Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação

RENAN SANTOS MENDES

Um algoritmo híbrido para uma classe de problemas de sequenciamento em máquinas paralelas

OURO PRETO - MG

2013

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Departamento de Computação

RENAN SANTOS MENDES

Um algoritmo híbrido para uma classe de problemas de sequenciamento em máquinas paralelas

Relatório Parcial para mudança de titularidade da bolsa, referente ao período agosto de 2012 a abril de 2013, apresentado à Universidade Federal de Ouro Preto (UFOP), como parte das exigências do Programa Institucional de Bolsas de Iniciação Científica do CNPq – PIBIC/CNPq.

Orientador:
Prof. Marcone Jamilson Freitas Souza, D.Sc.

OURO PRETO - MG

2013

Um algoritmo híbrido para uma classe de problemas de sequenciamento em máquinas paralelas

Renan Santos Mendes

Relatório Parcial para mudança de titularidade da bolsa, referente ao período agosto de 2012 a abril de 2013, apresentado à Universidade Federal de Ouro Preto (UFOP), como parte das exigências do Programa Institucional de Bolsas de Iniciação Científica do CNPq – PIBIC/CNPq.

Renan Santos Mendes
(Bolsista)

Prof Marcone Jamilson Freitas Souza, D.Sc. / DECOM-UFOP
(Orientador)

Ouro Preto, 29 de abril de 2013.

Resumo

Este trabalho tem seu foco no problema de programação da produção do tipo Job-Shop Scheduling com penalidade pelo atraso em relação a data de entrega. No problema tratado, cada job consiste de um conjunto de tarefas que devem ser processadas em uma dada máquina durante um período de tempo ininterrupto e predeterminado. Cada job tem uma data de entrega e o objetivo é minimizar o atraso na conclusão da operação. Propõe-se o algoritmo heurístico TWTJSSP-ILS para resolvê-lo. Inicialmente uma solução é gerada utilizando o método GRASP, combinando uma técnica de escolha randômica e outra seguindo a regra do maior tempo de processamento. Na fase de refinamento, usa-se o método Iterated Local Search (ILS) que possui como busca local o procedimento Descida em Vizinhança Variável (VND). O procedimento Path Relinking é acionado a cada iteração do ILS e conecta um ótimo local a uma solução elite gerada durante a busca. O algoritmo proposto foi testado em problemas-teste disponíveis na literatura e se mostrou capaz de gerar soluções de qualidade.

Palavras-chave: Job-Shop Scheduling com Penalidade pelo Tempo de Atraso, Iterated Local Search (ILS), Descida em Vizinhança Variável (VND).

Abstract

This work focuses on the Total Weighted Tardiness Job-Shop Scheduling Problem. In this problem, each job consists of a set of tasks that must be processed on a given set of machines for uninterrupted and predetermined time. Each job has a due date and a penalty associated with the delay in completion time. The objective is to minimize the total weighted tardiness. We propose an heuristic method based on Iterated Local Search (ILS) methodology, designated as TWTJSSP-ILS. The method applies a GRASP method to obtain an initial solution, combining a random choice technique with the longest processing time. In the refinement stage of the ILS method a Variable Neighborhood Descent (VND) procedure is applied as the as local search step. A path relinking procedure is performed for each ILS iteration and it connects a local optimum with an elite solution generated during the search. We present a computational experiment applying this method to the benchmark instances taken from the literature. The initial results lead to high quality solutions.

Keywords: Total Weighted Tardiness Job-Shop Scheduling Problem, Iterated Local Search (ILS), Variable Neighborhood Descent (VND).

Siglas e Abreviações

AG	:	Algoritmos Genéticos
BT	:	Busca Tabu
RC	:	Reconexão por Caminhos
TWTJSSP-ILS	:	Algoritmo heurístico proposto neste trabalho
TWTJSSP	:	*Total Weighted Tardiness Job-Shop Scheduling Problem*
ILS	:	*Iterated Local Search*
TS	:	*Tabu Search*
VND	:	*Variable Neighborhood Descent* (Descida em Vizinhança Variável)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

1.1 O Problema de Sequenciamento de Máquinas Jop-Shop

A busca pela eficiência em processos industriais está se intensificando cada vez mais. Tal fato se deve à grande competitividade promovida através da globalização. A fim de se ter eficiência é essencial ter um bom planejamento da produção. Aumento da produtividade e redução de custos são consequências de um bom planejamento. Ferramentas computacionais que auxiliam o desenvolvimento de um planejamento estão cada vez mais sendo utilizadas por indústrias.

Planejar o sequenciamento de tarefas em várias máquinas é um problema constantemente encontrado no âmbito industrial. O problema consiste em definir uma sequência de tarefas que devem ser alocadas em máquinas de forma que o tempo máximo de conclusão do processo ou o atraso da entrega seja o menor possível. O seqüenciamento das atividades ou scheduling é uma das atividades que compõem o planejamento da produção (programação da produção). Na programação da produção são levados em consideração uma série de elementos que disputam vários recursos por um período de tempo, recursos esses que possuem capacidade limitada.

Os elementos a serem processados são chamados de ordens de fabricação ou *jobs* (trabalho) e são compostos de partes elementares chamadas tarefas ou operações. Os principais objetivos tratados no problema de seqüenciamento podem ser resumidos no atendimento de prazos (ou datas de entrega), na minimização do tempo de fluxo dos estoques intermediários e na maximização da utilização da capacidade disponível, ou

mesmo na combinação destes objetivos.

A maioria dos problemas de programação da produção estudados aplica-se ao ambiente conhecido como Job-Shop. O problema Job-Shop (JSP) é um problema de alocação de um conjunto de jobs para as máquinas, de tal forma que os jobs sejam executados em um menor intervalo de tempo. Cada job pode consistir de diversas tarefas e cada tarefa deve ser processada numa máquina particular. Além disso, as tarefas em cada job estarão sujeitas à restrições de precedência.

Devido aos grandes esforços computacionais exigidos por modelos matemáticos de otimização para obtenção de solução ótima para o problema, propomos o estudo de um algoritmo heurístico, que tem como principio a Evolução Natural, para obtenção de boas soluções sem exigir grandes esforços computacionais.

1.2 Objetivos do trabalho

O objetivo geral deste projeto de pesquisa é desenvolver um algoritmo eficiente de otimização, baseado em metaheurísticas, para resolver o Problema de Sequenciamento de Máquinas do tipo *Job-Shop* com penalidade pelo atraso em relação a data de entrega (TWTJSSP, do inglês).

1.2.1 Objetivos específicos

São os seguintes os objetivos específicos:

1. Fazer uma revisão de literatura sobre os métodos utilizados para resolver o TWTJSSP;
2. Fazer uma revisão de literatura sobre técnicas heurísticas, em especial as metaheurísticas;
3. Testar o(s) algoritmo(s) desenvolvido(s);
4. Contribuir com a divulgação de técnicas de otimização aplicadas à resolução do problema, possibilitando aos desenvolvedores de aplicativos comerciais e/ou empresas do ramo industrial e de produção de manufaturas;
5. Contribuir com a divulgação de técnicas de otimização aplicadas à resolução do problema;

6. Contribuir com a formação de recursos humanos especializados nessa área do conhecimento;

7. Contribuir para a consolidação do grupo de pesquisa **Pesquisa Operacional**.

1.3 Estrutura do trabalho

O presente trabalho está dividido em seis capítulos, incluindo esta introdução.

O Capítulo 2 apresenta a definição do problema abordado neste trabalho, o Problema de Sequenciamento de Máquinas do tipo *Job-Shop* com penalidade pelo atraso em relação a data de entrega, também conhecido do inglês como *Total Weighted Tardiness Job-Shop Scheduling Problem* (TWTJSSP).

No Capítulo 3 é apresentada uma revisão bibliográfica sobre os diversos métodos utilizados na resolução do problema de sequenciamento de máquinas, bem como a forma com que diversos autores tratam esse problema. Apresenta-se também a descrição das metaheurísticas Descida em Vizinhança Variável (VND) e *Iterated Local Search* (ILS).

No Capítulo 4 é apresentado o algoritmo proposto, denominado TWTJSSP-ILS, para resolver o PRVCES. Para detalhar esse algoritmo, o qual é baseado na metaheurística *Iterated Local Search*, é mostrada como uma solução é representada, como são geradas as soluções iniciais, as estruturas de vizinhança utilizadas, a função de avaliação, o procedimento VND usado como busca local do ILS, a técnica Reconexão por Caminhos, bem como a estrutura auxiliar de dados utilizada no algoritmo.

No Capítulo 5 são apresentados e analisados os resultados computacionais e no Capítulo 6 são apresentadas as conclusões e apontadas as sugestões para trabalhos futuros.

Capítulo 2

Descrição do Problema Abordado

Descreve-se, a seguir, um problema da programação de produção do tipo *Job-Shop*, em sua forma básica. Um conjunto de n *jobs* J = $\{J_1, J_2, ... , J_n\}$ deve ser processado em um conjunto de m máquinas M = $\{M_1, M_2, ... , M_m\}$ disponíveis. Cada *job* possui uma ordem de execução específica entre as máquinas, ou seja, um *job* é composto de uma lista ordenada de tarefas, cada uma das quais definida pela máquina requerida e pelo tempo de processamento na mesma. As restrições que devem ser respeitadas são:

- Operações não podem ser interrompidas e cada máquina pode processar apenas uma tarefa de cada vez;
- Cada *job* só pode estar sendo processado em uma única máquina de cada vez;
- Cada *job* é executado por uma seqüência conhecida de tarefas.

Uma vez que as seqüências de máquinas de cada *job* são fixas, o problema a ser resolvido consiste em determinar as seqüências dos *jobs* em cada máquina, de forma que o tempo de execução transcorrido, desde o início do primeiro *job* até o término do último, não ultrapasse a data limite de término. Levando em consideração a abordagem deste trabalho, o objetivo é encontrar a solução com menor tempo de atraso para a execução dos *jobs*.

Na Figura 2.1 pode ser observado um exemplo de solução para o *Job-Shop Scheduling Problem* com $M = 3$ e $J = 3$. A máquina 1, denotada por **M1** na parte superior da Figura 2.1, esta executanto a primeira tarefa do *Job=3*, a segunda terefa do *Job=1* e por último, a primeira tarefa do *Job=2*. Percebe-se que todas as restrições do problema são atendidas.

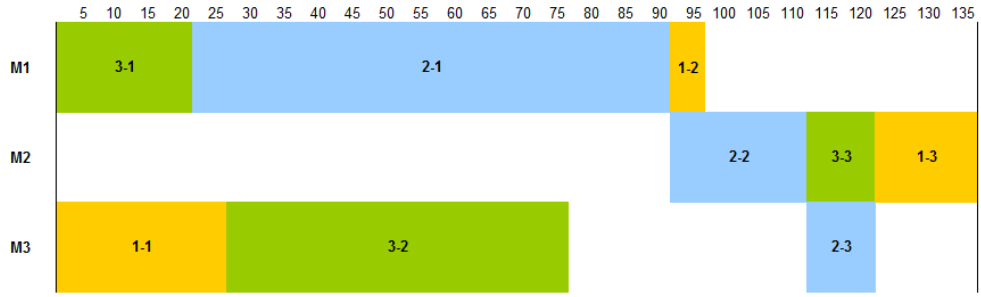

Figura 2.1: Exemplo de uma Solução para o Job-Shop Scheduling Problem

Capítulo 3

Revisão Bibliográfica

3.1 Introdução

O seqüenciamento das atividades ou *scheduling* é uma das atividades que compõem o planejamento da produção (programação da produção). Na programação da produção são levados em consideração uma série de elementos que disputam vários recursos por um período de tempo, recursos esses que possuem capacidade limitada. Portanto, a maioria dos problemas de programação da produção estudados aplica-se ao ambiente conhecido como *Job-Shop*.

O problema *Job-Shop scheduling*, que terá surgido no início de 1950 [Jain and Meeran, 1999], consiste em sequenciar um conjunto de trabalhos num grupo de máquinas, minimizando uma determinada medida de performance, seja ela, atendimento de prazos (ou datas de entrega), minimização do *makespan* (tempo de conclusão do processo), minimização do tempo de atraso em relação a entrega, minimização do tempo de fluxo dos estoques intermediários, maximização da utilização da capacidade disponível ou até mesmo na combinação destes objetivos [Walter, 1999].

Existe na literatura um grande número de métodos exatos e heurísticos para resolver o problema *Job-Shop*, porém a grande maioria deles são desenvolvidos para tratarem do *makespan* como objetivo. Uma das abordagens mais bem sucedidas para este critério é o algoritmo proposto por [Laarhoven *et al.*, 1992] baseado no método Simulated Annealing, o procedimento Busca Tabu implementado por [Nowicki and Smutnicki, 1996] e o *Itereted Local Search* em [Lourenço, 1995]. Algoritmos genéticos também têm sido aplicados com sucesso para o problema *Job-Shop*, como é o caso de [Volta *et al.*, 1995]. Uma aborda-

gem relevante que combina algoritmos genéticos com outras heurísticas de busca local é proposta por [Gonçalves *et al.*, 2005].

Umas das variantes do *Job-Shop scheduling*, que tem como objetivo minimizar o tempo de atraso em relação a data de entrega do produto, é chamada do inglês de *Total Weighted Tardiness Job-Shop Scheduling Problem* (TWTJSSP). É um problema de sequenciamento de máquinas da classe NP-difícil, portanto as principais abordagens presentes na literatura se baseiam em heurísticas, embora existam trabalhos que utilizam métodos exatos.

[Pinedo and Singer, 1998] apresenta e compara uma série de algoritmos baseados na técnica *Branch and Bound* para minimizar o atraso total no problema *Job-Shop Scheduling*. Em [Pinedo and Singer, 1999], a heurística *Shifting Bottleneck* (SB) proposta por [Adams *et al.*, 1988], é utilizada com o intuito de decompôr o problema em subproblemas menores com uma única máquina, sendo então resolvidos um após o outro, ou seja, cada máquina é programada de acordo com a solução de seu subproblema correspondente. Essa técnica se mostrou eficaz e produziu bons resultados.

[Kreipl, 2000] também aborda o problema de forma heurística utilizando uma técnica denominada de *large step random walk* que varia o tamanho da vizinhança de acordo com os paços seguidos pelo algoritmo. A técnica de algoritmos genéticos é combinada com o procedimento *Itereted Local Search* no trabalho proposto por [Essafi *et al.*, 2008], produzindo até então, umas das melhores soluções para o TWTJSSP. [Bülbül, 2011] propõe um algoritmo híbrido que contempla as heurísticas *Shifting Bottleneck* e Busca Tabu. Seu objetivo é particionar o problema, onde a Busca Tabu é responsável por realizar a busca local e o método SB se encarrega de diversificar as soluções.

Neste trabalho, o algoritmo denominado `TWTJSSP-ILS` incorpora o VND (*Variable Neighborhood Descent*) como busca local do método ILS (*Iterated Local Search*), além de uma estrutura auxiliar de dados que permite armazenar informações importantes ao longo da execução do algoritmo, contribuindo para fornecer suporte ao método de avaliação rápida. A técnica Reconexão por Caminhos é também utilizada com o objetivo de explorar melhor o espaço de soluções.

3.2 Metaheurísticas

Metaheurísticas são procedimentos de busca local destinados a resolver aproximadamente um problema de otimização, tendo a capacidade de escapar das armadilhas dos ótimos locais, ainda distantes de um ótimo global. Elas podem ser de busca local ou populacional.

Na primeira, a exploração do espaço de soluções é feita por meio de movimentos, os quais são aplicados a cada passo sobre a solução corrente, gerando outra solução promissora em sua vizinhança. Já na segunda, trabalha-se com um conjunto de soluções, recombinando-as com o intuito de aprimorá-las.

Neste trabalho, foram utilizadas as metaheurísticas Descida em Vizinhança Variável (VND) e *Iterated Local Search* (ILS), as quais são descritas nas Seções 3.2.1 e 3.2.2, respectivamente.

3.2.1 Descida em Vizinhança Variável

A Descida em Vizinhança Variável [Hansen and Mladenović, 2001], conhecida na literatura inglesa como *Variable Neighborhood Descent* - VND, é uma metaheurística que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança.

Seja s a solução corrente e N a vizinhança de uma solução estruturada em r vizinhanças distintas, isto é, $N = N^{(1)} \cup N^{(2)} \cup \cdots \cup N^{(r)}$. O VND inicia-se analisando a primeira estrutura de vizinhança $N^{(1)}$. A cada iteração, o método gera o melhor vizinho s' da solução corrente s na vizinhança $N^{(k)}$. Caso s' seja melhor que s, então s' passa a ser a nova solução corrente e retorna-se à vizinhança $N^{(1)}$. Caso contrário, passa-se para a próxima estrutura de vizinhança $N^{(k+1)}$. O método termina quando não é possível encontrar uma solução $s' \in N^{(r)}$ melhor que a solução corrente.

O Algoritmo 1 mostra o pseudocódigo do VND.

Algoritmo 1 Descida em Vizinhança Variável

```
1: Seja r o número de estruturas de vizinhança distintas;
2: k ← 1;
3: enquanto ( k ≤ r ) faça
4:    Encontre o melhor vizinho s' ∈ N^(k)(s);
5:    se ( f(s') < f(s) ) então
6:       s ← s';
7:       k ← 1;
8:    senão
9:       k ← k + 1;
10:   fim se
11: fim enquanto
12: Retorne s;
```

3.2.2 *Iterated Local Search*

O *Iterated Local Search* (ILS) [Stützle and Hoos, 1999] é uma metaheurística que possui quatro componentes básicos, a saber, a geração de uma solução inicial, o mecanismo de perturbação, o método de busca local e o critério de aceitação. Primeiramente, gera-se uma solução inicial e aplica-se uma busca local. Para escapar do ótimo local s gerado, é feita uma perturbação, gerando uma nova solução s'. Em seguida, essa solução perturbada é refinada pelo método de busca local, obtendo-se um novo ótimo local s''. Esta solução tornar-se a nova solução corrente caso s'' seja aprovada por um critério de aceitação; caso contrário, ela é descartada e nova perturbação é feita a partir da solução s. Esse procedimento é repetido até que um determinado critério de parada seja satisfeito, como, por exemplo, um número máximo de iterações sem melhora na solução corrente ou um tempo máximo de processamento.

Vale ressaltar que a intensidade das perturbações não pode ser nem tão pequena e nem tão grande. Se a intensidade for muito pequena, s' poderá voltar à região de atração de s e com isso a probabilidade de encontrar novas soluções é reduzida. Se a intensidade da perturbação for muito elevada, s' será uma solução aleatória e o método funcionaria como um algoritmo de reinício aleatório.

Uma análise mais detalhada sobre a metaheurística ILS pode ser encontrada em [Lourenço *et al.*, 2003].

O pseudocódigo do ILS básico é apresentado no Algoritmo 2.

Algoritmo 2 *Iterated Local Search*

1: Seja s_0 uma solução inicial;
2: $s \leftarrow BuscaLocal(s_0)$;
3: **enquanto** (*critério de parada não satisfeito*) **faça**
4: $\quad s' \leftarrow Perturbação(s)$;
5: $\quad s'' \leftarrow BuscaLocal(s')$;
6: $\quad s \leftarrow CritérioAceitação(s, s'')$;
7: **fim enquanto**
8: Retorne s;

3.3 Reconexão por Caminhos

A técnica Reconexão por Caminhos, do inglês *Path Relinking*, foi proposta por [Glover, 1997] como uma estratégia de intensificação. Para isso, são exploradas trajetórias que conectam boas soluções encontradas ao longo da busca, com o intuito de encontrar soluções ainda

melhores.

Para realizar a busca, é construído um conjunto chamado elite, que contém soluções geradas anteriormente pelo algoritmo. Geralmente, os critérios adotados para selecionar os membros são feitos através do valor da função de avaliação (que avalia a solução), e/ou da diversidade em relação às outras soluções do conjunto, para que o conjunto não contenham soluções muito parecidas.

Sendo assim, a Reconexão de Caminhos consiste em gerar e explorar caminhos no espaço de soluções, partindo de uma ou mais soluções elite e levando a outras soluções elite. Como estratégia para se gerar as soluções, são selecionados movimentos que introduzem atributos das soluções guia na solução corrente.

Essa técnica de intensificação pode ser aplicada segundo duas estratégias básicas:

- Reconexão por Caminhos aplicada como uma estratégia de pós-otimização entre todos os pares de soluções elite;
- Reconexão por Caminhos aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

Como pode ser observado no Algoritmo 3, primeiro é computado a diferença simétrica $\Delta(s, g)$ entre s e g, resultando no conjunto de movimentos que deve ser aplicado a uma delas, dita solução inicial s, para alcançar a outra, dita solução guia g. A partir da solução inicial, o melhor movimento ainda não executado de $\Delta(s, g)$ é aplicado à solução corrente $\bar{g}$ até que a solução guia seja atingida. A melhor solução encontrada ao longo desta trajetória é considerada como candidata à inserção no conjunto elite e a melhor solução já encontrada é atualizada.

Algoritmo 3 Reconexão-Caminhos

1: $\bar{g} \leftarrow s$;
2: Atribuir a g' a melhor solução entre s e g;
3: Calcular o conjunto de movimentos possíveis $\Delta(s, g)$;
4: **enquanto** ($|\Delta(s, g)| \neq 0$) **faça**
5: Atribuir a g"a melhor solução obtida aplicando o melhor movimento de $\Delta(s, g)$ a $\bar{g}$;
6: Excluir de $\Delta(s, g)$ este movimento;
7: $\bar{g} \leftarrow g$";
8: **se** ($f(\bar{g}) \leq f(g')$) **então**
9: $g' \leftarrow \bar{g}$;
10: **fim se**
11: **fim enquanto**
12: Retorne g';

Capítulo 4

Metodologia

Neste capítulo é apresentada a metodologia proposta para resolver o TWTJSSP. Na Seção 4.1 mostra-se como uma solução do problema é representada, enquanto na Seção 4.2 são apresentados os movimentos utilizados para explorar o espaço de soluções do problema. Na Seção 4.3 mostra-se como uma solução é avaliada. Os métodos de geração de uma solução inicial são apresentados na Seção 4.4. Na Seção 4.5 é descrito o algoritmo proposto para resolver o TWTJSSP, o qual faz uso de um método de busca local descrito na Seção 4.7.

4.1 Representação de uma solução

Uma solução s do TWTJSSP é representada como um vetor de listas. Em tal representação se tem um vetor v cujo tamanho é o número de máquinas, m, e cada posição desse vetor contém um número que representa uma máquina. O sequenciamento das tarefas em cada máquina, por sua vez, é representado por uma lista de pares de números, em que cada par representa o *job* e a tarefa. Para melhor compreensão desta representação tem-se como exemplo a Figura 4.1.

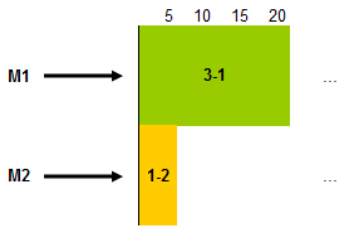

Figura 4.1: Representação de solução para o Job-Shop Scheduling Problem

4.2 Estruturas de vizinhança

Para explorar o espaço de soluções foram utilizados quatro tipos de movimentos: Troca, Realocação Monotarefa, Realocação Multitarefa e *2-Opt*. Todos eles serão detalhados nas Seções 4.2.0.1, 4.2.0.2, 4.2.0.3 e 4.2.0.4, respectivamente.

4.2.0.1 Movimento de Troca (*Exchange*)

O Movimento de Troca, também chamado de *Exchange*, consiste em selecionar duas tarefas T_i e T_j, e em seguida trocá-las de posição, conforme ilustrado na Figura 4.2.

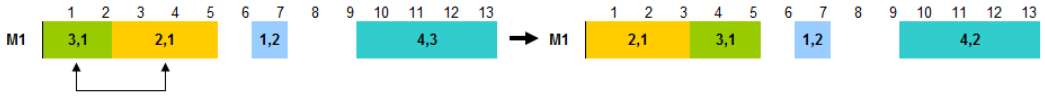

Figura 4.2: Exemplo da aplicação do Movimento de Troca (*Exchange*)

4.2.0.2 Movimento de Realocação Monotarefa

Para o Movimento de Realocação Monotarefa uma tareja T_i da sequência é indicada, assim como a posição j para onde ela será realocada. Um exemplo para este movimento por ser visto na Figura 4.2.

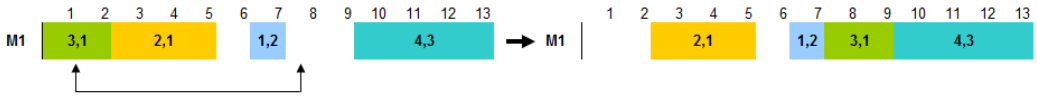

Figura 4.3: Exemplo da aplicação do Movimento de Realocação Monotarefa

4.2.0.3 Movimento de Realocação Multitarefa

O Movimento de Realocação Multitarefa é baseado na realocação de um bloco de k tarefas, com $2 \leq k \leq n-2$, sendo n o número total de tarefas de uma máquina. Portanto, um número k de tarefas (J_i, J_{i+1},...) são elegidas, assim como a posição j indicando a partir de onde elas serão alocadas. Um exemplo com $k = 2$ e posição $j = 8$ pode ser observado na Figura 4.4.

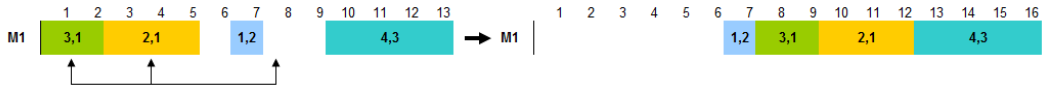

Figura 4.4: Exemplo da aplicação do Movimento de Realocação Multitarefa

4.2.0.4 Movimento *2-Opt*

O Movimento *2-Opt* consiste em remover dois arcos não adjacentes de forma que outros dois são adicionados e uma nova sequência de tarefas é gerada. Quandos os arcos são removidos, novos arcos são formados com o intuito de preencher o espaço vazio ocasionado na remoção. A Figura 4.5 ilustra a remoção dos arcos nela idicados com um "x" e a solução final gerada com este movimento.

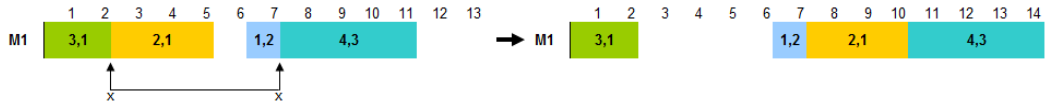

Figura 4.5: Exemplo da aplicação do Movimento *2-Opt*

4.3 Função de avaliação

Uma solução é avaliada pela função (4.1), a ser minimizada. A primeira parcela do somatório, representada por W_i, é um fator de importância para a execução do *job*, ou seja, alguns *jobs* possuem prioridades em relação aos demais. Este fator é fornecido pela instância e varia de 1 a 4, sendo que 20% do primeiros *jobs* tem peso 4 e alta prioridade, 60% dos *jobs* subsequentes tem peso 2 e média prioridade e 20% restantes tem peso 1 com baixa prioridade. A segunda parcela do somatório obtém o máximo valor entre 0 e o tempo de atraso na execução do *job*. Intuitivamente o tempo de atraso é definido pela diferença entre a data de conclusão C_i e a data de entrega d_i.

$$f(s) = \sum_{i \in J} W_i \cdot \max\{0, C_i - d_i\} \tag{4.1}$$

Na função de avaliação f, dada pela Equação (4.1), têm-se:

J: conjunto dos *jobs* a serem processados;

W_i: fator de importância do *job* $i \in J$;

C_i: Data de conclusão do *job* $i \in J$;

d_i: Data de entrega do *job* $i \in J$.

4.4 Geração de uma solução inicial

O algoritmo `TWTJSSP-ILS` gera uma solução inicial utilizando o método chmado de GRASP [Feo and Resende, 1995] que combina duas técnicas de escolha, uma randômica, onde a tarefa que vai ser inserida na máquina é escolhida alaetoriamente e outra seguindo a regra do maior tempo de processamento, onde é escolhida a tarefa candidata que possuir o maior tempo de execução entre o conjunto de tarefas candidatas da máquina. A busca local é realizada por meio do método Descida em Vizinhança Variável - VND [Mladenović and Hansen, 1997].

Inicialmente um número real aleatório (γ) entre $0,0$ e $0,7$ é gerado para determinar o índice de aleatoriedade do GRASP. Outro parâmetro utilizado no método é o critério de parada, que neste caso utiliza-se o tempo de execução (*time*).

A técnica de escolha randômica consiste em eleger uma tarefa de um determinado *job* de forma aleatória, verificar qual a máquina que deverá processar esta tarefa e por último alocá-la na máquina correspondente.

A regra gulosa que compõem o método GRASP se baseia em alocar as tarefas de acordo com o maior tempo de processamento que elas possuem. O primeiro passo é organizar as tarefas utilizando como critério a máquina onde serão executadas. Feita essa separação, deve-se ordenar essas tarefas em ordem decrescente de tempo, ou seja, do maior para o menor tempo de processamento. E por último as tarefas são direcionadas para as máquinas correspondentes seguindo esta ordem.

4.5 Algoritmo proposto

O algoritmo proposto, chamado de TWTJSSP-ILS, combina os procedimentos heurísticos GRASP, ILS, VND e *Path Relinking* (PR). O primeiro procedimento é usado para gerar uma solução inicial, sendo o refinamento feito pelo ILS. O ILS, por sua vez, utiliza os procedimentos VND como módulo de busca local. Adicionalmente, ele utiliza uma estratégia de Reconexão por Caminhos (do inglês *Path Relinking*) entre um ótimo local gerado pelo VND e uma solução do conjunto elite formada durante a busca. Também é agregado ao algoritmo uma estrutura auxiliar de dados que permite armazenar informações importantes ao longo da execução do algoritmo, contribuindo para fornecer suporte ao método de avaliação rápida. Seu pseudocódigo é apresentado pelo Algoritmo 4.

Como pode ser observado no Algoritmo 4, o método GRASP, descrito na Seção 4.4,

Algoritmo 4 *TWTJSSP-ILS*

1: $\gamma \leftarrow$ número real aleatório no intervalo $[0.0, 0.7]$;
2: $s \leftarrow GRASP(\gamma, Time)$;
3: $s \leftarrow VND(s)$;
4: $iter \leftarrow 1$;
5: **enquanto** ($iter \leq maxIter$) **faça**
6: $s' \leftarrow Perturbação(s)$;
7: $s'' \leftarrow VND(s')$;
8: $s \leftarrow Reconexao\_Por\_Caminhos(solElite, s'')$;
9: **se** ($f(s'') < f(s)$) **então**
10: $s \leftarrow s''$;
11: $iter \leftarrow 1$;
12: **senão**
13: $iter \leftarrow iter + 1$;
14: **fim se**
15: *Atualiza_Estrutura_Auxiliar_Dados(Job, Tarefa, Máquina, Tempo_Execução)*;
16: *Atualiza_Conjunto_Elite(s")*;
17: **fim enquanto**
18: Retorne s;

produz a solução inicial s, que é em seguida refinada por um procedimento VND (Seção 4.7). A solução inicial gerada é então refinada pelo ILS. Para escapar das armadilhas de ótimos locais, e se dirigir para outras regiões do espaço de busca, foram utilizadas perturbações descritas na Seção 4.6. O método VND (descrito na Seção 4.7) é utilizado como busca local, que é interriompido quando um número de iterações sem melhora na solução corrente ($maxIter$) é atingido. A cada iteração do ILS também é aplicada a técnica Reconexão por Caminhos (descrita na Seção 4.8) na solução gerada pela busca local s''.

4.6 Mecanismos de perturbação

As perturbações são realizadas por um dos dois procedimentos seguintes: Múltiplos Movimentos de Troca e Múltiplos Movimentos de Realocação Monotarefa. Os procedimentos consistem em k aplicações sucessivas dos movimentos citados, sendo k um valor inteiro aleatório enre 1 e 3.

4.7 Busca Local VND TWTJSSP-ILS

O VND implementado é baseado na proposta de [Mine *et al.*, 2010], onde são utilizadas duas estratégias adicionais em relação ao método clássico. A primeira consiste em definir aleatoriamente a ordem das vizinhanças a serem exploradas. Na outra estratégia, caracterizada pela intensificação da busca nas máquinas modificadas, após cada melhor vizinho escolhido aleatoriamente no passo anterior, é aplicada uma sequência aleatória de

procedimentos de otimização, no caso, envolvendo a busca local *Best Improvement* com os quatro movimentos descritos na Subseção 4.2. O pseudocódigo do VND é apresentado no Algoritmo 5.

Algoritmo 5 VND

```
1: Seja r o número de estruturas distintas de vizinhanças;
2: RN ← conjunto das vizinhanças N, descritas na Seção 4.2, em ordem aleatória;
3: k ← 1;
4: enquanto ( k ≤ r ) faça
5:    Encontre o melhor vizinho s' ∈ RN^(k)(s);
6:    se ( f(s') < f(s) ) então
7:       s ← s';
8:       k ← 1;
9:       {Intensificação nas máquinas alteradas}
10:      s ← BuscaLocal_movimentoTroca(s);
11:      s ← BuscaLocal_movimentoRealocaçãoMonoTarefa(s);
12:      s ← BuscaLocal_movimentoRealocaçãoMultitarefa(s) com k = 2, 3, 4;
13:      s ← BuscaLocal_movimento2-Opt(s);
14:   senão
15:      k ← k + 1;
16:   fim se
17: fim enquanto
18: Retorne s;
```

4.8 Reconexão por Caminhos

Como forma de fazer um balanço entre intensificação e diversificação, foi utilizada a técnica de Reconexão por Caminhos (PR, do inglês *Path Relinking*), que é aplicada após cada busca local do ILS.

O conjunto elite é composto por cinco soluções. Assim, quando uma nova solução é adicionada ao conjunto e sua capacidade é extrapolada, a solução de pior custo sai, dando lugar à nova solução. Para determinar quais soluções farão parte do conjunto elite foram utilizados os seguintes critérios:

- Se a solução corrente tiver a função de avaliação menor que a melhor solução encontrada até então;
- Se a solução corrente for melhor que a pior do conjunto elite, e for pelo menos 15% diferente das demais soluções do conjunto.

Adotou-se como estratégia utilizar cada uma das soluções do conjunto elite como solução base, e como guia o ótimo local gerado após a aplicação da busca local do procedimento ILS descrito na Seção 4.7. Ou seja, são realizadas cinco aplicações de Reconexão

por Caminhos. Caso durante a aplicação desta estratégia seja encontrada uma solução melhor que a melhor já obtida, o procedimento de Reconexão é abortado.

Cada iteração da Reconexão por Caminhos consiste em incluir na solução inicial (solução base), um atributo da solução guia. O atributo escolhido para ser inserido é aquele que produz na solução base o melhor valor para a função de avaliação. A seguir, à solução com o atributo inserido é aplicada uma busca local que não altere os atributos herdados, no caso, uma descida completa utilizando o Movimento de Troca. Foi considerado como atributo a ser herdado pela solução base, a posição de determinado cliente no vetor referente à solução guia. Este procedimento é repetido até que as soluções tenham as mesmas configurações.

4.9 Estrutura Auxiliar de Dados

Uma Estrutura Auxiliar de Dados, baseada no trabalho de [Penna *et al.*, 2013], foi implementada no algoritmo proposto com a intenção de armazenar informações importantes relacionadas às soluções durante o processo de busca local. Toda vez que se realiza um movimento, parte da solução é modificada, logo é necessário reavaliá-la para averiguar se houve melhora ou não. Portanto, faz-se necessário uma técnica para guiar o processo de busca e fornecer um suporte ao método de reavaliação da solução, já que um conjunto organizado de dados é de fácil acesso e muito útil para orientar o algoritmo nesta fase da execução.

Com o intuito de aperfeiçoar a busca nas vizinhanças, adotou-se uma matriz que armazena informações importantes sobre cada uma das máquinas:

1. **Job:** Indica o índice do *job* que está em execução em cada umas das máquinas;
2. **Tarefa:** Indica o índice da Tarefa que está em execução em cada umas das máquinas;
3. **Máquina:** Indica a máquina que está responsável por executar a Tarefa i do *Job* j;
4. **Ordem:** Indica qual a ordem que a Tarefa i do *Job* j assume durante a execução na máquina.

Esta matriz (ij) é declarada como uma estrutura, onde cada linha representa uma máquina e cada coluna um *job*. A célula desta matriz é capaz de armazenar os quatro diferentes tipos de dados descritos acima e deve ser atualizada com o valores correntes toda vez que se realiza um movimento onde a solução é modificada.

Uma vez consolidada, esta estrutura auxiliar de dados contribui para orientar na tarefa de reavaliação da solução. Toda vez que se realiza um movimento, a nova solução gerada deve ser avaliada para verificar o quão melhor ou pior ela é. Esta tarefa torna-se mais eficaz quando, a partir das informações fornecidas pela estrutura auxiliar de dados, o algoritmo é capaz de identificar a partir de onde a solução foi modificada. Identificado este ponto, é necessário avaliar a solução apenas desta parte em diante. Logo, poupa-se esforço computacional avaliando apenas o que fora modificado na solução e não toda ela.

Além do mais, é possível obter mais informações inerentes ao sequenciamento de máquinas simplesmente associando a estrutura auxiliar de dados com a matriz de tempo que é retirada do problema-teste. Desta maneira é facil identificar a escala de tarefas a serem executadas em uma determinada máquina, quanto tempo de processamento é necessário, quantificar a ociosidade de uma máquina e sobretudo informar as modificações na solução para orientar melhor o processo de busca.

Capítulo 5

Resultados Computacionais

O algoritmo `TWTJSSP-ILS` foi codificado na linguagem de programação C++ com auxílio do *framework OptFrame* desenvolvido por [Coelho *et al.*, 2011]. Para testá-lo, foi usado um microcomputador com processador Intel *Core 2 Duo*, 2,00 GHz e 3 GB de memória RAM e sistema operacional Windows Vista. Apesar de o microcomputador possuir dois núcleos, o algoritmo proposto não explora multiprocessamento.

Para validá-lo, foram usados 22 problemas-teste da literatura de tamanho 10×10, ou seja, com 10 máquinas e 10 *jobs*: abz5 e abz6 de [Adams *et al.*, 1988], LA16-LA24 de [Lawrence, 1994] e ORB01-ORB10 de [Applegate and Cook, 1991]. Os parâmetros adotados, obtidos experimentalmente em uma bateria preliminar de testes, foram os seguintes: $maxIter = 10000$, $time = 120$ segundos para a execução do GRASP e fator da data de entrega $f = 1{,}3$.

Dado seu caráter estocástico, o algoritmo foi executado 50 vezes em cada problema-teste. Os resultados alcançados por `TWTJSSP-ILS` podem ser visualizados na Tabela 5.1, onde o algoritmo proposto foi comparado com outros três algoritmos da literatura: o algoritmo heurístico que combina a técnica *large step random walk* de [Kreipl, 2000], o algoritmo genético com ILS de [Essafi *et al.*, 2008] e a Busca Tabu com *Shifting Bottleneck* de [Bülbül, 2011]. Nesta tabela, a coluna "Problema" indica o nome da instância e a coluna "MelhorLit" são os valores ótimos para cada um dos problemas-teste. As colunas "Melhor" e "$Desv^{Melhor}$" refere-se respectivamente, ao melhor valor encontrado por cada algoritmo e o desvio em relação ao melhor da literatura. O desvio de uma instância i é calculado pela equação (5.1), onde $Melhor_i^{Lit.}$ é o melhor valor da literatura para a instância i e $Melhor_i^{Alg}$ representa o melhor valor obtido pelo respectivo algoritmo.

$$Desv_i^{Melhor} = (Melhor_i^{Alg} - Melhor_i^{Lit.})/Melhor_i^{Lit.} \quad (5.1)$$

Considerando o conjunto de todos os 22 problemas-teste, em termos de número de melhores resultados, tem-se: [Essafi *et al.*, 2008]: 18; [Bülbül, 2011]: 10; `TWTJSSP-ILS`: 9 e [Kreipl, 2000]: 3. Já em termos de desvio médio, tem-se: [Essafi *et al.*, 2008]: 0,00%, `TWTJSSP-ILS`: 0,01%, [Bülbül, 2011]: 0,02% e [Kreipl, 2000]: 0,02%. Logo o algoritmo proposto supera o de [Kreipl, 2000] em relação ao número de melhores soluções encontradas, no caso 9, contra 3, além de obter desvio melhor que [Bülbül, 2011] e [Kreipl, 2000] que juntos alcançam 0,02%.

Tabela 5.1: Resultados

Problema	MelhorLit	*LSRW(200)* - [Kreipl, 2000]		*GLS* - [Essafi *et al.*, 2008]		*SB–TS* - [Bülbül, 2011]		*TWTJSSP-ILS*		
-	-	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Melhor**	**Desv^{Melhor}**	**Tempo (s)**
abz5	**1403**	1451	0,03	**1403**	0,00	1487	0,06	1440	0,03	15,00
abz6	**436**	**436**	0,00	**436**	0,00	**436**	0,00	**436**	0,00	12,54
la16	**1169**	1170	0,00	**1169**	0,00	**1169**	0,00	**1169**	0,00	19,50
la17	**899**	900	0,00	**899**	0,00	**899**	0,00	**899**	0,00	12,98
la18	**929**	**929**	0,00	**929**	0,00	**929**	0,00	960	0,03	28,64
la19	**948**	951	0,00	**948**	0,00	955	0,01	970	0,02	25,78
la20	**805**	809	0,00	**805**	0,00	**805**	0,00	**805**	0,00	13,62
la21	**463**	464	0,00	**463**	0,00	**463**	0,00	**463**	0,00	48,61
la22	**1064**	1086	0,02	**1064**	0,00	1084	0,02	1086	0,02	14,82
la23	**835**	875	0,05	**835**	0,00	877	0,05	841	0,01	27,40
la24	**835**	**835**	0,00	**835**	0,00	**835**	0,00	**835**	0,00	33,71
orb01	**2568**	2616	0,02	2570	0,00	2630	0,02	2593	0,01	46,29
orb02	**1408**	1434	0,02	**1408**	0,00	**1408**	0,00	1427	0,01	18,40
orb03	**2111**	2204	0,04	**2111**	0,00	2186	0,03	2164	0,03	66,19
orb04	**1623**	1674	0,03	**1623**	0,00	1652	0,02	**1623**	0,00	88,37
orb05	**1593**	1667	0,05	1662	0,04	1667	0,05	1615	0,01	59,11
orb06	**1790**	1802	0,01	**1790**	0,00	**1790**	0,00	1802	0,01	39,59
orb07	**590**	618	0,05	**590**	0,00	616	0,04	**590**	0,00	48,71
orb08	**2429**	2554	0,05	2439	0,00	2503	0,03	2491	0,03	27,56
orb09	**1316**	1334	0,01	**1316**	0,00	**1316**	0,00	**1316**	0,00	35,91
orb10	**1679**	1775	0,06	**1679**	0,00	1801	0,07	1753	0,04	44,71
Média	**-**	-	0,02	-	0,00	-	0,02	-	0,01	34,64

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho teve seu foco no Problema de Sequenciamento de Máquinas do tipo *Job-Shop Scheduling* com penalidade pelo tempo de atraso em relação a data de entrega. Dada a dificuldade de resolução desse problema em tempos computacionais aceitáveis, foi proposto um algoritmo heurístico que combina os procedimentos *Iterated Local Search*, *Variable Neighborhood Descent*, Reconexão por Caminhos e a técnica de estrutura auxiliar de dados para fornecer suporte ao método de avaliação rápida.

O algoritmo, nomeado `TWTJSSP-ILS`, foi testado em 22 problemas-teste da literatura e comparado com três algoritmos eficientes da literatura. Claramente, o algoritmo de [Essafi *et al.*, 2008] foi o de melhor desempenho em todos os quesitos analisados. Entretanto, esse algoritmo possui alta complexidade de implementação. Esses últimos, por sua vez, têm desempenho bastante semelhante, com destaque para o `TWTJSSP-ILS` pelo fato de produzir soluções com o segundo menor desvio médio e se aproximarem muito aos melhores resultados encontrados na literatura. Além disso, o `TWTJSSP-ILS` é um algoritmo mais simples de ser implementado e requerer a calibragem de muito menos parâmetros, sendo então, um trabalho promissor e eficiente.

A continuidade deste projeto estará a cargo do novo bolsista, o aluno Raphael Carlos Cruz, que contribuiu para o desenvolvimento deste trabalho desde seu início. Assim, a inserção do módulo matemático *Branch and Bound* e a calibração adequada do algoritmo, bem como testar o algoritmo em novos conjuntos de instâncias são etapas que serão realizadas nos próximos dois meses. Estes aperfeiçoamentos a serem feitos têm como objetivos melhorar a qualidade dos resultados.

Salienta-se que o projeto está cumprindo com o cronograma em dia, conforme pode ser visualizado abaixo (6.1):

ATIVIDADES DESENVOLVIDAS (**AA** - atividades em andamento; **AC** - atividades concluídas; **AF** – atividades futuras)

Atividade prevista no plano de trabalho	AA	AC	AF	Observações
Estudo do problema abordado: Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times (UPMSP).		X		Foi realizado um estudo sistemático sobre os Problemas de Sequenciamento em Máquinas Paralelas. Em particular, pesquisou-se também algumas variantes do problema como é o caso do Job-Shop Scheduling, conforme se justifica no campo 8. (Informações complementares do projeto)
Estudo do problema abordado: Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times (UPMSP).		X		Realizou-se um estudo dos métodos heurísticos indicados com o intuito de entender suas aplicações no problema abordado.
Estudo do método Branch and Bound		X		O estudo do método Branch and Bound foi iniciado e tem o objetivo de resolver o problema associado a cada máquina. A Implementação está em andamento.
Pesquisa sobre métodos solução do UPMSP		X		Um vasto levantamento bibliográfico foi feito para conhecer os métodos mais usados para a resolução do problema abordado. Este estudo contemplou formas de representação da solução, vizinhanças, heurísticas e validação de algoritmos.
Estudo alg. [HSCC11]		X		Foi realizada uma investigação do algoritmo HSCC11 para realizar os aprimoramentos necessários.
Desenvolvimento e implementação de um novo procedimento construtivo para o UPMSP		X		Já se iniciou esta fase testando novos métodos de geração de solução inicial.
Desenvolvimento e implementação de novas estratégias de perturbação e buscas locais para o algoritmo proposto	X			Já se iniciou este fase testando novas estruturas de vizinhança. Estudos e aprimoramentos ainda devem ser realizados.
Validação/Testes	X			Realizado com Instâncias de tamanho 10 x 10 (10 máquinas e 10 Jobs). Maiores instâncias serão testadas.
Síntese/Análise dos Resultados	X			Realizado com Instâncias de tamanho 10 x 10 (10 máquinas e 10 Jobs). Maiores instâncias serão testadas.
Redação artigos/Relatórios	X			Desde o início foram documentadas todas as informações inerentes ao desenvolvimento do projeto para auxiliar na construção de artigos e relatórios.

Figura 6.1: Cronograma do Projeto

Referências

[Adams *et al.*, 1988] Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401.

[Applegate and Cook, 1991] Applegate, D. and Cook, W. (1991). A computational study of the job shop scheduling problem. *Journal of Computing*, 3:149–156.

[Bülbül, 2011] Bülbül, K. (2011). A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research*, 38:967–983.

[Coelho *et al.*, 2011] Coelho, I. M., Munhoz, P. L. A., Haddad, M. N., Coelho, V. N., Silva, M. M., Souza, M. J. F., and Ochi, L. S. (2011). A computational framework for combinatorial optimization problems. *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, 1:51–54.

[Essafi *et al.*, 2008] Essafi, I., Mati, Y., and Pérès, S. D. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35:2599–2616.

[Feo and Resende, 1995] Feo, T. A. and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133.

[Glover, 1997] Glover, F. (1997). A template for scatter search and path relinking. *Lecture Notes in Computer Science*. J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers, (Eds.).

[Gonçalves *et al.*, 2005] Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95.

[Hansen and Mladenović, 2001] Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

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

[Kreipl, 2000] Kreipl, S. (2000). A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3:125–138.

[Laarhoven *et al.*, 1992] Laarhoven, P. J. M. V., Aarts, E. H. L., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125.

[Lawrence, 1994] Lawrence, S. (1994). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. PhD thesis, Carnegie Mellon University, Estados Unidos.

[Lourenço, 1995] Lourenço, H. R. (1995). Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–364.

[Lourenço *et al.*, 2003] Lourenço, H. R., Martin, O., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

[Mine *et al.*, 2010] Mine, M. T., Silva, M. S. A., Ochi, L. S., and Souza, M. J. F. (2010). O problema de roteamento de veículos com coleta e entrega simultânea: uma abordagem via iterated local search e genius. In *Transporte em transformação XIV: trabalhos vencedores do prêmio CNT de Produção Acadêmica 2009*, pages 59–78. Editora Positiva, Brasília.

[Mladenović and Hansen, 1997] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100.

[Nowicki and Smutnicki, 1996] Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the job shop problem. *Management Science*, 42:797–813.

[Penna *et al.*, 2013] Penna, P. H. V., Subramanian, A., and Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, 19:201–232.

[Pinedo and Singer, 1998] Pinedo, M. and Singer, M. (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 30:109–118.

[Pinedo and Singer, 1999] Pinedo, M. and Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46:1–17.

[Stützle and Hoos, 1999] Stützle, T. and Hoos, H. H. (1999). Analyzing the run-time behaviour of iterated local search for the tsp. In *Proceeding of the Third Metaheuristics International Conference*, pages 449–453, Angra dos Reis, Rio de Janeiro.

[Volta *et al.*, 1995] Volta, R., Croce, G. D., and Tadei, F. (1995). A genetic algorithm for the job shop scheduling problem. *Computers & Operations Research*, 22:15–24.

[Walter, 1999] Walter, C. (1999). *Planejamento e Controle da Produção - PCP*. PhD thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre.