UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

RODRIGO RIBEIRO FRANCO

UM ALGORITMO ITERATED LOCAL SEARCH COM BUSCA LOCAL BASEADA EM VERY LARGE-SCALE NEIGHBORHOOD SEARCH PARA A PROGRAMAÇÃO DE MÁQUINAS PARALELAS

Ouro Preto, 2019

RODRIGO RIBEIRO FRANCO

UM ALGORITMO ITERATED LOCAL SEARCH COM BUSCA LOCAL BASEADA EM VERY LARGE-SCALE NEIGHBORHOOD SEARCH PARA A PROGRAMAÇÃO DE MÁQUINAS PARALELAS

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Marcone Jamilson Freitas Souza

Coorientador: Paulo Marcos de Barros Monteiro

Ouro Preto
Escola de Minas – UFOP
12/2019

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

F825u Franco, Rodrigo Ribeiro .
Um algoritmo Iterated Local Search com busca local baseada em Very Large-scale Neighborhood search para a programação de máquinas paralelas. [manuscrito] / Rodrigo Ribeiro Franco. Rodrigo Ribeiro Franco. - 2019.
45 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Marcone Jamilson Freitas Souza.
Coorientador: Prof. Dr. Paulo Marcos de Barros Monteiro.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. Programação paralela (Computação) . 2. Otimização combinatória - Very Large-scale Neighborhood Search (VLNS) . 3. otimização combinatória. 4. Otimização combinatória - Iterated Local Search (ILS). I. Franco, Rodrigo Ribeiro. II. Monteiro, Paulo Marcos de Barros . III. Souza, Marcone Jamilson Freitas. IV. Universidade Federal de Ouro Preto. V. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Digite o nome do autor (Apenas a primeira letra de cada nome, maiúscula)

Rodrigo Ribeiro Franco

Digite o título (Use letras maiúsculas apenas nos casos previstos nas regras ortográficas)

Um Algoritmo Interated Local Search com busca local baseada em Very Large-Scale Neighborhood Search para programação de máquinas paralelas

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 17 de dezembrode 2019

Membros da banca

Dr. Marcone Jamilson Freitas Souza - Orientador (ICEB - UFOP)
Dr. Paulo Marcos de Barros Monteiro -Coorientador (Escola de Minas - UFOP)
MSc Débora Nasser Diniz (ICEB - UFOP)

Paulo Marcos de Barros Monteiro, coorientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 27/05/2021

Documento assinado eletronicamente por **Paulo Marcos de Barros Monteiro**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 27/05/2021, às 20:50, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0176403** e o código CRC **51853E08**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591533 - www.ufop.br

AGRADECIMENTOS

À minha mãe Lourdes e ao meu pai Flávio, os grandes responsáveis pela minha formação, pelo amor e confiança. Ao meu irmão e minha família, por todas as vezes que me apoiaram e confiaram em mim.

Ao professor Marcone, que não mediu esforços para me ajudar e tornou este trabalho possível, participando como orientador, pesquisador e amigo, sendo para mim uma referência em todos os aspectos.

A todos os meus amigos que me ajudaram e contribuíram para esta conquista.

À Escola de Minas, por me proporcionar os ensinamentos do que é ser um engenheiro ético e competente.

A todos os professores que participaram, da minha formação profissional e crescimento pessoal.

À Capes, FAPEMIG e ao CNPq, pelos recursos disponibilizados para o desenvolvimento deste trabalho.

Finalmente, a todos que, direta ou indiretamente, contribuíram para a conclusão deste trabalho.

“Ideias e somente ideias podem iluminar a escuridão.” Ludwig von Mises

RESUMO

Este trabalho trata o problema de programação de máquinas paralelas com minimização do atraso. Neste problema, para cada tarefa é conhecido o tempo de processo, a data de entrega e o peso por dias de atraso. O objetivo é alocar as tarefas às máquinas minimizando a soma dos atrasos ponderados. Para resolvê-lo, foram desenvolvidas duas versões de um algoritmo baseado na metaheurística *Iterated Local Search* (ILS). Esse algoritmo explora o espaço de soluções usando movimentos de realocação e troca entre tarefas de máquinas diferentes, assim como movimentos de troca e inversão de subsequências de tarefas de uma mesma máquina. A versão denominada ILS-Clássica usa nas buscas locais somente os movimentos clássicos definidos acima, enquanto a versão denominada ILS-VLNS utiliza a técnica *Very Large-scale Neighborhood Search* (VLNS) como método de busca local do ILS que realiza operações do tipo *n-optimal*. Foram realizados testes computacionais com instâncias da literatura nas quais as soluções ótimas são conhecidas e também com instâncias maiores, nas quais as soluções ótimas não são conhecidas. Foi possível verificar que a versão ILS-VLNS produziu resultados melhores do que a versão ILS-Clássica; porém seus resultados não superaram os da literatura.

PALAVARAS-CHAVE. Programação de máquinas paralelas, Very Large-scale Neighborhood Search, Iterated Local Search.

Área principal: Logística e Transportes, Metaheurísticas

ABSTRACT

This work presents two versions of the Iterated Local Search (ILS) metaheuristic for the parallel-machine total tardiness problem. In this problem, we have known the processing time, the due date, and the weight for tardiness for each job. The objective is to assign jobs to the machines minimizing the sum of the weighted tardiness. We developed two versions of an algorithm based on the Iterated Local Search (ILS) metaheuristic to address it. This algorithm explores the solution space using moves of reallocation and exchange between jobs of different machines and moves of swap and twist of subsequences of jobs of the same machine. The version called ILS-Classical uses only the classic moves defined above. In contrast, the version called ILS-VLNS uses the Very Large-scale Neighborhood Search (VLNS) technique to realize n-optimal local searches. We perform computational tests with instances of the literature in which the optimal solutions are known and also with larger instances, in which the optimal solutions are unknown. It was possible to verify that the ILS-VLNS version produced better results than the ILS-Classic version, but its results did not outperform those of the literature.

KEYWORDS. Parallel-machine Total Tardiness Problem, Very Large-scale Neighborhood Search, Iterated Local Search.

LISTAS DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BIS *Best Improvement Search*

DP Desvio Padrão

EDD *Earliest Due Date*

FIS *First Improvement Search*

FO Função Objetivo

GPI *Generalized Pairwise Interchange*

ILS Busca Local Iterativa (*Iterated Local Search*)

PAMP Problema de Atraso total ponderado em Máquinas Paralelas

RIS Random Improvement Search

RPD *Relative Percentage Deviation*

SA *Simulated Annealing*

VLNS *Very Large-Scale Neighborhood Search*

VNS *Variable Neighborhood Search*

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização e revisão de literatura

O problema de programação de máquinas paralelas com minimização da soma do atraso ponderado consiste em atribuir um conjunto de *n* tarefas a um conjunto de *m* máquinas de tal forma que cada tarefa seja executada por uma única máquina. Em determinados casos, as tarefas exigem um determinado tempo de preparação da máquina para a sua execução. Este tempo de preparação pode ser constante, como também pode depender da sequência em que as tarefas são realizadas.

Existe uma grande variedade de problemas de sequenciamento de tarefas. Este trabalho trata do Problema de Atraso total ponderado em Máquinas Paralelas (PAMP) e idênticas sem tempo de preparação, ou seja, o *parallel-machine total tardiness problem*, denotado na literatura por $Pm||\sum w_j T_j$.

Em problemas deste tipo, é dado um conjunto de tarefas $N = \{1, 2, ..., n\}$, com o tempo de processamento p_j, pesos w_j e as datas de entrega d_j para cada tarefa $j \in N$. As tarefas são processadas em um conjunto $M = \{1, 2, ..., m\}$ de máquinas idênticas paralelas de modo que a soma ponderada dos atrasos T_j na execução de cada tarefa em relação às suas datas de entrega seja minimizada. A Equação (1) mostra como uma solução *s* é avaliada pela função *T* de atraso.

$$T(s) = \sum_{j=1}^{n} w_j T_j = \sum_{j=1}^{n} w_j \max\{C_j - d_j, 0\} \quad (1)$$

Na Equação (1), w_j é a penalização por atraso e C_j é o tempo de término da tarefa *j*, calculado como $C_j = \max\{0, C_j - d_j\}$. Assim, se $C_j > d_j$, isto é, se a tarefa *j* for entregue com atraso, então o atraso $C_j - d_j$ será ponderado pelo fator w_j de penalização; caso contrário, não haverá penalização.

De acordo com Lenstra *et al.* (1977), este problema é NP-difícil; por esta razão, são utilizadas metaheurísticas para se obter boas soluções em um tempo de processamento razoável (Du & Leung, 1990). Para $m = 1$, temos o problema de atraso total de uma única máquina, bem estudado e resolvido, tanto com métodos exatos quanto com métodos heurísticos (Congram *et al.*, 2002; Grosso *et al.*, 2004; Pan e Shi, 2007). A seguir são destacados os primeiros trabalhos que abordam o problema tratado neste trabalho.

Koulamas (1997) apresenta uma extensão do princípio da decomposição do problema de atraso em uma única máquina para uma configuração de máquinas paralelas. Os subproblemas gerados pela decomposição são resolvidos por uma heurística eficaz que produz soluções de tal forma que a programação em qualquer máquina do conjunto satisfaz o princípio de decomposição de uma única máquina. O autor também apresenta um desenvolvimento híbrido da metaheurística *Simulated Annealing* para resolver o problema. Os resultados computacionais mostram a eficiência e eficácia da heurística de decomposição.

No trabalho de Bilge *et al*. (2004) foi considerado o problema de sequenciamento de um conjunto de tarefas independentes com tempo de *setup* dependente da sequência, em um conjunto de máquinas paralelas e uniformes com minimização do atraso total. Os autores tratam o problema com a metaheurística Busca Tabu (BT). A fim de obter um mecanismo de busca robusta, vários componentes chave da BT, como estratégias para considerar a lista de vizinhos candidatos, classificação de movimentos tabus, duração tabu e estratégias de intensificação e diversificação são exploradas. Abordagens alternativas para cada uma destas questões são desenvolvidas e testadas com um conjunto de problemas obtidos a partir da literatura. Os resultados apresentados são consideravelmente melhores do que aqueles relatados anteriormente, e constituem as melhores soluções conhecidas, até então, para instâncias *benchmark* do problema.

Anghinolfi e Paolucci (2007) abordam o problema por meio de uma metaheurística híbrida (HMH) que integra várias funcionalidades de BT, *Simulated Annealing* (SA) e *Variable Neighborhood Search* (VNS) em um algoritmo configurável de sequenciamento. Em particular, pode ser usada uma estratégia determinística ou randômica para gerar a lista de vizinhos candidatos de uma solução. Tanto um mecanismo de lista tabu como uma regra probabilística do tipo SA pode ser adotada para aceitar soluções, e a dimensão da vizinhança explorada pode ser dinamicamente modificada. O problema definido por Bilge *et al.* (2004) é abordado neste trabalho. Várias configurações alternativas da HMH foram testadas com o mesmo conjunto de dados utilizado por Bilge *et al*. (2004). Os resultados obtidos destacam a adequação da abordagem proposta.

Rodrigues *et al.* (2008) apresentam um algoritmo heurístico para o problema, cuja principal característica inovadora está em representar a solução do problema de programação de múltiplas máquinas por uma única sequência, simplificando o tratamento desse problema. A representação como uma única sequência é otimizada por uma Busca Local Iterativa (*Iterated Local Search* – ILS) utilizando os movimentos definidos como *Generalized*

Pairwise Interchange – GPI, melhorado com um critério de desempate adequado. A ideia do critério de desempate é de que mesmo que a solução corrente não tenha nenhum vizinho GPI com melhor custo, a busca não para se houver vizinhos melhorando o critério proposto. Testes extensivos em instâncias com 2 e 4 máquinas, e com até 50 postos de trabalho, obtiveram as melhores soluções em quase todos os casos.

A busca *Very Large-scale Neighborhood Search* (VLNS) é um método de busca local que explora vizinhanças complexas de problemas de otimização combinatória. Tais vizinhanças são caracterizadas por ter um número exponencial de soluções vizinhas, mas que podem ser exploradas em tempo polinomial por métodos exatos ou heurísticos (Ahuja, 2002). Os trabalhos de Congram *et al*. (2002) e Ergun e Orlin (2006) são exemplos da aplicação da técnica VLNS, chamada *dynasearch*, para o problema $1||\sum w_j T_j$.

Della Croce *et al.* (2012) apresentam uma implementação da metaheurística ILS para resolver o PAMP. A metaheurística combina a busca em vizinhança GPI, a busca do tipo VLNS e uma nova vizinhança baseada nas máquinas (*machine-based*) cujo tamanho é não-polinomial em relação ao número de máquinas. Particularmente, os autores introduzem uma busca em vizinhança do tipo *dynasearch* em cada máquina. Neste caso, a busca *dynasearch* foi implementada por meio da técnica de Programação Dinâmica para fazer uma busca em uma vizinhança de tamanho exponencial, em tempo polinomial (Congram *et al.*, 2002.

Conforme descrito anteriormente, Della Croce *et al.* (2012) propõem um algoritmo ILS para o problema baseada em operadores GPI e na busca *dynasearch*. Baseado no trabalho desses autores, implementamos a metaheurística ILS, utilizando a técnica VLNS como método de busca local. Diferentemente de Della Croce *et al.* (2012), a busca local foi empregada para realizar a otimização entre máquinas, e não na otimização intra-máquinas como fizeram aqueles autores. Na etapa de otimização intra-máquina foram implementados os operadores de busca local do tipo GPI.

Esta monografia reproduz o trabalho publicado em Franco e Silva (2016), sem nenhuma atualização da literatura desde então e sem nenhum teste adicional. Entretanto, procura-se detalhar melhor os algoritmos desenvolvidos naquele trabalho.

1.2 Justificativa

O problema de sequenciamento de máquinas é muito estudado e tem grande importância prática, pois está associado à alocação de recursos a um conjunto de tarefas em um dado tempo. Escalonar as tarefas significa designar recursos para a sua execução até que

todas as tarefas tenham sido executadas sob as restrições impostas, com o objetivo de minimizar os custos envolvidos com a operação. Aplicações desse problema podem ser encontradas em diversas situações reais, como no planejamento da produção de uma empresa, no gerenciamento de projetos e no sequenciamento de tarefas das unidades centrais de processamento (CPU) de uma rede de computadores. A minimização do atraso nesses problemas é de vital importância para as empresas, porque entregas além da data prevista, além de gerarem insatisfação do cliente, são acompanhadas de multas contratuais. Assim, entregar a produção no prazo elimina esses dois aspectos da relação entre a empresa e o cliente.

Devido à grande concorrência existente entre as empresas, o custo de produção tem grande importância. A redução desses custos pode implicar em um menor preço do produto e/ou serviço, proporcionando à organização maior competitividade frente ao mercado global.

1.3 Objetivos

1.3.1 Geral

Este trabalho tem como objetivo estudar e implementar duas variantes de um algoritmo heurístico baseado na metaheurística *Iterated Local Search* (ILS) para resolver o problema de sequenciamento de tarefas em máquinas paralelas com minimização da soma ponderada dos atrasos (*Parallel-machine total weighted tardiness problem*). Na primeira variante, denominada ILS-Clásico, será utilizado um método clássico de busca local; enquanto a segunda variante, denominada ILS-VLNS, terá o método VLNS (Ahuja et al., 2002) como método de busca local.

1.3.2 Específicos

- Analisar e estudar trabalhos da literatura que tratam o problema abordado;
- Analisar e estudar técnicas de solução de problemas de otimização combinatória;
- Implementar uma heurística construtiva gulosa para o problema;
- Implementar heurísticas de busca local intra-máquina e entre-máquinas;
- Implementar algoritmos metaheurísticos baseados em ILS utilizando dois tipos de busca local, *Best Improvement Search* (BIS) e *Very Large-scale Neighborhood Search* (VLNS) (Ahuja *et al*. 2002);
- Testar os algoritmos metaheurísticos com instâncias *benchmark* disponíveis na literatura.

1.4 Estrutura do trabalho

O restante desta monografia está assim estruturado. No Capítulo 2, é apresentado o referencial teórico deste trabalho. No Capítulo 3, são apresentados os algoritmos desenvolvidos para tratar o problema objeto de estudo. No Capítulo 4, são apresentados e discutidos os resultados obtidos pela aplicação dos algoritmos desenvolvidos. Por fim, este trabalho é concluído no Capítulo 5.

2 REFERENCIAL TEÓRICO

Neste capítulo é apresentado o referencial teórico de técnicas de otimização utilizadas neste trabalho para tratar o problema em estudo.

2.1 Métodos Heurísticos

Um método heurístico é um método aproximado projetado com base nas propriedades estruturais ou nas características das soluções dos problemas, com complexidade reduzida em relação à dos algoritmos exatos, que fornece, em geral, soluções viáveis de boa qualidade para um problema de otimização.

2.1.1 Heurísticas Construtivas

Segundo Souza (2011), uma heurística construtiva constrói uma solução, elemento por elemento. A escolha do elemento que será inserido na solução corrente vai depender da função objetivo do problema.

Esses métodos são comumente utilizados para a construção de uma solução inicial, pois, geralmente são rápidos. A solução obtida não será necessariamente satisfatória. Isto implica que esta solução pode requerer um refinamento após sua construção. Eles também podem ser utilizados como ponto de partida para outras heurísticas de refinamento ou metaheurísticas.

2.1.1 Heurísticas de Refinamento

Ainda de acordo com Souza (2011), as heurísticas de refinamento, também chamadas de métodos de busca local, são métodos de otimização que consistem no refinamento de uma solução. A partir dela, define-se uma vizinhança através de uma regra, chamada de movimento, que a modifica.

Estas heurísticas iniciam com uma solução do problema, que pode ser gerada a partir de uma heurística construtiva ou aleatoriamente, buscando-se, a cada iteração, em sua vizinhança já definida, novas soluções que possam melhorar o valor da função de avaliação do problema. A eficiência desse método depende da solução inicial e da definição de uma vizinhança que estabelece uma relação entre as soluções no espaço de decisões.

Para problemas de otimização, este método pode ser utilizado para melhorar um conjunto de soluções dominantes geradas por uma heurística construtiva.

2.1.1.1.1 Best Improvement Search

Para Souza (2011), o Método de Melhor Melhora ou *Best Improvement Search* (BIS), baseia-se na análise de toda vizinhança de uma solução qualquer, movendo somente para aquele que representar uma melhora no valor atual da função objetivo.

O Algoritmo 1 mostra o pseudocódigo do método BIS aplicado à minimização da função *T* dada pela Equação (1) a partir de uma solução inicial conhecida *s* e considerando a busca em uma dada vizinhança *N*(.). Na linha 1, começa uma estrutura de repetição, que verifica se existe uma solução *s'* vizinha de *s* na vizinhança *N* melhor do que *s* segundo a função *T*. Na linha 2, o algoritmo analisa toda a vizinhança da solução *s* e retorna o melhor vizinho *s** da solução corrente *s*. Na linha 3, essa solução *s** passa a ser a nova solução *s* corrente. Por fim, na linha 5 é retornada a melhor solução encontrada pelo algoritmo.

```
procedimento BIS(T(.), N(.), s);
   1. enquanto ∃ s'∈ N(s) | T(s') < T(s) faça
   2.   Seja s* tal que T(s*) = arg min{T(s') | s'∈ N(S)};
   3.   s ← s*;
   4. fim_enquanto;
   5. retorne s;
   fim BIS;
```

Algoritmo 1: Pseudocódigo do método *Best Improvement Search*

2.1.1.1.2 First Improvement Search

Segundo Souza (2011), o método de Descida Primeira Melhora ou *First Improvement Search* (FIS) explora a vizinhança de uma solução corrente, porém ela interrompe a exploração quando um vizinho melhor é encontrado. Ou seja, ao contrário do método BIS, esse método não analisa toda a vizinhança para depois escolher o melhor movimento de descida. Entretanto, quando não há um movimento de melhora, toda a vizinhança é explorada. Assim, o método também retorna um ótimo local com relação à vizinhança considerada.

Neste método é desejável que a ordem de exploração das soluções vizinhas seja alterada a cada passo; do contrário, privilegia-se apenas um caminho determinístico no espaço de soluções. Por exemplo, em um problema de programação de tripulações, imagine que um

movimento consista em realocar uma tarefa de uma tripulação para outra tripulação. Se a cada passo, a ordem de exploração das soluções vizinhas começar com a movimentação da primeira tarefa do primeiro tripulante para o segundo tripulante, depois para o terceiro, a seguir para o quarto e assim sucessivamente, então os tripulantes iniciais serão privilegiados. Para evitar isso, uma alternativa de implementação é utilizar uma sequência diferente de tripulações a cada iteração do método. Assim, na primeira iteração do método usa-se uma sequência de tripulantes, por exemplo, (1, 2, 3, 4, · · ·, n); outra sequência diferente na segunda iteração, por exemplo, (7, 4, 1, 9, · · ·) e assim por diante.

2.1.1.1.3 Random Descent Search

De acordo com Souza (2011), o método Descida Randômica (RDS, das iniciais em inglês *Random Descent Search*), consiste em analisar um vizinho qualquer e o aceitar somente se ele for melhor que a solução corrente; não sendo, a solução corrente permanece inalterada e outro vizinho é gerado. O procedimento é interrompido após um número fixo de iterações sem melhora no valor da melhor solução obtida até então.

Como neste método não é feita a exploração de toda a vizinhança da solução corrente, não há garantia de que a solução final seja um ótimo local.

No Algoritmo 2 mostra-se o pseudocódigo do Método de Descida Randômica aplicado ao refinamento de uma solução *s* em um problema de minimização do valor retornado pela função *T*(.), utilizando uma estrutura de vizinhança *N*(.). Neste algoritmo, *IterMax* representa o número máximo de iterações sem melhora no valor da função de avaliação.

Inicialmente, na linha 1, uma variável contadora do número de iterações sem melhora é inicializada com o valor 0. Na linha 2 começa uma estrutura de repetição que tem como critério de parada o número *IterMax* de iterações. Na linha 4, uma solução vizinha *s*' da solução *s* é selecionada aleatoriamente. Na linha 5, é testado se essa solução vizinha é melhor que a solução corrente. Se for melhor, na linha 6 a variável contadora do número de iterações sem melhora retorna ao seu valor inicial e na linha 7, a solução *s* recebe a nova solução *s*' melhorada. O procedimento termina quando após *IterMax* iterações não houver melhora na solução corrente.

```
procedimento RDS(T(.), N(.), IterMax, s);
1.Iter ← 0;
2. enquanto (Iter < IterMax) faça
3.  Iter ← Iter + 1;
4.  Selecione aleatoriamente s' ∈ N(s);
5.  se (T(s') < T(s)) então
6.    Iter ← 0;
7.    s ← s';
8.  fim-se;
9.fim_enquanto;
10.retorne s;
fim DescidaRandomica;
```

Algoritmo 2: Pseudocódigo do método Descida Randômica

2.1.1.2 *Very Large-Scale Neighborhood Search*

Esta técnica de busca *Very Large Neighborhood Search* (VLNS) é uma generalização da vizinhança de realocação e troca aos pares, onde um vizinho é obtido através da realização de uma *troca cíclica* ou de uma *troca em caminho*. A vizinhança de troca cíclica envolve a troca aos pares, mas também permite que tarefas de três ou mais máquinas estejam envolvidas na troca. Considerando o caso de três máquinas e de troca cíclica, uma tarefa da máquina 1 é realocada para a máquina 2, que por sua vez tem uma tarefa realocada para a máquina 3, que finalmente tem uma tarefa realocada para a máquina 1. No caso da troca em caminho, a máquina 3 não realoca qualquer tarefa para a máquina 1 e após o movimento a máquina 1 terá uma tarefa a menos, assim como a máquina 3 contará com uma tarefa além do que havia anteriormente.

Para que seja possível realizar uma busca na vizinhança de troca cíclica ou em caminho, de forma eficiente, é necessário construir um grafo direcionado que represente a vizinhança de uma dada solução *s*. Tal grafo, denominado *grafo de melhoria G*(*s*), é

definido para cada solução factível *s* do problema. Considerando *n* tarefas e *m* máquinas, seja s[*j*] a máquina que contém a tarefa *j*, então, $G(s) = (\boldsymbol{N}, \boldsymbol{E})$ é um grafo direcionado com o conjunto de nós $\boldsymbol{N}$ contendo um nó para cada tarefa $i = 1, .., n$, um nó para cada máquina $j = n+1, ..., n+m$, e um super-nó $t = n+m+1$. O conjunto $\boldsymbol{E}$ contém os seguintes arcos:

- (i, j) de uma tarefa *i* para outra tarefa *j* que representa a substituição da tarefa *j* pela tarefa *i* na máquina *s*[*j*] que contém a tarefa *j*. O custo deste tipo de arco é dado por $T(\{i\} \cup s[j] \setminus \{j\}) - T(s[j])$;
- (i, m_j) de uma tarefa *i* para uma máquina m_j, sendo $m_j \neq s[i]$, que representa a realocação da tarefa *i* para a máquina m_j. Neste caso, o custo do arco é calculado pela expressão $T(\{i\} \cup m_j) - T(m_j)$;
- (t, j) do super-nó *t* para cada tarefa *j*, que considera a retirada da tarefa *j* da máquina em que se encontra. O custo deste arco é dado por $T(s[j] \setminus \{j\}) - T(S[j])$;
- (m_j, t) de cada máquina m_j para o super-nó *t*, que permite a formação de ciclos. Esses arcos têm custo zero.

Um ciclo negativo no grafo $G(s)$ corresponde a uma troca cíclica que melhora o valor da função objetivo referente à solução *s*. Esta é uma maneira eficiente de buscar por soluções vizinhas de *s* que melhoram a função objetivo. Portanto, basta encontrar um ciclo negativo no grafo direcionado $G(s)$ para se obter um vizinho melhor. Neste trabalho foi implementado o algoritmo de *Walk to the root* (Cherkassky e Goldberg, 1999) que tem complexidade $O(n^2m)$, ou seja, apresenta complexidade polinomial.

No Algoritmo 3, mostra-se um pseudocódigo do algoritmo VLNS. Nesse algoritmo, s, descrito na linha 1, representa uma solução corrente. Com essa solução, na linha 2 é gerado um grafo de melhoria *G(D)*. Uma vez encontrado um ciclo negativo, as trocas referentes ao ciclo são realizadas, e o grafo de melhoria é atualizado na linha 6. O processo é repetido até que nenhum ciclo negativo seja encontrado no grafo de melhoria. Quando o grafo não apresentar qualquer ciclo negativo, então a solução corrente é uma solução ótima local segundo a vizinhança considerada.

```
procedimento VLNS
   1. s ← Solução corrente;
   2. Construção do grafo de melhoria G(D) referente a s;
   3. enquanto G(D) tiver ciclos negativos faça
   4.   Identifique um ciclo negativo em G(D);
   5.   Melhore a solução s devido às trocas do ciclo
        negativo;
   6.   Atualize o grafo de melhoria G(D);
   7. fim-enquanto;
   8. retorne s;
   fim VLNS;
```

Algoritmo 3: Pseudocódigo do método de busca VLNS

2.1.2 Metaheurísticas

Para Souza (2011), as metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, ou seja, cada método de busca de uma heurística, a qual tem que ser modelada para cada problema específico.

Contrariamente às heurísticas convencionais, as metaheurísticas são de caráter geral e providas de mecanismos para evitar que se fique preso em ótimos locais.

As metaheurísticas diferenciam-se entre si basicamente pelo mecanismo usado para não ficar preso em ótimos locais. Elas se dividem em duas categorias, de acordo com o princípio usado para explorar o espaço de soluções: busca local e busca populacional.

Nas metaheurísticas baseadas em busca local, 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. Busca Tabu, *Simulated Annealing*, Busca em Vizinhança Variável (*Variable Neighborhood Search*) e *Iterated Local Search* e Busca Populacional são exemplos de métodos que se enquadram nesta categoria.

2.1.2.1 Metaheurística *Iterated Local Search* (ILS)

Souza (2011) afirma que o método ILS é baseado na ideia de que um procedimento de busca local pode ser melhorado gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução ótima local.

Para aplicar um algoritmo ILS, quatro componentes têm que ser especificadas:

(a) Função GeraSolucaoInicial(), que gera uma solução inicial s_0 para o problema;

(b) Função BuscaLocal, que retorna uma solução possivelmente melhorada s'' ou s caso contrário;

(c) Função Perturbacao, que modica a solução corrente s guiando a uma solução intermediária s ′

(d) Função CriterioAceitacao, que decide de qual solução a próxima perturbação será aplicada.

No Algoritmo 4 mostra-se o pseudocódigo do algoritmo ILS clássico.

```
procedimento ILS
  1. s0 ← GeraSolucaoInicial();
  2. s ← BuscaLocal(s0);
  3. enquanto (os critérios de parada não estiverem
     satisfeitos) faça
  4.   s' ← Perturbacao(histórico, s);
  5.   s'' ← BuscaLocal(s');
  6.   s ← CriterioAceitacao(s, s'', histórico);
  7. fim-enquanto;
  fim ILS;
```

Algoritmo 4: Pseudocódigo da metaheurística ILS

O sucesso do ILS é centrado no conjunto de amostragem de ótimos locais, juntamente com a escolha do método de busca local, das perturbações e do critério de aceitação. Em princípio, qualquer método de busca local pode ser usado, mas o desempenho do ILS com respeito à qualidade da solução final e a velocidade de convergência depende fortemente do método escolhido. Normalmente um método de descida é usado, mas também é possível aplicar algoritmos mais sofisticados, tais como Busca Tabu ou outras metaheurísticas.

A intensidade da perturbação deve ser forte o suficiente para permitir escapar do ótimo local corrente e permitir explorar diferentes regiões. Ao mesmo tempo, ela precisa ser fraca o suficiente para guardar características do ótimo local corrente.

O critério de aceitação é usado para decidir de qual solução se continuará a exploração, bem como qual será a perturbação a ser aplicada. Um aspecto importante do critério de aceitação e da perturbação é que eles induzem aos procedimentos de intensificação e diversificação. A intensificação consiste em permanecer na região do espaço onde a busca se encontra, procurando explorá-la de forma mais efetiva; enquanto a diversificação consiste em se deslocar para outras regiões do espaço de soluções. A intensificação da busca no entorno da melhor solução encontrada é obtida, por exemplo, pela aplicação de pequenas perturbações sobre ela. A diversificação, por sua vez, pode ser realizada aceitando-se quaisquer soluções s ″ e aplicando "grandes" perturbações na solução ótima local.

Um critério de aceitação comumente utilizado é mover-se para o ótimo local *s″* somente se ele for melhor que o ótimo local corrente *s*, isto é, somente se $f(s'') < f(s)$ em um problema de minimização, ou se $f(s'') > f(s)$ em um problema de maximização.

A Figura 2 ilustra o funcionamento do método ILS em um problema de minimização. A partir de um ótimo local s, é feita uma perturbação que guia a uma solução intermediária *s′*. Após a aplicação de um método de busca local a *s'* é gerado um novo ótimo local *s″*. Considerando como critério de aceitação o fato de *f(s″)* ser melhor que *f*(s), então a busca prossegue a partir de *s ″*.

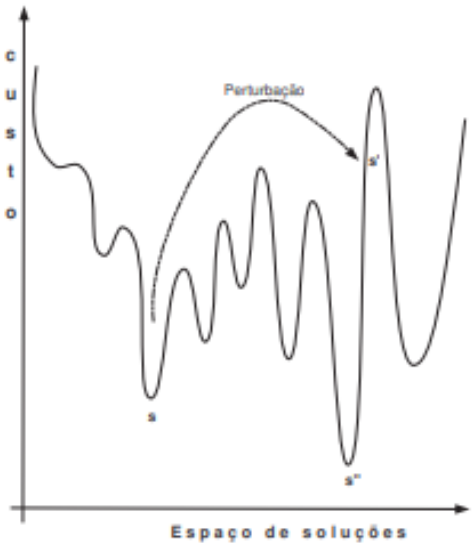

Figura 1: Representação esquemática do funcionamento do ILS

Para definir o que seria uma perturbação no Problema do Caixeiro Viajante, consideremos uma estrutura de vizinhança que utilize movimentos de troca de posição de

duas cidades para gerar vizinhos. Uma perturbação poderia ser dividida em vários níveis de intensidade. Assim, por exemplo, uma perturbação de nível 1 poderia consistir na realização de duas trocas aleatórias. A perturbação de nível 2 consistiria na execução de três trocas aleatórias sobre uma mesma solução e assim sucessivamente. O algoritmo então funcionaria da seguinte maneira: para cada nível de perturbação seria realizada uma busca local, a qual, se bem sucedida, faria retornar ao nível mínimo de perturbação; caso contrário, seriam realizadas mais algumas buscas locais no mesmo nível de perturbação. Em caso de insucesso destas buscas locais, o nível de perturbação seria gradativamente aumentado. O método encerraria após um certo número de iterações sem melhora ou quando um tempo limite fosse atingido.

Para o problema da mochila 0-1, em que se considera como movimento a troca no valor de um bit, uma perturbação de nível 1 poderia ser a troca no valor de dois bits simultaneamente, a perturbação de nível 2 seria a troca no valor de três bits simultaneamente e assim sucessivamente. Igualmente ao caso anterior, para cada nível de perturbação (*nperturb*) seriam executadas várias buscas locais, digamos *nbuscas*.

3 DESENVOLVIMENTO

Descreve-se, a seguir, a adaptação feita ao ILS para tratar o problema objeto de estudo. A metaheurística ILS é um arcabouço de busca local que pode ser visto como um *tradeoff* entre a metaheurística *Multi-Start* (MS) é uma metaheurística mais complexa como a Busca Tabu ou Algoritmos Genéticos. Na MS, uma busca local é iniciada a partir de diferentes soluções geradas aleatoriamente. Este processo se repete normalmente um várias vezes a fim de escapar de ótimos locais e explorar um grande conjunto de soluções. Em metaheurísticas mais complexas, um número sofisticado de estratégias como memória de curto e longo prazo, cruzamentos genéticos, entre outros, são aplicados a fim de fugir de ótimos locais. No ILS, a busca é recomeçada a partir de uma perturbação realizada nas melhores soluções conhecidas. Com este tipo de recomeço, a solução de partida de cada busca local não é completamente aleatória e a perturbação, no caso chamada de *kick*, tenta projetar a busca "não muito longe" do ótimo local anteriormente explorado, ou seja, sem perder completamente suas estruturas parcialmente otimizadas (Lourenço et al. 2001). A seguir são descritos os procedimentos que compõem a metaheurística ILS implementada neste trabalho.

3.1 Geração da Solução Inicial

Este procedimento gera uma solução factível, de boa qualidade, para dar início ao processo de otimização. No caso foi empregada a estratégia do *Earliest Due Date* – EDD (Baker, 1984), que consiste em ordenar as tarefas priorizando aquelas com data de entrega mais próxima. Após a ordenação crescente da data de entrega das tarefas, deve ser verificado no final de qual máquina a tarefa de menor data de entrega deve ser alocada, de tal forma que incorra no menor atraso possível. A tarefa é alocada no final desta máquina e o processo se repete até que todas as tarefas tenham sido alocadas

3.2 Estruturas de Vizinhança

3.2.1 Estrutura de Vizinhança para uma Única Máquina

Dois tipos diferentes de movimentos foram usados para definir a vizinhança N_1, que explora o espaço de soluções do problema considerando apenas as operações em uma única máquina. Esses movimentos, nomeados troca intra-máquina e *twist*, funcionam como segue.

Seja a sequência de tarefas em uma máquina dada por $\sigma = \alpha i \pi \omega j \beta$, onde as tarefas i e j estão nas posições k e l, respectivamente e α, π, ω e β são outras tarefas. No movimento troca intra-máquina, a sequência $\alpha i \pi \omega j \beta$ resulta em $\alpha j \pi \omega i \beta$, como mostrado na Figura 3.

Dois tipos diferentes de movimentos foram usados para definir a vizinhança N_1, que explora o espaço de soluções do problema considerando apenas as operações em uma única máquina. Esses movimentos, nomeados troca intra-máquina e *twist*, funcionam como segue.

Seja a sequência de tarefas em uma máquina dada por $\sigma = \alpha i \pi \omega j \beta$, onde as tarefas i e j estão nas posições k e l, respectivamente e α, π, ω e β são outras tarefas. No movimento troca intra-máquina, a sequência $\alpha i \pi \omega j \beta$ resulta em $\alpha j \pi \omega i \beta$, tal como mostrado na Figura 3.

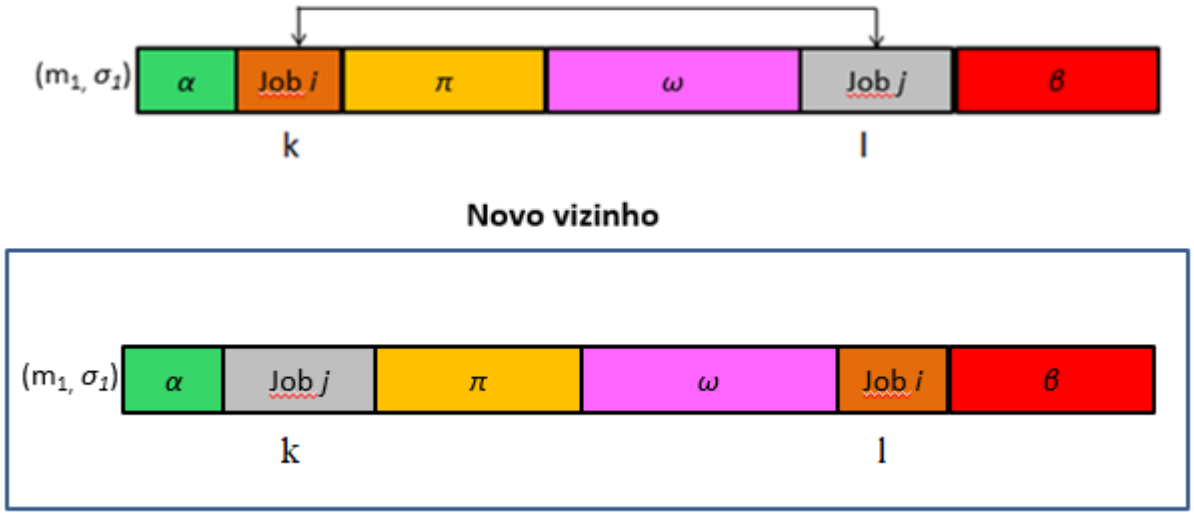

Figura 2: Movimento de Troca Intra-Máquina

Por outro lado, com o movimento *twist*, a sequência $\alpha i \pi \omega j \beta$ torna-se $\alpha j \omega \pi i \beta$, como na Figura 4.

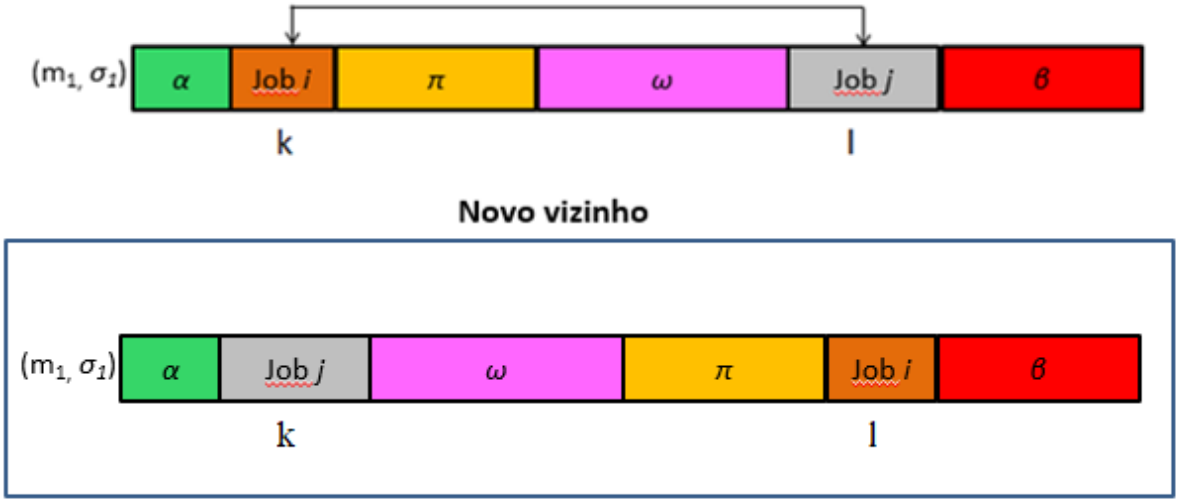

Figura 3: Movimento *Twist*

3.2.2 Estruturas de Vizinhança para Máquinas Paralelas

Dois outros tipos diferentes de movimentos foram usados para definir a vizinhança N_2, que explora o espaço de soluções do problema considerando apenas operações entre máquinas. Esses movimentos, nomeados realocação entre-máquinas e troca entre-máquinas, funcionam como segue.

Seja um par de máquinas m_1 e m_2, com os respectivos sequenciamentos de tarefas σ_1 e σ_2. Os movimentos considerados são os seguintes:

i) realocação entre-máquinas: extrai-se uma tarefa *i* de σ_1 e a insere em σ_2, ou seja, faz-se a *realocação* da tarefa *i* da máquina m_1 para a máquina m_2;

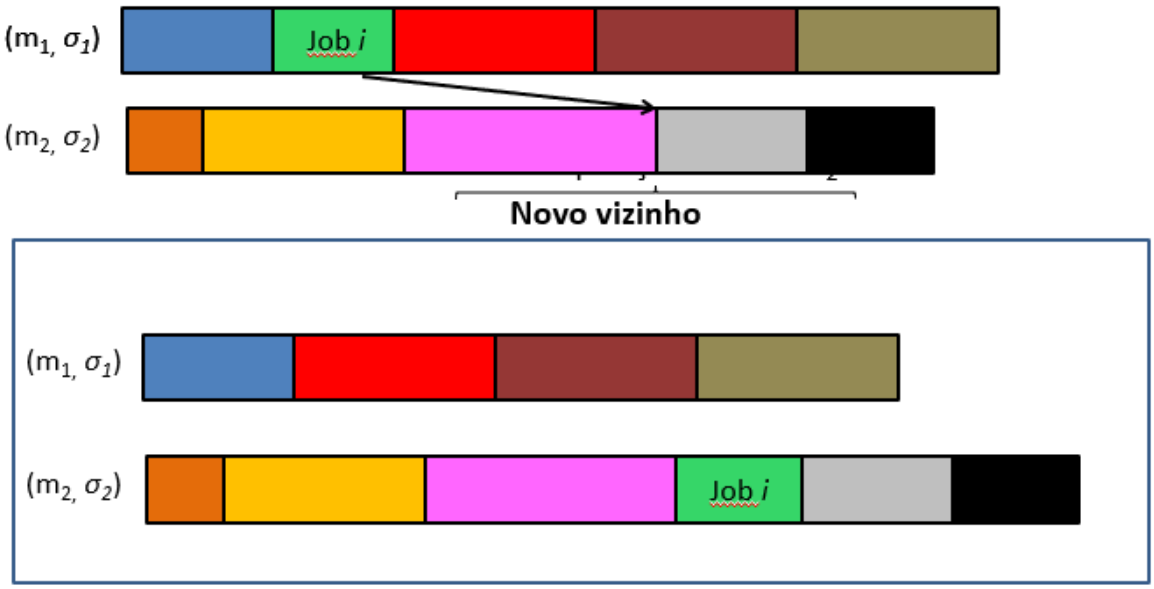

Figura 4: Realocação Entre-Máquinas

ii) troca entre-máquinas: extrai-se uma tarefa *i* pertencente à σ_1 e uma tarefa *j* pertencente a σ_2. Insere-se *i* em σ_2 na posição onde estava *j*, e insere-se *j* em σ_1 na posição onde estava *i*, ou seja, faz-se a *troca* das tarefas *i* e *j* entre as máquinas m_1 e m_2. A Figura 6 ilustra essa operação.

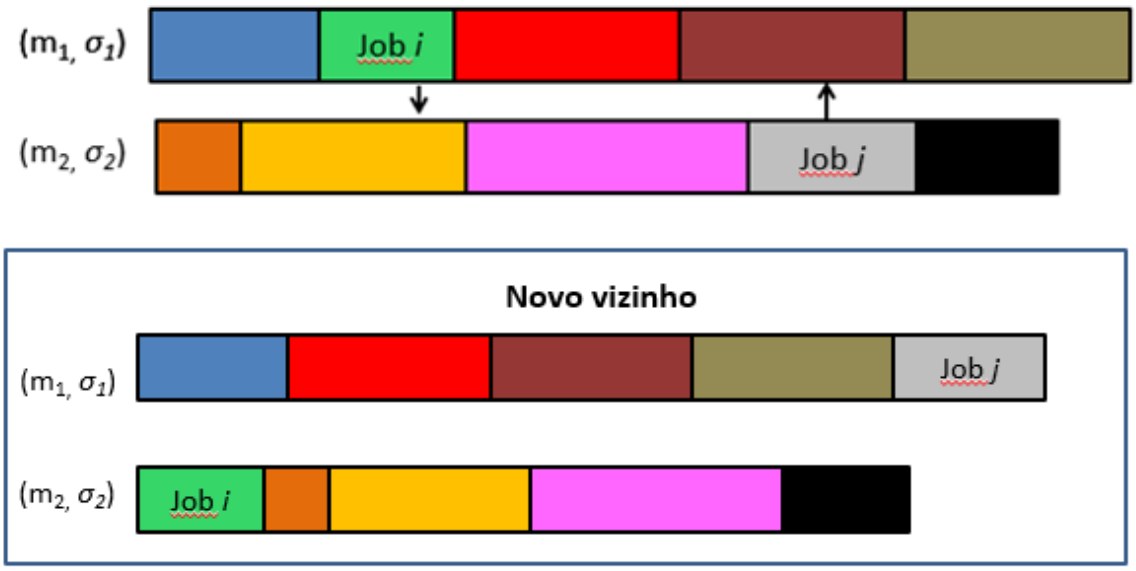

Figura 5: Troca Entre-Máquinas

iii) Troca Cíclica: Uma troca cíclica pode ser definido como custo de todos os arcos, sendo que um arco nada mais é que a realocação de uma única tarefa de uma máquina para outra até

fazer um ciclo. Como exemplo, seja o ciclo da Figura 7, que envolve as tarefas *i*, *j*, *k* e *l* pertencentes às sequências σ_1, σ_2, σ_3 e σ_4, respectivamente, isto é, $i \in \sigma_1$, $j \in \sigma_2$, $k \in \sigma_3$ e $l \in \sigma_4$. Nesse ciclo, extraia uma tarefa *i* de σ_1 e a insira em $\sigma_2 \setminus \{j\}$, depois insira a tarefa *j* em $\sigma_3 \setminus \{k\}$. Em seguida, insira a tarefa *k* em $\sigma_4 \setminus \{l\}$. Por fim, insira a tarefa *l* em $\sigma_1 \setminus \{i\}$.

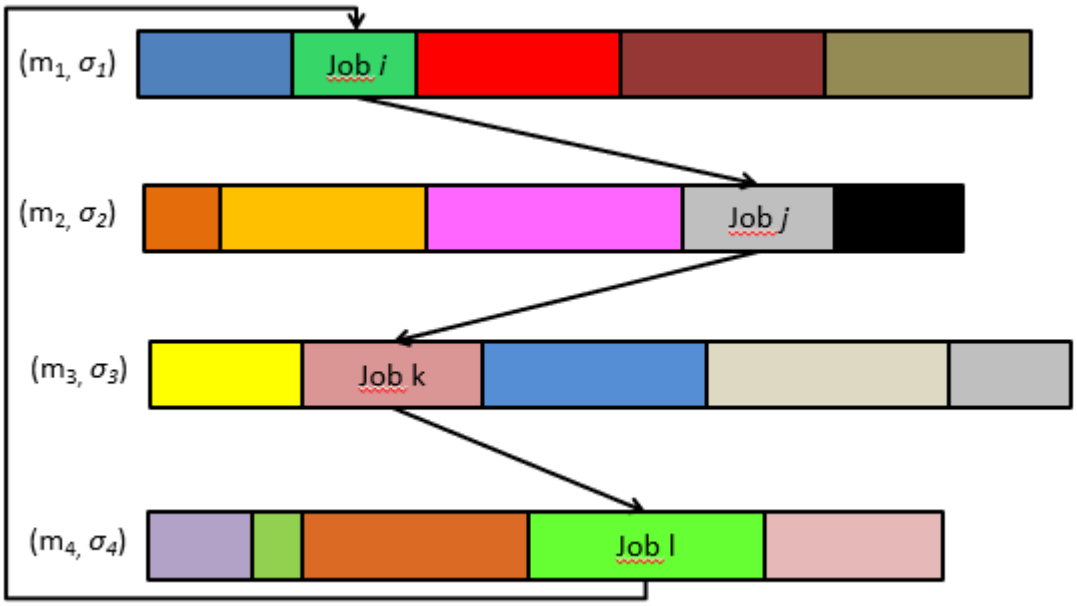

Figura 6: Troca Cíclica

Para realizar a troca cíclica, é necessário construir um grafo direcionado que represente a vizinhança de uma solução s. Tal grafo, denominado *grafo de melhoria G*(*s*), é definido para cada solução factível *s* do problema. O algoritmo é executado até que não haja mais nenhum ciclo válido ou ciclo negativo. Um ciclo negativo pode ser definido como custo de todos os arcos, sendo que a soma de todos esses arcos tem que ser um valor negativo. Na Figura 8 mostra um exemplo de ciclo negativo, onde cada círculo representa uma máquina e cada seta representa um arco.

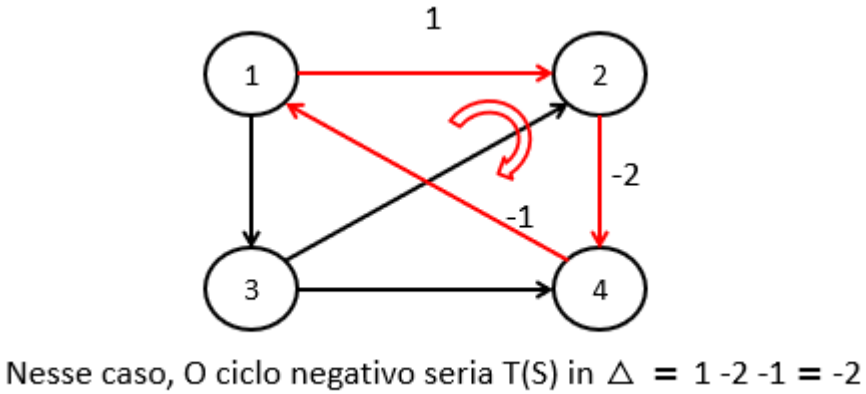

Figura 7: Exemplo de um ciclo negativo

3.3 Buscas Locais

Foram implementadas quatro heurísticas clássicas de busca local: *Best Improvement Search*, *First Improvement Search*, *Random Improvement Search* e *Very Large-scale Neighborhood Search*.

As heurísticas de busca local BIS, FIS e RIS foram aplicadas usando-se a vizinhança N_1, enquanto nas heurísticas BIS e VLNS utiliza-se a vizinhança N_2.

3.4 Perturbação de uma Solução

A perturbação de uma solução é a etapa da ILS que tem como objetivo escapar de um ótimo local e explorar outras regiões do espaço de soluções. Ela deve ser calibrada para não ser muito fraca, impossibilitando a saída do "vale" do ótimo local corrente, e nem muito forte que perca boas características da solução corrente. Na implementação desenvolvida, a perturbação consistiu em aplicar um número limitado de movimentos aleatórios do tipo troca intra-máquina e de movimento troca entre-máquinas.

3.5 Critério de Aceitação e de Parada

Estes procedimentos, determina qual deve ser a nova solução corrente para a próxima iteração. O critério de aceitação foi determinístico, ou seja, uma nova solução só será aceita se ela for melhor do que a melhor solução conhecida. Foi considerado como critério de parada o tempo de processamento limitado a 15 minutos.

3.6 Algoritmos ILS para o Problema Pm||∑WjTj

Os algoritmos ILS implementados dão origem a diferentes versões, cada qual relacionada com o tipo de busca local utilizado. O Algoritmo 5 mostra o pseudocódigo do ILS implementado. Os parâmetros N_1 e N_2 são as buscas locais intra-máquina e entre-máquinas. Na linha dois, é definida a solução inicial utilizando o algoritmo EDD. Já na descida1, descrita na linha 4, é uma busca local que usa a vizinhança N_1 e pode ser implementada nas estratégias BIS, FIS e RDS. Por outro lado, descida2, descrita na linha 6, é uma busca local que usa a vizinhança N_2 e pode ser implementada nas estratégias BIS e VLNS.

Na linha 19 ocorre a perturbação intra-máquinas aplicando a função *kick1*(), ou seja, são modificadas as posições das tarefas em máquinas escolhidas aleatoriamente.

Posteriormente, na linha 22, é realizada a perturbação via *kick2*(), que consiste em aplicar movimentos de troca de tarefas escolhidas de pares de máquinas tomadas aleatoriamente.

```
ILS(N1, N2, T(), EDD(), Tempo, MaxSemMelhora)
  1. s* ← s ← EDD();
  2. IterCont = 0;
  3. enquanto tempo de processamento < Tempo faça
  4. s1 ← descida1(s, N1);
  5. Repita
  6.       s2 ← descida2(s1, N2);
  7.       se(T(s2) < T(s1) então
  8.         s1 ← s2;
  9.  fim_se;
  10.    até que N2 não melhore s1
  11.    se T(s1) < T(s*) então
  12.       S* ← S1;
  13.       interCont = 0;
  14.    senão
  15.       interCont = interCont + 1;
  16.    fim_se;
  17.    se N2 falhar em melhorar s1 então
  18.       se iterCont > MaxSemMelhora então
  19.        s ← kick1(s*);
  20.        interCont = 0;
  21.       senão
  22.        s ← kick2(s2);
  23.       fim_se;
  24.    fim_se;
  25.    fim_enquanto;
  26.    retorna s*;
```

Algoritmo 5: Pseudocódigo da implementação do ILS.

4 RESULTADOS COMPUTACIONAIS

Os algoritmos desenvolvidos foram implementados na linguagem C e testados em um computador com processador i7, com 8 GB de memória RAM sob sistema operacional Windows 7.

Inicialmente foram realizados testes para determinar quais são os métodos mais eficientes de busca local, assim como as estruturas de vizinhança que produzem os melhores resultados. Posteriormente, foram realizados testes com as duas versões do ILS em diferentes conjuntos de instâncias *benchmark*. São destacados em negrito os melhores resultados obtidos.

4.1 Teste do ILS usando-se buscas locais intra-máquina

Inicialmente, decidiu-se analisar qual das estratégias de busca local intra-máquinas tem o melhor desempenho. Para tanto, foi aplicado o Algoritmo 1 sem a busca local descida2().

As Tabelas 1 e 2 mostram os resultados obtidos pelas buscas locais *BIS*, *FIS* e *RDS* nas vizinhanças troca (*swap*) intra-máquina e *twist*. O critério de parada da busca RIS foi executar 100 iterações sem melhora na função objetivo (FO). Nas Tabelas 1 e 2 são apresentados os valores da FO e o tempo de processamento (CPU em segundos) de cada combinação.

Tabela 1: Resultados dos métodos BIS, FIS e RDS nas estruturas de vizinhança troca intra-máquina em instâncias de 40 a 250 tarefas.

Instância	# Tarefas	*Swap*					
		BIS	Tempo (s)	FIS	Tempo (s)	RDS	Tempo (s)
1	40	**956**	0,028	**956**	0,051	1594	0,003
2	50	**2300**	0,036	**2300**	0,06	3079	0,002
3	100	**6077**	0,776	6426	1,167	8749	0,004
4	150	8203	2,691	**7814**	5,516	19632	0,008
5	200	17133	12,15	**17072**	41,417	31752	0,02
6	250	**26228**	26,617	26432	126,163	54579	0,018

Analisando-se a Tabela 1, verifica-se que o movimento troca intra-máquina teve melhor desempenho com as buscas *BIS* e *FIS*, considerando a quantidade de melhores soluções, uma vez que houve empate nos testes realizados. Entretanto, o tempo de CPU da busca *BIS* foi menor em todos os casos.

Um comportamento semelhante ao movimento troca intra-máquina foi verificado para o movimento *twist*, tendo em vista os resultados da Tabela 2. A combinação *Twist* com o método de busca local *BIS* produziu os melhores resultados em 5 dentre as 6 instâncias utilizadas para teste.

Tabela 2: Resultados dos métodos BIS, FIS e RIS nas estruturas de vizinhança *Twist*.

Instância	# Tarefas	*Twist*					
		BIS	Tempo (s)	FIS	Tempo (s)	RDS	Tempo (s)
1	40	1666	0,036	**996**	0,103	2713	0,004
2	50	**2345**	0,063	3074	0,119	2889	0,004
3	100	**7455**	0,973	8204	1,531	12050	0,004
4	150	**10706**	6,038	10952	18,233	28868	0,009
5	200	**25033**	25,711	27394	101,490	88393	0,020
6	250	**37440**	87,218	45807	327,200	139242	0,021

Para se ter uma visualização melhor dos resultados contidos na Tabela 2, foi gerado o Gráfico 1 com os dados dessa tabela. Analisando-o, pode-se verificar que a eficiência da busca BIS com na vizinhança *twist* fica melhor à medida em que a dimensão da instância cresce.

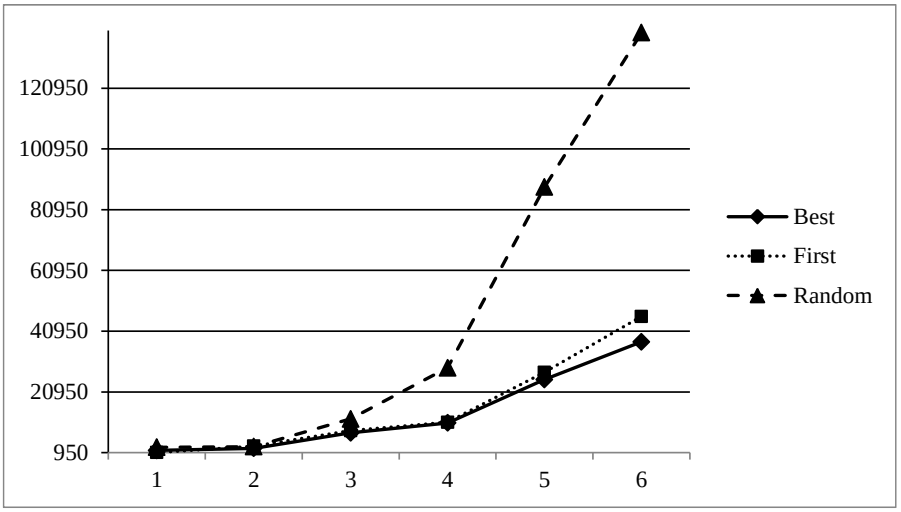

Gráfico 1: Valor da FO (eixo das ordenadas) nas 6 instâncias (eixo das abscissas) testadas, considerando a vizinhança *twist* com os métodos de busca local *Best*, *First* e *Random*.

Em vista desses resultados, a combinação *twist* com *Best Improvement* com o movimento intra-máquina foi adotada para a função descida1() do algoritmo ILS.

4.2 Teste do ILS usando-se buscas locais entre-máquinas, exceto VLNS

Nestes testes foi verificada a eficiência das buscas locais usando-se movimentos de realocação e troca de tarefas entre máquinas e os métodos de busca local BIS, FIS e o RDS. Para o movimento de trocas entre-máquinas foram utilizadas instâncias com 50 tarefas e o número de máquinas variando entre 2 a 6. A Tabela 3 mostra o valor da função objetivo obtida por cada variante do ILS nessas instâncias.

Tabela 3: Resultados dos métodos de busca local BIS, FIS e RDS nas estruturas de vizinhança troca e de realocação de tarefas.

# Máquinas	Vizinhança de Troca			Vizinhança de Realocação		
	BIS	*FIS*	*RIS*	*BIS*	*FIS*	*RIS*
2	**162**	178	178	175	175	187
3	**150**	**150**	**150**	162	162	194
4	**154**	**154**	165	171	171	175
5	**155**	**155**	**155**	161	161	161
6	**161**	**161**	**161**	179	179	179

Pelos resultados apresentados na Tabela 3, pode ser observado que o método de busca local BIS, com a vizinhança de troca de tarefas, foi a combinação mais eficiente e, portanto, foi a busca local adotada como descida2() no algoritmo ILS.

4.3 Comparação de resultados do ILS usando-se a busca local VLNS

O objetivo desta seção é comparar os resultados do algoritmo ILS com as melhores estratégias de busca local e movimentos até então, com o ILS no qual a busca local descida2() seja o VLNS. Esta versão será denominada ILS-VLNS, e logo em seguida, comparar os melhores resultados das buscas locais com os resultados presentes na literatura.

Baseado nos testes preliminares, o algoritmo ILS com o melhor desempenho até então é aquele no qual a função *descida1*( ), apresentada no Algoritmo 1, faz a busca intra-máquina utilizando o movimento *twist* associado à estratégia *BIS*, e a função *descida2*() utiliza o movimento de troca entre-máquinas associado à heurística *BIS*. Doravante, essa combinação será denominada ILS-Clássica.

A seguir, na Tabela 4, são apresentados os resultados de testes computacionais realizados com um conjunto de problemas para os quais são conhecidas as soluções ótimas que está disponível na literatura[1]. Nesse conjunto de problemas, temos n = 20 e m = 4. De acordo com Della Croce *et al.* (2012), as instâncias da Tabela 4 foram obtidas da adaptação de um conjunto de problemas com uma única máquina disponíveis na *OR-library*.

Na Tabela 4, as colunas R e T se referem à distribuição uniforme utilizada para gerar os respectivos problemas. A coluna %otm se refere ao percentual da diferença entre o valor da FO obtida e o valor ótimo. DP é o desvio padrão das 5 execuções e QT é o número de vezes que foi obtida a solução ótima. Infelizmente, uma comparação com os resultados de Della Croce *et al*. (2012) para as instâncias da Tabela 4 não foi possível, porque os resultados de cada instância obtidos pelos autores não estão disponíveis no link apontado no referido artigo.

[1] https://sites.google.com/site/negocindosica/pmttbench_n%3D20.rar

Tabela 4: Resultados obtidos pelos algoritmos ILS-Clássico e ILS-VLNS em instâncias cuja solução ótima é conhecida.

Instância			ILS – Clássico				ILS – VLNS			
#	R	T	%otm	Média	DP	QT	%otm	Média	DP	QT
1	0,2	0,2	0	144	0	5	0	144	0	5
2	0,2	0,4	0	48	0	5	0	48	0	5
3	0,2	0,6	0	0	0	5	0	0	0	5
4	0,2	0,8	0	0	0	5	0	0	0	5
5	0,2	1,0	0	0	0	5	0	0	0	5
6	0,4	0,2	0	487	0	5	0	487	0	5
7	0,4	0,4	0	386	0	5	0	386	0	5
8	0,4	0,6	0	301	0	5	0	301	0	5
9	0,4	0,8	0	300	0	5	0	300	0	5
10	0,4	1,0	0	324	0	5	0	324	0	5
11	0,6	0,2	0	1069	0	5	0	1069	0	5
12	0,6	0,4	**0,19**	1035	0	0	0	1033	0	5
13	0,6	0,6	0	1036	0	5	0	1036	0	5
10	0,6	0,8	0	1070	0	5	0	1070	0	5
15	0,6	1,0	0	887	0	5	0	887	0	5
16	0,8	0,2	0	1866	0	5	0	1866	0	5
17	0,8	0,4	0	1904	0	5	0	1904	0	5
18	0,8	0,6	0	1681	0	5	0	1681	0	5
19	0,8	0,8	0	1461	0	5	0	1461	0	5
20	0,8	1,0	0	1261	0	5	0	1261	0	5
21	1,0	0,2	**1,86**	2690	0	0	**0,53**	2655	0	0
22	1,0	0,4	**0,46**	2392	0	0	**0,46**	2392	0	0
23	1,0	0,6	0	2150	0	5	0	2150	0	5
24	1,0	0,8	0	1904	0	5	0	1904	0	5
25	1,0	1,0	0	1681	0	5	0	1681	0	5

Pode-se observar que ambas as versões do ILS conseguiram obter a melhor solução na maioria dos casos; porém a versão ILS-VLNS (a que contém o VLNS como busca local)

apresentou um desempenho ligeiramente superior, uma vez que ela obteve 23 valores ótimos contra 22 ótimos obtidos pela versão ILS-Clássico dentre as 25 instâncias. Quanto às duas instâncias nas quais ambas não obtiveram a solução ótima, a versão ILS-VLNS obteve um valor para a FO distante 0,53% em relação ao valor ótimo, enquanto o ILS-Clássico obteve uma função objetivo 1,86% distante do valor ótimo. Os resultados dos testes mostram que as versões ILS-Clássico e ILS-VLNS produziram resultados muito bons, pois obtiveram a solução ótima de 88% e 92% do valor ótimo das instâncias, respectivamente.

Nas Tabelas 5 e 6 são apresentados os resultados dos testes realizados com as instâncias na internet[2]. Tais tabelas contêm, em suas colunas, as seguintes informações: o número de identificação de cada problema, o valor mínimo, máximo e a média da FO de cinco execuções realizadas em cada instância e cada versão do ILS. A coluna DP se refere ao desvio padrão de cinco valores obtidos para a função objetivo.

Na Tabela 5 são apresentados os resultados dos testes realizados com um grupo de instâncias com $n = 50$ e $m = 4$, ou seja, que contam com 50 tarefas a serem distribuídas entre 4 máquinas.

Tabela 5: Resultados obtidos pelas duas versões da metaheurística ILS para instâncias com $n = 50$ e $m = 4$ e cuja solução ótima não é conhecida.

Grupo de Instância	ILS-Clássico				ILS-VLNS			
	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	3438	3451	3442,2	6,02	**3434**	3435	3434,4	0,55
20	21079	21132	21097,6	20,08	**20986**	20990	20987,6	1,82
30	61	64	62,4	1,52	**59**	61	60	1,00
40	8502	8553	8524,2	18,78	**8428**	8443	8434,8	5,89
50	56027	56145	56075	47,8	**55757**	55763	55759,2	2,28
60	2289	2296	2293,6	2,79	**2288**	2293	2289,6	1,95
70	23467	23681	23588,4	84,52	**22982**	23014	23002,8	12,52
80	**0**	0	0	0	**0**	0	0	0
90	6756	6884	6807,2	47,98	**6324**	6333	6329,2	3,7
100	34920	35066	34980,6	58,52	**34440**	34461	34451,2	9,58

[2] https://sites.google.com/site/negocindosica/instancias.rar

Na Tabela 6 são apresentados os resultados de instâncias com 50 tarefas alocadas em 10 máquinas.

Tabela 6: Resultados obtidos pelas duas versões da metaheurística ILS para problemas com n = 50 e m = 10 e cuja solução ótima não é conhecida.

Grupo de Instância	ILS-Clássico				ILS-VLNS			
	Mínimo	Máximo	Média	DP	Mínimo	Máximo	Média	DP
10	2160	2174	2166,8	5,02	**2115**	2123	2119,2	3,49
20	11452	11487	11463,4	14,24	**11364**	11373	11369	3,39
30	173	177	175,8	1,79	**165**	168	166,8	1,3
40	5775	5801	5787,4	9,24	**5630**	5643	5637,4	5,6
50	27923	28008	27958,8	33,34	**27568**	27570	27568,8	1,1
60	2012	2036	2026,2	8,98	**1918**	1961	1940,8	17,24
70	13360	13433	13408	28,63	**13164**	13173	13169	3,54
80	**0**	0	0	0	**0**	0	0	0
90	5602	5627	5615,4	9,63	**5381**	5402	5389,4	8,53
100	18087	18124	18105,6	13,41	**17806**	17812	17810,4	2,51

Analisando os resultados apresentados nas Tabelas 5 e 6, pode ser verificado que a versão ILS-VLNS foi superior à versão ILS-Clássica na maioria das instâncias. Apenas na instância 80, os resultados gerados por ambas as versões foram iguais. Os percentuais de melhoria do ILS-VLNS em relação ao ILS-Clássico, dado por P = 100×(FO(ILS-Clássico) - FO(ILS-VLNS))/FO(ILS-Clássico), variaram entre 0,12% e 6,39% para as instâncias da Tabela 5 e entre 0,77% e 4,67% para as da Tabela 6.

Nas Tabelas 7 e 8 são apresentadas as comparações de resultados dos algoritmos ILS-VLNS implementados neste trabalho com os resultados do algoritmo ILS de Della Croce[3] nos mesmos grupos de instâncias das tabelas 5 e 6. Tais tabelas contêm, em suas colunas, as seguintes informações: Instância, o valor de função objetivo do ILS de Della-Croce, o melhor valor da função objetivo do ILS-VLNS implementado neste trabalho (ILS-VLNS best), o valor médio da função objetivo do ILS-VLNS de cinco execuções realizadas para cada instância (ILS-VLNS best), o Desvio Relativo Percentual (RPD) do melhor valor do ILS-

[3] http://www.di.unito.it/~grosso/solution.tar.7z

VLNS (RPD best) e o RPD do valor médio (RDP avg) do ILS-VLNS, esses últimos calculados em relação aos valores obtidos pelo algoritmo ILS de Della Croce. O valor de RDP best para uma dada instância é calculado pela expressão: RPD_best = [melhor valor de FO do algoritmo ILS-VLNS – valor de FO do algoritmo ILS-Della Croce] / (valor de FO do algoritmo ILS-Della Croce) × 100. De forma análoga é calculo o valor RDP avg, mas neste caso, tomam-se os valores médios obtidos pelo algoritmo ILS-VLNS.

Tabela 7: Resultados obtidos pelas versões da metaheurística ILS de Della Groce e VLNS para problemas com $n = 50$ e $m = 4$ e cuja solução ótima não é conhecida.

Grupo de Instância	ILS-Della Croce	ILS-VLNS best	ILS-VLNS avg	RPD best	RPD avg
10	**3434**	3434	3434,4	0	0,01
20	**20977**	20986	20987,6	0,04	0,05
30	**59**	**59**	60	0	1,69
40	**8420**	8428	8434,8	0,1	0,18
50	**55755**	55757	55759.2	0	0,01
60	**2283**	2288	2289,6	0,22	0,29
70	**22952**	22982	23002.8	0,13	0,22
80	**0**	**0**	0	0	0
90	**6285**	6324	6329,2	0,62	0,7
100	**34427**	34440	34451,2	0,04	0,07

Tabela 8: Resultados obtidos pelas versões da metaheurística ILS de Della Groce e VLNS para problemas com $n = 50$ e $m = 10$ e cuja solução ótima não é conhecida.

Instância	ILS-Della Croce	ILS-VLNS best	ILS-VLNS avg	RPD best	RPD avg
10	**2109**	2115	2119,2	0,28	0,48
20	**11359**	11364	11369	0,04	0,09
30	**165**	**165**	166,8	0,00	1,09
40	**5623**	5630	5637,4	0,12	0,26
50	**27565**	27568	27568,8	0,01	0,01
60	**1869**	1918	1940,8	2,62	3,84
70	**13157**	13164	13169	0,05	0,09
80	**0**	**0**	0	0	0
90	**5375**	5381	5389,4	0,11	0,27
100	**17804**	17806	17810,4	0,01	0,04

Analisando os resultados apresentados nas Tabelas 7 e 8, pode ser verificado que a versão ILS-Della Croce foi superior à versão ILS-VLNS na maioria das instâncias. Apenas nas instâncias 30 e 80, os resultados gerados por ambas as versões foram iguais. O RPD best do ILS-VLNS em relação ao algoritmo ILS-Della Croce variou entre 0% e 0,62% na Tabela 7 e 0,01% e 2,62 na Tabela 8. Já o RPD médio do ILS-VLNS variou entre 0,01% e 1,69% nas instâncias da Tabela 7 e entre 0% e 3,84% nas instâncias da Tabela 8.

4.4 Análise dos Resultados

De acordo com os resultados apresentados na Tabela 4, pode-se verificar que a versão ILS-VLNS foi capaz de encontrar a solução ótima em 92% dos problemas contra 88% da versão ILS-Clássica. Ambas as versões foram eficientes na obtenção das soluções ótimas, mesmo com um tempo de processamento reduzido. O mesmo comportamento pode ser verificado nos grupos de problemas das Tabelas 5 e 6. Para esses dois grupos, o ILS-VLNS obteve soluções melhores do que o ILS-Clássico em todas as instâncias. O desvio padrão (DP) entre as soluções obtidas para cada problema é inferior no ILS-VLNS, o que mostra que esta versão é mais robusta do que a versão clássica da metaheurística. Os testes realizados sugerem que inferir que a versão ILS-VLNS é mais eficiente do que a versão ILS-Clássica na resolução do problema. No entanto, a versão se mostrou inferior aos resultados presentes na literatura, como é mostrado nas tabelas 7 e 8, mostrando que ainda é preciso fazer mais estudos para melhorar o desempenho desse algoritmo.

5 CONCLUSÕES

Este trabalho faz um estudo de buscas locais da metaheurística ILS para resolver o problema de sequenciamento de máquinas paralelas e uniformes considerando a aplicação de diferentes versões de busca local. São aplicados movimentos clássicos envolvendo uma única máquina e movimentos entre máquinas, assim como diferentes estratégias de busca local. Inicialmente, foram realizados testes para verificar qual a combinação de movimentos clássicos e estratégias de busca é a melhor. Essa combinação resultou no algoritmo denominado ILS-Clássico. Em seguida, testou-se outra combinação, denominada ILS-VLNS, na qual a busca local do ILS é feita pelo algoritmo VLNS, de busca em vizinhança de grande porte.

No primeiro conjunto de instâncias, em que se conhecia a solução ótima, foi verificada a capacidade das duas versões de atingir a otimalidade. Observou-se uma vantagem da versão ILS-VLNS, que obteve maior número de soluções ótimas assim como soluções mais próximas da ótima nos casos em que a solução ótima não foi encontrada. No segundo conjunto de instâncias, para as quais não se conhecia a solução ótima, também foi confirmada a superioridade da versão ILS-VLNS sobre a versão ILS-Clássica, com melhorias variando de 0,77% e 4,67% de diferença entre a função objetivo das duas versões, porém, comparando com os resultados da literatura, a versão ILS-VLNS foi levemente inferior.

Desta forma, este trabalho contribuiu testando uma combinação ainda não relatada na literatura.

6 REFERÊNCIAS

Ahuja, R. K., Ergun, Ö., Orlin, J. B. e Punnen, A. P. (2002). **A survey of very large-scale neighborhood search techniques.** Discrete Applied Mathematics, 123(1-3), 75–102.

Anghinolfi, D. e Paolucci, M. (2007). **Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach.** Computers & Operations Research, 34(11), 3471-3490.

Baker, K. R. (1984). **Sequencing rules and due-date assignments in a job shop.** Management Science, 30(9), 1093–1104.

Bilge, Ü., Kıraç, F., Kurtulan, M. e Pekgün, P. (2004). **A tabu search algorithm for parallel machine total tardiness problem.** Computers & Operations Research, 31(3), 397-414.

Cherkassky, B. V e Goldberg, A. V. (1999). Negative-cycle detection algorithms. Mathematical Programming, 85(2), 277–311.

Congram, R. K., Potts, C. N. e van de Velde, S. L. (2002). **An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem.** INFORMS Journal on Computing, 14(1), 52-67.

Della Croce, F., Garaix, T. e Grosso, A. (2012). **Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem.** Computers & Operations Research, 39(6), 1213–1217.

Du, J. e Leung, J. Y. T. (1990). **Minimizing total tardiness on one machine is NP-hard.** Mathematics of Operations Research, 15(3), 483-0,495.

Ergun, Ö. e Orlin, J. B. (2006). **Fast neighborhood search for the single machine total weighted tardiness problem.** Operations Research Letters,34(1), 41-45.

FRANCO, R. R.; SILVA, G. P. **A metaheuristic Iterated Local Search and Very Large-scale Neighborhood Search to solve the Parallel Machine Scheduling Problem.**

Proceedings of the XVIII Latin-Iberoamerican Conference on Operations Research, p. 327-334, 2016.

Grosso, A., Della Croce, F. e Tadei, R. (2004). **An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem.** Operations Research Letters, 32(1), 68-72.

Koulamas C. (1997) **Decomposition and hybrid simulated annealing heuristics for the parallel-machine total tardiness problem.** Naval Research Logistics, 44, 109–25.

Lenstra, J. K., Rinnooy Kan, A. H. G. e Brucker, P. (1977). **Complexity of machine scheduling problems.** Annals of Discrete Mathematics, 1, 343–362.

Lourenco, H. R., Martin, O. C. e Stutzle, T. (2001). **Iterated Local Search. In R. Glover e G. Kochenberger (Eds.)**, Handbook of Metaheuristics, 253–321. ISORMS.

Pan, Y. e Shi, L. (2007). **On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems.** Mathematical Programming, 110(3), 543-559.

Rodrigues, R., Pessoa, A., Uchoa, E. e Poggi de Aragão, M. (2008). **Heuristic algorithm for the parallel machine total weighted tardiness scheduling problem.** Relatórios de Pesquisa em Engenharia de Produção, 8(10), disponível em http://www.producao.uff.br/conteudo/rpep/volume82008/RelPesq_V8_2008_10.pdf. Acesso em 01/10/2019.