UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO E TESTE DE UMA HEURÍSTICA DE DESCIDA EM VIZINHANÇA VARIÁVEL PARA UM PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS

TRABALHO DE GRADUAÇÃO

Artur Ferreira Brum

Santa Maria, RS, Brasil
2013

IMPLEMENTAÇÃO E TESTE DE UMA HEURÍSTICA DE DESCIDA EM VIZINHANÇA VARIÁVEL PARA UM PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS

por

Artur Ferreira Brum

Trabalho de Graduação apresentado ao Curso de Bacharelado em Ciência da Computação, Área de Concentração em Otimização, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Ciência da Computação.**

Orientador: Prof. Dr. Benhur de Oliveira Stein

Coorientador: Prof. Dr. Olinto César Bassi de Araújo

Trabalho de Graduação N. 346

Santa Maria, RS, Brasil
2013

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

IMPLEMENTAÇÃO E TESTE DE UMA HEURÍSTICA DE DESCIDA EM VIZINHANÇA VARIÁVEL PARA UM PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS

elaborado por
Artur Ferreira Brum

Como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____
Prof. Dr. Olinto César Bassi de Araújo (UFSM)
(Presidente/Coorientador)

_____
Prof. Msc. Guilherme Dhein (UFSM)

_____
Prof. Dr. Vinícius Jacques Garcia (UFSM)

Santa Maria, 21 de fevereiro de 2013

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

IMPLEMENTAÇÃO E TESTE DE UMA HEURÍSTICA DE DESCIDA EM VIZINHANÇA VARIÁVEL PARA UM PROBLEMA DE PROGRAMAÇÃO DE TAREFAS EM MÁQUINAS PARALELAS

Autor: Artur Ferreira Brum
Orientador: Prof. Dr. Benhur de Oliveira Stein
Coorientador: Prof. Dr. Olinto César Bassi de Araújo
Local e data da defesa final: Santa Maria, 21 de fevereiro de 2012

Neste trabalho é apresentado um estudo sobre o problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da máquina e da sequência. Este problema é de interesse prático e ocorre, por exemplo, em fábricas com linhas de produção cujas máquinas são tecnologicamente diferentes. Para a resolução deste problema é implementada a abordagem heurística de descida em vizinhança variável proposta por Fleszar et al. (2011). Os testes computacionais consideram o conjunto de instâncias de Vallada e Ruiz (2011), além daquele utilizado no artigo original. Os resultados obtidos são comparados com a finalidade de verificar se o método é de fato robusto, eficaz e eficiente como reivindicado em Fleszar et al. (2011).

Palavras-chave: Otimização combinatória. Máquinas paralelas. Programação inteira mista.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

IMPLEMENTATION AND TESTING OF A VARIABLE NEIGHBORHOOD DESCENT HEURISTIC FOR A PARALLEL MACHINES TASK SCHEDULING PROBLEM

Author: Artur Ferreira Brum
Advisor: Prof. Dr. Benhur de Oliveira Stein
Co-advisor: Prof. Dr. Olinto César Bassi de Araújo
Place and date of presentation: Santa Maria, february 21st , 2012

In this work is presented a study on the unrelated parallel machine task scheduling with machine and sequence dependent setup times problem. This problem is of practical interest and occurs, for instance, in large facilities with production lines where machines are technologically different. The heuristic approach based in a variable neighborhood descent algorithm proposed by Fleszar et al. (2011) is implemented to solve this problem. Computational tests were performed considering the set of instances from Vallada and Ruiz (2011), besides the ones used in the original article. The results are compared in order to verify whether the method is in fact robust, efficient and effective as claimed by Fleszar et al. (2011).

Keywords: Combinatorial optimization. Parallel machines. Mixed-integer programming.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACO – Ant Colony Optimization

AP – Assignment Problem

COP – Contract or Patch

GA – Genetic Algorithm

GRASP – Greedy Randomized Adaptive Search Procedure

LNS – Large Neighborhood Search

MetaRaPS – Meta-heuristic for Randomized Priority Search

MVND – Multi-start Variable Neighborhood Descent

SNS – Short Neighborhood Search

TSP – Traveling Salesman Problem

VND – Variable Neighborhood Descent

SUMÁRIO

1 INTRODUÇÃO

O problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da sequência considera um conjunto de *n* tarefas que devem ser, uma a uma, processadas sem interrupção por exatamente uma única máquina dentre *m* disponíveis. É considerado que as máquinas são distintas, ou não relacionadas, e por isso o tempo de processamento de uma tarefa é definido para cada uma das máquinas na qual pode ser processada. Da mesma forma, dada a diferença entre as máquinas, os tempos de preparação (*setup time*) são dependentes da máquina e da sequência, isto é, o *setup time* computado na máquina *i* entre as tarefas *j* e *k* é diferente do *setup time* computado na mesma máquina entre as tarefas *k* e *j*. Ainda, o *setup time* entre as tarefas *j* e *k* na máquina *i* é diferente do *setup time* entre as tarefas *j* e *k* na máquina *i'*. O objetivo é determinar sequências para as máquina disponíveis que incluam todas as tarefas de tal modo que o tempo de processamento total da máquina com maior carga (*makespan*) seja o menor possível. Em Graham et al. (1979) é proposta uma classificação em notação de três campos para problemas de programação de tarefas, onde o primeiro campo descreve o ambiente das máquinas, o segundo as características das tarefas e o terceiro a função objetivo. O problema em estudo é denotado por $R|S_{ijk}|C_{max}$ em Pinedo (2008). No decorrer deste trabalho a denominação em três campos será utilizada para fazer referência ao problema em estudo.

Este problema é de especial interesse prático em um ambiente industrial, no qual é comum a existência de recursos paralelos distintos capazes de desempenhar uma mesma tarefa em tempos diferentes, de acordo com suas características. Idealmente todos os recursos deveriam ter a mesma capacidade, no entanto, essa situação dificilmente ocorre devido a dificuldade em sincronizar a depreciação e a aquisição de maquinário mais moderno. A consideração de intervalos de preparação advém da necessidade de executar operações como limpeza ou regulagem de uma máquina durante a troca de tarefas. Estas aplicações podem ser encontradas em fábricas de semicondutores, têxteis, químicos e plásticos (Rabadi et al. 2006).

Neste trabalho é implementado e testado o método de resolução para este problema proposto por Fleszar et al. (2011), que considera uma heurística de descida em vizinhança variável com elementos de programação inteira. Os resultados computacionais são realizados com dois conjuntos de instâncias propostos, separadamente, por Rabadi et al. (2006) e Vallada e Ruiz (2011). O primeiro conjunto permite validar a fidelidade da implementação e o segundo verificar a robustez do método de Fleszar et al (2011).

Este trabalho está organizado na forma que segue. No Capítulo 2 uma descrição formal matemática do problema é apresentada. O Capítulo 3 apresenta uma revisão bibliográfica de algoritmos de otimização que hibridizam heurísticas e métodos de programação matemática, assim como os trabalhos da literatura com foco no problema estudado. O Capítulo 4 detalha a implementação da heurística de descida em vizinhança variável com elementos de programação inteira de Fleszar et al. (2011). No capítulo 5 é apresentada uma alternativa implementada neste trabalho para a otimização de sequência de tarefas. No Capítulo 6 os resultados computacionais são apresentados e discutidos. As conclusões finais são apresentadas no Capítulo 7 e, por fim, as referências bibliográficas.

2 DESCRIÇÃO FORMAL DO PROBLEMA

A descrição textual permite um primeiro entendimento do problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da sequência, no entanto, dada a falta de rigor da linguagem natural, algumas características importantes podem não ser completamente entendidas. O modelo matemático apresentado a seguir não é a única forma de expressar rigorosamente o problema, mas permite o melhor entendimento de todas as restrições e considerações que são tomadas em conta para a resolução deste.

Para modelar adequadamente é utilizada uma tarefa artificial de índice 0 que corresponde a primeira tarefa designada a cada um das máquinas. Essa tarefa possui tempo de processamento nulo e o tempo de preparação das demais tarefas em relação a artificial é igual ao tempo de preparação para iniciar a sequência em uma máquina vazia.

Parâmetros:

n : número de tarefas;
m : número de máquinas;
M : $\{1,...,m\}$, representa o conjunto das máquinas;
N : $\{1,...,n\}$, representa o conjunto das tarefas;
N_0 : $N \cup \{0\}$ representa o conjunto das tarefas que inclui a tarefa artificial 0;

V : um número suficientemente grande;
p_{ik} : tempo de processamento da tarefa i, $i \in N$, na máquina k, $k \in M$.

s_{ijk} : tempo de preparação (*setup time*) da tarefa j, $j \in N$, para execução após o processamento da tarefa i, $i \in N_0$, na máquina k, $k \in M$.

Variáveis:

$$x_{ijk} = \begin{cases} 1 & \text{se a tarefa } i \text{ precede a tarefa } j \text{ na máquina } k \\ 0 & \text{caso contrário} \end{cases}$$

$C_i \geq 0$: tempo de final de processamento da tarefa i, $i \in N$.

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

Modelo $R|S_{ijk}|C_{max}$

$$\text{Min} \quad C_{max} \tag{1}$$

s.a.

$$\sum_{\substack{i \in N_0 \\ i \neq j}} \sum_{k \in M} x_{ijk} = 1 \qquad \forall\, j \in N \tag{2}$$

$$\sum_{\substack{j \in N \\ j \neq i}} \sum_{k \in M} x_{ijk} \leq 1 \qquad \forall\, i \in N \tag{3}$$

$$\sum_{j \in N} x_{0\,jk} \leq 1 \qquad \forall\, k \in M \tag{4}$$

$$\sum_{\substack{h \in N_0 \\ h \neq j,\, h \neq i}} x_{hjk} \geq x_{jik} \qquad \forall\, j, i \in N,\, j \neq i, \forall\, k \in M \tag{5}$$

$$C_j + V\left(1 - x_{ijk}\right) \geq C_i + s_{ijk} + p_{ik} \qquad \forall\, j \in N, \forall\, i \in N_0, i \neq j, \forall\, k \in M \tag{6}$$

$$C_{max} \geq C_i \qquad \forall\, i \in N \tag{7}$$

$$C_0 = 0 \tag{8}$$

$$C_i \geq 0 \qquad \forall\, i \in N \tag{9}$$

$$x_{ijk} \in \{0,1\} \qquad \forall\, i \in N_0, \forall\, j \in N, \forall\, k \in M \tag{10}$$

A função objetivo (1) considera a minimização do *makespan*, o maior tempo total de processamento dentre todas as máquinas. O conjunto de restrições (2) determinam que cada tarefa deve ter um e somente um predecessor, excetuando a tarefa artificial 0. De forma análoga, em (3) é determinado que cada tarefa pode ter no máximo um sucessor. A não obrigatoriedade de um sucessor é devido ao fato da sequência das máquinas não ser um ciclo como, por exemplo, uma rota de caixeiro viajante. O conjunto de restrições determina que a tarefa artificial, em cada máquina, pode ter no máximo um sucessor. As restrições (2) - (6), quando tomadas em conjunto, determinam sequências sem subciclos em todas as máquinas. Em (5) é imposto que o sucessor e predecessor de uma determinada tarefa devem estar na

mesma máquina, e em (6) subciclos são proibidos. Este último conjunto de restrições é comum em modelos de problemas de roteamento de veículos. Ainda, em (6) é determinado o tempo final de processamento de cada tarefa. O conjunto de restrições (7) determina que o menor valor para C_{max} corresponde a $\max_{i \in N}(C_i)$. A restrição (8) estabelece um tempo de processamento nulo para tarefa artificial 0. Em (9) e (10) o domínio das variáveis é definido.

3 REVISÃO BIBLIOGRÁFICA

A otimização combinatória tem como objetivo a resolução de problemas referentes à alocação de recursos, tipicamente limitados, com o intuito de alcançar determinados objetivos. Problemas deste tipo surgem em diferentes situações práticas e a resolução destes auxilia na tomada de decisões em diversos setores produtivos. São exemplos destes problemas planejamento e controle da produção, roteamento de veículos, escalonamento de enfermeiras, entre outros. A escolha da melhor decisão frente a recursos limitados pode representar uma grande economia e maior produtividade. Por exemplo, Anbil et al. (1991) descreve um problema de escalas de tripulações em uma empresa aérea no qual os custos de operação são da ordem de bilhões de dólares por ano. Neste problema, o custo de alocação de pessoal é um dos componentes mais caros, mesmo pequenas melhoras na eficiência podem levar a substanciais economias.

Problemas de otimização combinatória possuem uma interessante particularidade: usualmente são fáceis de descrever, mas difíceis de resolver (Osman e Kelly, 1996). O objetivo reside em encontrar uma solução que receba a melhor avaliação e, ao mesmo tempo, consiga satisfazer todas as restrições impostas. Isto é conhecido como solução ótima. Dado que geralmente o conjunto de soluções possíveis é discreto e finito, um método simples de resolução é o de examinar exaustivamente os seus elementos e encontrar a solução ótima ou concluir que esta não existe. Logo, qualquer problema, em princípio, pode ser resolvido, mas isto não significa que possa ser resolvido em tempo aceitável.

A teoria da complexidade computacional de Cook (1971) classifica problemas de otimização em *P* ou *NP*. A classe *P* consiste de todos os problemas de decisão para os quais existe um algoritmo de tempo polinomial. A classe *NP* representa os problemas de decisão que podem ser resolvidos em tempo polinomial por algoritmos não determinísticos. Isso significa se uma solução for escolhida de forma aleatória, é possível avaliar a qualidade em tempo polinomial. Também existem os problemas *NP-difíceis*, nos quais os problemas de decisão envolvidos podem ser polinomialmente redutíveis a um problema da classe *NP*. Acredita-se que seja muito improvável a existência de métodos eficientes para resolver estes problemas na otimalidade. Muitos problemas de otimização combinatória são *NP-difíceis*.

Existem numerosas abordagens para resolver problemas de otimização combinatória, duas correntes, oriundas de diferentes comunidades científicas, têm alcançado significativo sucesso:

1 Programação inteira como uma abordagem exata, oriunda da comunidade de pesquisa operacional e baseada em conceitos de programação linear.
2 Busca local com várias extensões e variantes desenvolvidas independentemente, denominadas meta-heurísticas, como uma abordagem heurística.

Entre os métodos exatos estão *branch-and-bound*, programação dinâmica, métodos baseados em relaxação lagrangeana, e métodos baseados em programação linear e inteira, tal como *branch-and-cut*, *branch-and-price,* e *branch-and-cut-and-price*. Algoritmos exatos dão garantia de encontrar e provar que uma solução é ótima para qualquer instâncias de problemas de otimização combinatória. Por outro lado, dada a complexidade computacional, o tempo de execução frequentemente aumenta exponencialmente com o tamanho da instância, e isso torna impraticável o uso destes algoritmos na resolução de instâncias de problemas reais, que usualmente são de média e grande dimensão.

Sob a designação de meta-heurísticas é possível incluir técnicas como *simulated annealing*, busca tabu, *GRASP* e vários modelos baseados em populações como algoritmos evolucionários e *scatter search*. Neste caso, a garantia de encontrar soluções ótimas é sacrificada em prol de se obter boas soluções em um tempo razoável. Essa característica é plenamente aceitável na prática, uma vez que um modelo é tão somente uma aproximação da realidade e a solução deste, mesmo sendo ótima, também o será. Deste modo, uma boa solução para o modelo, invariavelmente, também é uma boa solução para o problema em estudo. A desvantagem destes métodos recai na especialização das técnicas para cada problema em estudo, o que torna difícil a inclusão de novas características ou restrições. Uma visão geral destes métodos pode ser encontrada em Blum e Roli (2003).

Apenas recentemente, algoritmos híbridos que buscam unir métodos exatos e meta-heurísticos foram propostos na literatura. Resumidamente, alguns destes algoritmos objetivam obter soluções ótimas com tempos de execução menores, enquanto outros buscam obter melhores soluções heurísticas.

No trabalho de Puchinger e Raidl (2005) algumas combinações existentes são descritas e categorizadas em duas classes principais:

- **Combinação colaborativa**: quando os algoritmos trocam informações, mas não fazem parte um do outro. Algoritmos exatos e heurísticos podem ser executados sequencialmente, entremeados ou em paralelo.
- **Combinação integrativa**: quando uma técnica é um componente subordinado de outra técnica. Assim, existe um algoritmo mestre distinto, que pode ser um algoritmo exato ou meta-heurístico, e, ao menos, um escravo integrado.

Trabalhos relacionados com a primeira categoria, combinação colaborativa, podem ser encontrados em Clements et al. (1997), Applegate et al. (1998), Klau et al. (2004), Plateau et al. (2002), Vasquez e Hao (2001), Lin et al. (2004), Nagar et al. (1995), Tamura et al. (1994), entre outros. Para trabalhos referentes a combinações integrativas veja Burke et al. (2001), Thompson et al. (1993), Cotta e Troya (2003), French et al. (2001), Filho e Lorena (2000), entre outros.

O método estudado neste trabalho, proposto por Fleszar et al. (2011), pode ser incluído na segunda categoria, uma vez que um modelo matemático, cuja solução é encontrada por um resolvedor genérico, é subordinado a um método heurístico. Neste caso busca-se obter soluções de melhor qualidade em um tempo computacional viável. Obviamente, para que isto seja possível o tamanho dos modelos matemáticos não devem ter custo computacional elevado.

Fleszar et al. (2011) propõem uma heurística que supervisiona procedimentos de programação matemática. A heurística pode ser definida com um método de múltiplos inícios no qual, a cada iteração, duas vizinhanças são exploradas: vizinhança pequena e vizinhança grande. A vizinhança pequena é composta por movimentos denominados “simples” e a vizinhança grande por movimentos compostos por um ou mais movimentos simples. A exploração destas vizinhanças ocorre segundo os preceitos da descida em vizinhança variável (em inglês, VND – Variable Neighborhood Descent). O procedimento de programação matemática é utilizada na exploração da vizinhança grande, a partir de um modelo matemático que calcula os movimentos compostos e é resolvido pelo resolvedor CPLEX (pacote proprietário de otimização).

Vários trabalhos na literatura abordam o problema de programação de tarefas em máquinas não relacionadas, como França et al. (1996), Weng et al. (2001), Kim et al. (2002), Chen (2005), Low (2005), Chen (2006), Chen e Wu (2006), Rabadi et al. (2006), De Paula et

al. (2007), Logendran et al. (2007) e Armentano e De França (2007), para citar alguns. Neste trabalhos diferentes critérios de avaliação são considerados, algumas vezes a combinação de dois ou mais, como *makespan* e atraso de entrega. Os trabalhos que tratam especificamente o problema em questão são resumidamente descritos a seguir.

Rabadi et al. (2006) propõem uma meta-heurística baseada em busca randômica priorizada (em inglês, Meta-heuristic for Randomized Priority Search ou Meta-RaPS) para resolver o $R|S_{ijk}|C_{max}$. Esta meta-heurística é composta por uma estratégia que combina heurísticas de construção e refinamento, e é fortemente baseada nos preceitos do método *Greedy Randomized Adaptive Search Procedure* (GRASP). A heurística construtiva consiste em sequenciar tarefas que possuem o menor tempo de processamento ajustado na máquina com menor tempo de processamento total, que é definido pela soma do tempo de processamento somado ao tempo de preparação. Um componente aleatório é utilizado nessa heurística construtiva, semelhante à fase de construção do GRASP, para gerar soluções diversas. Na fase de refinamento, são aplicadas três buscas locais com movimentos de inserções entre máquinas, troca de pares de tarefas entre máquinas distintas e trocas aleatórias na mesma máquina. Os autores disponibilizam as instâncias utilizadas no endereço http://schedulingresearch.com.

Helal et al. (2006) implementam uma Busca Tabu para resolução do $R|S_{ijk}|C_{max}$. A solução inicial é construída a partir do menor tempo médio de processamento, calculado por meio da simulação de designações das tarefas em todas as máquinas. A Busca Tabu considera movimentos na mesma máquina, que visam melhorar o sequenciamento, e movimentos entre máquinas. Os resultados computacionais são realizados com as instâncias de Rabadi et al. (2006).

Um método baseado em colônia de formigas (em inglês, Ant Colony Optimization – ACO) é proposto por Arnaout et al. (2010). Este método foi projetado para considerar o caso em que o número de tarefas e o número de máquinas é grande. O algoritmo é executado em dois estágios, o primeiro trata da atribuição de tarefas a máquinas e o segundo do sequenciamento destas tarefas nas suas respectivas máquinas. São utilizados dois feromônios, um para favorecer a designação de uma tarefa a uma máquina e o outro para favorecer o sequenciamento das tarefas. Também são feitas buscas locais por meio de movimentos de inserção e troca. Testes computacionais são apresentados utilizando as instâncias de Rabadi et al. (2006).

Vallada e Ruiz (2011) propõem um Algoritmo Genético para a resolução do $R|S_{ijk}|C_{max}$. A formação da população inicial utiliza-se da múltipla inserção de Kurz e Askin (2001) para criar um indivíduo, enquanto os demais são gerados aleatoriamente. Ainda na definição da população inicial, uma busca local é aplicada a cada indivíduo. Nos cruzamentos também são aplicados procedimentos de buscas locais. Cada nova geração é construída com os melhores indivíduos da geração anterior. Os autores não consideram tempos de preparação para as primeiras tarefas alocadas nas máquinas. As instâncias dos testes computacionais são disponibilizadas no endereço http://soa.iti.es/problem-instances.

Stefanello et al. (2011) propõe uma heurística que hibridiza busca local com métodos de programação matemática. Movimentos de inserção e expulsão são denominados movimentos atômicos e um modelo matemático é utilizado para calcular uma ou mais sequências disjuntas de movimentos atômicos. Dois tipos de sequência de movimentos podem ocorrer: em ciclo ou em cadeia. No caso da sequência em ciclo vários movimentos atômicos de expulsão são realizados, sempre com uma tarefa assumindo o lugar da próxima e a última da sequência assumindo o lugar da primeira. A sequência em cadeia é semelhante, sendo que nenhuma tarefa assume o lugar da primeira e a última não expulsa nenhuma tarefa. Para realizar estes movimentos as tarefas predecessoras e sucessoras daquelas que compõem uma sequência são proibidas de participar de outros movimentos. Os resultados computacionais são apresentados sobre as instâncias de Vallada e Ruiz (2011).

4 MÉTODO DE DESCIDA EM VIZINHANÇA VARIÁVEL

A ideia principal subjacente a abordagem de Fleszar et al. (2011) para resolver o $R|S_{ijk}|C_{max}$ reside na decomposição deste em duas partes: o problema mestre, no qual é considerado a designação das tarefas às máquinas para minimizar o *makespan*, e subproblemas referentes ao sequenciamento das tarefas com o objetivo de minimizar o tempo de processamento de cada máquina. No que se refere aos métodos de resolução de cada parte da decomposição, um algoritmo de descida em vizinhança variável (em inglês, VND - Variable Neighborhood Descent) e uma heurística inicialmente proposta para o caixeiro viajante assimétrico são utilizados para a resolução do problema mestre e dos subproblemas, respectivamente.

A Figura 1 apresenta o pseudocódigo do Multi-start Variable Neighborhood Descent (MVND). Em cada execução, uma solução inicial aleatória é gerada e, no laço interno, as duas vizinhanças são exploradas. Uma busca na vizinhança menor é realizada até que melhoras na função objetivo não sejam possíveis. A vizinhança maior é utilizada no intuito de fazer um único movimento que retorne uma solução com custo de função objetivo menor, momento a partir do qual o fluxo de execução do algoritmo retorna para a vizinhança menor. Caso não seja possível encontrar uma melhor solução com a vizinhança maior uma nova iteração do MVND é realizada. Todo o método é repetido por 10 vezes (daí o nome – Multi-start VND) e ao final a melhor solução encontrada é retornada.

Inicializa X^* com uma solução vazia

Para todo *itera* = 1, 2, …, 10 **faça**
$X \leftarrow$ solução inicial

Repita
$X \leftarrow$ ótimo local de X utilizando SNS

$X \leftarrow$ tentativa de encontrar um movimento de melhora para X utilizando LNS

Até *LNS* falhar em melhorar a solução
Se X é melhor que X^* **então** $X^* \leftarrow X$ **Fim Se**

Fim para
Retornar X^*

Figura 4.1 – Pseudocódigo do algoritmo MVND (Fleszar et al., 2011)

A seguir cada procedimento do MVND é apresentado e detalhes de implementação são discutidos.

4.1 Solução inicial

A solução inicial é construída conforme o pseudocódigo apresentado na Figura 4.1.1. O processo é iniciado assegurando-se que nenhuma máquina tenha tarefas alocadas em si, e em seguida gerando-se uma lista que contenha todas as tarefas ordenadas aleatoriamente. Percorre-se então, para cada tarefa, todas as máquinas, verificando a melhor posição para inserção, ou seja, aquela que gera menor aumento no tempo de processamento total (*span)* da respectiva máquina ao inserir-se a tarefa. Ao encontrar a posição, encerra-se a iteração e repete-se o processo para a tarefa seguinte até que todas estejam alocadas a uma máquina. Finalmente, é executada a otimização da sequência das tarefas para cada máquina através do algoritmo SeqOpt.

Para todo $k \in M$ **faça**

$S_k \leftarrow$ sequência vazia

Fim Para

$L \leftarrow$ lista de tarefas em ordem aleatória

Para todo $j \in L$ **faça**

Para todo $k \in M$ **faça**

Encontre a melhor posição q_k para inserir *j* em S_k

Memorize q_k e o novo C_k

Fim para

$k^* \leftarrow arg\ min_k C_k$

Insira *j* na sequência S_{k^*} na posição q_{k^*}

Fim para

Para todo $k \in M$ **faça**

$S_k \leftarrow SeqOpt(S_k)$

Fim para

Retornar $X = (S_{1,} \ldots, S_k)$

Figura 4.1.1 – Pseudocódigo do algoritmo para solução inicial

4.2 Otimização da Sequência (SeqOpt)

A otimização da sequência é composta pelos passos descritos a seguir. Primeiro, uma dada sequência é melhorada utilizando um procedimento denominado SeqLS. A seguir, uma nova sequência é construída utilizando o algoritmo Contract or Patch (COP) proposto por Glover et al. (2001) e melhorada com SeqLS. Ao final, as duas sequências são comparadas e aquela com menor tempo de execução é escolhida como retorno do procedimento. O pseudocódigo do algoritmo pode ser visto na Figura 4.2.4.

O método SeqLS, utilizado para otimizar a sequência de tarefas em cada máquina, iterativamente, executa o melhor movimento de realocação de uma única tarefa até que não seja mais possível diminuir o tempo de processamento total da máquina em questão. Na Figura 4.2.1 é ilustrada em (b), da esquerda para a direita, a realocação de cada uma das tarefas de uma sequência inicial definida em (a). Note que as sequências hachuradas já foram avaliadas anteriormente e não é necessário portanto avaliá-las outra vez.

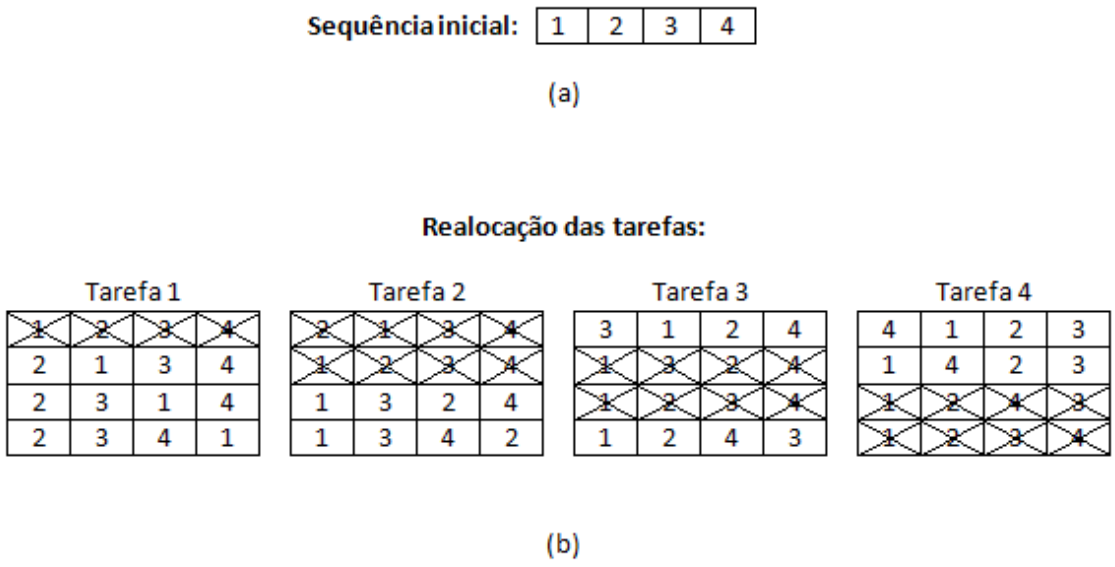

Figura 4.2.1 – Exemplo de realocação de tarefas pelo SeqLS

O algoritmo Contract or Patch (COP) foi proposto por Glover et al. (2001) como forma de solução para o problema do caixeiro viajante assimétrico (ATSP).

Em sua primeira etapa é usado o procedimento de Jonker e Volgenant (1987) para a resolução de um problema de designação, denotado por AP, que retorna uma estrutura associando um único predecessor e um único sucessor a cada um dos nós representados pelas tarefas. Se esta estrutura contiver apenas um ciclo, a sequência por ela representada é a solução ótima para o problema, caso contrário representa um limite inferior. Para o caso de mais ciclos, são executadas outras duas etapas para contração e/ou junção de ciclos.

Na implementação sugerida do COP utiliza-se um limiar *t* de valor igual a cinco, que definirá a contração de todos os ciclos menores ou igual a *t*. Na operação de contração o arco com maior peso é encontrado e removido, quebrando o ciclo e gerando um caminho. Este caminho então é contraído e passa a ser representado por um único nó. Os pesos dos arcos com origem ou destino a este nó, representado por *p*, devem então ser recalculados. A distância de um nó *x* até *p* é igual a distância de *x* até o primeiro nó do caminho que é representado por *p*, e a distância de *p* até *x* é igual a distância do último nó do caminho representado por *p* até *x*. Para o novo conjunto de nós e arcos é aplicado novamente o método AP. A execução alternada dos dois procedimentos citados é repetida até que se tenha apenas um ciclo ou nenhum ciclo menor ou igual ao limiar *t*. A Figura 4.2.2 ilustra passo a passo, para um valor limiar igual a três, a operação de contração. Em (a) tem-se um par de ciclos, tendo o da esquerda comprimento igual a cinco, e o da direita igual a três. Desta forma será contraído apenas o ciclo da direita. Supondo que o arco mais pesado seja o que vai do nó 6 até o nó 2, em (b) é feita a quebra do ciclo através da remoção do arco de maior peso. O caminho obtido é representado por um novo nó em (c).

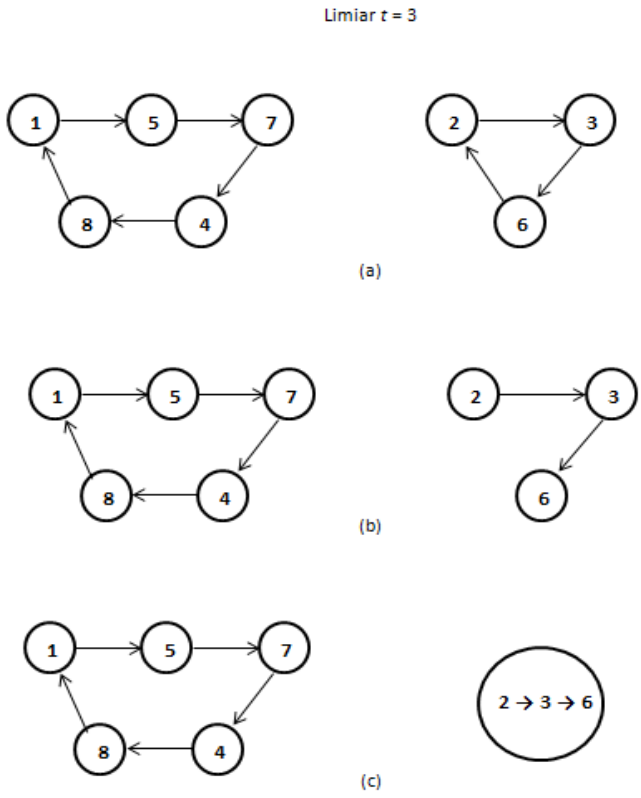

Figura 4.2.2 – Exemplo de contração

Na etapa seguinte é feita a junção de todos os ciclos, caso haja mais que um, com o método de Karp e Steele (1985) modificado por Glover et al. (2001) e denotado por GKS, que seleciona um par de arcos de diferentes ciclos cuja junção resulte no menor peso possível. O diferencial introduzido por Glover et al. (2001) no GKS é uma matriz de custos pré-calculada para as junções, que melhora o desempenho do procedimento em termos de tempo de processamento. A solução encontrada pelo GKS representa a melhor sequência para o conjunto de tarefas que se desejava otimizar. A Figura 4.2.3 mostra um exemplo de junção de dois ciclos apresentados em (a). O método GKS escolhe um par de arcos e faz a junção em (b). A Figura 4.2.5 apresenta ainda o pseudocódigo do COP.

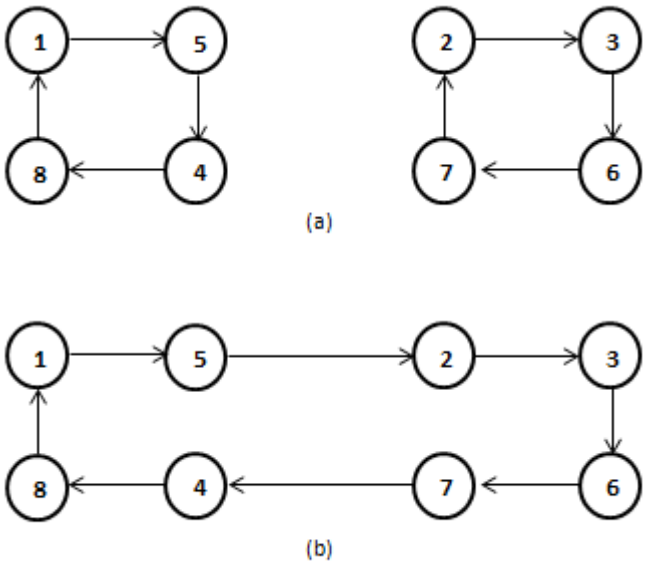

Figura 4.2.3 – Exemplo de junção

Entrada: Sequência de tarefas S_k

$S_k \leftarrow SeqLS(S_k)$

$S_{k^*} \leftarrow COP(S_k)$

$S_{k^*} \leftarrow SeqLS(S_{k^*})$

Se o $C_{max}(S_{k^*}) < C_{max}(S_k)$ **então** $S_k \leftarrow S_{k^*}$

Retornar S_k

Figura 4.2.4 – Pseudocódigo do SeqOpt

Entrada: Sequência de tarefas S_k

$t \leftarrow limiar$

$S_{k^*} \leftarrow AP(S_k)$

Enquanto há ciclos de tamanho até *t* em S_{k^*}

Contraia todos os ciclos menores ou iguais a *t* em S_{k^*}

$S_{k^*} \leftarrow AP(S_{k^*})$

Fim enquanto

Se há mais de um ciclo em S_{k^*} então faça a junção dos ciclos e armazene em S_k

Retornar S_k

Figura 4.2.5 – Pseudocódigo do COP

4.3 Vizinhança Pequena (SNS)

Os movimentos realizados durante a exploração das vizinhanças são compostos de três operações elementares:

Adição: uma tarefa é adicionada a uma máquina.

Remoção: uma tarefa é removida de uma máquina

Substituição: uma tarefa é removida e outra adicionada a uma máquina.

Na figura 4.3.1 são ilustrados os três movimentos elementares.

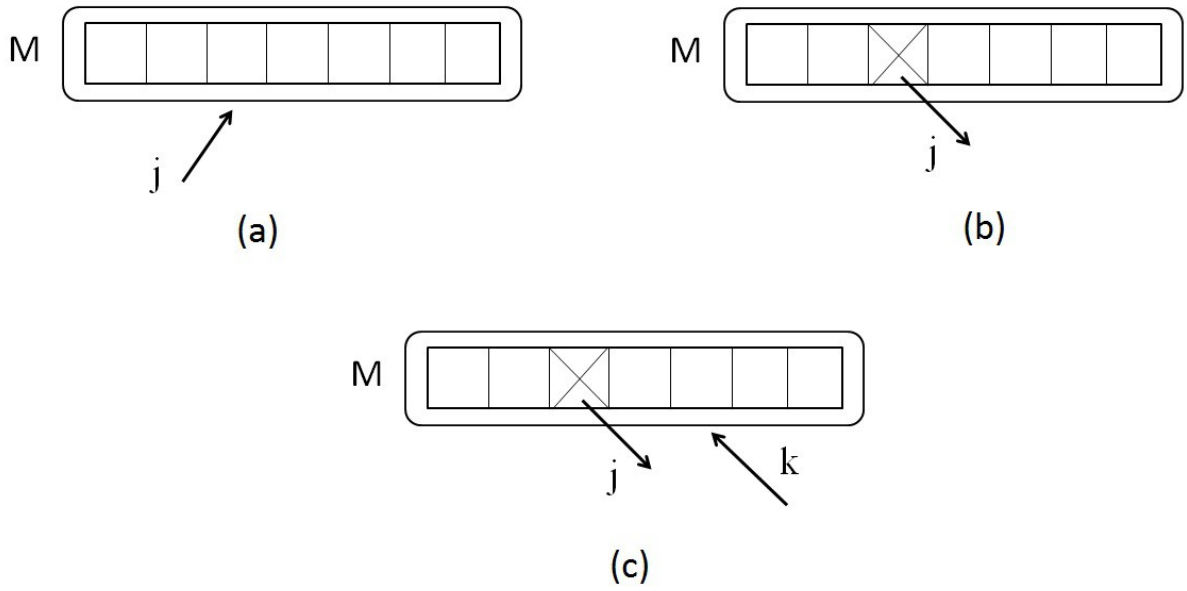

Figura 4.3.1 – Movimentos elementares. A adição está representada em (a), remoção em (b) e substituição em (c).

Dado que o custo de cada movimento é calculado com a otimização da sequência após as operações elementares a descrição destes movimentos pode ser feita considerando apenas as máquinas envolvidas no processo, conforme segue.

Transferência: uma tarefa é removida de uma máquina e inserida em outra diferente.

Troca: para duas tarefas *i* e *j* alocadas em diferentes máquinas, *i* é inserida da máquina de *j*, e *j* é inserida na máquina original de *i*.

Uma transferência é composta de uma operação de remoção, seguida de uma adição. Na troca são realizadas quatro operações elementares, uma de remoção e uma de adição para cada uma das tarefas envolvidas. Na figura 4.3.2 podem ser vistos os movimentos de transferência e troca.

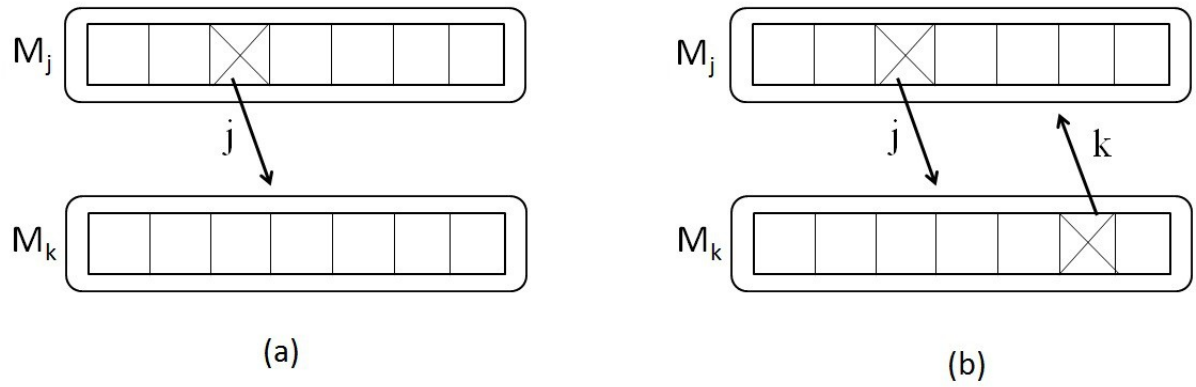

Figura 4.3.2 – Movimentos da vizinhança pequena. A transferência está representada em (a) e a troca em (b).

Para a busca em vizinhança pequena (SNS) são avaliados movimentos de transferência e troca, que são dispostos em uma lista ordenada aleatoriamente. A lista inclui inclusive movimentos inválidos, como troca de duas tarefas alocadas em uma mesma máquina. Movimentos que melhoram a solução são imediatamente adotados, e a busca continua até que não exista movimento capaz de minimizar a função objetivo. O algoritmo de Fleszar et al. (2011) implementa ainda um cache de movimentos com intuito de reduzir custo computacional. Como as operações elementares podem fazer parte de mais de um movimento tanto na vizinhança pequena quanto na grande, é possível evitar reavaliações desnecessárias armazenando a operação em uma estrutura.

4.4 Vizinhança Grande (LNS)

Para a vizinhança grande são considerados os seguintes movimentos:

Caminho: para uma sequência de tarefas $j_1, j_2, \ldots, j_n$ cada tarefa j_i é transferida para a máquina de j_{i+1} e a última tarefa j_n é transferida para uma máquina k diferente das máquinas de $j_1, j_2, \ldots, j_n$.

Cíclico: para uma sequência de tarefas $j_1, j_2, \ldots, j_n$ cada tarefa j_i é transferida para a máquina de j_{i+1} e a última tarefa j_n é transferida para a máquina da primeira tarefa j_1 .

Composto: um movimento composto combina movimentos cíclicos e de caminho, de maneira que cada máquina esteja envolvida em até um dele.

Nota-se também que os movimentos cíclicos e de caminho são compostos por trocas e transferências, logo todos os movimentos descritos podem ser decompostos nas operações elementares apresentadas inicialmente.

A busca em vizinhança grande avalia movimentos cíclicos e de caminho e através da resolução do modelo matemático descrito a seguir e determina um movimento do tipo composto. Os movimentos de caminho e cíclico são ilustrados na Figura 4.4.1 a seguir.

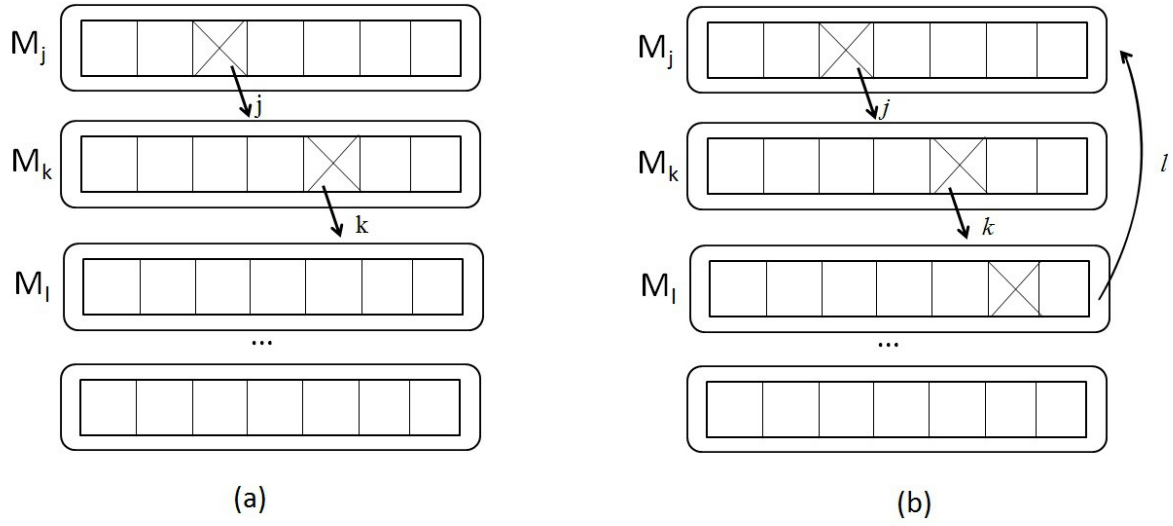

Figura 4.4.1 – Movimentos da vizinhança grande. O movimento de caminho está representado em (a) e o cíclico em (b).

4.4.1 Modelo matemático para a LNS

Parâmetros:

C_{max}^{0} : *makespan* da solução incumbente;

C_{k}^{0} : tempo total de processamento da máquina *k*, $k \in M$;

δ_{i0k} : acréscimo ao tempo de processamento da máquina *k* quando a tarefa *i* é adicionada, $k \in M, i \in N$;

δ_{0jk} : decréscimo ao tempo de processamento da máquina *k* quando a tarefa *j* é removida, $k \in M, j \in N$;

δ_{ijk} : diferença no tempo de processamento da máquina *k* quando a tarefa *i* é adicionada e a tarefa *j* é removida, $k \in M, i, j \in N$;

A_k : conjunto de todas as operações sobre a máquina *k* que não aumentam o tempo de processamento total;

ω : peso do *makespan* na função objetivo.

Variáveis:

$$y_{ijk} = \begin{cases} 1 & \text{se a tarefa } i \text{ é adicionada e a tarefa } j \text{ é removida da máquina } k \\ 0 & \text{caso contrário} \end{cases}$$

$$y_{i0k} = \begin{cases} 1 & \text{se a tarefa } i \text{ é adicionada a máquina } k \\ 0 & \text{caso contrário} \end{cases}$$

$$y_{0jk} = \begin{cases} 1 & \text{se a tarefa } j \text{ é removida da máquina } k \\ 0 & \text{caso contrário} \end{cases}$$

$C_k \geq 0$: tempo total de processamento da máquina *k*, $k \in M$.

$C_{max} \geq 0$: *makespan*.

Modelo LNS:

$$\text{Min} \quad \omega\left(C_{max}-C_{max}^{0}\right)+\frac{1}{m}\sum_{k\in M}\left(C_k-C_k^{0}\right) \tag{18}$$

s.a.

$$C_k \le C_{max} \qquad \forall\, k\in M \tag{19}$$

$$C_k = C_k^{0}+\sum_{(i,\,j)\in A_k}\delta_{ijk}\, y_{ijk} \qquad \forall\, k\in M \tag{20}$$

$$\sum_{(i\,,\,j)\in A_k} y_{ijk} \le 1 \qquad \forall\, k\in M \tag{21}$$

$$\sum_{k\in M}\ \sum_{j\,:\,(h\,,\,j)\in A_k} y_{hjk} = \sum_{k\in M}\ \sum_{i\,:\,(i\,,\,h)\in A_k} y_{ihk} \qquad \forall\, h\in N \tag{22}$$

$$y_{ijk}\in\{0,1\} \qquad \forall\, k\in M\,,\forall(i\,,\,j)\in A_k \tag{23}$$

$$C_k \ge 0 \qquad \forall\, k\in M \tag{24}$$

$$0 \le C_{max} \le C_{max}^{0} \tag{25}$$

A função objetivo (18) considera dois critérios de otimalidade: a minimização do *makespan* e a média dos tempos de processamento das máquinas (em inglês, *mean flow time*). Os coeficientes das variáveis determinam a prioridade dada a cada critério, especificamente nesse caso, $\omega \gg 1/m$, ou seja, a minimização do *makespan* é prioritária. Os parâmetros relativos aos tempos de processamento das máquinas e ao *makespan* são importantes para melhor entendimento do modelo, no entanto, na implementação, estes podem ser desprezados por serem constantes. Isso pode ser melhor observado se (18) for escrita da forma que segue.

$$\omega\, C_{max}+\frac{1}{m}\sum_{k\in M} C_k \quad -\omega\, C_{max}^{0}-\frac{1}{m}\sum_{k\in M} C_k^{0}$$

Não é difícil observar que as duas últimas parcelas são constantes.

Ainda sobre o modelo, as restrições (19) determinam o valor do *makespan*. As restrições (20) alteram o tempo de processamento de cada máquina de acordo com o movimento escolhido. Apenas um movimento pode ser aplicado em cada máquina, conforme definido pelas restrições (21). As restrições (22) garantem que se uma tarefa for removida de uma máquina então esta deve ser inserida em outra. As restrições (23) – (25) determinam o domínio das variáveis.

Uma observação pertinente sobre este modelo se refere ao índice *k* das variáveis *y,* o qual endereça as máquinas. Dado que o modelo calcula alterações em uma solução existente, ou seja, no qual é conhecida a designação inicial de cada tarefa a uma máquina, é possível facilmente interpretar um movimento y_{ij} . Para tanto, basta verificar quais eram as máquinas da designação inicial das tarefas *i* e *j*. Para o caso em que uma tarefa é inserida em outra máquina sem que qualquer outra tarefa desta seja removida, basta criar um índice para cada máquina, como tarefas artificiais. Ainda que essa observação não seja relevante no que se refere a formalização teórica do modelo, uma vez que o índice *k* não aumenta o número de variáveis nem a complexidade do modelo, no que tange a implementação computacional pode ser relevante no sentido de economizar o uso de memória.

5 MODELO PARA O TSP

Neste trabalho foi implementado também um modelo matemático para o problema do caixeiro viajante (Traveling Salesman Problem - TSP) como método alternativo para o SeqOpt, que serviu para fins de comparação com o procedimento COP. Nota-se que o problema estudado é análogo ao TSP assimétrico, uma vez que os custos de preparo são dependentes de sequência.

Como o seguinte modelo refere-se ao TSP, para fins de apresentação foram mantidos os termos cidade e rota, que representam tarefa e a soma de tempo de processamento com tempo de preparo, respectivamente.

Parâmetros:

n : número de cidades;

T : $\{1,...,n\}$, representa o conjunto de cidades;

V : um número suficientemente grande, neste caso, $V = |T|$;

d_{ij} : distância da cidade i para cidade j, $i, j \in T$.

Neste caso, a distância d_{ij} entre os nós i e j é calculada a partir da soma do tempo de processamento da tarefa j com o tempo de preparação quando executada imediatamente após a tarefa i.

Variáveis:

$$x_{ij} = \begin{cases} 1 & \text{se a cidade } i \text{ precede a cidade } j \\ 0 & \text{caso contrário} \end{cases}$$

$c_i \geq 0$: determina a posição da cidade i rota, $i \in N$, .

Modelo TSP:

$$\text{Min} \sum_{i \in T} \sum_{\substack{j \in T \\ j \neq i}} c_{ij} x_{ij} \qquad (11)$$

s.a.

$$\sum_{\substack{i \in T \\ i \neq j}} x_{ij} = 1 \qquad \forall\, j \in T \tag{12}$$

$$\sum_{\substack{j \in T \\ j \neq i}} x_{ij} = 1 \qquad \forall\, i \in T \tag{13}$$

$$c_j \geq 1 + c_i + (x_{ij} - 1)V \qquad \forall\, i \in T, \forall\, j \in T \setminus \{0\} \tag{14}$$

$$c_0 = 0 \tag{15}$$

$$c_i \geq 0 \qquad \forall\, i \in T \setminus \{0\} \tag{16}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, i, j \in T \tag{17}$$

A função objetivo (11) considera a minimização da distância total da rota. As restrições (12) e (13) determinam que cada cidade só pode ter um predecessor e um sucessor, respectivamente. As restrições (14) são evitam subciclos. Quando a cidade j_0 sucede a cidade i_0 temos $x_{i_0 j_0} = 1$, e a restrição assume a forma $c_{j_0} \geq 1 + c_{i_0} + (1-1)V \Rightarrow c_{j_0} \geq 1 + c_{i_0}$. Deste modo, c_{j_0} deve ser, no mínimo, uma unidade maior que c_{i_0}. Agora, se $x_{i_0 j_0} = 1$ e $x_{j_0 i_0} = 1$ então $c_{i_0} \geq 1 + c_{j_0}$ e, ao mesmo tempo, $c_{j_0} \geq 1 + c_{i_0}$, o que é impossível. Esse raciocínio pode ser facilmente estendido para demonstrar que subciclos maiores também são impossíveis. Dado que V é um número suficientemente grande, quando $x_{i_0 j_0} = 0$ a restrição assume a forma $c_{j_0} \geq 1 + c_{i_0} + (0-1)V \Rightarrow c_{j_0} \geq 1 + c_{i_0} - V$, que é trivialmente satisfeita. As restrições (15) – (17) determinam o domínio das variáveis.

6 RESULTADOS COMPUTACIONAIS

O MVND foi implementado na linguagem C++ no ambiente Microsoft Visual Studio 2010, com o resolvedor de programação inteira IBM CPLEX 12.4. O período previsto no projeto para a implementação do trabalho iria até o início do mês de janeiro de 2013, mas acabou estendendo-se até a primeira semana do mês seguinte devido às dificuldades encontradas. A memória auxiliar para as vizinhanças não foi implementada e o tempo computacional tornou-se um obstáculo para realizar os testes com instâncias de maior número de tarefas. Por esse motivo a plataforma de *hardware* não é relevante e os tempos computacionais não são reportados. No entanto, foi possível realizar os testes com um número representativo de instâncias, uma vez que o algoritmo possui um critério de parada bem determinado e relacionado com o número de iterações.

Dois conjuntos de instâncias teste foram utilizados. O primeiro, devido a Rabadi et al. (2006), consiste de três configurações distintas: tempo de processamento e tempo de preparação são semelhantes; tempo de processamento dominante; e tempo de preparação dominante. Para cada uma dessas configurações, seis grupos de número de máquinas foram considerados (2, 4, 6, 8, 10 e 12) e seis grupos de número de tarefas (20, 40, 60, 80, 100 e 120). Para cada uma das combinações de número de máquinas e número de tarefas 15 instâncias foram geradas, o que determina um conjunto com 1620 instâncias.

O segundo conjunto de instâncias teste foi proposto em Vallada e Ruiz (2011), e é dividido em dois grupos: 640 instâncias consideradas de pequeno porte, 1000 instâncias consideradas de grande porte. As instâncias de pequeno porte foram geradas a partir da combinação de $n = \{6, 8, 10, 12\}$ e $m = \{2, 3, 4, 5\}$, enquanto as instâncias de grande porte foram geradas a partir da combinação de $n = \{50, 100, 150, 200, 250\}$ e $m = \{10, 15, 20, 25, 30\}$. Neste trabalho serão consideradas as instâncias de grande porte.

A Tabela 1 apresenta uma comparação do MVND implementado neste trabalho com outras abordagens conforme reportado no trabalho de Fleszar et al. (2011). As primeiras colunas denotam as dimensões de cada instância (número de tarefas e número de máquinas). A relação das instâncias corresponde ao que foi possível testar dentro da restrição de tempo imposta pela falta da memória auxiliar das vizinhanças na implementação do MVND. Ainda

nesta tabela, as próximas seis colunas apresentam o número de soluções obtidas com o MVND com avaliação de melhor, igual ou pior em relação as abordagens consideradas na comparação (MetaRaPS (MR) - Rabadi et al. (2006) e ACO - Arnaout et al (2010)). As últimas três colunas mostram o desvio relativo de cada abordagem com relação ao limitante inferior relatado em Fleszar et al. (2011). O cálculo do desvio relativo é dado por $(C_{max} - LB)/LB * 100$, sendo C_{max} o valor do *makespan* e *LB* o valor do limitante inferior. Nesta tabela é possível observar que o MVND encontra melhores soluções em quase todos os casos. O pior desempenho ocorre quanto o número de tarefa é igual a 20, com o número de máquinas entre 2 e 8 e quando o número de tarefas é igual a 40 ou 60 com 2 máquinas. Considerando o número de tarefas, o MVND apresenta melhor desempenho quando a avaliação considera o desvio médio relativo.

A Tabela 2 é um extrato do artigo de Fleszar et al. (2011) considerando o MVND implementados pelos autores, neste trabalho identificado por $MVND_O$. É possível verificar, a partir das tabelas 1 e 2, que o MVND implementado neste trabalho apresenta um desempenho médio ligeiramente inferior ao $MVND_O$. É possível atribuir esta diferença à interpretação do método COP apresentado em Glover et al. (2001), especificamente no que se refere ao parâmetro limiar *t* deste método que não é definido em Fleszar et al. (2011). Dado a falta desta informação, foi utilizado o valor 5 proposto por Glover et al. (2001).

A Tabela 3 apresenta a diferença no número de soluções melhores, iguais e piores encontradas pelo método MVND quanto comparado com $MVND_O$. Nota-se que quanto maior o número de máquinas mais os resultados são fidedignos ao trabalho original. Uma vez que as instâncias de Vallada e Ruiz (2011) consideram um número mínimo de 10 máquinas, considerou-se a implementação do MVND apresentada neste trabalho como uma aproximação suficientemente boa do algoritmo original $MVND_O$ para fazer os testes computacionais relatados a seguir.

A comparação do MVND com o algoritmo genético de Vallada e Ruiz (2011) é exibida nas tabelas 4 e 5. Novamente, a relação das instâncias correspondem ao que foi possível testar dentro do tempo disponível. Nesta tabela a primeira coluna relaciona a dimensão das instâncias e a segunda coluna os diferentes intervalos para o tempo de preparação. A seguir são apresentados os números de soluções obtidas com o MVND com avaliação melhor, pior

ou igual ao algoritmo genético. A última coluna refere-se ao desvio relativo calculado por $(Cmax_{MVND} - Cmax_{AG}) / min(Cmax_{MVND} - Cmax_{AG}) * 100$, em que $Cmax_{MVND}$ corresponde ao *makespan* obtido com o MVND e $Cmax_{AG}$ o *makespan* obtido com o algoritmo genético. Observa-se que o desempenho médio do MVND é notadamente superior nas configurações com 100 tarefas. Quando o número de tarefas é igual a 50 o desempenho do MVND é melhor quando o número de máquinas é superior a 15.

Como uma forma de testar ao extremo a estratégia idealizada para o MVND, testes considerando o modelo matemático para o TSP assimétrico definido no Capítulo 5 foram realizados. Devido ao custo computacional proibitivo, somente as instâncias com configuração 50 tarefas, 10 e 30 máquinas e tempo de preparação dentro do intervalo 1-124 foram utilizadas.

Para as dez instâncias com n x m = 50 x 10 o MVND foi capaz melhorar, em média, 2,21% os resultados do algoritmo genético. Para as instâncias com n x m = 50 x 30 a melhora média foi de 6,55%. Este resultado pode ser considerado contraintuitivo, uma vez que o MVND possui uma forte conotação de otimização da sequência das tarefas em cada máquina, e os melhores resultados foram obtidos para instâncias com menor relação de tarefas por máquinas.

Outro teste realizado com o modelo TSP foi utilizar somente a vizinhança grande. Neste caso, para as dez instâncias com n x m = 50 x 10 o desempenho do MVND foi, em média, 1,88% pior que o algoritmo genético. Esse resultado pode ser atribuído a uma convergência prematura para um ótimo local de pior qualidade. Pode-se concluir que a vizinhança pequena permite pertubações menores que contribuem para uma convergência a um ótimo local de melhor qualidade.

Tabela 1 – Comparação do MVND com MetaRaPS (MR) Rabadi et al. (2006) e ACO Arnaout et al. (2010) – instâncias de Rabadi et al. (2006)

		MVND versus MR			**MVND versus ACO**			**Desvio médio relativo**		
n	***m***	Melhor	Igual	Pior	Melhor	Igual	Pior	MR	ACO	MVND
20	2	12	5	28	9	8	28	3.23	3.18	4.07
	4	6	19	20	35	4	6	6.45	7.33	6.00
	6	0	31	14	32	10	3	22.12	23.27	22.51
	8	9	24	12	40	5	0	25.72	27.37	25.37
	10	32	12	1	44	1	0	11.86	14.65	10.31
	Total	59	91	75	160	28	37			
	Média							13.88	15.16	13.65
40	2	36	2	7	2	0	43	2.52	1.82	2.34
	4	42	1	2	44	0	1	4.83	5.09	4.40
	6	43	2	0	44	0	1	8.82	10.15	9.18
	8	45	0	0	45	0	0	7.11	9.52	6.46
	10	45	0	0	45	0	0	10.30	11.10	8.40
	Total	211	5	9	180	0	45			
	Média							6.72	7.54	6.16
60	2	33	6	6	0	0	45	2.33	1.37	2.19
	4	44	0	1	43	2	0	4.49	3.88	3.70
	6	45	0	0	45	0	0	5.43	6.22	4.40
	8	45	0	0	45	0	0	9.75	11.41	9.33
	10	45	0	0	45	0	0	6.83	9.53	5.86
	Total	212	6	7	178	2	45			
	Média							5.77	6.47	5.10
Total		482	102	91	518	30	127			
Média								8.79	9.72	8.30

Tabela 2 – Extrato da Tabela 1 de Fleszar et al. (2011)

		$MVND_o$ versus MR			**$MVND_o$ versus ACO**			**Desvio médio relativo**		
n	***m***	Melhor	Igual	Pior	Melhor	Igual	Pior	MR	ACO	MVND
20	2	32	10	3	26	16	3	3.23	3.18	3.01
	4	23	21	1	41	4	0	6.45	7.33	6.24
	6	0	42	3	34	11	0	22.12	23.27	22.19
	8	13	27	5	40	5	0	25.72	27.37	25.41
	10	34	10	1	44	1	0	11.86	14.65	11.10
	Total	102	110	13	185	37	3			
	Média							13.88	15.16	13.59
40	2	45	0	0	41	1	3	2.52	1.82	1.58
	4	45	0	0	45	0	0	4.83	5.09	3.54
	6	44	1	0	44	0	1	8.82	10.15	7.49
	8	45	0	0	45	0	0	7.11	9.52	5.77
	10	45	0	0	45	0	0	10.30	11.10	6.60
	Total	224	1	0	220	1	4			
	Média							6.72	7.54	5.00
60	2	45	0	0	44	1	0	2.33	1.37	1.08
	4	45	0	0	45	0	0	4.49	3.88	2.50
	6	45	0	0	45	0	0	5.43	6.22	3.59
	8	45	0	0	45	0	0	9.75	11.41	8.43
	10	45	0	0	45	0	0	6.83	9.53	4.87
	Total	225	0	0	224	1	0			
	Média							5.77	6.47	4.10
Total		551	111	13	629	39	7			
Média								8.79	9.72	7.56

Tabela 3 – Variação na classificação de soluções obtidas em relação às de Fleszar et al. (2011)

		MVND versus MR			MVND versus ACO		
n	***m***	Melhor	Igual	Pior	Melhor	Igual	Pior
20	2	-20	-5	+25	-17	-8	+25
	4	-17	-2	+19	-6	0	+6
	6	0	-11	+11	-2	-1	+3
	8	-4	-3	+7	0	0	0
	10	-2	+2	0	0	0	0
	Total	-43	-19	+62	-25	-9	+34
40	2	-9	+2	+7	-39	-1	+40
	4	-3	+1	+2	-1	0	+1
	6	-1	+1	0	0	0	0
	8	0	0	0	0	0	0
	10	0	0	0	0	0	0
	Total	-13	+4	+9	-40	-1	+41
60	2	-12	+6	+6	-44	-1	+45
	4	-1	0	+1	-2	+2	0
	6	0	0	0	0	0	0
	8	0	0	0	0	0	0
	10	0	0	0	0	0	0
	Total	-13	+6	+7	-46	+1	+45
Total		-69	-9	+78	-111	-9	+120

Tabela 4 – Comparação do MVND com o algoritmo genético (GA) de Vallada e Ruiz (2011) para instâncias de 50 tarefas.

		MVND versus GA			**Desvio médio relativo**
n x m	***Setup***	Melhor	Igual	Pior	
50 x 10	1-124	1	0	9	8.02
	1-99	1	1	8	5.19
	1-49	1	2	7	2.33
	1-9	2	4	4	0.47
	Total	5	7	28	
	Média				4.00
50 x 15	1-124	2	2	6	3.80
	1-99	4	1	5	1.92
	1-49	5	1	4	-0.29
	1-9	4	3	3	-0.17
	Total	15	7	18	
	Média				1.31
50 x 20	1-124	3	3	4	0.74
	1-99	7	0	3	-1.33
	1-49	3	2	5	2.12
	1-9	3	4	3	0.94
	Total	16	9	15	
	Média				0.62
50 x 25	1-124	6	1	3	-3.43
	1-99	5	4	1	-2.96
	1-49	5	3	2	-2.77
	1-9	2	6	2	1.84
	Total	18	14	8	
	Média				-1.83
50 x 30	1-124	6	1	3	-2.83
	1-99	4	2	4	-0.91
	1-49	6	2	2	-2.45
	1-9	0	8	2	3.47
	Total	16	13	11	
	Média				-0.68
Total		70	50	80	
Média					0.68

Tabela 5 – Comparação do MVND com o algoritmo genético (GA) de Vallada e Ruiz (2011) para instâncias de 100 tarefas.

		MVND versus GA			**Desvio médio relativo**
n x m	***Setup***	Melhor	Igual	Pior	
100 x 10	1-124	1	1	8	-4.42
	1-49	3	0	7	1.23
	Total	4	1	15	
	Média				-1.59
100 x 20	1-124	6	0	4	-0.76
	1-99	2	4	4	2.13
	1-49	7	0	3	-2.85
	1-9	9	0	1	-5.84
	Total	24	4	12	
	Média				-1.83
100 x 30	1 -124	4	1	5	-0.93
	1-99	6	1	3	-2.02
	1-49	7	3	0	-4.76
	1-9	7	2	1	-3.90
	Total	24	7	9	
	Média				-2.90
Total		52	12	36	
Média					-2.11

7 CONCLUSÃO

Neste trabalho foi estudado e implementado um método de descida em vizinhança variável como forma de solução para um problema de programação de tarefas em máquinas não relacionadas considerando tempos de preparação dependentes da máquina e sequência. Foi possível obter uma boa aproximação dos resultados do artigo original e concluir que o método implementado é competitivo quando comparado com o algoritmo genético utilizado nos testes comparativos (Vallada e Ruiz, 2011). O uso do modelo matemático TSP dentro do método SeqOpt mostrou que a estratégia subjacente ao MVND é adequada para o problema em estudo e o uso de técnicas com baixo custo computacional e bons desempenho para o sequenciamento pode melhorar os resultados.

Uma etapa especialmente trabalhosa foi a implementação da otimização da sequência, especificamente do método COP, cujos autores detalharam sua utilização apenas de forma sucinta, o que gerou uma necessidade de aplicação de grande esforço para a compreensão e implementação. Algumas dúvidas persistiram, como o valor de limiar não especificado e a sua influência no algoritmo de maneira geral. O atraso na implementação desta etapa do trabalho ainda acabou afetando os testes com os conjuntos de instâncias, que demandaram um tempo considerável. Embora os resultados do trabalho não tenham coincidido perfeitamente com os dos autores do método original, foi possível estabelecer uma comparação para os conjuntos de dados pretendidos e conclusões podem ser formadas.

O desenvolvimento deste trabalho contribuiu de fato para o estudo dos objetivos específicos pretendidos, e do objetivo geral de um trabalho de graduação, que é exercitar o conhecimento adquirido ao longo do curso. A aquisição de novos conhecimentos sobre otimização combinatória, heurísticas, meta-heurísticas e resolvedores matemáticos genéricos foi considerável, visto que estes temas foram pouco abordados durante a graduação. Os objetivos específicos foram portanto alcançados. A implementação de maneira geral também contribuiu para o exercício e melhoria de técnicas de programação, uma vez que o código teve de ser o mais otimizado possível para minimizar o tempo de execução.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ANBIL, R.; GELMAN, E.; PATTY, B.; TANGA, R. (1991) **Recent Advances in Crew-Pairing Optimization at American Airlines**. Interfaces, v. 21, n.1, p. 62-74.

APPLEGATE, D.; BIXBY, R; CHVATAL, V.; COOK, W. (1998) **On the solution of traveling salesman problems.** Documenta Mathematica, Extra Volume ICM, vol. III: pp. 645-656.

ARMENTANO, V. A.; DE FRANÇA, M. F. (2007) **Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach**. European Journal of Operational Research, v. 183, n.1, p. 100-114.

ARNAOUT, J. P., RABADI, G., MUSA, R. (2010). **A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times**. Journal of IntelligentManufacturing, 21(6), 693–701.

BLUM, C.; ROLI, A. (2003) **Metaheuristics in combinatorial optimization: Overview and conceptual comparison**. ACM Computing Surveys, v. 35, n. 1, p. 268-308.

BURKE, E.; COWLING, P.; KEUTHEN, R. (2001) **Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem**. EvoWorkshops 2001. vol 2037 of LNCS, pp. 203-212.

CHEN, J. F. (2006) **Minimization of maximum tardiness on unrelated parallel machines with process restrictions and setups**. International Journal of Advanced Manufacturing Technology, v. 29, n. 5, p. 557-563.

CHEN, J. F.; WU, T. H. (2006) **Total tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints**. Omega-International Journal of Management Science, v. 34, n. 1, p. 81-89.

CLEMENTS, D.; CRAWFORD, J.; JOSLIN, D.; NEMHAUSER, G.; PUTTLITZ, M.; SAVELSBERGH, M. (1997) **Heuristic Optimization: A hybrid AI/OR approach**. Proceedings of the Workshop on Industrial Constraint-Directed Scheduling.

COOK, S. (1971). **The complexity of theorem proving procedures.** Proceedings of the Third Annual ACM Symposium on Theory of Computing, p. 151–158.

COTTA, C.; TROYA, J. (2003) **Embedding Branch and Bound within Evolutionary Algorithms**. Applied Intelligence, vol. 18: pp. 137-153.

DE PAULA, M. R.; RAVETTI, M. G.; MATEUS, G. R.; PARDALOS, P. M. (2007) **Solving parallel machines scheduling problems with sequence-dependent setup times using Variable Neighbourhood Search**. IMA Journal of Management Mathematics, v.18, n. 2, p. 101-115.

FILHO, G; LORENA, L. (2000) **Constructive Genetic Algorithm and Column Generation: an Application to Graph Coloring.** Proceedings of the APORS´2000 - The fifth conference of the association of Asian-pacific Operations Research Societies.

FLESZAR, K.; CHARALAMBOUS, C.; HINDI, K. (2011) **A variable neighborhood descent heuristic for the problem of makespan minimization on unrelated parallel machines with setup times**. J Intell Manuf 23:1949-1958.

FRANÇA, P.; GENDREAU, M.; LAPORTE, G.; MÜLLER, F. (1996) **A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times**. International Journal of Production Economics, v. 43, n. 2-3, p. 79-89.

FRENCH, A.; ROBINSON, A.; WILSON, J. (2001) **Using a Hybrid Genetic-Algorithm/Branch and Bound Approach to Solve Feasibility and Optimization Integer Programming Problems**. Journal of Heuristics, vol. 7: pp. 551-564.

GAREY, M.; JOHNSON, D. (1979) **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman, ISBN 0-7167-1045-5

GRAHAM, R. L.; LAWLER, E. L.; LENSTRA, J. K.; RINNOOY KAN, A. H. G. (1979) **Optimization and approximation in deterministic sequencing and scheduling: A survey.** Annals of Discrete Mathematics, v. 5, p. 287-326.

HELAL, M.; RABADI, G., AL-SALEM, A. (2006). **A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times**. International Journal of Operations Research, 3(3), 182–192

JONKER, R., VOLGENANT, A. (1987). **A shortest augmenting path algorithm for dense and sparse linear assignment problems**. Computing, 38, 325–340.

KIM, D.W.; KIM, K.H.; JANG, W.; CHEN, F.F. (2002) **Unrelated parallel machine scheduling with setup times using simulated annealing**. Robotics and Computer-Integrated Manufacturing, v. 18, n. 3-4, p. 223-231.

KLAU, G.; LJUBIC, I.; MOSER, A.; MUTZEL, P., NEUMER, P.; PFERSCHY, U.; RAIDL, G.; WEISKIRCHER, R. (2004) **Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem**. GECCO 2004. vol. 3102, pp. 1304-1315.

KURZ, M.; ASKIN, R. **Heuristic scheduling of parallel machines with sequence-dependent set-up times**. International Journal of Production Research 39, 3747– 3769.

LIN, Z.; BEAN, C.; WHITE-III, C. (2004) **A Hybrid Genetic Optimization Algorithm for Finite-Horizon, Partially Observed Markov Decision Processes**. INFORMS Journal on Computing, vol. 16(1): pp. 27-38.

LOGENDRAN, R.; MCDONELL, B.; SMUCKER, B. (2007) **Scheduling unrelated parallel machines with sequence-dependent setups**. Computers & Operations Research, v. 34, n. 11, p. 3420-3438.

NAGAR, A.; HERAGU, S.; HADDOCK, J. (1995) **A meta-heuristic algorithm for a bi-criteria scheduling problem.** Annals of Operations Research, vol. 63: pp. 397-414.

OSMAN, I.; KELLY, J. (1996) **Meta-Heuristics: Theory and Applications.** Journal of Global Optimization, v. 15, p. 105-107

PINEDO, M. (2008) **Scheduling: Theory, Algorithms, and Systems**. 3th ed, New Jersey: Prentice Hall, 678 p.

PLATEAU, A.; TACHAT, D.; TOLLA (2002) **A hybrid search combining interior point method and metaheuristics for 0-1 programming.** International Transactions in Operational Research, vol. 9: pp. 731-746.

PUCHINGER, J.; RAIDL, G.R. (2005) **Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification**. International Work-Conference on the Interplay Between Natural and Artificial Computation - IWINAC, Part II. LNCS 3562, p. 41–53.

RABADI, G.; MORAGA, R.; SALEM, A. (2006) **Heuristics for the unrelated parallel machine scheduling problem with setup times**. Journal of Intelligent Manufacturing, v. 17, n. 1, p. 85-97.

STEFANELLO, F.; ARAÚJO, O.; MÜLLER, F.; GARCIA, V. (2011) **Uma vizinhança de grande porte para o problema de programação de tarefas em máquinas paralelas não relacionadas com tempo de preparação dependente da sequência**. XLII Simpósio Brasileiro de Pesquisa Operacional, 1-12

TAMURA, H.; HIRAHARA, A.; HATONO, I.; UMANO, M. (1994) **An approximate solution method for combinatorial optimisation**. Transactions of the Society of Instrumentand Control Engineers, vol. 130: pp. 329-336.

THOMPSON, P; PSARAFTIS, H. (1993) **Cycle transfer algorithm for multivehicle routing and scheduling problems**. Operations Research, vol. 41: pp. 935-946.

VALLADA, E.; RUIZ, R. (2011) **Genetic algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times**. European Journal of Operational Research, v. 211, p. 612-622.

VASQUEZ, M.; HAO, J. (2001) **A Hybrid Approach for the 0-1 Multidimensional Knapsack problem**. Proceedings of the International Joint Conference on Artificial Intelligence 2001, pp. 328-333.

WENG, M. X.; LU, J.; REN, H. (2001) **Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective**. International Journal of Production Economics, v. 70, n. 3, p. 215-226.