

	F1	F2	F3
P1	1	2	3
P2	4	5	6
P3	7	8	9
P4	10	11	12

(a)

1	2	3		
4	5	6	7	8
9	10	11	12	

(b)

	F1	F2	F3
P1	9	10	11
P2	12	4	5
P3	6	7	8
P4	1	2	3

(c)

Fonte: (Autor)

O número de tarefas em cada parte varia a cada iteração, de tal modo que todas as combinações sejam testadas.

5.3 APRESENTAÇÃO DOS ALGORITMOS

Todos os algoritmos propostos utilizam o *relax-and-fix* apresentado na Seção 5.1 para gerar uma solução inicial.

Depois de encontrá-la, é necessário aplicar um procedimento de busca local, e outro de perturbação, além de definir um critério de aceitação que indicará a partir de qual solução cada perturbação será realizada.

Nesta Seção será apresentado cada um destes procedimentos e critérios para os algoritmos criados, assim como suas respectivas estruturas de vizinhança.

Vale salientar de antemão que para aumentar as chances de escapar dos ótimos locais, todos os algoritmos criados utilizam a ideia básica do VNS (Seção 2.2.3) de utilizar mais de uma estrutura de vizinhança.

Além disso, as quantidades de trocas e inserções realizadas durante as etapas de perturbação foram adotadas a partir de alguns testes preliminares realizados em instâncias menores do problema. O mesmo vale para os critérios de aceitação em que não há melhoria na solução.

5.3.1 Algoritmos baseados na metaheurística ILS

Nesta Seção foram desenvolvidos quatro algoritmos baseados na metaheurística ILS.

Como será visto a seguir, a principal diferença entre eles é a estrutura de vizinhança e o mecanismo de perturbação utilizados.

5.3.1.1 Algoritmo ILS 1

A busca local é realizada utilizando como vizinhança a troca de posição de duas tarefas (Seção 5.2.1) e a troca de tarefas entre dois fornos (Seção 5.2.3). Todas as trocas possíveis são avaliadas.

A melhor solução encontrada por esta busca local é selecionada e comparada com a solução corrente, sendo aceita sempre que o melhor vizinho encontrado for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais melhora no resultado.

Quando o processo encontra um ótimo local, realiza-se a operação de perturbação, a qual utiliza como vizinhança a troca de posição de dois elementos (Seção 5.2.1).

A cada perturbação, a quantidade de trocas realizada é determinada gerando–se um número aleatoriamente, compreendido entre os valores um e o número total de tarefas do problema.

Para tentar escapar dos ótimos locais, a solução gerada pela perturbação é aceita depois de 10.n tentativas sem melhora no resultado, mesmo que esta seja pior que a atual.

O processo é encerrado após um tempo máximo de execução.

5.3.1.2 Algoritmo ILS 2

Este algoritmo segue basicamente os mesmos passos do anterior, mas possui algumas diferenças na estrutura de vizinhança, no mecanismo de perturbação e no critério de aceitação utilizados.

A busca local utiliza como vizinhança a troca de posição de duas tarefas (Seção 5.2.1), a troca de tarefas entre dois fornos (Seção 5.2.3), e a troca de posição de dois pares (Seção 5.2.2). Todas as trocas possíveis são avaliadas.

A melhor solução encontrada entre todas as anteriores é selecionada e comparada com a solução corrente, sendo aceita sempre que o melhor vizinho encontrado for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais evolução no resultado.

Quando um ótimo local é alcançado, utiliza-se a troca de posição de dois pares (Seção 5.2.2) para realizar a operação de perturbação.

A cada perturbação, a quantidade de trocas realizada é determinada gerando–se um número aleatoriamente, compreendido entre os valores um e o número total de pares do problema.

Com o intuito de tentar escapar dos ótimos locais, a solução gerada pela perturbação é aceita depois de um número de tentativas sem melhora na solução igual a seis vezes o número de pares do problema, mesmo que esta seja pior que a atual.

O processo é encerrado após um tempo máximo de execução.

5.3.1.3 Algoritmo ILS 3

Este algoritmo utiliza as mesmas três vizinhanças do algoritmo ILS 2: troca de posição de duas tarefas (Seção 5.2.1), troca de tarefas entre dois fornos (Seção 5.2.3), e troca de posição de dois pares (Seção 5.2.2); e da mesma forma que aquele, avalia todos os vizinhos possíveis.

A melhor solução encontrada entre todas as anteriores é escolhida e comparada com a solução corrente, sendo aceita sempre que o melhor vizinho encontrado for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais alteração no resultado.

Quando o algoritmo encontra um ótimo local, realiza-se a operação de perturbação, constituída de duas etapas.

Na primeira etapa, utiliza-se a operação de destruição e reconstrução (Seção 5.2.5). A cada nova solução gerada por esta operação, realiza-se uma nova busca local completa, utilizando as mesmas estruturas de vizinhança citadas anteriormente. É importante destacar que a estrutura de vizinhança de destruição e reconstrução, ao ser utilizada somente durante a fase de perturbação, complementa a busca local e dificulta o retorno à solução anterior quando a busca local for aplicada novamente.

Depois que todas as possibilidades de destruição e reconstrução forem esgotadas sem que uma solução melhor seja encontrada, tem início a segunda etapa, que utiliza a troca de posição de dois pares (Seção 5.2.2). A cada perturbação, a quantidade de trocas realizada é determinada gerando–se um número aleatoriamente, compreendido entre os valores um e o número total de pares do problema.

Com o intuito de tentar escapar dos ótimos locais, a solução gerada por esta última perturbação é sempre aceita, mesmo que esta seja pior que a solução atual.

O processo é encerrado após um tempo máximo de execução.

5.3.1.4 Algoritmo ILS 4

Este algoritmo apresenta pequenas alterações em relação ao modelo da Seção 5.4.1.3.

Para realizar a busca local, o algoritmo utiliza, além das vizinhanças do algoritmo ILS 3 - troca de posição de duas tarefas (Seção 5.2.1), troca de tarefas entre dois fornos (Seção 5.2.3), e troca de posição de dois pares (Seção 5.2.2), a vizinhança inserção de uma tarefa

numa nova posição (Seção 5.2.4). Em todas elas, é realizada uma busca completa entre os vizinhos possíveis.

Depois de encontrar o melhor vizinho, este é comparado com a solução corrente, sendo aceito sempre que for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais nenhuma modificação no resultado.

Quando um ótimo local é alcançado, realiza-se a operação de perturbação, constituída das duas etapas descritas na Seção 5.3.1.3. A única diferença entre elas é que a quantidade de trocas realizada na segunda etapa da perturbação é determinada gerando–se um número aleatoriamente, compreendido entre os valores três e o número total de pares do problema. Assim como no algoritmo anterior, a estrutura de vizinhança de destruição e reconstrução, ao ser utilizada somente durante a fase de perturbação, complementa a busca local e dificulta o retorno à solução anterior quando a busca local for aplicada novamente.

Da mesma forma que os algoritmos anteriores, o algoritmo termina após um tempo máximo de processamento.

5.3.2 Algoritmo baseado na metaheurística VNS

O algoritmo VNS utilizado neste trabalho baseia-se no método GVNS, apresentado na Seção 2.2.3.

Para gerar os vizinhos da solução corrente, utiliza-se a troca de posição de duas tarefas (Seção 5.2.1).

Inicialmente, gera-se aleatoriamente um vizinho para esta solução, e realiza-se uma busca local, até que um ótimo local seja encontrado. Se a solução encontrada não for melhor que a solução corrente, gera-se, aleatoriamente, um vizinho mais distante, isto é, aumenta-se o número de trocas ao longo das iterações para gerar novas soluções vizinhas da solução corrente; e realiza-se uma nova busca local a partir deste ponto. Quando a solução encontrada for melhor que a solução atual, ela a substitui, e repete-se o procedimento desde o início.

Para realizar a busca local, utiliza-se o método VND apresentado na Seção 2.2.3, o qual emprega a inserção de uma tarefa numa nova posição (Seção 5.2.4), a troca de posição de duas tarefas (Seção 5.2.1), e a troca de posição de dois pares (Seção 5.2.2).

O processo é encerrado após um tempo máximo de execução.

5.3.3 Algoritmos híbridos baseados nas metaheurísticas ILS e VND

Nesta Seção foram desenvolvidos dois algoritmos que combinam as metaheurísticas ILS e VND.

Como será visto a seguir, a principal diferença entre eles é o mecanismo de perturbação utilizado.

5.3.3.1 Algoritmo ILS-VND 1

Para realizar a busca local, o algoritmo emprega o método VND apresentado na Seção 2.2.3. As estruturas de vizinhança utilizadas são: inserção de uma tarefa numa nova posição (Seção 5.2.4), troca de posição de duas tarefas (Seção 5.2.1), e troca de posição de dois pares (Seção 5.2.2). Em todas elas, é realizada uma busca completa entre os vizinhos possíveis.

A cada iteração, compara-se o resultado da busca local com a solução corrente, sendo o melhor vizinho aceito sempre que for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais nenhuma modificação no resultado.

Quando o processo encontra um ótimo local, realiza-se a operação de perturbação, constituída de trocas de posições de pares de tarefas (Seção 5.2.2). A quantidade de trocas realizada nesta operação é determinada gerando–se um número aleatoriamente, compreendido entre os valores três e o número total de pares do problema.

Com o intuito de tentar escapar dos ótimos locais, a solução gerada por esta perturbação é sempre aceita, mesmo que esta seja pior que a solução atual.

Da mesma forma que os modelos anteriores, o algoritmo termina após um tempo máximo de processamento.

5.3.3.2 Algoritmo ILS-VND 2

Para realizar a busca local, o modelo emprega o método VND apresentado na Seção 2.2.3. As estruturas de vizinhança utilizadas são as mesmas do modelo anterior: inserção de uma tarefa numa nova posição (Seção 5.2.4), troca de posição de duas tarefa (Seção 5.2.1), e troca de posição de dois pares (Seção 5.2.2). Em todas elas, é realizada uma busca completa entre os vizinhos possíveis.

A cada iteração, compara-se o resultado da busca local com a solução corrente, sendo o melhor vizinho aceito sempre que for melhor que a solução vigente. A seguir, realiza-se uma nova busca local, até que não haja mais nenhuma modificação no resultado.

Quando um ótimo local é encontrado, realiza-se a operação de perturbação sobre a melhor solução conhecida. Ao contrário do algoritmo anterior, a cada iteração, utiliza-se uma estrutura de vizinhança diferente para gerar a perturbação (inserção de uma tarefa numa nova posição - Seção 5.3.4, troca de posição de duas tarefas - Seção 5.2.1, e troca de posição de dois pares - Seção 5.2.2). Em cada uma, gera–se um número aleatoriamente, compreendido entre os valores três e o número total de pares do problema, que determina o número de trocas ou inserções que serão realizadas nesta operação.

Da mesma forma que os modelos anteriores, o algoritmo termina após um tempo máximo de processamento.

5.3.4 Algoritmos baseados na metaheurística LNS

Nesta Seção foram desenvolvidos dois algoritmos que utilizam o método LNS.

Como será visto a seguir, a principal diferença entre eles é a quantidade de tarefas selecionadas durante a fase de destruição.

5.3.4.1 Algoritmo LNS 1

Na fase de destruição, dois pares de tarefas e outras duas tarefas quaisquer, totalizando seis elementos, são removidos da solução atual de forma aleatória.

Na fase de reconstrução, os elementos removidos na etapa anterior são reinseridos em novas posições, com o auxílio do modelo matemático do caso D (Seção 4.7), até que uma nova solução seja obtida. Desta forma, o LNS proposto funciona como um método *fix-and-optimize*.

Uma solução é aceita somente se for melhor que a melhor conhecida até o momento.

O algoritmo termina depois de atingir um tempo máximo de execução.

5.3.4.2 Algoritmo LNS 2

Este modelo é idêntico ao anterior, mas remove sete elementos da solução corrente na fase de destruição, ao invés de seis. Os sete elementos são constituídos por dois pares de tarefas e três outras tarefas quaisquer, todas escolhidas de forma aleatória.

O Quadro 5 apresenta um resumo com as vizinhanças utilizadas nos algoritmos propostos durante as etapas de perturbação e busca local.

Quadro 5 – Vizinhanças dos algoritmos propostos

Algoritmo	Busca Local	Perturbação
ILS 1	5.2.1 e 5.2.3	5.2.1
ILS 2	5.2.1, 5.2.2 e 5.2.3	5.2.2
ILS 3	5.2.1, 5.2.2 e 5.2.3	5.2.2 e 5.2.5
ILS 4	5.2.1, 5.2.2, 5.2.3 e 5.2.4	5.2.2 e 5.2.5
VNS	5.2.1, 5.2.2 e 5.2.4	-
ILS-VND 1	5.2.1, 5.2.2 e 5.2.4	5.2.2
ILS-VND 2	5.2.1, 5.2.2 e 5.2.4	5.2.1, 5.2.2 e 5.2.4
LNS 1	Remoção e inserção de 6 elementos	
LNS 2	Remoção e inserção de 7 elementos	

5.4 TRATAMENTO DE SOLUÇÕES INVIÁVEIS

No problema estudado, qualquer estrutura de vizinhança poderá gerar uma solução inviável, dependo das tarefas e das posições que forem movimentadas.

De uma forma geral, os movimentos de trocas de posições de dois pares, e de trocas de tarefas entre fornos geram menos soluções inviáveis do que os outros movimentos. Em alguns problemas específicos, estes movimentos podem gerar apenas soluções viáveis (por exemplo, em problemas onde nenhuma tarefa excede a capacidade de nenhum forno, a troca de posições entre pares e a troca de posições entre fornos jamais gerará uma inviabilidade). Além disso, mesmo quando são obtidas soluções inviáveis através destes movimentos, estas são facilmente revertidas.

Os demais movimentos, embora gerem mais soluções inviáveis, são importantes, pois ajudam a diversificar a busca.

A estratégia utilizada para lidar com as soluções inviáveis é a seguinte: se for encontrada uma solução inviável durante o processo de busca local, ou de perturbação, adota-se um valor enorme em sua função-objetivo, para que os algoritmos tratem todas as soluções inviáveis como uma solução ruim. Com isso, qualquer solução viável será sempre melhor que qualquer solução inviável, e o algoritmo caminhará naturalmente para uma solução viável.

Assim, durante o processo de busca local, uma solução inviável nunca é aceita.

No caso das perturbações, as soluções inviáveis são aceitas em alguns casos e rejeitadas em outros.

As soluções geradas durante o processo de perturbação com as vizinhanças 5.2.1 e 5.2.2 são sempre aceitas, mesmo quando inviáveis. Muitas vezes, a aceitação destas soluções inviáveis permite encontrar soluções viáveis que não seriam obtidas partindo-se diretamente de uma solução viável.

Como a estrutura de vizinhança 5.2.5 pode gerar inviabilidades difíceis de serem revertidas, todas as soluções geradas por ela durante o processo de perturbação (inclusive as inviáveis) passam por um processo de busca local separadamente do algoritmo principal. Se o resultado desta busca local for uma solução melhor que a melhor conhecida, ela é aceita e vai para o algoritmo principal. Caso contrário, o algoritmo principal continua com a solução anterior à perturbação.

6. RESULTADOS COMPUTACIONAIS

Neste capítulo serão apresentados os resultados obtidos através da modelagem matemática do problema e dos algoritmos propostos.

Os resultados foram gerados a partir dos modelos matemáticos dos casos B, D e E, descritos nas seções 4.3, 4.7 e 4.9, respectivamente, além dos quatro algoritmos baseados na metaheurística ILS, apresentados na Seção 5.4.1; do algoritmo VNS, descrito na Seção 5.4.2; dos dois algoritmos VNS-ILS, exibidos na Seção 5.4.3; nos dois algoritmos LNS, apresentados na Seção 5.4.4; e do *relax-and-fix* (RF), apresentado na Seção 5.1.

Os modelos matemáticos foram implementados no software CPLEX 12.1, enquanto os algoritmos das metaheurísticas, na linguagem de programação C++.

Os testes computacionais foram executados em um computador com processador Intel Core 2 Duo de 3 GHz, e memória de 2 GB.

Inicialmente, os algoritmos propostos baseados em metaheurísticas foram utilizados para resolver as instâncias pequenas do Capítulo 4. Todos foram capazes de encontrar as soluções ótimas de todos os problemas.

A seguir, foram resolvidos 45 problemas reais, referentes à produção diária de uma fábrica, contendo de 20 a 25 tarefas (cujos tempos de processamento, já acrescidos dos tempos de *set-up*, variavam de 100 a 275 minutos) e cinco máquinas.

Os testes computacionais das instâncias reais foram divididos em dois grupos: no primeiro, as instâncias foram resolvidas por cada um dos algoritmos propostos apenas uma vez, numa rodada longa que possuía um tempo máximo de processamento definido em 5400 segundos; no segundo, foram realizadas 10 rodadas curtas utilizando as mesmas instâncias do grupo anterior, num tempo máximo de processamento definido em 600 segundos por rodada.

6.1 RESULTADOS OBTIDOS COM AS RODADAS LONGAS

Esta seção apresenta os limitantes inferiores obtidos pelos modelos matemáticos, as soluções obtidas por cada método, os *GAPs* entre estas soluções e o melhor limitante inferior conhecido, e o tempo de processamento gasto para encontrar a melhor solução.

A Tabela 5 apresenta os limitantes inferiores gerados pelos modelos matemáticos após o término do tempo de execução.

Tabela 5 – Limitantes inferiores

Prob	n	Pares	Caso B	Caso D	Caso E
1	20	7	173	714	139
2	20	7	64	711	136
3	20	8	159	600	119
4	20	8	148	616	129
5	20	9	36	601	128
6	20	9	44	586	115
7	20	9	158	556	120
8	20	9	133	623	126
9	20	9	26	606	117
10	21	9	159	594	118
11	21	9	162	654	121
12	21	9	164	671	126
13	21	10	82	651	116
14	21	10	159	623	144
15	22	8	38	680	121
16	22	9	159	640	129
17	22	9	161	727	129
18	22	10	83	629	119
19	22	10	162	640	122
20	22	10	158	639	117
21	22	10	158	640	124
22	22	10	158	638	124
23	22	10	158	633	118
24	23	10	26	655	117
25	23	10	160	658	118
26	23	10	158	638	129
27	23	10	44	639	117
28	23	10	50	684	121
29	23	10	160	696	124
30	23	11	160	681	120
31	23	11	185	733	120
32	23	11	118	747	121
33	24	9	54	709	121
34	24	9	45	672	113
35	24	10	44	700	119

Tabela 5 – Limitantes inferiores

Prob	n	Pares	Caso B	Caso D	Caso E
36	24	10	66	702	117
37	25	11	33	838	132
38	25	11	31	732	118
39	25	11	30	732	119
40	25	11	44	727	120
41	25	11	30	731	120
42	25	11	41	681	116
43	25	11	41	726	135
44	25	11	161	733	117
45	25	11	45	726	135

A Tabela 5 confirma o que foi observado na Seção 4.3: os limitantes inferiores gerados pelo modelo matemático do caso B são de baixíssima qualidade. Ao serem introduzidas as expressões (27), (28), (29) e (30), obtêm-se limitantes inferiores muito melhores, mesmo em problemas reais.

Do mesmo modo, pode-se constatar que os limitantes inferiores obtidos pelo caso E (que havia obtido bons resultados em instâncias menores) também são de baixíssima qualidade.

A Tabela 6 apresenta as soluções encontradas por cada algoritmo após o término do tempo de execução. Os melhores resultados encontram-se destacados em negrito.

Tabela 6 - Soluções dos problemas reais

Prob	n	Pares	B	D	E	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	-	774	836	2444	**734**	736	**734**	**734**	**734**	**734**	**734**	735	735
2	20	7	-	783	808	824	736	**735**	**735**	**735**	**735**	**735**	**735**	757	741
3	20	8	1299	649	-	842	**626**	**626**	**626**	**626**	**626**	**626**	**626**	639	**626**
4	20	8	-	678	-	759	**639**	**639**	**639**	**639**	**639**	**639**	**639**	660	653
5	20	9	-	713	730	780	**660**	**660**	**660**	**660**	**660**	**660**	**660**	685	662
6	20	9	-	662	-	703	628	**625**	**625**	**625**	**625**	**625**	**625**	649	633
7	20	9	-	710	-	719	628	**621**	**621**	**621**	**621**	**621**	**621**	645	648
8	20	9	-	846	-	728	680	678	**676**	**676**	**676**	**676**	**676**	703	694
9	20	9	-	702	-	745	**639**	**639**	**639**	**639**	**639**	**639**	**639**	663	659
10	21	9	1383	697	-	783	646	644	**642**	**642**	**642**	**642**	**642**	665	646
11	21	9	-	845	-	890	701	708	**700**	**700**	**700**	**700**	**700**	726	721

Tabela 6 - Soluções dos problemas reais

Prob	n	Pares	B	D	E	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
12	21	9	-	1014	-	1744	**708**	**708**	**708**	**708**	**708**	**708**	**708**	750	**708**
13	21	10	-	816	-	822	704	**703**	**703**	**703**	**703**	**703**	**703**	710	710
14	21	10	-	881	-	758	697	695	**686**	**686**	695	**686**	695	717	702
15	22	8	1759	803	-	861	723	723	**722**	**722**	724	724	724	741	729
16	22	9	-	902	-	846	683	**682**	686	**682**	686	689	**682**	698	690
17	22	9	-	1016	-	881	**777**	780	**777**	**777**	**777**	**777**	**777**	793	798
18	22	10	-	-	-	2512	711	**700**	**700**	**700**	707	**700**	**700**	727	737
19	22	10	-	843	-	786	**698**	**698**	**698**	**698**	**698**	**698**	**698**	725	**698**
20	22	10	-	909	-	777	710	**700**	**700**	**700**	**700**	**700**	**700**	727	720
21	22	10	1132	927	-	761	692	697	**690**	**690**	691	**690**	**690**	731	691
22	22	10	-	745	-	830	696	**685**	687	**685**	**685**	689	**685**	718	690
23	22	10	-	754	-	753	688	684	684	**671**	**671**	672	684	701	704
24	23	10	-	1074	-	752	**683**	**683**	**683**	**683**	**683**	**683**	**683**	706	709
25	23	10	-	991	-	847	695	699	**693**	**693**	**693**	**693**	**693**	728	727
26	23	10	-	767	-	738	682	**673**	**673**	**673**	**673**	**673**	**673**	694	694
27	23	10	-	795	-	739	678	674	**671**	**671**	675	**671**	**671**	699	686
28	23	10	-	1010	1198	838	748	**721**	**721**	**721**	**721**	736	**721**	772	743
29	23	10	-	1061	-	938	751	748	**746**	**746**	748	747	**746**	770	753
30	23	11	-	1271	-	1109	771	761	762	**758**	761	760	**758**	776	767
31	23	11	-	-	-	821	814	**793**	**793**	**793**	**793**	**793**	**793**	814	814
32	23	11	-	928	-	907	799	795	795	**793**	795	795	795	803	795
33	24	9	-	1018	1113	927	**734**	**734**	**734**	**734**	**734**	736	**734**	747	758
34	24	9	-	896	-	1101	**696**	**696**	**696**	**696**	**696**	**696**	**696**	718	**696**
35	24	10	-	898	-	906	750	**732**	734	**732**	**732**	**732**	**732**	792	761
36	24	10	-	1011	-	785	759	749	749	**748**	749	749	750	772	765
37	25	11	-	1124	-	969	872	870	869	867	867	**866**	**866**	870	868
38	25	11	-	945	-	1141	**770**	**770**	**770**	**770**	**770**	**770**	**770**	**770**	789
39	25	11	-	979	-	1051	791	**770**	**770**	**770**	**770**	771	771	808	796
40	25	11	-	1169	-	917	796	789	781	**774**	778	783	**774**	824	796
41	25	11	-	1162	-	919	782	779	**775**	779	**775**	780	**775**	800	785
42	25	11	-	1145	-	1057	755	**730**	**730**	**730**	**730**	**730**	**730**	737	765
43	25	11	-	1396	-	917	796	782	**781**	**781**	782	793	787	826	815
44	25	11	-	1216	-	881	794	798	784	**779**	**779**	784	**779**	818	811
45	25	11	-	1396	-	917	788	**781**	**781**	**781**	785	794	791	795	816

Analisando as soluções, observa-se a fragilidade do modelo matemático do caso B até mesmo para encontrar uma solução viável. Dos 45 problemas analisados, ele só encontrou uma solução viável em quatro ocasiões, e mesmo assim, de baixa qualidade.

O mesmo ocorreu com o modelo matemático do caso E, que encontrou apenas cinco soluções viáveis de baixa qualidade.

Mais uma vez, as expressões adicionais utilizadas no modelo matemático do caso D melhoraram o resultado obtido: o modelo só não conseguiu encontrar uma solução viável em duas ocasiões.

Entretanto, comparando-se as soluções, observa-se a superioridade das metaheurísticas em relação à formulação matemática, para o tempo de processamento utilizado. Observa-se também que todas as metaheurísticas melhoraram consideravelmente a solução inicial gerada pelo método *relax-and-fix*.

Dentre todos os algoritmos, o ILS 4 foi o que obteve o melhor desempenho, encontrando a melhor solução (dentre aquelas conhecidas) em 43 ocasiões. Ele foi seguido pelo ILS-VND 2, que foi capaz de atingir a melhor solução em 37 problemas; pelo ILS 3 que encontrou a melhor solução em 35 casos; pelo VNS, que encontrou a melhor solução em 31 problemas; pelo ILS-VND 1, que chegou à melhor solução em 29 problemas; pelo ILS 2, que atingiu a melhor solução em 25 ocasiões; pelo ILS 1, que obteve os melhores resultados 12 vezes; pelo LNS 2, que atingiu os melhores resultados por quatro vezes; e pelo LNS 1, que encontrou a melhor solução apenas uma vez.

Os modelos matemáticos e a heurística *relax-and-fix* não foram capazes de encontrar a melhor solução em nenhum dos problemas apresentados.

A Tabela 7 complementa os resultados obtidos, mostrando o *GAP* entre os limitantes superior e inferior. No caso dos algoritmos baseados em metaheurísticas, o *GAP* foi calculado considerando-se a solução encontrada por cada algoritmo e o melhor limitante inferior conhecido.

Tabela 7 - *GAP*s dos problemas reais

Prob	n	Pares	B	D	E	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	100%	8%	83%	71%	3%	3%	3%	3%	3%	3%	3%	3%	3%
2	20	7	100%	9%	83%	14%	3%	3%	3%	3%	3%	3%	3%	6%	4%
3	20	8	88%	8%	100%	29%	4%	4%	4%	4%	4%	4%	4%	6%	4%
4	20	8	100%	9%	100%	19%	4%	4%	4%	4%	4%	4%	4%	7%	6%
5	20	9	100%	16%	82%	23%	9%	9%	9%	9%	9%	9%	9%	12%	9%

Tabela 7 - *GAP*s dos problemas reais

Prob	n	Pares	B	D	E	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
6	20	9	100%	11%	100%	17%	7%	6%	6%	6%	6%	6%	6%	10%	7%
7	20	9	100%	22%	100%	23%	11%	10%	10%	10%	10%	10%	10%	14%	14%
8	20	9	100%	26%	100%	14%	8%	8%	8%	8%	8%	8%	8%	11%	10%
9	20	9	100%	14%	100%	19%	5%	5%	5%	5%	5%	5%	5%	9%	8%
10	21	9	89%	15%	100%	24%	8%	8%	7%	7%	7%	7%	7%	11%	8%
11	21	9	100%	23%	100%	27%	7%	8%	7%	7%	7%	7%	7%	10%	9%
12	21	9	100%	34%	100%	62%	5%	5%	5%	5%	5%	5%	5%	11%	5%
13	21	10	100%	20%	100%	21%	8%	7%	7%	7%	7%	7%	7%	8%	8%
14	21	10	100%	29%	100%	18%	11%	10%	9%	9%	10%	9%	10%	13%	11%
15	22	8	98%	15%	100%	21%	6%	6%	6%	6%	6%	6%	6%	8%	7%
16	22	9	100%	29%	100%	24%	6%	6%	7%	6%	7%	7%	6%	8%	7%
17	22	9	100%	28%	100%	17%	6%	7%	6%	6%	6%	6%	6%	8%	9%
18	22	10	100%	100%	100%	75%	12%	10%	10%	10%	11%	10%	10%	13%	15%
19	22	10	100%	24%	100%	19%	8%	8%	8%	8%	8%	8%	8%	12%	8%
20	22	10	100%	30%	100%	18%	10%	9%	9%	9%	9%	9%	9%	12%	11%
21	22	10	86%	31%	100%	16%	8%	8%	7%	7%	7%	7%	7%	12%	7%
22	22	10	100%	14%	100%	23%	8%	7%	7%	7%	7%	7%	7%	11%	8%
23	22	10	100%	16%	100%	16%	8%	7%	7%	6%	6%	6%	7%	10%	10%
24	23	10	100%	39%	100%	13%	4%	4%	4%	4%	4%	4%	4%	7%	8%
25	23	10	100%	34%	100%	22%	5%	6%	5%	5%	5%	5%	5%	10%	9%
26	23	10	100%	17%	100%	14%	6%	5%	5%	5%	5%	5%	5%	8%	8%
27	23	10	100%	20%	100%	14%	6%	5%	5%	5%	5%	5%	5%	9%	7%
28	23	10	100%	32%	90%	18%	9%	5%	5%	5%	5%	7%	5%	11%	8%
29	23	10	100%	34%	100%	26%	7%	7%	7%	7%	7%	7%	7%	10%	8%
30	23	11	100%	46%	100%	39%	12%	11%	11%	10%	11%	10%	10%	12%	11%
31	23	11	100%	100%	100%	11%	10%	8%	8%	8%	8%	8%	8%	10%	10%
32	23	11	100%	20%	100%	18%	7%	6%	6%	6%	6%	6%	6%	7%	6%
33	24	9	100%	30%	89%	24%	3%	3%	3%	3%	3%	4%	3%	5%	6%
34	24	9	100%	25%	100%	39%	3%	3%	3%	3%	3%	3%	3%	6%	3%
35	24	10	100%	22%	100%	23%	7%	4%	5%	4%	4%	4%	4%	12%	8%
36	24	10	100%	31%	100%	11%	8%	6%	6%	6%	6%	6%	6%	9%	8%
37	25	11	100%	25%	100%	14%	4%	4%	4%	3%	3%	3%	3%	4%	3%
38	25	11	100%	23%	100%	36%	5%	5%	5%	5%	5%	5%	5%	5%	7%
39	25	11	100%	25%	100%	30%	7%	5%	5%	5%	5%	5%	5%	9%	8%

Tabela 7 - *GAP*s dos problemas reais

Prob	n	Pares	B	D	E	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
40	25	11	100%	38%	100%	21%	9%	8%	7%	6%	7%	7%	6%	12%	9%
41	25	11	100%	37%	100%	20%	7%	6%	6%	6%	6%	6%	6%	9%	7%
42	25	11	100%	41%	100%	36%	10%	7%	7%	7%	7%	7%	7%	8%	11%
43	25	11	100%	48%	100%	21%	9%	7%	7%	7%	7%	8%	8%	12%	11%
44	25	11	100%	40%	100%	17%	8%	8%	7%	6%	6%	7%	6%	10%	10%
45	25	11	100%	48%	100%	21%	8%	7%	7%	7%	8%	9%	8%	9%	11%
		Média	99%	29%	98%	24%	7,1%	6,5%	6,3%	6,2%	6,3%	6,4%	6,3%	9,3%	8,2%

Analisando a Tabela 7, observa-se que os *GAPs* nos modelos matemáticos são muito elevados, especialmente nos modelos dos casos B e E (99% e 98%, em média, contra 29% do caso D).

Observa-se também que os *GAP*s do *relax-and-fix*, embora sejam, em geral, menores do que os encontrados pelos modelos matemáticos, continuam apresentando um valor alto (24% em média).

Entre os demais algoritmos, o *GAP* foi, em média, de 6,2% para o ILS 4; 6,3% para o ILS 3, para o ILS-VND 2 e para o VNS; 6,4% para o ILS-VND 1; 6,5% para o modelo ILS 2; 7,1% para o ILS 1; 8,2% para o LNS 2; e 9,3% para o LNS 1. No caso do ILS 4 e do ILS-VND 2, nenhum *GAP* foi superior a 10%. Estes valores são perfeitamente aceitáveis do ponto de vista da aplicação industrial.

A Tabela 8 apresenta o tempo gasto por cada algoritmo para encontrar a melhor solução.

Tabela 8 – Tempo de execução

Prob	n	Pares	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	250	1770	6	205	144	119	1503	3847	32	6
2	20	7	276	1137	45	77	19	75	296	48	11	384
3	20	8	236	103	22	1	17	80	1	19	2	524
4	20	8	243	6	76	122	359	231	641	118	10	12
5	20	9	257	25	324	89	11	44	55	43	4	886
6	20	9	352	340	462	56	58	3	42	34	8	282
7	20	9	295	4	46	265	1	1	27	2	2	117
8	20	9	357	269	76	3114	302	2158	372	1001	1	6

Tabela 8 – Tempo de execução

Prob	n	Pares	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
9	20	9	298	200	11	11	13	16	41	13	2	13
10	21	9	257	96	1	817	302	1661	14	612	2	494
11	21	9	312	1025	9	204	276	149	1229	55	30	35
12	21	9	464	46	2033	1390	149	15	12	12	19	123
13	21	10	289	1041	763	42	1	943	2687	101	37	338
14	21	10	283	282	9	1538	367	155	3647	8	1	9
15	22	8	275	40	141	4858	1809	57	3121	3113	2	142
16	22	9	302	39	12	4028	4681	2615	1248	414	27	29
17	22	9	320	449	9	330	1	107	1083	1167	113	49
18	22	10	265	84	9	197	1301	96	2249	147	50	472
19	22	10	410	955	3	131	1	61	43	36	3	193
20	22	10	339	598	1295	715	1	79	3031	2005	19	282
21	22	10	338	278	60	1011	982	1482	2606	534	6	4635
22	22	10	509	1041	31	984	3404	3571	1336	529	4	1980
23	22	10	395	22	1488	181	2047	3804	2240	25	147	855
24	23	10	445	61	275	1809	1280	347	1654	473	32	296
25	23	10	345	581	35	23	525	5388	1634	19	1	7
26	23	10	455	132	112	365	320	995	3454	28	2	211
27	23	10	347	210	494	4713	221	7	3398	3	343	786
28	23	10	377	584	1748	1933	1850	145	3084	620	1	74
29	23	10	312	269	76	4230	1172	2018	2483	466	189	313
30	23	11	278	2034	36	181	310	78	4339	3739	8	252
31	23	11	311	5	243	590	295	153	594	143	1	11
32	23	11	313	552	219	810	2261	435	1088	19	2	34
33	24	9	379	631	3197	2223	1747	114	3425	85	59	22
34	24	9	254	180	190	1	1	50	112	23	157	82
35	24	10	297	76	3610	550	740	1103	384	1519	5	496
36	24	10	434	230	4607	1	5429	1569	3344	53	3	51
37	25	11	346	651	10	869	1232	309	3870	32	28	1591
38	25	11	433	1083	65	160	57	73	645	16	12	30
39	25	11	406	584	3	373	2266	295	1864	1288	4	552
40	25	11	540	494	24	25	1895	1907	465	404	28	217
41	25	11	625	1039	5320	2709	363	3411	3476	562	155	139
42	25	11	301	19	1633	122	1081	1149	644	145	72	332

Tabela 8 – Tempo de execução

Prob	n	Pares	RF	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
43	25	11	349	831	1767	1214	2905	2705	2229	1250	41	484
44	25	11	545	19	19	3188	517	1060	5139	321	54	20
45	25	11	365	732	395	444	5139	2427	2368	62	7	84
		Média	351	463	689	1042	1063	961	1716	559	39	399

Analisando o tempo médio necessário para que a melhor solução fosse encontrada, observa-se que os algoritmos com os quatro maiores *GAP*s correspondem aos algoritmos com os quatro menores tempos de processamento. Os tempos foram de 39 segundos para o LNS 1, 351 segundos para o *relax-and-fix*, 399 segundos para o LNS 2, e 463 segundos para o ILS 1.

Observa-se ainda que os algoritmos com os seis menores *GAP*s correspondem aos algoritmos com os seis maiores tempos de processamento. Os tempos foram de 559 segundos para o ILS-VND 2, 689 segundos para o ILS 2, 961 segundos para o VNS, 1042 segundos para o ILS 3, 1063 segundos para o ILS 4, e 1716 segundos para o ILS-VND 1.

Vale ressaltar que todos os valores encontrados também são considerados aceitáveis para o ambiente industrial (mesmo o tempo limite de 5400 segundos já seria considerado aceitável).

Comparando-se as metaheurísticas ILS entre si, os algoritmos com maior *GAP* levaram menos tempo para encontrar sua melhor solução. Estes resultados mostram que os algoritmos ILS mais complexos, com mais estruturas de vizinhanças, por exemplo, encontraram resultados melhores, mas a um custo computacional maior. É importante destacar que a média dos *GAPs* passou de 7,1% para 6,2% (uma evolução inferior a 1%), aumentando-se o tempo médio de processamento em 2,3 vezes, de 463 para 1063 segundos.

Da mesma forma, o algoritmo LNS 2 também encontrou soluções melhores que o LNS 1, mas utilizando um tempo de processamento maior, em função de sua maior vizinhança. A média dos *GAPs* passou de 9,3% para 8,2% (uma evolução ligeiramente superior a 1%), aumentando-se o tempo médio de processamento em 10 vezes, de 39 para 399 segundos.

Já o algoritmo ILS-VND 2 encontrou, em média, uma solução 0,1 % melhor que o ILS-VND 1, mas num tempo 3 vezes menor.

A Tabela 9 apresenta um resumo dos resultados obtidos com as rodadas longas.

Tabela 9 – Resumo das rodadas longas

Modelo	Melhor Solução	GAP Médio	Tempo Médio (s)
ILS 4	43	6,2%	1063
ILS-VND 2	37	6,3%	559
ILS 3	35	6,3%	1042
VNS	31	6,3%	961
ILS-VND 1	29	6,4%	1716
ILS 2	25	6,5%	689
ILS 1	12	7,1%	463
LNS 2	4	8,2%	399
LNS 1	1	9,3%	39
Relax-and-fix	0	24%	351
CPLEX D	0	29%	5400
CPLEX E	0	98%	5400

6.2 RESULTADOS OBTIDOS COM AS RODADAS CURTAS

Esta seção apresenta o número de vezes que a melhor solução conhecida foi encontrada por cada algoritmo, a média dos *GAPs* entre as 10 soluções obtidas e o melhor limitante inferior conhecido, e uma análise sobre a dispersão dos resultados.

A Tabela 10 mostra quantas vezes os algoritmos foram capazes de encontrar a melhor solução conhecida após o término do tempo de execução.

Tabela 10 – Total de melhores soluções encontradas

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	3	0	3	5	1	3	5	3	1
2	20	7	3	4	4	10	5	10	10	0	0
3	20	8	2	3	10	10	3	10	10	1	1
4	20	8	3	2	10	6	0	3	10	0	0
5	20	9	0	4	10	10	3	10	10	0	0
6	20	9	0	2	10	10	7	10	10	1	0
7	20	9	3	10	10	10	4	10	10	0	0
8	20	9	1	2	2	10	2	7	7	0	0
9	20	9	3	4	10	10	1	10	10	0	0

Tabela 10 – Total de melhores soluções encontradas

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
10	21	9	1	10	4	10	1	10	5	1	0
11	21	9	0	0	2	10	3	1	10	1	0
12	21	9	3	2	2	10	6	3	10	0	1
13	21	10	1	4	6	10	2	10	10	0	0
14	21	10	0	1	3	3	1	0	0	0	0
15	22	8	0	2	3	4	1	0	0	0	0
16	22	9	0	4	0	2	3	1	1	0	1
17	22	9	3	0	1	10	1	4	7	0	0
18	22	10	0	1	0	5	1	0	2	0	0
19	22	10	2	10	7	10	10	4	10	0	1
20	22	10	0	1	1	10	0	0	1	0	0
21	22	10	0	3	0	3	2	0	10	0	4
22	22	10	0	1	0	1	2	0	5	0	0
23	22	10	0	0	1	2	2	0	0	0	0
24	23	10	0	5	2	2	4	1	10	1	0
25	23	10	0	4	6	10	10	3	10	0	0
26	23	10	0	10	3	10	3	0	10	0	0
27	23	10	0	3	0	10	5	3	2	0	0
28	23	10	0	0	1	2	3	2	4	0	1
29	23	10	0	1	0	0	1	0	1	0	0
30	23	11	0	2	0	2	0	0	0	0	2
31	23	11	0	4	2	10	4	0	10	0	0
32	23	11	0	1	4	2	0	1	1	1	0
33	24	9	2	2	1	3	2	0	3	0	1
34	24	9	2	1	10	10	1	6	10	1	1
35	24	10	0	2	1	3	7	0	1	0	0
36	24	10	0	0	1	0	5	2	0	0	0
37	25	11	0	1	1	2	1	0	2	0	2
38	25	11	0	6	10	10	10	2	10	1	0
39	25	11	0	2	1	2	1	0	2	0	0
40	25	11	0	0	2	1	2	0	1	0	0
41	25	11	0	2	3	0	2	0	2	0	1
42	25	11	0	3	4	5	10	2	4	0	0
43	25	11	0	1	1	0	1	0	0	1	0

Tabela 10 – Total de melhores soluções encontradas

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
44	25	11	0	1	1	3	0	0	1	0	0
45	25	11	0	2	0	2	1	0	2	0	0
		Total	32	123	153	260	134	128	239	12	17

Analisando os resultados da Tabela 10, observa-se que o comportamento dos algoritmos durante as rodadas curtas foi semelhante ao observado durante as rodadas longas. Dentre todos os algoritmos, o ILS 4 foi o que obteve o melhor desempenho, encontrando a melhor solução (dentre aquelas conhecidas) em 260 ocasiões. Ele foi seguido pelo ILS-VND 2, que foi capaz de atingir a melhor solução 239 vezes; pelo ILS 3 que encontrou a melhor solução em 153 casos; pelo VNS, que encontrou a melhor solução em 134 rodadas; pelo ILS-VND 1, que chegou à melhor solução em 128 vezes; pelo ILS 2, que atingiu a melhor solução em 123 ocasiões; pelo ILS 1, que obteve os melhores resultados 32 vezes; pelo LNS 2, que atingiu os melhores resultados por 17 vezes; e pelo LNS 1, que encontrou a melhor solução apenas 12 vezes.

A Tabela 11 complementa os resultados obtidos, mostrando a média dos *GAPs* encontrados ao longo das 10 rodadas.

Tabela 11 – Médias dos *GAPs* das rodadas curtas

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	2,9%	3,1%	2,8%	2,8%	3,1%	2,8%	2,8%	2,8%	2,9%
2	20	7	3,5%	4,0%	3,5%	3,3%	3,4%	3,3%	3,3%	7,1%	4,0%
3	20	8	4,5%	5,1%	4,2%	4,2%	5,1%	4,2%	4,2%	5,9%	5,4%
4	20	8	3,7%	4,2%	3,6%	3,7%	4,5%	3,8%	3,6%	6,9%	5,8%
5	20	9	9,8%	9,5%	8,9%	8,9%	9,4%	8,9%	8,9%	12,4%	9,3%
6	20	9	6,5%	6,6%	6,2%	6,2%	6,3%	6,2%	6,2%	9,6%	7,3%
7	20	9	10,9%	10,5%	10,5%	10,5%	10,7%	10,5%	10,5%	13,4%	14,6%
8	20	9	8,3%	8,0%	7,9%	7,8%	8,4%	7,9%	7,9%	10,4%	10,2%
9	20	9	5,6%	5,4%	5,2%	5,2%	6,0%	5,2%	5,2%	7,8%	8,8%
10	21	9	8,0%	7,5%	7,7%	7,5%	7,9%	7,5%	7,7%	10,3%	8,0%
11	21	9	8,5%	7,3%	7,5%	6,6%	7,3%	6,8%	6,6%	10,1%	10,0%
12	21	9	5,8%	5,9%	6,1%	5,2%	5,3%	5,7%	5,2%	10,6%	6,2%
13	21	10	7,8%	7,5%	7,5%	7,4%	7,8%	7,4%	7,4%	8,5%	8,6%

Tabela 11 – Médias dos *GAPs* das rodadas curtas

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
14	21	10	11,1%	10,0%	9,6%	9,6%	10,0%	10,4%	10,0%	13,8%	10,5%
15	22	8	6,2%	5,9%	6,1%	6,0%	6,1%	6,8%	6,4%	8,5%	6,9%
16	22	9	7,5%	6,5%	7,0%	6,7%	6,5%	7,4%	6,9%	8,7%	7,4%
17	22	9	6,6%	8,6%	6,8%	6,4%	7,1%	6,7%	6,5%	7,9%	8,9%
18	22	10	11,5%	11,1%	11,3%	10,4%	11,5%	11,0%	10,8%	14,0%	14,2%
19	22	10	8,6%	8,3%	8,3%	8,3%	8,3%	8,4%	8,3%	12,7%	9,3%
20	22	10	9,9%	9,2%	9,3%	8,7%	9,4%	9,7%	9,1%	11,7%	11,2%
21	22	10	9,0%	7,6%	7,9%	7,6%	7,7%	9,1%	7,2%	11,9%	7,3%
22	22	10	8,4%	10,4%	7,7%	7,2%	7,3%	7,9%	7,0%	9,7%	7,4%
23	22	10	8,1%	7,0%	7,0%	5,9%	6,7%	7,2%	6,8%	9,8%	10,6%
24	23	10	6,0%	4,2%	4,8%	4,3%	5,0%	4,8%	4,1%	6,4%	8,2%
25	23	10	5,8%	5,2%	5,1%	5,1%	5,1%	5,2%	5,1%	9,2%	9,7%
26	23	10	6,0%	5,2%	5,4%	5,2%	5,6%	5,6%	5,2%	8,0%	9,2%
27	23	10	5,7%	5,0%	5,2%	4,8%	5,0%	5,2%	5,1%	8,2%	7,1%
28	23	10	7,9%	8,1%	6,5%	6,1%	6,5%	5,8%	5,7%	12,4%	8,5%
29	23	10	8,0%	7,9%	7,2%	7,0%	7,8%	7,4%	7,0%	9,6%	7,5%
30	23	11	12,7%	10,4%	12,0%	10,4%	10,8%	11,3%	10,5%	12,2%	10,9%
31	23	11	9,0%	7,9%	8,1%	7,6%	7,8%	8,6%	7,6%	10,0%	10,1%
32	23	11	6,8%	6,0%	6,0%	6,0%	6,2%	6,3%	6,0%	7,1%	6,0%
33	24	9	3,9%	5,5%	3,8%	3,7%	4,4%	4,2%	3,6%	4,7%	6,4%
34	24	9	3,6%	5,3%	3,4%	3,4%	4,0%	3,5%	3,4%	6,3%	4,4%
35	24	10	5,8%	5,0%	5,0%	4,5%	4,4%	4,7%	4,6%	9,9%	8,7%
36	24	10	7,4%	6,9%	7,1%	6,5%	6,2%	6,6%	6,8%	9,7%	8,6%
37	25	11	4,3%	3,5%	3,7%	3,4%	3,4%	3,5%	3,3%	3,6%	3,4%
38	25	11	5,8%	5,0%	4,9%	4,9%	4,9%	5,1%	4,9%	6,8%	6,8%
39	25	11	6,7%	5,6%	5,1%	5,2%	5,1%	5,9%	5,1%	9,7%	8,0%
40	25	11	9,6%	7,6%	6,7%	6,9%	7,3%	8,1%	6,8%	10,8%	7,9%
41	25	11	7,4%	6,0%	6,0%	6,3%	6,5%	6,6%	6,4%	8,9%	6,3%
42	25	11	9,4%	6,9%	6,9%	6,8%	6,7%	7,0%	6,8%	7,7%	12,2%
43	25	11	9,7%	7,7%	8,0%	7,7%	7,9%	8,5%	8,4%	11,6%	11,1%
44	25	11	9,0%	7,0%	6,9%	6,3%	6,8%	7,3%	6,7%	10,5%	10,1%
45	25	11	9,0%	7,8%	8,0%	7,3%	8,0%	9,0%	7,7%	8,8%	11,0%
		Média	7,4%	6,9%	6,6%	6,3%	6,7%	6,8%	6,4%	9,2%	8,4%

Analisando a Tabela 11, observa-se que os menores *GAPs* ratificam o comportamento observado na Tabela 9. O *GAP* foi, em média, de 6,3% para o ILS 4; 6,4% para o ILS-VND 2, 6,6% para o ILS 3, 6,7% para o VNS; 6,8% para o ILS-VND 1; 6,9% para o modelo ILS 2; 7,4% para o ILS 1; 8,4% para o LNS 2; e 9,2% para o LNS 1.

Finalmente, a Tabela 12 apresenta os coeficientes de variação das soluções encontradas, a fim de se avaliar a dispersão dos resultados.

Tabela 12 – Coeficiente de Variação das soluções

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
1	20	7	0,11%	0,23%	0,07%	0,07%	0,25%	0,07%	0,07%	0,10%	0,09%
2	20	7	0,24%	0,74%	0,24%	0,00%	0,23%	0,00%	0,00%	1,46%	0,41%
3	20	8	0,25%	1,31%	0,00%	0,00%	0,82%	0,00%	0,00%	1,01%	0,52%
4	20	8	0,12%	0,55%	0,00%	0,15%	0,58%	0,14%	0,00%	1,82%	1,16%
5	20	9	0,70%	0,74%	0,00%	0,00%	0,44%	0,00%	0,00%	0,78%	0,18%
6	20	9	0,13%	0,21%	0,00%	0,00%	0,08%	0,00%	0,00%	2,20%	0,72%
7	20	9	0,50%	0,00%	0,00%	0,00%	0,29%	0,00%	0,00%	1,10%	2,57%
8	20	9	0,29%	0,12%	0,06%	0,00%	0,59%	0,07%	0,07%	1,81%	1,71%
9	20	9	0,42%	0,40%	0,00%	0,00%	0,58%	0,00%	0,00%	1,63%	1,18%
10	21	9	0,33%	0,00%	0,26%	0,00%	0,31%	0,00%	0,28%	2,14%	0,23%
11	21	9	0,54%	0,43%	0,78%	0,00%	0,75%	0,15%	0,00%	2,33%	1,43%
12	21	9	0,62%	0,57%	0,76%	0,00%	0,13%	0,64%	0,00%	3,72%	0,46%
13	21	10	0,27%	0,15%	0,18%	0,00%	0,35%	0,00%	0,00%	0,61%	0,60%
14	21	10	0,69%	0,50%	0,50%	0,41%	0,59%	0,56%	0,37%	2,52%	1,27%
15	22	8	0,29%	0,09%	0,21%	0,17%	0,22%	0,36%	0,25%	1,44%	0,47%
16	22	9	0,89%	0,36%	0,31%	0,39%	0,36%	0,59%	0,56%	1,51%	0,72%
17	22	9	0,15%	1,28%	0,16%	0,00%	0,64%	0,31%	0,06%	0,88%	1,41%
18	22	10	0,74%	0,71%	0,65%	0,40%	0,94%	0,55%	0,51%	2,44%	2,75%
19	22	10	0,20%	0,00%	0,07%	0,00%	0,00%	0,11%	0,00%	1,56%	0,67%
20	22	10	0,66%	0,32%	0,40%	0,00%	0,35%	0,28%	0,33%	1,96%	1,82%
21	22	10	0,75%	0,39%	0,41%	0,27%	0,39%	1,14%	0,00%	3,75%	0,07%
22	22	10	0,53%	2,54%	0,53%	0,18%	0,43%	0,51%	0,13%	2,15%	0,31%
23	22	10	1,09%	0,70%	0,71%	0,20%	0,79%	0,65%	0,58%	1,30%	1,49%
24	23	10	1,15%	0,16%	0,69%	0,16%	1,05%	0,55%	0,00%	1,92%	1,87%
25	23	10	0,39%	0,14%	0,12%	0,00%	0,00%	0,13%	0,00%	2,57%	2,00%
26	23	10	0,33%	0,00%	0,28%	0,00%	0,36%	0,23%	0,00%	1,32%	1,02%

Tabela 12 – Coeficiente de Variação das soluções

Prob	n	Pares	ILS1	ILS2	ILS3	ILS4	VNS	ILS VND1	ILS VND2	LNS1	LNS2
27	23	10	0,46%	0,20%	0,17%	0,00%	0,28%	0,28%	0,19%	2,46%	1,21%
28	23	10	1,26%	1,44%	1,06%	0,78%	1,28%	0,44%	0,60%	2,87%	1,84%
29	23	10	0,92%	0,88%	0,27%	0,14%	0,96%	0,30%	0,18%	1,54%	0,42%
30	23	11	1,36%	0,17%	0,98%	0,21%	0,35%	0,39%	0,15%	1,14%	0,71%
31	23	11	1,16%	0,37%	0,53%	0,00%	0,23%	0,84%	0,00%	1,43%	1,28%
32	23	11	0,42%	0,09%	0,17%	0,11%	0,19%	0,30%	0,12%	0,67%	0,12%
33	24	9	0,39%	1,67%	0,19%	0,26%	0,89%	0,29%	0,18%	0,96%	2,04%
34	24	9	0,12%	1,18%	0,00%	0,00%	0,54%	0,12%	0,00%	1,90%	0,91%
35	24	10	0,75%	0,62%	0,49%	0,12%	0,12%	0,19%	0,10%	3,76%	1,64%
36	24	10	0,78%	0,38%	0,79%	0,32%	0,07%	0,42%	0,48%	2,14%	1,46%
37	25	11	0,32%	0,17%	0,27%	0,09%	0,12%	0,14%	0,05%	0,25%	0,13%
38	25	11	0,53%	0,07%	0,00%	0,00%	0,00%	0,10%	0,00%	1,12%	1,30%
39	25	11	1,00%	0,68%	0,11%	0,16%	0,09%	0,73%	0,10%	1,88%	1,94%
40	25	11	1,13%	0,66%	0,40%	0,54%	0,93%	0,99%	0,56%	3,50%	1,37%
41	25	11	1,20%	0,18%	0,36%	0,49%	0,73%	0,68%	0,69%	1,97%	0,51%
42	25	11	1,08%	0,16%	0,23%	0,16%	0,00%	0,18%	0,12%	0,45%	2,74%
43	25	11	1,33%	0,62%	0,64%	0,48%	0,61%	0,79%	0,41%	3,19%	2,63%
44	25	11	1,48%	0,90%	0,65%	0,42%	0,76%	0,82%	0,55%	1,83%	2,22%
45	25	11	1,24%	0,72%	0,45%	0,26%	0,61%	0,71%	0,66%	1,17%	2,53%
		Média	0,65%	0,53%	0,34%	0,15%	0,45%	0,35%	0,19%	1,78%	1,20%

De acordo com a Tabela 12, percebe-se que, de uma maneira geral, os algoritmos que encontraram as melhores soluções foram também os que apresentaram uma menor dispersão. As médias do coeficiente de variação foram de 0,15% para o ILS 4, 0,19% para o ILS-VND 2, 0,34% para o ILS 3, 0,35% para o ILS-VND 1, 0,45% para o VNS, 0,53% para o ILS 2, 0,65% para o ILS 1, 1,20% para o LNS 2, e 1,78% para o LNS 1.

A Tabela 13 apresenta um resumo dos resultados obtidos com as rodadas curtas.

Tabela 13 – Resumo das rodadas curtas

Modelo	Melhor Solução	GAP Médio	Coef. de Variação
ILS 4	260	6,3%	0,15%
ILS-VND 2	239	6,4%	0,19%
ILS 3	153	6,6%	0,34%
VNS	134	6,7%	0,45%
ILS-VND 1	128	6,8%	0,35%
ILS 2	123	6,9%	0,53%
ILS 1	32	7,4%	0,65%
LNS 2	17	8,4%	1,20%
LNS 1	12	9,2%	1,78%

6.3 ENVIO DAS MELHORES SOLUÇÕES PARA O SOFTWARE COMERCIAL

O *software* CPLEX 12.1 permite que uma solução inicial seja fornecida para que ele a adote como solução corrente durante o processo de otimização. De acordo com o manual do fabricante (IBM ILOG CPLEX V12.1 - USER'S MANUAL FOR CPLEX, 2009), esta solução inicial permite a eliminação de uma porção do espaço de busca.

Dessa forma, decidiu-se utilizar novamente o modelo matemático do caso D (Seção 4.7) para tentar melhorar os resultados, fornecendo os valores das variáveis de decisão e da função-objetivo da melhor solução conhecida de cada instância, e limitando o tempo máximo de processamento em oito horas.

Entretanto, mesmo depois deste procedimento, não foi possível melhorar nenhum dos resultados obtidos pelos algoritmos propostos.

7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o problema de sequenciamento da produção em máquinas paralelas, sem interrupções, sem *buffers*, com tempos de processamento dependentes da sequência, restrições de capacidade, e com sincronismo de execução das tarefas; assim como diversos métodos para resolvê-lo.

Os objetivos traçados na Seção 1.2 foram alcançados: o modelo matemático foi apresentado no Capítulo 4; os métodos de solução, no Capítulo 5; e sua aplicação em problemas reais, no Capítulo 6.

Na Seção 4.3 verificou-se que a restrição de sincronismo de execução de tarefas dificulta consideravelmente a resolução do problema. Mesmo instâncias relativamente pequenas, com até 10 cilindros, que são rapidamente resolvidas pelo software de otimização comercial em problemas tradicionais, não puderam ser resolvidas até a otimalidade quando esta restrição foi adicionada.

Constatou-se também que a relaxação proposta na Seção 4.6, embora melhore o desempenho do modelo matemático, fazendo-o superar a performance de uma adaptação de um modelo matemático clássico encontrado na literatura, não é capaz de encontrar soluções de boa qualidade para problemas maiores.

Para encontrar soluções melhores, combinou-se o método *relax-and-fix* com alguma metaheurística. Os resultados obtidos comprovaram a viabilidade de se utilizar tal combinação na resolução deste tipo de problema.

7.1 CONTRIBUIÇÃO DO *RELAX-AND-FIX*

A principal contribuição do método *relax-and-fix* foi obter uma solução inicial viável para cada um dos problemas propostos, já que a presença da restrição de sincronismo de execução de tarefas dificulta consideravelmente a obtenção de soluções viáveis de forma aleatória, como foi apresentado na Seção 5.1.

O *relax-and-fix* utilizado neste trabalho teve êxito em seu objetivo, uma vez que foi capaz de encontrar soluções viáveis para todos os problemas propostos num tempo computacional relativamente pequeno (351 segundos, em média). Entretanto, tais soluções não eram de boa qualidade. Como pode ser visto através da Tabela 6, o método *relax-and-fix*

encontrou soluções com um *GAP* de 24%, em média. Mesmo no melhor dos resultados, o GAP por ele encontrado foi de 11%.

7.2 CONTRIBUIÇÃO DAS METAHEURÍSTICAS

O grande objetivo ao se utilizar as metaheurísticas foi o de melhorar consideravelmente a solução gerada pelo *relax-and-fix*.

Todas as metaheurísticas atingiram este objetivo, proporcionando uma grande melhoria na qualidade das soluções dentro de um tempo computacional aceitável. No pior dos casos, o *GAP* médio passou para 9,3% entre os métodos LNS, 7,1% entre os métodos ILS, 6,4% entre os métodos ILS-VND, e 6,3% para o VNS. No melhor dos casos (método ILS 4), o *GAP* médio foi reduzido para 6,2%.

Os resultados são ainda mais significativos considerando-se que não foi possível melhorar nenhum dos resultados depois de se utilizar os melhores valores conhecidos de cada problema como solução inicial para o software de otimização comercial, mesmo depois de oito horas de processamento.

7.3 AVALIAÇÃO DAS METAHEURÍSTICAS

Analisando a qualidade das soluções obtidas, constatou-se uma superioridade dos algoritmos ILS, VNS e ILS-VND (que apresentaram um *GAP* médio de até 7,1%) em relação aos algoritmos LNS (que apresentaram um *GAP* médio acima de 8%).

7.3.1 ILS

Como foi observado no Capítulo 6, analisando os algoritmos ILS isoladamente, os algoritmos mais complexos, com mais estruturas de vizinhanças e diferentes mecanismos de perturbação, encontraram resultados melhores, mas a um custo computacional maior. Vale ressaltar, entretanto, que o aumento no tempo de processamento não inviabilizou nenhum dos algoritmos.

Dessa forma, o ILS 4, composto de quatro estruturas de vizinhança e de dois mecanismos de perturbação, obteve soluções melhores que o ILS 3, composto de três

estruturas de vizinhança e de dois mecanismos de perturbação. Este por sua vez, encontrou resultados melhores que o ILS 2, constituído das mesmas estruturas de vizinhança, mas de apenas um mecanismo de perturbação. Os piores resultados foram obtidos pelo ILS 1, constituído de apenas duas estruturas de vizinhança e de um mecanismo de perturbação.

É importante destacar também que os algoritmos ILS que obtiveram os melhores resultados (ILS 4 e ILS 3), ao contrário do ILS 2 e do ILS 1, utilizam uma estrutura de vizinhança durante a fase de perturbação diferente daquelas utilizadas durante a fase de busca local. Esta estratégia faz com que a operação de perturbação complemente a etapa de busca local, e dificulte um retorno à solução anterior.

Analisando as rodadas longas, o ILS 4 apresentou um *GAP* médio de 6,2%, e foi capaz de encontrar a melhor solução conhecida em 43 problemas. O ILS 3, com um *GAP* médio de 6,3%, obteve o melhor resultado em 35 problemas. Já o ILS 2, com um *GAP* médio de 6,5%, atingiu o melhor resultado em 25 ocasiões. Finalmente, o ILS 1, com um *GAP* médio de 7,1%, encontrou a melhor solução conhecida em 12 problemas. Um comportamento semelhante foi observado para as rodadas curtas.

Finalmente, os algoritmos que encontraram as melhores soluções também foram aqueles que apresentaram uma menor dispersão dos resultados. As médias dos coeficientes de variação foram de 0,15% para o ILS 4, 0,34% para o ILS 3, 0,53% para o ILS 2 e 0,65% para o ILS 1; o que representa uma dispersão quatro vezes maior deste último em relação ao melhor dos ILS.

7.3.2 ILS-VND

Com relação aos algoritmos ILS-VND, o fator que influenciou o desempenho de cada um foi a estratégia utilizada para escapar dos ótimos locais, uma vez que todos utilizaram as mesmas estruturas de vizinhança no processo de busca local.

O algoritmo ILS-VND 1, que utilizava um mecanismo de perturbação constituído de uma única estrutura de vizinhança, que permitia diversos movimentos de troca a cada iteração para gerar uma nova solução, e assim tentar escapar dos ótimos locais, foi o que obteve o pior desempenho. Ao substituir esta estratégia por uma que utilizava uma nova estrutura de vizinhança a cada iteração e que permitia diversos movimentos de troca em cada uma delas (ILS-VND 2), obteve-se um resultado melhor, num tempo médio de processamento inferior ao utilizado pelo ILS-VND 1.

Durante as rodadas longas, o ILS-VND 2 apresentou um *GAP* médio de 6,3%, e foi capaz de encontrar a melhor solução conhecida em 37 problemas. O ILS-VND 1, com um *GAP* médio de 6,4%, obteve o melhor resultado em 29 problemas. Para as rodadas curtas, o ILS-VND 2 também obteve um melhor desempenho, com um *GAP* médio de 6,4%, contra 6,8% do ILS-VND 1.

Com relação à dispersão dos resultados, o ILS-VND 1 apresentou um coeficiente de variação quase duas vezes maior que o ILS-VND 2: 0,35% para o primeiro, e 0,19% para o segundo.

7.3.3 VNS

O algoritmo VNS apresentou um desempenho próximo dos desempenhos dos melhores algoritmos ILS e ILS-VND, especialmente para as rodadas mais longas.

Com relação ao *GAP* médio, analisando as rodadas longas, o VNS ficou atrás apenas do ILS 4 (6,2%), igualando os 6,3% conseguidos com o ILS 3 e com o ILS-VND 2, e superando os algoritmos ILS-VND (6,4%), ILS 2 (6,5%), ILS 1 (7,1%) e os dois LNS (mais de 8%). Ao conseguir encontrar a melhor solução conhecida em 31 casos, o VNS ficou atrás do ILS 4 (43), ILS-VND 2 (37), e ILS 3 (35), mantendo-se à frente dos demais algoritmos.

O *GAP* médio do VNS nas rodadas curtas (6,7%) ficou um pouco mais distante daqueles obtidos pelos três melhores algoritmos (ILS 4 – 6,3%; ILS-VND 2 – 6,4%; ILS 3 – 6,6%), mas, mesmo assim, manteve-se à frente do ILS-VND (6,8%), ILS 2 (6,9%), ILS 1 (7,4%) e dos dois LNS (mais de 8%).

O VNS apresentou um coeficiente de variação três vezes maior que o ILS 4, ficando atrás também do ILS-VND 2, do ILS 3, e do ILS-VND 1.

7.3.4 LNS

Finalmente, analisando-se os algoritmos LNS, verifica-se que o LNS 2, que possuía uma maior vizinhança, encontrou soluções melhores que o LNS 1, mas utilizando um tempo de processamento maior. Assim como nos métodos ILS, o aumento no tempo de processamento não inviabilizou este método.

Nas rodadas longas, o LNS 2 apresentou um *GAP* médio de 8,2%, e foi capaz de encontrar a melhor solução conhecida em quatro problemas; enquanto o LNS 1, com um *GAP* médio de 9,3%, obteve o melhor resultado de apenas um problema. Já nas rodadas curtas, os *GAP*s foram de 8,4% para o LNS 2 e de 9,2% para o LNS 1.

Os algoritmos LNS foram os que apresentaram a maior dispersão nos resultados entre todos os algoritmos. A média dos coeficientes de variação foi de 1,2% para o LNS 2, e de 1,78% para o LNS 1.

7.4 IMPLANTAÇÃO NA EMPRESA

A empresa espera que a implantação deste trabalho traga benefícios tanto para o departamento de programação da produção quanto para o setor produtivo.

Além de reduzir o tempo de elaboração da programação diária de produção, espera-se que haja uma melhor utilização dos fornos de fusão, e consequentemente, um aumento de produtividade. Além disso, por serem equipamentos elétricos que atingem temperaturas da ordem de 1800ºC, qualquer antecipação no desligamento dos equipamentos depois do cumprimento do programa de produção resultará numa economia significativa no consumo de energia elétrica da fábrica de cilindros como um todo.

7.5 FUTURAS DIREÇÕES DE PESQUISA

Após a elaboração desta tese, foram identificados alguns tópicos a serem abordados em futuros trabalhos:

a) Aplicar a relaxação proposta na Seção 4.6 juntamente com a formulação matemática apresentada na Seção 4.4 em problemas tradicionais de sequenciamento da produção, onde a restrição de sincronismo de execução de tarefas não existe; e avaliar seu desempenho em relação às formulações já conhecidas.

b) Empregar métodos de relaxação aos problemas com restrição de sincronismo de execução de tarefas, tais como relaxação lagrangeana, relaxação *surrogate* e relaxação

lagrangeana/surrogate (ESPEJO; GALVÃO; 2002), para tentar obter limitantes inferiores de melhor qualidade.

c) Empregar metaheurísticas populacionais, como, por exemplo, algoritmos genéticos e colônias de formigas aos problemas com restrição de sincronismo de execução de tarefas e avaliar seu desempenho em relação aos algoritmos propostos neste trabalho.

d) Resolver as instâncias deste trabalho considerando outros critérios de otimização, e avaliar o desempenho das soluções propostas.

e) Desenvolver novas alternativas para gerar as soluções iniciais e verificar a influência da qualidade destas soluções no desempenho dos algoritmos criados.

f) Verificar o comportamento e o desempenho dos algoritmos propostos ao se utilizar uma solução inicial inviável.

g) Desenvolver novos meios de se determinar o valor do parâmetro *m* (número total de posições na sequência de produção de cada máquina).

REFERÊNCIAS

ALLAHVERDI, A.; GUPTA, J. N. D.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **Omega - International Journal of Management Science**, v. 27, n. 2, p. 219-239, abr. 1999.

ALLAHVERDI, A.; NG, C. T.; CHENG, T. C. E.; KOVALYOV, M. Y. A survey of scheduling problems with setup times or costs. **European Journal of Operational Research**, v. 187, n. 3, p. 985-1032, jun. 2008.

ARAUJO, S. A.; ARENALES, M. N.; CLARK, A. R. Lot sizing and furnace scheduling in small foundries. **Journal Computers & Operations Research**, v. 35, n. 3, p. 916-932, mar. 2008.

ARAÚJO JUNIOR, L. O. **Método de programação de sistemas de manufatura do tipo job shop dinâmico não determinístico**. 2006. 178 f. Tese (Doutorado em Engenharia) – Universidade de São Paulo, São Paulo, 2006.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa Operacional**. Rio de Janeiro: Elsevier Editora, 2007. 523p.

BERALDI, P.; GHIANI, G.; GRIECO, A.; GUERRIERO, E. Rolling-horizon and fix-and-relax heuristics for the parallel machine lot-sizing and scheduling problem with sequence-dependent set-up costs. **Journal Computers & Operations Research**, v. 35, n. 11, p. 3644-3656, nov. 2008.

BERTRAND, J. W. M.; FRANSOO, J. C. Operations management research methodologies using quantitative modeling. **International Journal Of Operations & Production Management**, v.22, n.2, p.241-254, 2002.

BICALHO, L. H. C.; SANTOS, A. G.; ARROYO, J. E. C. GBT: GRASP + busca tabu em problemas de scheduling. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 43., 2011, Ubatuba. **Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional**. Ubatuba: Sociedade Brasileira de Pesquisa Operacional, 2011.

BLAZEWICZ, J.; LENSTRA, J. K.; KAN, A. H. G. R. Scheduling subject to resource constraints: classification and complexity. **Journal of Discrete Applied Mathematics**, v.5, n.1, p. 11 – 24, 1983.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. **Journal of Computing Surveys**, v. 35, n. 3, p. 268-308, set. 2003.

BRUCKER, P. **Scheduling algorithms.** 5. ed. Berlim: Springer, 2010, 384 p.

CANTIERE, P. C.; BOIKO, T. J. P. Programação da produção em sistemas com máquinas paralelas com tempos de setup separados dos tempos de processamento. In: ENCONTRO DE PRODUÇÃO CIENTÍFICA E TECNOLÓGICA, 6., 2011, Campo Mourão. **Anais do VI Encontro de Produção Científica e Tecnológica**. Campo Mourão: FECILCAM/NUPEM, 2011.

CARCHRAE T.; BECK J. C. Principles for the Design of Large Neighborhood Search. **Journal of Mathematical Modelling and Algorithms**, v. 8, n. 3, p. 245-270, ago. 2009.

CHEN, C. L. An iterated local search for unrelated parallel machines problem with unequal ready times. In: INTERNATIONAL CONFERENCE ON AUTOMATION AND LOGISTICS, 2008, Qingdao. **Proceedings of the ICAL**, Qingdao: IEEE, 2008. p. 2044 – 2047.

CORNÉLIO, G. T. **Caracterização de materiais utilizados na fabricação de cilindros de laminação submetidos ao desgaste abrasivo.** 2006. 119 p. Dissertação (Mestrado em Engenharia Mecânica - Materiais) – Faculdade de Engenharia de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2006.

DOMINGOS, J. C.; RODRIGUES, C. V.; PEREIRA, N. A.; POLITANO, P. R.; BACHEGA, S. J. Um sistema de apoio à decisão para scheduling em job shop, utilizando lógica fuzzy. In: XXVIII ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 28., 2008, Rio de Janeiro. **Anais do XXVIII Encontro Nacional de Engenharia de Produção.** Rio de Janeiro: Associação Brasileira de Engenharia de Produção, 2008.

DRIESSEL, R.; MÖNCH, L. Scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times using variable neighborhood search. In: INTERNATIONAL CONFERENCE ON COMPUTERS & INDUSTRIAL ENGINEERING, 39., 2009, Troyes. **Anais: Proceedings of the CIE**. Troyes: IEEE, 2009. p. 273-278.

DRIESSEL, R.; MÖNCH, L. Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times. **Journal of Computers & Industrial Engineering**, v. 61, n. 2, p. 336-345, set. 2011.

ESPEJO, L.G.A.; GALVÃO, R.D. O uso das relaxações lagrangeana e surrogate em problemas de programação inteira. **Pesquisa Operacional**, v.22, n.3, p. 387-402, jul. 2002.

FERREIRA, D.; MORABITO, R.; RANGEL, S. Um modelo de otimização inteira mista e heurísticas relax and fix para a programação da produção de fábricas de refrigerantes de pequeno porte. **Revista Produção**, v. 18, n. 1, p. 76-88, jan. 2008.

FERREIRA, D.; MORABITO, R.; RANGEL, S. Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants. **Journal Computers & Operations Research**, v. 37, n. 4, p. 684-691, abr. 2010.

FURLAN, M. M. **Métodos heurísticos para o problema de dimensionamento de lotes multiestágio com limitação de capacidade.** 2011. 113 f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos, 2011.

GENTNER, K.; NEUMANN, K.; SCHWINDT, C.; TRAUTMANN, N. Batch production Scheduling in the process industries. In:___. **Handbook of scheduling: algorithms, models, and performance analysis**, Nova York: Chapman & Hall/CRC, 2004. p. 1035-1055.

GOMES, H. C. **Relaxação lagrangeana com fixação de variáveis aplicada ao problema de seqüenciamento em uma máquina**. 2008. 43 f. Dissertação (Mestrado em Engenharia de Produção) – Universidade Federal de Minas Gerais, Belo Horizonte, 2008.

GOMES JÚNIOR, A. C. **Problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso: modelagem e resolução.** 2007. 86 f. Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal de Minas Gerais, Belo Horizonte, 2007.

GOMES, R. M.; ARENALES, M. N. Otimização linear aplicada ao plantio sustentável de vegetais. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 42., 2010, Bento Gonçalves. **Anais do XLII Simpósio Brasileiro de Pesquisa Operacional**. Bento Gonçalves: Sociedade Brasileira de Pesquisa Operacional, 2010.

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; KAN, A. H. G. R. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: ANNALS OF DISCRETE MATHEMATICS, 5., 1979. p. 287 – 326.

GUIMARÃES, K. F. **Escalonamento genético FJSP com tempo de configuração dependente da sequência**. 2007. 100 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Uberlândia, Uberlândia, 2007.

GUPTA, S. R.; SMITH, J. S. Algorithms for single machine total tardiness scheduling with se-quence dependent setups. **European Journal of Operational Research**, v. 175, n. 2, p. 722–739, dez. 2006.

HANSEN, P.; MLADENOVIC, N. Variable neighborhood search. In:___. **Handbook of metaheuristics**, Nova York: Springer, 2003. p. 145-184.

HARI, Y.; YANG, C. L.; SURYANI, E. Maximizing space utilization in plant factory through crop scheduling. **Jurnal Sistem Informasi**, v.4, n3, p. 201-206, set. 2012.

HELAL, M.; RABADI, G.; AL-SALEM, A. A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. **International Journal of Operations Research**, v.3, n.3, p. 182-192, jan. 2006.

HELBER, S.; SAHLING, F. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. **International Journal of Production Economics**, v.123, n.2, p. 247-256, feb. 2010.

HURINK, J.; JURISCH, B.; THOLE, M. Tabu search for the job-shop scheduling problem with multi-purpose machines. **Journal Operations Research Spektrum**, v.15, n.4, p. 205-215, 1994.

INTERNATIONAL BUSINESS MACHINES. **IBM ILOG CPLEX V12.1 - User's manual for CPLEX**. IBM, 2009. 952 p.

JAMES, R. J. W.; ALMADA-LOBO, B. Single and parallel machine capacitated lotsizing and scheduling: new iterative MIP-based neighborhood search heuristics. **Journal Computers & Operations Research**, v. 38 n. 12, p. 1816-1825, dez. 2011.

KAMPKE, E. H. **Metaheurística para o problema de programação de tarefas em máquinas paralelas com tempos de preparação dependentes da sequência e de recursos.** 2010. 112 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Viçosa, Viçosa, 2010.

KAWAMURA, M. S. **Dimensionamento e sequenciamento de lotes de produção na indústria de bens de consumo de higiene pessoal**. 2011. 106 f. Tese (Doutorado em Engenharia de Produção) – Universidade de São Paulo, São Paulo, 2011.

KAWAMURA, M. S.; RONCONI, D. P. Aplicação da heurística *relax-and-fix* no problema de dimensionamento e sequenciamento de lotes de produção em máquinas distintas em paralelo. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 42., 2010, Bento Gonçalves. **Anais do XLII Simpósio Brasileiro de Pesquisa Operacional**. Bento Gonçalves: Sociedade Brasileira de Pesquisa Operacional, 2010.

KAZAMA, E. K. **Heurísticas construtivas para programação de operações em sistemas de produção flowshop permutacional: classificação de suas fases construtivas**. 2011. 110 f. Trabalho de Graduação (Graduação em Engenharia de Produção Mecânica) – Universidade de São Carlos, São Carlos, 2011.

KHALILI, M. An iterated local search algorithm for flexible flow lines with sequence dependent setup times to minimize total weighted completion. **International Journal of Management Science and Engineering Management**, v. 7, n. 1, p. 64-68, 2012.

LAKATOS, E. M.; MARCONI, M. A. **Fundamentos da metodologia científica**. 6 ed. São Paulo: Atlas, 2005, 203 p.

LANG, J. C.; SHEN, Z. J. M. Fix-and-optimize heuristics for capacitated lot-sizing with sequence-dependent setups and substitutions. **European Journal of Operational Research**, v.214, n3, p. 595-605, nov. 2011.

LEUNG, J. Y. T. Introduction and notation. In:___. **Handbook of scheduling: algorithms, models, and performance analysis**, Nova York: Chapman & Hall/CRC, 2004. p. 1-10.

LIANG, Y. C.; CHEN, A. H. L.; TIEN, C. Y. Variable neighborhood search for multi-objective parallel machine scheduling problems. In: PROCEEDINGS OF THE EIGHTH INTERNATIONAL CONFERENCE ON INFORMATION AND MANAGEMENT SCIENCES, 8., 2009, Kunming. **Proceedings of the IMS**, Kunming: 2009. p. 519 – 522.

LIN, M. T.; LIN, Y. Y.; FANG, K. T. Two-machine flowshop scheduling of polyurethane foam production. **International Journal of Production Economics**, v.141, n1, p. 286-294, jan 2013.

LIU J.; REEVES, C. R. Constructive and composite heuristic solutions to the $P//\Sigma C_j$ scheduling problem. **European Journal of Operational Research**, v. 132, n. 2, p. 439–452, jul. 2001.

LOPES, M. J. P. **Resolução de problemas de programação de máquinas paralelas pelo método de partição e geração de colunas.** 2004. 168 f. Tese (Doutorado em Produção e Sistemas) – Universidade do Minho, Braga, 2004.

LOURENÇO, H. R.; MARTIN, O.; STUTZLE, T. A beginner's introduction to iterated local search. In: METAHEURISTICS INTERNATIONAL CONFERENCE, 4., 2001, Porto. **Anais: Proceedings of the 4th Metaheuristics International Conference**. Porto, 2001. p. 1-11.

LOURENÇO, H. R.; MARTIN, O.; STUTZLE, T. Iterated local search. In:___. **Handbook of metaheuristics**, Nova York: Springer, 2003. p. 320-353.

MARTINS, W. A. **Busca em vizinhança variável aplicado na solução do problema de planejamento da expansão do sistema de transmissão de energia elétrica.** 2009. 85 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual Paulista, Ilha Solteira, 2009.

MELO, V. A. **PQA: investigações sobre a metaheurística VNS e sobre o uso da variância em problemas de isomorfismo de grafos**. 2010. 128 f. Tese (Doutorado em Engenharia de Produção) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2010.

MENDES A. B. **Programação de frota de apoio a operações "offshore" sujeita à requisição de múltiplas embarcações para uma mesma tarefa**. 2007. 224 f. Tese (Doutorado em Engenharia) – Universidade de São Paulo, São Paulo, 2007.

MÉNDEZ C. A; CERDÁ, J.; GROSSMANN, I. E.; HARJUNKOSKI, I.; FAHL, M. State-of-the-art review of optimization methods for short-term scheduling of batch processes. **Journal of Computers & Chemical Engineering**, v. 30, n. 6-7, p. 913-946, mai. 2006.

MIGUEL, P. A. C.; FLEURY, A.; MELLO, C. H. P.; NAKANO, D. N.; LIMA, E. P.; TURRIONI, J. B.; HO, L. L.; MORABITO, R.; MARTINS, R. A.; SOUSA, R.; COSTA, S. E. G.; PUREZA, V. **Metodologia de pesquisa em engenharia de produção e gestão de operações**. 2 ed. Rio de Janeiro: Campus-Elsevier, 2012, 280 p.

MOHAMMADI, M.; GHOMI, S. M. T. F. Relax and fix heuristics for simultaneous lot sizing and sequencing the permutation flow shops with sequence-dependent setups. **International Journal of Industrial Engineering & Production Research**, v. 21, n. 3, p. 147-153, set. 2010.

NADERI, B.; RUIZ, R.; ZANDIEH, M. Algorithms for a realistic variant of flowshop scheduling. **Journal Computers & Operations Research**, v. 37, n. 2, p. 236-246, fev. 2010.

NUNES, G. V. P.; ARROYO, J.E.C. Algoritmo GRASP-ILS para a minimização do atraso total no problema de programação de tarefas em uma máquina com setup time. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 17., 2008, Juiz de Fora. **Anais do XVII Congresso Brasileiro de Automática**. Juiz de Fora: Sociedade Brasileira de Automática, 2008.

PACHECO, R. F.; SANTORO. M. C. Proposta de classificação hierarquizada dos modelos de solução para o problema de job shop scheduling. **Revista Gestão & Produção**, v. 6, n. 1, p. 1-15, abr. 1999.

PACINO, D.; HENTENRYCK, P. V. Large neighborhood search and adaptive randomized decompositions for flexible jobshop scheduling. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 22., 2011, Barcelona. **Anais: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence**. Barcelona, 2011. p. 1997-2002.

PAN, Q. K.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization. **European Journal of Operational Research**, v. 222, n. 1, p. 31-43, out. 2012.

PAN, Q. K.; TASGETIREN, M. F.; LIANG, Y. C. A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. **Journal Computers and Operations Research**, v. 35, n. 9, p. 2807-2839, set. 2008.

PAULA, M. R. **Heurísticas para a minimização dos atrasos em sequenciamento de máquinas paralelas com tempos de preparação dependentes da sequência**. 2008. 109 f. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Minas Gerais, Belo Horizonte, 2008.

PAULA, M. R.; RAVETTI, M. G.; MATEUS, G. R.; PARDALOS, P. M. Solving parallel machines scheduling problems with sequence-dependent setup times using variable neighbourhood search. **Journal of Management Mathematics**, v. 18, n. 2, p. 101-115, abr. 2007.

PAULA, M. R.; RAVETTI, M. G.; PARDALOS, P. Abordagem variable neighborhood search para o problema de sequenciamento com máquinas paralelas e tempos de preparação dependentes da sequência. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 38., 2006, Goiânia. **Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional**. Goiânia: Sociedade Brasileira de Pesquisa Operacional, 2006.

PFUND, M.; FOWLER, J. W.; GADKARI, A.; CHEN, Y.. Scheduling jobs on parallel machines with setup times and ready times. **Journal of Computers and Industrial Engineering**, v. 54, n. 4, p. 764-782, mai. 2008.

PINEDO, M. L. **Scheduling: Theory, Algorithms and Systems**. 4 ed. Nova York: Springer, 2012. 673 p.

PISINGER, D. ROPKE, S. Large neighborhood search. In:___. **Handbook of metaheuristics**, Nova York: Springer, 2010. p. 399-420.

POTTS, C. N.; KOVALYOV, M. Y. Scheduling with batching: A review. **European Journal of Operational Research**, v. 120, n. 2, p. 228-249, jan. 2000.

POTTS, C. N.; STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. **Journal of The Operational Research Society**, v. 60, n. 1, p. 41-68, mai. 2009.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Journal Transportation Science**, v. 40, n. 4, p. 455-472, nov. 2006.

ROSHANAEI, V.; NADERI, B.; JOLAI, F.; KHALILI, M. A variable neighborhood search for job shop scheduling with set-up times to minimize makespan. **Journal Future Generation Computer Systems**, v. 25, n. 6, p. 654-661, jun. 2009.

RUIZ, R.; ANDRÉS, C. Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. In: MULTIDISCIPLINARY INTERNATIONAL CONFERENCE ON SCHEDULING, 3., 2007, Paris. **Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications**. Paris, 2007, p. 439-446.

RUIZ, R.; ROMANO, C. A. Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times. **International Journal of Advanced Manufacturing Technology**, v. 57, n. 5-8, p. 777-794, nov. 2011.
RUIZ, R.; STUTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, v. 177, n. 3, p. 2033-2049, mar. 2007.

SANTOS, M. L. R. **Programação de rotação de culturas – modelos e métodos de solução**. 2009. 95 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Universidade de São Paulo, São Carlos, 2009.

SANTOS, M. L. R.; COSTA, A. M.; ARENALES, M. N.; SANTOS, R. H. S. Sustainable vegetable crop supply problem. **European Journal of Operational Research**, v. 204, n. 3, p. 639-647, ago. 2010.

SANTOS, M. L. R.; MICHELON, P.; ARENALES, M. N.; SANTOS, R. H. S. Crop rotation scheduling with adjacency constraints. **Annals of Operations Research**, v. 190, n. 1, p. 165-180, out. 2011.

SERAFINI, P. Scheduling jobs on several machines with the job splitting property. **Journal of Operations Research**, v. 44, n. 4, p. 617-628, jul. 1996.

SIDHOUM, S. K.; SOURD, F. Fast neighborhood search for the single machine earliness–tardiness scheduling problem. **Journal Computers & Operations Research**, v. 37, n. 8, p. 1464-1471, ago. 2010.

STEBEL, S. L.; NEVES JR.; ARRUDA, L. V. R.; RODRIGUES, L. C. A. Scheduling de processos contínuos baseado nos recursos de estocagem. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 25., 2003, Natal. **Anais do XXXV Simpósio Brasileiro De Pesquisa Operacional**. Natal, 2003.

STEFANELLO, F.; ARAÚJO, O. C. B.; MÜLLER, F. M.; GARCIA, V. J. Uma vizinhança de grande porte para o problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da sequência. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 42., 2010, Bento Gonçalves. **Anais do XLII Simpósio Brasileiro de Pesquisa Operacional**. Bento Gonçalves: Sociedade Brasileira de Pesquisa Operacional, 2010.

SOURD, F. Scheduling Tasks on Unrelated Machines: Large Neighborhood Improvement Procedures. **Journal of Heuristics**, v. 7, n. 6, p. 519-531, nov. 2001.

TAHAR, D. N.; YALAOUI, F.; CHU, C.; AMODEO, L. A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent set-up times. **International Journal of Production Economics**, v. 99, n. 1-2, p. 63-73, fev. 2006.

TANG, L.; LUO, J. A new ILS algorithm for parallel machine scheduling problems. **Journal of Intelligent Manufacturing**, v. 17, n. 5, p. 609-619, out. 2006.

TOFFOLO, T. A. M. **Otimização do fluxo de produtos de uma empresa mineradora**. 2009. 104 f. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Minas Gerais, Belo Horizonte, 2009.

WANG, C.; LI, X.; WANG, Q. Iterative local search algorithm for no-wait flowshop scheduling problems to minimize makespan. In: INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK IN DESIGN, 12., 2008, Nanjing. **Proceedings of the CSCWD**, Nanjing: IEEE, 2008. p. 908 – 912.

XAVIER, R. R. **Efeito do tratamento térmico nas propriedades mecânicas e resistência ao desgaste de um ferro fundido branco multicomponente**. 2010. 95 p. Dissertação (Mestrado em Engenharia Mecânica - Materiais) – Faculdade de Engenharia de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2010.

XING, W.; ZHANG, J. Parallel machine scheduling with splitting jobs. **Discrete Applied Mathematics**, v. 103, n. 1-3, p. 259-269, jul. 2000.

YU, L., SHIH, H. M., PFUND, M., CARLYLE, W. M., FOWLER J.W. Scheduling of unrelated parallel machines: an application to PWB manufacturing, **IIE Transactions**, v.34, n.11, p.921-931, 2002.