André L. B. Costa

Novos Algoritmos para linhas de montagem flowshop com tempo de setup

Rio Tinto, Paraíba, Brasil

26 de Maio de 2017

André L. B. Costa

Novos Algoritmos para linhas de montagem flowshop com tempo de setup

Trabalho monográfico acadêmico que tem como objetivo apresentar novos algoritmos como soluções para problemas flowshop com tempo de setup.

Universidade Federal da Paraíba – UFPB

Centro de Ciências Aplicadas e Educação – CCAE – Campus IV

Departamento de Ciências Exatas - DCX

Orientador: Wagner E. Costa

Rio Tinto, Paraíba, Brasil

26 de Maio de 2017

C837n Costa, André L. B..
Novos algoritmos para linhas de montagem flowshop. / André L. B. Costa. – Rio Tinto: [s.n.], 2017.
54 f. : il.-

Orientador(a): Prof. Dr. Wagner E. Costa.
Monografia (Graduação) – UFPB/CCAE.

1. Sistemas de informação. 2. Programação de produção. 3. Flowshop - programação - informática.

UFPB/BS-CCAE CDU: 004(043.2)

.

André L. B. Costa

Novos Algoritmos para linhas de montagem flowshop com tempo de setup

Trabalho monográfico acadêmico que tem como objetivo apresentar novos algoritmos como soluções para problemas flowshop com tempo de setup.

Trabalho aprovado. Rio Tinto, Paraíba, Brasil, 07 de abril de 2017:

D.Sc.Wagner E. Costa
Orientador

D.Sc.Marcus Carvalho
Professor

D.Sc.Joelson Carvalho
Professor

Rio Tinto, Paraíba, Brasil
26 de Maio de 2017

Resumo

O ambiente de produção flowshop com tempo de setup, que tem o critério de minimizar o makespan, vem ganhando atenção de pesquisadores da área de produção e otimização combinatória. Com o aumento de sua pesquisa, o desempenho de suas abordagens algorítmicas tem sido rapidamente elevados. Neste trabalho apresenta-se novos métodos para o problema de programação no ambiente de produção flowshop com tempos de setup das máquinas, separados dos tempos de processamento das tarefas. Os novos métodos são heurísticos, dos tipos *Buscas por Vizinhança Variável* (VNS) e sua variação *Descida Variável pela Vizinhança* (VND)
Palavras-chave: Flowshop. Programação de Produção. Busca por Vizinhança Variável. Tempos de *Setup*.

Abstract

The production environment flowshop with setup time, which has the criterion of minimizing the makespan, has been gaining attention from researchers in the area of production and combinatorial optimization. With the increase of their research, the performance of their algorithmic approaches has been rapidly raised. In this work we present new methods for the scheduling problem in the flowshop production environment with machine setup times, separated from task processing times. The new methods are heuristic, of type *Variable Neighborhood Searches* (VNS) and their variation *Variable Neighborhood Descent* (VND)

Keywords: Flowshop. Scheduling problem. Variable Neighborhood Searches.

Sumário

Introdução

O problema de programação no ambiente de produção flowshop se compromete na execução de n tarefas que devem ser processadas, na mesma sequência, em m máquinas distintas - Gupta (1979). Toda tarefa requer um tempo conhecido, determinado e não-negativo para sua execução em uma determinada máquina, que não pode ser interrompido. Em flowshop, as tarefas devem ser processadas em uma determinada sequência nas máquinas, mantendo a mesma sequência para todas elas, desde a primeira até a última.

O objetivo da resolução do problema é determinar dentre as n fatoriais $(n!)$ sequências possíveis de tarefas, aquela que otimize algum critério de desempenho da programação. Os critérios mais usuais são a minimização do tempo total de execução das tarefas, que é referida como makespan (Cmax), ou a minimização do tempo médio de fluxo. O problema no ambiente de produção flowshop tem sido uma área de pesquisa altamente estudada pelos mais diversos cientistas da área, desde sua primeira pesquisa e resultado publicado por Johnson (1954) pelo problema com somente duas máquinas. Desde a publicação de Johnson, o problema de programação flowshop teve seu estudo ampliado e revisado por Campbell, Dudek e Smith (1970), Dannenbring (1977), Garey, Johnson e Sethi (1976), Gupta (1979), Osman e Potts (1989), Taillard (1993), Hejazi e Saghafian (2005) e vários outros pesquisadores de renome.

Gupta (1979), descreveu 21 restrições sobre o ambiente *flowshop* do problema de programação tradicional no ambiente flowshop, originalmente concebido por Johnson (1954). Segue abaixo o detalhamento das restrições

- Restrições relativas às tarefas

    1. Cada tarefa fica disponível à fabrica ao início do período de programação.
    2. Cada tarefa tem sua data de validade que é fixa, e não está sujeita a mudança.
    3. Cada tarefa é independente uma da outra.
    4. Cada tarefa consiste em operações específicas, onde cada uma das quais é executada por apenas uma máquina.
    5. Cada tarefa tem uma ordem prescrita que é a mesma para todas as outras tarefas, que é fixa.
    6. Cada tarefa possui um tempo conhecido e finito de processamento para ser processado pelas máquinas. Esse tempo de processamento inclui os tempos de transporte e de preparação, se houver, e é independente das tarefas anteriores e posteriores.

7. Cada tarefa é processada não mais que uma vez em qualquer máquina.
8. Cada tarefa tem que esperar entre uma máquina e outra, elas são armazenadas em uma área enquanto aguardam.

- Restrições relativas às máquinas

1. Cada estação central consiste em apenas uma máquina. Assim, a fábrica contém apenas uma máquina para cada tipo.
2. Cada máquina está inativa até o início do período de programação.
3. Cada máquina opera independentemente das outras máquinas, portanto é capaz de operar com sua própria taxa máxima de saída.
4. Cada máquina só pode processar uma tarefa de cada vez. Isso elimina as máquinas que são projetadas para processar várias tarefas simultaneamente.
5. Cada máquina está disponível continuamente para processar tarefas durante todo o período de programação, sem interrupções por avarias, manutenção ou qualquer outra causa.

- Restrições relativas às políticas operacionais

1. Cada tarefa é processada o mais cedo possível. Assim, não há nenhuma tarefa em espera ou máquina em tempo ocioso intencionalmente.
2. Cada tarefa é considerada uma entidade indivisível mesmo que possa ser composta por um número de unidades individuais.
3. Cada tarefa, uma vez aceita, é processada até sua conclusão, ou seja, não é permitido o cancelamento das tarefas.
4. Cada tarefa, uma vez iniciada na máquina, é mantida até sua conclusão antes que outra tarefa possa ser iniciada na máquina.
5. Cada tarefa é processada não mais do que uma máquina por vez.
6. Cada máquina tem um espaço de espera adequado para permitir que as tarefas sejam aguardadas antes de iniciar o processo.
7. Cada máquina é totalmente alocada para as tarefas durante todo o tempo de programação, ou seja, as máquinas não são utilizadas para qualquer outra finalidade ao longo do período de programação.
8. Cada máquina processa as tarefas na mesma sequência, ou seja, não é permitida nenhuma passagem ou ultrapassagem de tarefas.

Uma variação interessante do problema de programação flowshop é quando as tarefas tem um tempo de preparação(setup) em cada máquina distinta, onde esse

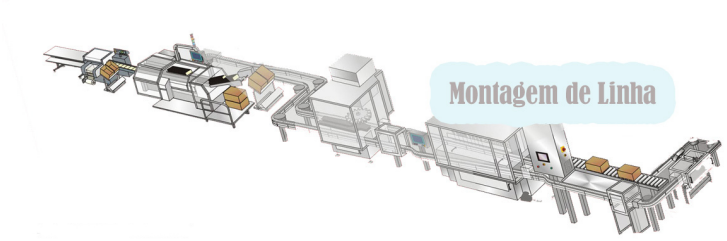

Figura 1 – Exemplo de linha de produção (Fonte: http://www.remakaracatuba.com.br/)

tempo de preparação da próxima tarefa que sera processada depende da tarefa que está sendo executada no momento na máquina. Assim, esse tempo que é determinado e não-negativo, denotado por S_{ijk}, requer a preparação da máquina i para a tarefa j que será executada imediatamente após a saída da tarefa k. De acordo com Dong, Huang e Chen (2008) os problemas de ambiente de produção flowshop com o critério de minimização do makespan é fortemente NP-Completo, sendo com mais de duas máquinas, e com isso é improvável que exista algum algoritmo polinomial para resolução geral do problema.

Como exemplo, para resumir o problema em ambiente flowshop com o tempo de setup, é possível citar a fabricação de placas mãe, onde as mesmas seguem uma linha de montagem rígida, passando por várias máquinas na mesma ordem, sendo que as máquinas precisam ser preparadas para cada modelo de placa. A fábrica contém várias máquinas de soldagem e colocação de peças que precisam ser calibradas e recarregadas com peças de acordo com o modelo da placa mãe fabricada, exigindo que as placas mantenham a ordem de acordo com o modelo. As placas mãe que serão fabricadas representam as tarefas, são postas em produção e cada máquina tem que ser configurada para aquele modelo de placa, o que caracteriza o tempo de setup (tempo de preparação). Cada placa tem um tempo de soldagem ou colocação de peças em cada máquina, que caracteriza como tempo de processamento. E apenas uma placa pode passar pela máquina, respeitando a ordem para não haver problemas.

Este trabalho tem como objetivo propor novos algoritmos heurísticos para o problema de minimizar o makespan em flowshop com tempo de setup. A heurística é Buscas por Vizinhança Variável (VNS - Variable Neighbourhood Search) e sua variante, a Descida Variável pela Vizinhança (VND - Variable Neighbourhood Descent) .

Este documento está organizado como segue, o capítulo 1 descreve o problema de flowshop com tempo de setup, suas equações e os algoritmos que as implementa. O capítulo 2 descreve o método VNS. O Capítulo 3 apresenta a metodologia de afinação, junto com seu resultado, o capítulo 4 apresenta os estudos, metodologias e resultados do experimento e o capitulo 5 apresenta as conclusões finais.

1 Flowshop com tempo de setup

Essa pesquisa se baseia em problemas flowshop com tempos de preparação(setup) dependentes das tarefas, o tornando dependente da ordenação. Porém, de acordo com CHENG, GUPTA e WANG (2000), existem, no geral, variações de operações da tarefa, as quais podemos citar:

1. Tempo de setup dependente da tarefa a ser processada.
2. Tempo de setup independente da tarefa a ser processada.
3. Tempo de processamento da tarefa a ser processada.
4. Tempo de remoção dependente da tarefa que foi processada.
5. Tempo de remoção independente da tarefa que foi processada.

Neste capítulo iremos discutir sobre os tempos de preparação(setup) para os problemas flowshop, destacando o tempo de setup dependente da tarefa a ser processada, que é utilizado como estudo principal dessa pesquisa.

Nosso foco são os problemas de flowshop assumindo o tempo de setup, definidos como um tempo necessário de preparação para mudar de uma tarefa para outra em uma determinada máquina. Podem ser distintos do tempo de processamento, onde podem ser processados antes do início de uma determinada tarefa, podem ser incluídos no tempo de processamento ou até mesmo ignorados por serem insignificantes.

De acordo com CHENG, GUPTA e WANG (2000), quando o tempo de setup é separado do tempo de processamento, ele pode ser iniciado antecipadamente assim que a máquina ficar livre do processo anterior, sabendo que o sistema já conhece qual será a próxima tarefa a ser executada. Assim, a máquina específica poderá utilizar seu tempo ocioso para completar o tempo de setup da tarefa.

Quando o tempo de setup é acoplado ao tempo de processamento, ele não pode ser iniciado antecipadamente, significando que o os procedimentos de preparação só podem ser iniciados assim que uma tarefa é alocada na máquina. O problema de programação em ambiente de programação flowshop com tempo de setup é aplicado em vários casos reais, em diversos tipos de máquinas usadas para os mais diferentes propósitos. Um bom exemplo são as indústrias de fabricação de carros, onde máquinas de pintura precisam ser ajustadas para uma nova cor quando um novo lote de carros está prestes a ser pintado. Outro exemplo claro é a indústria de papéis, onde as máquinas de corte de papéis são calibraddas quando mudam de um novo lote de corte para outro.

O *Makespan*, como foi citado, é um critério de desempenho de programação para ambientes de produção, como Flowshop, que proporciona a minimização do tempo total de execução de todas as tarefas em todas as máquinas. Seu cálculo é efetuado pela soma dos tempos de processamento das tarefas, porém, como se faz presente os tempos de setup que são dependentes da sequência, o cálculo do makespan depende deste sequenciamento.

O conjunto de regras utilizadas nesse trabalho para calcular o makespan está resumido no algoritmo 1, onde se é iniciado o vetor *makespan* que armazena os tempos calculados de processamento de cada tarefa, em cada máquina, o tempo de setup e tempos ociosos. O tempo do makespan é o tempo do processamento da última tarefa na última máquina que pode ser retornado por $makespan[k_{max} - 1][m_{max} - 1]$.

O algoritmo 1 calcula o tempo de makespan. Na linha 1 o vetor que armazena os tempos é iniciado com o tempo da primeira tarefa na primeira máquina. Na linha 3, a primeira tarefa passa por todas as outras máquinas, onde é calculado o tempo de processamento junto aos tempos nas máquinas anteriores. A linha 6 mostra as demais tarefas passando pela primeira máquina, onde é calculado o seu tempo de processamento junto ao tempo de setup e o tempo calculado das tarefas anteriores. Nas linhas 8 e 9 são iniciados os loops para as demais tarefas passarem pelas demais máquinas. A linha 10 mostra um comparativo dos tempos calculados. Se o tempo das tarefas anteriores e o tempo de setup da tarefa na máquina somados for maior que o tempo da tarefa na máquina anterior, será executado até a linha 11 onde o tempo de setup, o tempo de processamento e o tempo anterior das tarefas são calculados. Caso seja menor, irá até a linha 13 onde o tempo de processamento e o tempo da tarefa na máquina anterior são calculados.

Algorítimo 1 Calculo-Makespan(S^*, C^*, ST^{**}, k_{max}, m_{max})

Entrada: A sequência ordenada S^*, os tempos de processamento de cada tarefa C^*, os tempos de setup ST^{**} de cada tarefa em cada máquina, o número total de tarefas k_{max} e máquinas m_{max}.

1: $makespan[0][0] = C[S[0]][0]$
2: **para** $j = 1$ até m_{max} **faça**
3: $\quad makespan[0][j] = C[S[0]][j] + makespan[0][j-1]$
4: **fim para**
5: **para** $i = 1$ até k_{max} **faça**
6: $\quad makespan[i][0] = ST[0][S[i-1]][S[i]] + C[S[i]][0] + makespan[i-1][0]$
7: **fim para**
8: **para** $i = 1$ até k_{max} **faça**
9: $\quad$ **para** $j = 1$ até m_{max} **faça**
10: $\quad\quad$ **se** $makespan[i-1][j] + ST[j][S[i-1][S[i]] > makespan[i][j-1]$ **então**
11: $\quad\quad\quad makespan[i][j] = ST[j][S[i-1]][S[i] + C[S[i]][j] + makespan[i-1][j]$
12: $\quad\quad$ **senão**
13: $\quad\quad\quad makespan[i][j] = C[S[i]][j] + makespan[i][j-1]$
14: $\quad\quad$ **fim se**
15: $\quad$ **fim para**
16: **fim para**

2 VNS

De acordo com Hansen, Mladenović e Pérez (2009) o **Variable Neighbourhood Search** (VNS), *traduzindo Busca variável pela vizinhança*, é uma meta-heurística, ou um método para construção de heurísticas, visando resolver problemas de otimização combinatória e global, baseado nas mudanças sistemáticas da vizinhança tanto em fase de descida, para encontrar o mínimo local, como em fase de desordem para emergir da situação correspondente. Sua proposta foi feita pela primeira vez por Mladenović (1995), que desde então, tem sido estudado e desenvolvido rapidamente, tanto em seus métodos como em sua aplicação. O VNS se baseia nas seguintes regras:

1. Um mínimo local em relação a estrutura de vizinhanças não é necessariamente um mínimo local para outra estrutura de vizinhanças.
2. Um mínimo global é um mínimo local em relação a todas as estruturas de vizinhanças possíveis.
3. Para vários problemas, os mínimos locais são próximos uns dos outros em relação a várias outras vizinhanças.

A otimização, ou programação matemática, refere-se ao estudo de problemas que buscam escolher o melhor elemento, entre um conjunto de alternativas, por meio de uma função objetiva específica que apresenta ou tenta se aproximar de um valor desejado. O melhor elemento pode variar entre um valor inteiro que maximiza ou minimiza uma função de variável real até estruturas mais complexas. Os conhecidos "método da descida mais íngreme"(steepest descent) e o método dos mínimos quadrados (MMQ) são técnicas de otimização utilizadas, atribuídas ao grande matemático **Gauss**.

A meta-heurística é um método de solução para construção de heurística utilizado em problemas de otimizações globais. De acordo com Glover, Kochenberger e Gary (2003), são métodos de solução que orquestram e interagem entre os processos de melhoria local e estratégias de nível superior para criar um processo capaz de escapar de mínimos locais e realizar buscas robustas no espaço de soluções de um problema. Após, foi implementado na definição para abranger quaisquer procedimentos que empregassem estratégias para escapar de mínimos locais em espaços de busca de soluções complexas.

Como dito antes, sua ideia principal se baseia na mudança sistemática do vizinho combinado com uma busca local. Desde seu surgimento, foi desenvolvido e expandido através de várias pesquisas com foco nas regras básicas do VNS e seus principais

métodos, sendo aplicado nos mais variados campos. O VNS, ao contrário de várias outras meta-heurísticas, em seu esquema básico e extensões, tem uma estrutura simples e requerem poucos ou até nenhum parâmetro.

2.1 Estrutura

De acordo com Hansen, Mladenović e Pérez (2009), o VNS implementa uma heurística de pesquisa local para resolver problemas de otimização combinatória e global, permitindo a mudança das estruturas da vizinhança dentro da busca.

2.1.1 Busca Local

De acordo com Coopin (2004) **Busca Local** se inicia a partir de uma configuração (geralmente aleatória), onde é feita pequenas alterações nessa configuração, até alcançar um estado que não é mais possível atingir melhorias. Desta forma, um algoritmo de busca local define uma solução inicial x, onde uma das formas é percorrer a vizinhança $N(x)$ dessa solução em busca de outro valor menor $f(x)$ (para um problema de minimização) entre os descendentes de x. Caso seja encontrado uma nova solução vizinha, se torna a solução corrente e o algoritmo continua. Caso não seja encontrado uma nova solução, a solução corrente x é definida como ótimo local em relação à vizinhança.

De acordo com Hansen, Mladenović e Pérez (2009) a heurística de busca local pode ser feita a partir de uma solução inicial x, descobrindo a descida em x, dentro da sua vizinhança $N(x)$, indo na mesma direção em busca do mínimo de $f(x)$ dentro da vizinhança $N(x)$. Caso não contenha mais descida, a heurística para, caso contenha ela é iterada. Geralmente referida como *descida mais inclinada* ou *melhor melhoria*, que está resumido no Algoritmo 2. A saída consiste no mínimo local, denotado por x. A estrutura da vizinha $N(x)$ é definida para todo os $x \in X$. Em cada etapa, a vizinhança $N(x)$ de x é explorada completamente. Devido ao maior consumo de tempo, geralmente uma alternativa é usar a *primeira descida*, também conhecido como *primeira melhoria*, que está resumido no algoritmo 3. Vetores $x_i \in N(x)$ são enumerados sistematicamente e um movimento é feito assim que uma descida é encontrada. A *Melhor-Troca* que é definido no algorítimo 4, é uma busca local implementada no projeto, que consiste na execução do método de troca de vizinhos para uma descida na vizinhança $N(x)$ em busca do mínimo local. O algoritmo realiza o método de troca na linha 4 com duas tarefas e após verifica se houve melhoria, caso não, a alteração anterior é cancelada e volta a realizar a troca com outras tarefas. Caso aja melhoria, é retornado o mínimo local. Os algoritmos 5 e 6 realizam os mesmos passos, só que o algoritmo 5 utiliza o

Algorítimo 2 Melhor-Melhoria(x)

1: **repetir**
2: $x' \leftarrow x$
3: $x \leftarrow \arg\min_{y \in N(x)} f(y)$
4: **até que** $f(x) \geq f(x')$

Algorítimo 3 Primeira-Melhoria(x)

1: **repetir**
2: $x' \leftarrow x$
3: $i \leftarrow 0$
4: **repetir**
5: $i \leftarrow i + 1$
6: $x \leftarrow$arg min $\{f(x), f(x_i)\}$, $x_i \in N(x)$
7: **até que** ($f(x) < f(x_i)$ ou $i = |N(x)|$)
8: **até que** $f(x) \geq f(x')$

Algorítimo 4 Melhor-Troca(x, k_{max})

Entrada: Um valor de índice x e o numero de tarefas k_{max}

1: $k \leftarrow 0$
2: **repetir**
3: $x' \leftarrow x$
4: $x \leftarrow$Troca-Vizinhos(k,x)
5: $k \leftarrow k + 1$
6: **até que** ($k = k_{max}$ & $f(x) > f(x')$)

método de inserção e o algoritmo 6 utiliza inversão de tarefas.

Algorítimo 5 Melhor-Insercao(x, k_{max})

Entrada: Um valor de índice x e o numero de tarefas k_{max}

1: $k \leftarrow 0$
2: **repetir**
3: $x' \leftarrow x$
4: $x \leftarrow$Insercao-Vizinho(k)
5: $k \leftarrow k + 1$
6: **até que** ($k = k_{max}$ & $f(x) > f(x')$)

2.1.2 VND

De acordo com Hansen, Mladenović e Pérez (2009) O **Variable Neighbourhood Descent** (VND) traduzindo "Descida Variável pela Vizinhança", é um método que é obtido se a mudança de vizinhanças é realizada de forma determinística. Uma solução inicial x é dada na descrição dos algoritmos que seguem. A maioria das buscas locais heurísticas em sua fase de descida usa poucas vizinhanças. Note que a solução final é

Algorítimo 6 Melhor-Inversao(x, k_{max})

Entrada: Um valor de índice x e o numero de tarefas k_{max}

1: $k \leftarrow 0$
2: **repetir**
3: $x' \leftarrow x$
4: $x \leftarrow$Inversão-Vizinhos(k,x)
5: $k \leftarrow k + 1$
6: **até que** ($k = k_{max}$ & $f(x) > f(x')$)

um mínimo local em relação a todas as vizinhanças K'_{max}, então as chances de encontrar um global são maiores quando se usa o VND do que com uma única estrutura de vizinhança. Além disso, a partir da ordem sequencial da estrutura de vizinhanças em VND citada, pode-se desenvolver uma estratégia aninhada. Essa abordagem é aplicada em Brimberg et al. (2000), Hansen e Mladenović (2001).

No algoritimo 7 é mostrado resumidamente o VND implementado no projeto, onde o mesmo se utiliza de três buscas locais para encontrar a ótima local e assim a ótima global. Da linha 4 até 6 é exibida a utilização da busca local *Melhor-Troca* em todas as tarefas. Caso nenhuma melhoria tenha sido obtida na busca anterior, como mostra na linha 7, é passada para a próxima busca local *Melhor-Inserção* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado a busca local *Melhor-Inversão* com todas as tarefas. Caso haja alguma melhoria nas buscas locais, o processo é reiniciado, caso não, o algoritmo é encerrado e retornado o melhor local (possível global) mais próximo do global. Outros algoritmos implementados do VND são mostrados resumidamente, pois são similares. Os algoritmos de 8 à 18 diferem apenas na quantidade de buscas presente e na ordem de execução das mesma.

O algoritimo 8 é um resumo do VND que se utiliza de três buscas locais: Melhor-Troca, Melhor-Inversão e Melhor-Inserção. Iniciando com o Melhor-Troca assim como o algoritmo 7. Da linha 4 até 6 é exibido a utilização da busca local *Melhor-Troca* em todas as tarefas. Caso nenhuma melhoria tenha sido obtido na busca anterior, é passado para a próxima busca local *Melhor-Inversão* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado a busca local *Melhor-Inserção* com todas as tarefas.

O algoritmo 9 é outro do VND que se utiliza de três buscas locais: Melhor-Inserção, Melhor-Troca e Melhor-Inversão. Esse VND inicia com o Melhor-Inserção, se saindo um pouco mais diferente do algoritmo 7 e 8. Da linha 4 até 6 é exibida a utilização da busca local *Melhor-Inserção* em todas as tarefas. Caso nenhuma melhoria tenha sido obtida na busca anterior, é passado para a próxima busca local *Melhor-Troca* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado

Algorítimo 7 VND-MelhorTroca1(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Troca(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inserção(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
  se Nenhuma melhoria **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inversão(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que Nenhuma melhoria seja obtida

Algorítimo 8 VND-MelhorTroca2(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Troca(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inversão(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inserção(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

a busca local *Melhor-Inversão* com todas as tarefas.

Algorítimo 9 VND-MelhorInserção1(k_{max})

Entrada: Numero de tarefas k_{max}

```
1:  repetir
2:    k ← 0
3:    repetir
4:      Melhor-Inserção(k, k_max)
5:      k ← k + 1
6:    até que k = k_max
7:    se nenhuma melhoria for obtida então
8:      k ← 0
9:      repetir
10:       Melhor-Troca(k, k_max)
11:       k ← k + 1
12:     até que k = k_max
13:   fim se
14:   se nenhuma melhoria for obtida então
15:     k ← 0
16:     repetir
17:       Melhor-Inversão(k, k_max)
18:       k ← k + 1
19:     até que k = k_max
20:   fim se
21: até que nenhuma melhoria seja obtida
```

No algoritmo 10 é mostrado o VND que se utiliza três buscas locais: Melhor-Inserção, Melhor-Inversão e Melhor-Troca. Parecido com o algoritmo 9. Da linha 4 até 6 é exibida a utilização da busca local *Melhor-Inserção* em todas as tarefas. Caso nenhuma melhoria tenha sido obtida na busca anterior, é passado para a próxima busca local *Melhor-Inversão* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado a busca local *Melhor-Troca* com todas as tarefas.

No algoritmo 11 é mostrado o VND que se utiliza três buscas locais: Melhor-Inversão, Melhor-Troca e Melhor-Inserção. Da linha 4 até 6 é exibida a utilização da busca local *Melhor-Inversão* em todas as tarefas. Caso nenhuma melhoria tenha sido obtida na busca anterior, é passado para a próxima busca local *Melhor-Troca* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado a busca local *Melhor-Inserção* com todas as tarefas.

O algoritmo 12 resume o VND que se utiliza três buscas locais: Melhor-Inversão, Melhor-Inserção e Melhor-Troca. Da linha 4 até 6 é exibida a utilização da busca local *Melhor-Inversão* em todas as tarefas. Caso nenhuma melhoria tenha sido obtida na

Algorítimo 10 VND-MelhorInserção2(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inserção(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inversão(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Troca(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

Algorítimo 11 VND-MelhorInversão1(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inversão(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Troca(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inserção(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

busca anterior, é passado para a próxima busca local *Melhor-Inserção* na linha 10 com todas as tarefas. Caso nenhuma melhoria tenha sido obtida, é executado a busca local *Melhor-Troca* com todas as tarefas.

Algorítimo 12 VND-MelhorInversão2(k_{max})

Entrada: Numero de tarefas k_{max}

```
1:  repetir
2:    k ← 0
3:    repetir
4:      Melhor-Inversão(k, k_max)
5:      k ← k + 1
6:    até que k = k_max
7:    se nenhuma melhoria for obtida então
8:      k ← 0
9:      repetir
10:       Melhor-Inserção(k, k_max)
11:       k ← k + 1
12:     até que k = k_max
13:   fim se
14:   se nenhuma melhoria for obtida então
15:     k ← 0
16:     repetir
17:       Melhor-Troca(k, k_max)
18:       k ← k + 1
19:     até que k = k_max
20:   fim se
21: até que nenhuma melhoria seja obtida
```

O algorítimo 13 mostra um VND com 2 buscas locais: Melhor-Troca e Melhor-Inserção. Em 4 ele utiliza a busca local *Melhor-Troca* e na 10 utiliza *Melhor-Inserção*, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

No algorítimo 14 resume um VND com 2 buscas locais: Melhor-Troca e Melhor-Inversão. Em 4 ele utiliza a busca local *Melhor-Troca* e na 10 utiliza *Melhor-Inversão*, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

No algorítimo 15 é um pouco diferente, onde o mesmo contém apenas 2 buscas locais, em 4 ele utiliza a busca local *Melhor-Inserção* e na 10 utiliza *Melhor-Troca*, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

O algorítimo 16 resume outro VND com 2 buscas locais: Melhor-Inserção e

Algorítimo 13 VND-Dois-MelhorTroca1(k_{max})

Entrada: Numero de tarefas k_{max}

```
1:  repetir
2:     k ← 0
3:     repetir
4:        Melhor-Troca(k, k_max)
5:        k ← k + 1
6:     até que k = k_max
7:     se nenhuma melhoria for obtida então
8:        k ← 0
9:        repetir
10:          Melhor-Inserção(k, k_max)
11:          k ← k + 1
12:       até que k = k_max
13:    fim se
14: até que nenhuma melhoria seja obtida
```

Algorítimo 14 VND-Dois-MelhorTroca2(k_{max})

Entrada: Numero de tarefas k_{max}

```
1:  repetir
2:     k ← 0
3:     repetir
4:        Melhor-Troca(k, k_max)
5:        k ← k + 1
6:     até que k = k_max
7:     se nenhuma melhoria for obtida então
8:        k ← 0
9:        repetir
10:          Melhor-Inversão(k, k_max)
11:          k ← k + 1
12:       até que k = k_max
13:    fim se
14: até que nenhuma melhoria seja obtida
```

Melhor-Inversão. Em 4 ele utiliza a busca local *Melhor-Inserção* e na 10 utiliza *Melhor-Inversão*, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

No algorítimo 17 resume um VND com 2 buscas locais: Melhor-Inversão e Melhor-Troca. Em 4 ele utiliza a busca local *Melhor-Inversão* e na 10 utiliza *Melhor-Troca*, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

No algorítimo 18 resume um VND com 2 buscas locais: Melhor-Inversão e Melhor-Inserção. Em 4 ele utiliza a busca local *Melhor-Inversão* e na 10 utiliza *Melhor-*

Algorítimo 15 VND-Dois-MelhorInserção1(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inserção(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Troca(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

Algorítimo 16 VND-Dois-MelhorInserção2(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inserção(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inversão(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

Inserção, sempre verificando se houve melhoria para continuação, caso não, é encerrado o processo.

Algorítimo 17 VND-Dois-MelhorInversão1(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inversão(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Inserção(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

Algorítimo 18 VND-Dois-MelhorInversão2(k_{max})

Entrada: Numero de tarefas k_{max}

repetir
  $k \leftarrow 0$
  repetir
    Melhor-Inversão(k, k_{max})
    $k \leftarrow k + 1$
  até que $k = k_{max}$
  se nenhuma melhoria for obtida **então**
    $k \leftarrow 0$
    repetir
      Melhor-Troca(k, k_{max})
      $k \leftarrow k + 1$
    até que $k = k_{max}$
  fim se
até que nenhuma melhoria seja obtida

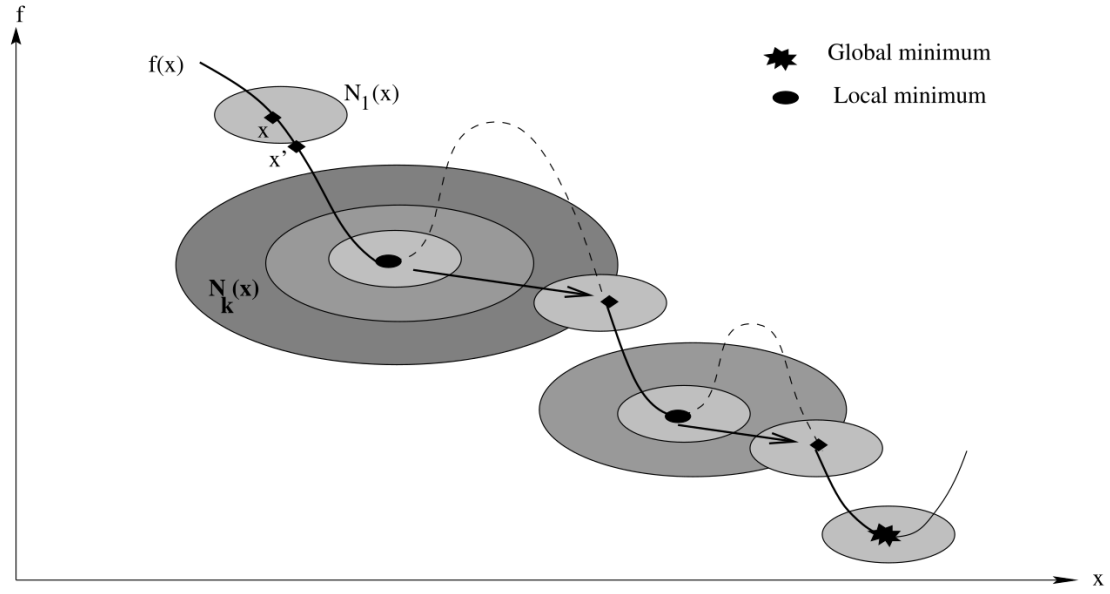

Figura 2 – VNS Básico (Fonte: Hansen, Mladenović e Pérez (2009))

2.1.3 VNS Básico

O método **Basic VNS** (BVNS) ou *VNS Básico* combina mudanças determinísticas e estocásticas na vizinhança. Muitas vezes, vizinhos sucessivos N_k serão aninhados. Seus passos são mostrados na figura 2 e resumidos no algoritmo 19, que foi implementado no projeto. Observe que esse VNS se faz da utilização, na linha 2, do VND, resumido no algoritmo 7 que o mesmo se utiliza de buscas locais. Se observa também a utilização do **Kick()** na linha 3 para evitar a estagnação, o que pode ocorrer se uma regra determinística for aplicada.

Algorítimo 19 VNS(k_{max}, t_{max})

Entrada: O numero de tarefas k_{max} e o tempo limite t_{max}

1: **repetir**
2: VND-MelhorTroca1(k_{max})
3: Kick()
4: $t \leftarrow$ CpuTime()
5: **até que** ($t \geq t_{max}$)

Kick é um procedimento de agitação. Esse passo é fundamental no VNS, mas fácil de implementar. Sua implementação pode ser feita em poucas linhas de código. Por exemplo: na implementação do nosso trabalho, para resolver o problema de parada, a agitação mais simples é a realocação de uma tarefa k aleatória, escolhida da sequência, para outra que também é escolhida aleatoriamente. O algoritmo 20 mostra os passos realizados pelo **Kick** nesse trabalho. Na linha 1 a variável k recebe o valor 0. Na linha 3 a variável x recebe o valor do método *rand()* onde o mesmo retorna um valor

inteiro positivo aleatório que pode ser menor ou igual ao número de tarefas. Na linha 4 a variável y recebe o valor do método *rand()* onde o mesmo retorna um valor inteiro positivo aleatório que pode ser menor ou igual ao número de tarefas. Na linha 5 é executada o método *Troca-Vizinhos()* que é passado as variáveis x e y onde as mesmas contém as posições que serão trocadas pelo método. Na linha 6 a variável k é acrescentado o valor de 1. Na linha 7 é a condição da repetição, onde a mesma só deve acontecer enquanto a variável k for menor que a força do Kick.

Algorítimo 20 Kick($strength, k_{max}$)

Entrada: A força do Kick $strength$ e o numero de tarefas k_{max}

1: $k \leftarrow 0$
2: **repetir**
3: $x \leftarrow (rand() \leq k_{max})$
4: $y \leftarrow (rand() \leq k_{max})$
5: Troca-Vizinhos(x,y)
6: $k \leftarrow k + 1$
7: **até que** ($k >= strength$)

3 Afinação

O objetivo desse capítulo é apresentar a metodologia de afinação utilizada e considerações realizadas para realizações dos experimentos dessa monografia. Explicando sua utilização e estudo dos quais operadores e valores de parâmetros devem ser utilizados para que o método proposto tenha um bom desempenho. Junto com os resultados obtidos dos experimentos, foi possível relatar qual parâmetro deve ser utilizado para melhor desempenho.

O problema com o qual é possível deparar ao tentar adaptar um método metaheurístico a um problema específico, consiste em uma série de definições relativas tanto ao tipo de método que se quer propor, quanto a que parâmetros afinar, e que valores ou atributos podem ser atribuídos a cada parâmetro. No nosso experimento, a vizinhança e a força do Kick são passados como parâmetros, onde a vizinhança é passada como instância e a força do Kick deve ser afinada para cada VNS.

Para os experimentos independentes, um subconjunto das instâncias de Ruben Ruiz[1] foi separado, como sendo representativo do conjunto de testes. Este subconjunto, denominado conjunto de afinação, é composto das 6 instâncias de cada grupo com N =50 ou N = 100 tarefas, totalizando 24 instâncias. As instância de afinação foram: SDST1031, SDST1041, SDST1051, SDST1061, SDST1071, SDST1081, SDST5031, SDST5041, SDST5051, SDST5061, SDST5071, SDST5081, SDST10031, SDST10041, SDST10051, SDST10061, SDST10071, SDST10081, SDST12531, SDST12541, SDST12551, SDST12561, SDST12571, SDST12581. Um experimento independente consiste na realização de 5 execuções de cada versão do método em teste sobre cada instância do conjunto de afinação, resultando em um conjunto de 120 amostras aleatórias para cada versão testada. Cada execução terá o mesmo limite de tempo, definido por Dubois-Lacoste, Pagnozzi e Stützle (2016), onde o tempo limite fica: $k \times (m/2) \times 60$ milissegundos. Ele será utilizado como critério de parada, pois os métodos disponíveis para comparação adotam o mesmo critério. Esses conjuntos são comparados usando testes estatísticos para averiguar se há diferenças significativas entre eles. O computador utilizado nos testes possui processador Intel Xeon E5-2640 - 2.0GHz.

Para que a comparação seja possível, os resultados das execuções são transformados em desvios relativos percentuais (DRP), calculados usando a equação 4.1. Na equação 4.1, $\Pi_{Execucao}$ é a solução oriunda de urna execução, e $\Pi_{EMelhor}$ é a melhor solução obtida dentre todas as geradas no experimento.

$$DRP(\%) = 100 \times \frac{makespan(\Pi_{Execucao}) - makespan(\Pi_{EMelhor})}{makespan(\Pi_{EMelhor})} \quad (4.1)$$

[1] (http://soa.iti.es)

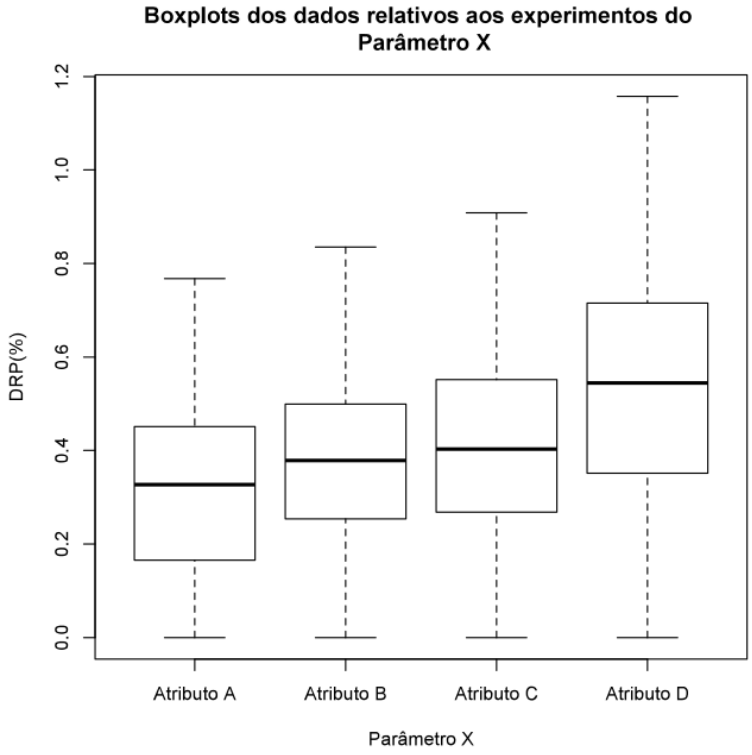

Figura 3 – Boxplots dos dados relativos aos experimentos do Parâmetro X

O uso de DRP como métrica de comparação já é prática comum nas pesquisas relativas a PFSP.

O teste de Kruskal-Wallis (Conover (1999)) é um método não paramétrico utilizado para testar se um conjunto de amostras provêm da mesma distribuição. Kruskal-Wallis é utilizado para mais de duas amostras, sendo uma alternativa ao teste de Mann-Whitney. O teste foi utilizado, empregando as ferramentas de R Core Team (2017), já que possui mais de dois valores para o Kick. E, em um paralelo da metodologia de Ruiza e Stützleb (2007), utiliza-se a mediana para discernir qual atributo testado exibe a melhor performance, uma vez que a mediana é um indicador de tendência central independente de distribuição.

Os dados são analisados pelos gráficos de boxplots e testes. O gráfico boxplots é utilizado para analisar a variação de uma variável dentre os diferentes grupos de dados. Mcgill, Tukey e Larsen (1978) definem o bloxplot sendo um sumário visual dos dados amostrados. O eixo vertical representa a variável a ser analisada, o eixo horizontal representa o fator de interesse e a linha central mais escura marca o valor mediano. A 3 mostra quatro boxplots correspondentes aos dados de quatro atribuições em comparação

Independente do teste utilizado, qualquer p-valor obtido igual ou inferior a 5% (0,05) é considerado significativo (CONOVER, 1999)(DOWDY; WEARDEN; CHILKO,

Kick	min	max	mean	median	std
3	0.0000000	3.0881459	0.7414137	0.5700663	0.7034894
4	0.0000000	2.1155015	0.7570115	0.6096745	0.6194217
5	0.0000000	2.9239766	0.8519562	0.6677446	0.6743127
6	0.0000000	3.2948328	0.9648380	0.8903360	0.6774217
7	0.0000000	3.0516717	0.9988353	0.8119994	0.8008520

Tabela 1 – Valores DRP(%) dos resultados de SW-INS-INV

2004).

Em resumo, a metodologia de afinação de parâmetros consiste em, inicialmente, definir que parâmetros devem ser ajustados para cada método. Com base na literatura do método e na literatura do PFSP com critério makespan, especulam-se possíveis atributos para cada parâmetro, e determina-se uma configuração inicial de parâmetros. Então executam-se experimentos comparativos independentes, cada experimento coleta amostras relativas a atributos distintos de um parâmetro, usando um subconjunto de 24 instâncias do conjunto Ruben Ruiz, e realizando 5 execuções sobre cada instância. Desta forma é possível coletar 120 amostras para cada atributo em comparação. Os resultados são transformados em DRPs (usando a equação 4.1), e examinados usando boxplots e testes não-paramétricos. Caso o teste aponte a presença de diferenças significativas entre a performance dos atributos testados, utiliza-se a mediana para discernir a melhor opção dentre eles.

Com o uso da metodologia, aplicando os testes de Kruskal-Wallis para descobrir qual força utilizar no kick, entre os valores de 3,4,5,6,7. Em sua pesquisa, Dong, Huang e Chen (2008) testou vários números para movimento de mudança, e seus resultados indicaram que 4 a 7 movimentos de mudança são necessários para usar quando se utiliza a busca local de inserção.

Na figura 4 representa o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Troca, Melhor-Inserção e Melhor Inversão. O seu p-valor é maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 1 apresenta os resultados detalhados do gráfico.

Na figura 5 apresenta o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Troca, Melhor-Inversão e Melhor-Inserção. O seu p-valor é maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 2 apresenta os resultados detalhados do gráfico.

Na figura 6 apresenta o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Inversão, Melhor-Troca e Melhor-Inserção. O seu p-valor é

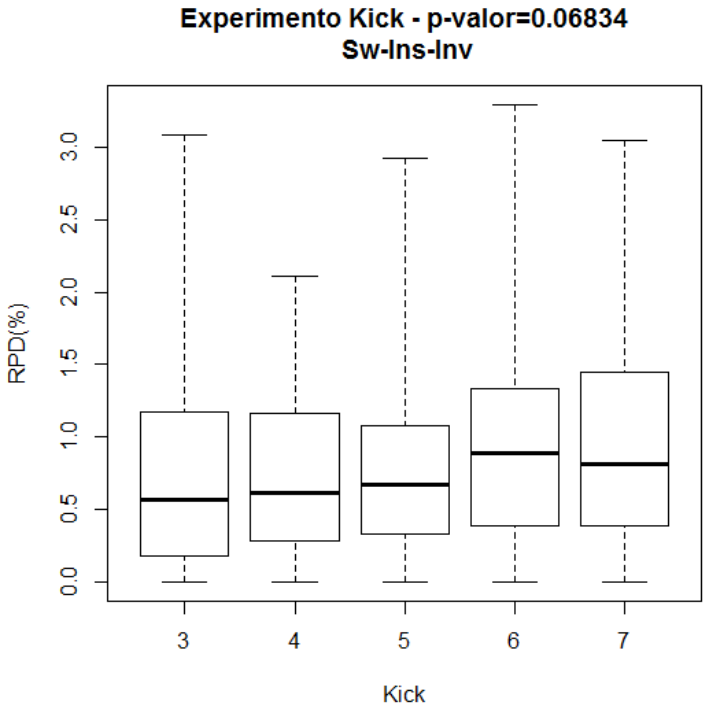

Figura 4 – Boxplots VNS SW-INS-INV

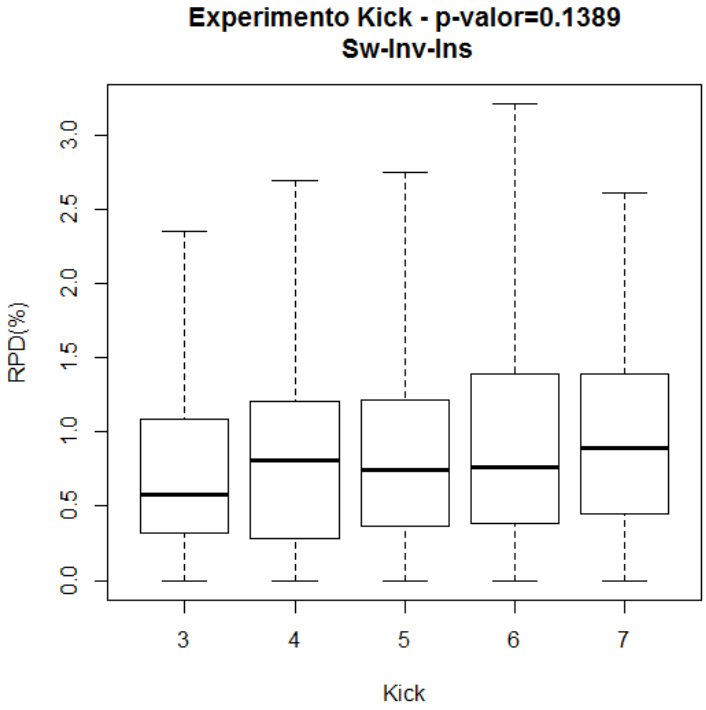

Figura 5 – Boxplots VNS SW-INV-INS

Kick	min	max	mean	median	std
3	0.0000000	2.3475315	0.7352274	0.5761024	0.6044185
4	0.0000000	2.6935045	0.7743637	0.8070716	0.5688877
5	0.0000000	2.7468538	0.8931971	0.7481247	0.7143924
6	0.0000000	3.2066796	0.9194828	0.7649089	0.7072217
7	0.0000000	2.6066351	0.9806039	0.8904750	0.6497856

Tabela 2 – Valores DRP(%) dos resultados de SW-INV-INS

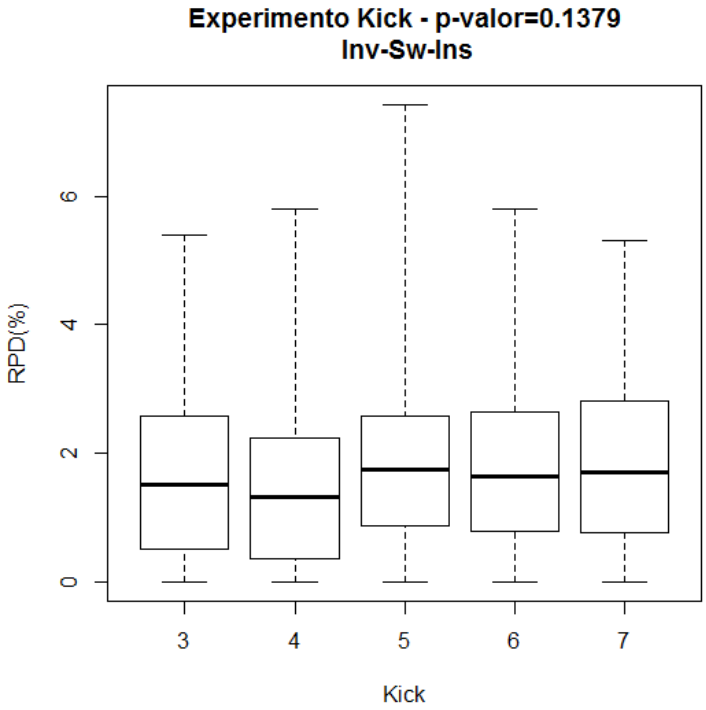

Figura 6 – Boxplots VNS INV-SW-INS

Kick	min	max	mean	median	std
3	0.0000000	5.402637	1.633834	1.498324	1.253805
4	0.0000000	5.797101	1.473712	1.312947	1.280481
5	0.0000000	7.420330	1.882819	1.749181	1.345647
6	0.0000000	5.806924	1.923198	1.629310	1.388005
7	0.0000000	5.302793	1.867270	1.698418	1.364522

Tabela 3 – Valores RPD(%) dos resultados de INV-SW-INS

maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 3 apresenta os resultados detalhados do gráfico.

Na figura 7 apresenta o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Inversão, Melhor-Inserção e Melhor-Troca. O seu p-valor é menor que 5% o tornando significativo, como resultado o kick utilizado será o de 3,

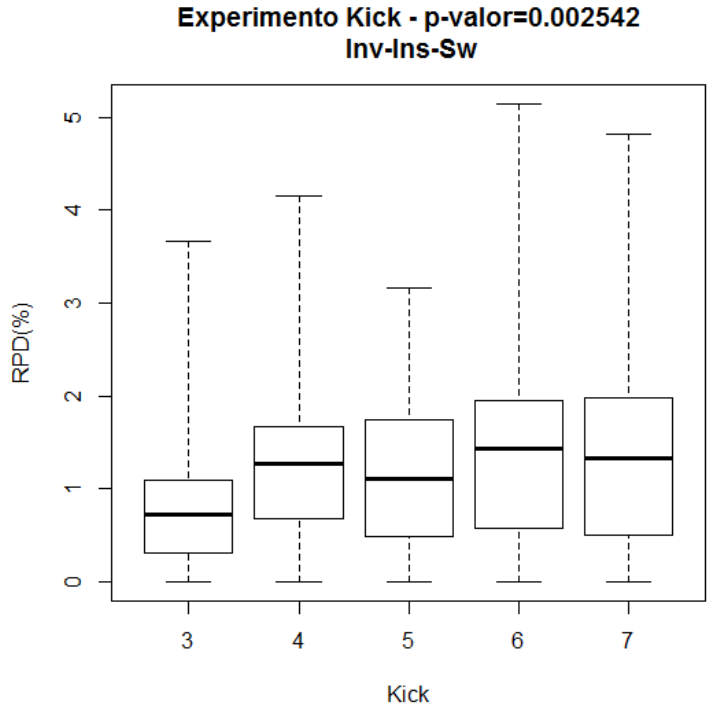

Figura 7 – Boxplots VNS INV-INS-SW

Kick	min	max	mean	median	std
3	0.0000000	3.6602701	0.8579952	0.7250114	0.7721779
4	0.0000000	4.1577825	1.2764387	1.2615977	0.8634344
5	0.0000000	3.1691044	1.1663178	1.1045828	0.7920039
6	0.0000000	5.806924	1.390508	1.431515	1.009196
7	0.0000000	4.821132	1.356880	1.329346	1.022454

Tabela 4 – Valores DRP(%) dos resultados de INV-INS-SW

com o p-valor de 0.002542. A tabela 4 apresenta os resultados detalhados do gráfico.

Na figura 8 apresenta o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Inserção, Melhor-Troca e Melhor-Inversão. O seu p-valor é maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 5 apresenta os resultados detalhados do gráfico.

Na figura 9 apresenta o boxplots do VNS que utiliza o VND com três buscas locais na ordem: Melhor-Inserção, Melhor-Inversão e Melhor-Troca. O seu p-valor é menor que 5% o tornando significativo, como resultado o kick utilizado será o de 3, com o p-valor de 0.03729. A tabela 6 apresenta os resultados detalhados do gráfico.

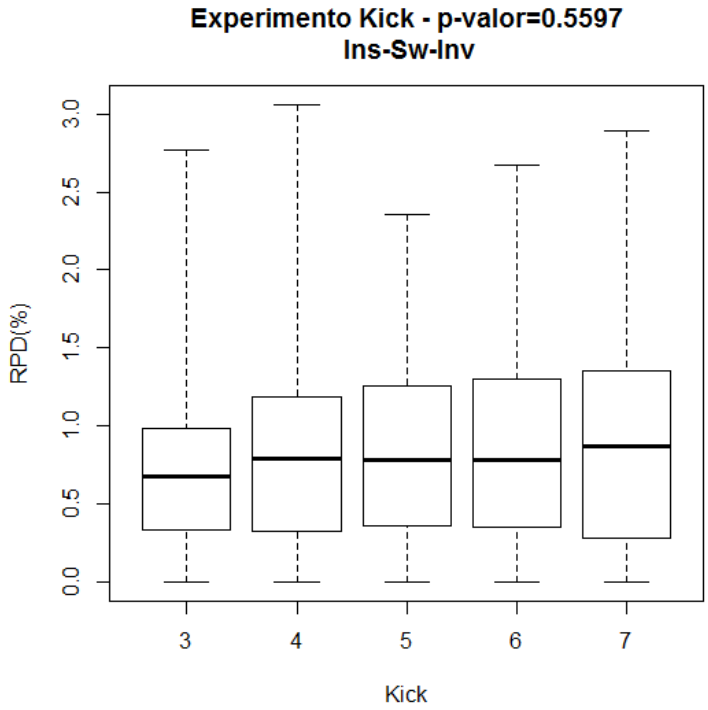

Figura 8 – Boxplots VNS INS-SW-INV

Kick	min	max	mean	median	std
3	0.0000000	2.7683341	0.7234274	0.6728041	0.5303999
4	0.0000000	3.0597377	0.8447981	0.7936688	0.6518149
5	0.0000000	2.3546372	0.8530043	0.7817484	0.6309635
6	0.0000000	2.6711996	0.8698102	0.7812431	0.6211880
7	0.0000000	2.8897523	0.9130372	0.8651368	0.7005777

Tabela 5 – Valores DRP(%) dos resultados de VNS INS-SW-INV

Kick	min	max	mean	median	std
3	0.0000000	2.0399399	0.6875894	0.5794637	0.5032035
4	0.0000000	2.0016821	0.7469149	0.7072702	0.5412163
5	0.0000000	2.3396425	0.8949691	0.8581042	0.5706011
6	0.0000000	2.3641791	0.8750197	0.9110562	0.5637558
7	0.0000000	2.6948989	0.9840117	0.9513245	0.6736669

Tabela 6 – Valores DRP(%) dos resultados de VNS INS-INV-SW

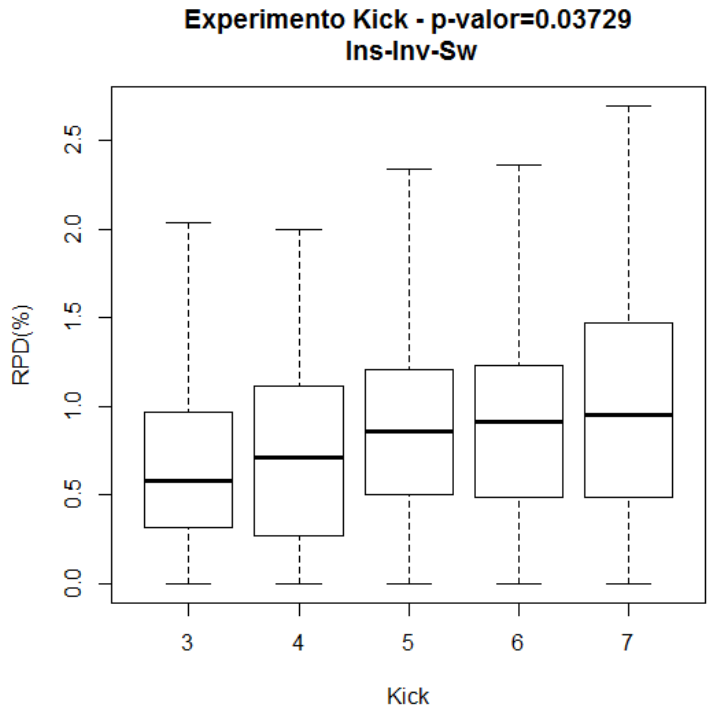

Figura 9 – Boxplots VNS INS-INV-SW

Kick	min	max	mean	median	std
3	0.0000000	3.633933	0.717735	0.568308	0.631985
4	0.0000000	3.1842489	0.7826881	0.6934242	0.6623042
5	0.0000000	3.9279655	0.9089584	0.6171985	0.7438184
6	0.0000000	2.2054026	0.9231671	0.8739140	0.6224123
7	0.0000000	2.3396425	0.9405790	0.8992617	0.5814433

Tabela 7 – Valores DRP(%) dos resultados de VNS SW-INS

Na figura 10 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Troca e Melhor-Inserção. O seu p-valor é maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 7 apresenta os resultados detalhados do gráfico.

Na figura 11 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Troca e Melhor-Inversão. O seu p-valor é menor que 5% o tornando significativo, como resultado o kick utilizado será o de 3, com o p-valor de 0.001 758. A tabela 8 apresenta os resultados detalhados do gráfico.

Na figura 12 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Inserção e Melhor-Troca. O seu p-valor é maior que 5% o

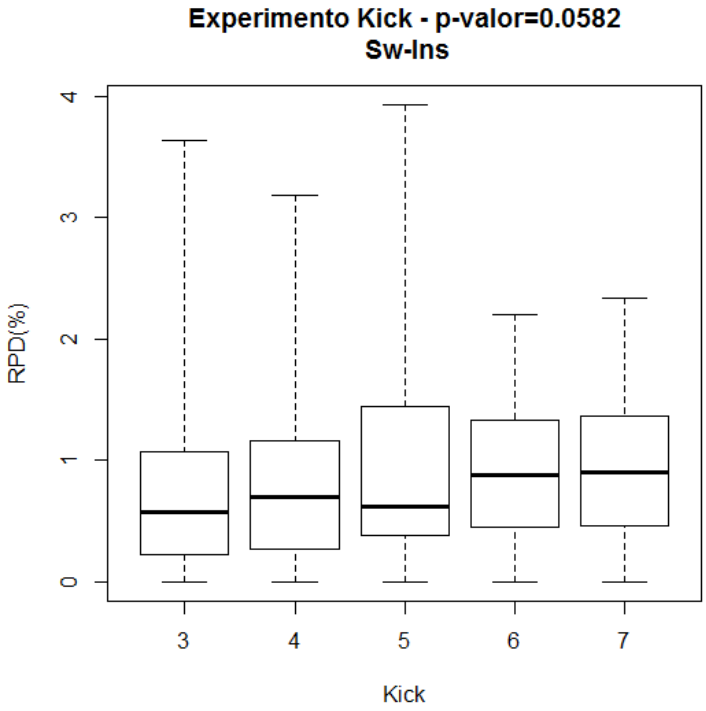

Figura 10 – Boxplots VNS SW-INS

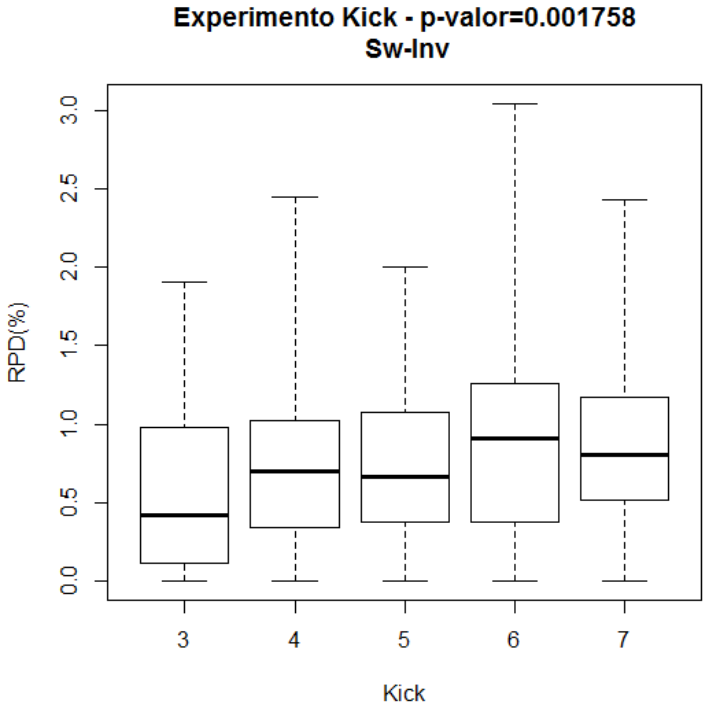

Figura 11 – Boxplots VNS SW-INV

Kick	min	max	mean	median	std
3	0.0000000	1.9076422	0.5683225	0.4215084	0.5125252
4	0.0000000	2.4483065	0.7338715	0.7013178	0.5073042
5	0.0000000	2.0024029	0.7528176	0.6617497	0.5194327
6	0.0000000	3.0445262	0.9294033	0.9060272	0.6712367
7	0.0000000	2.4296675	0.8618420	0.8035175	0.4794151

Tabela 8 – Valores DRP(%) dos resultados de VNS SW-INV

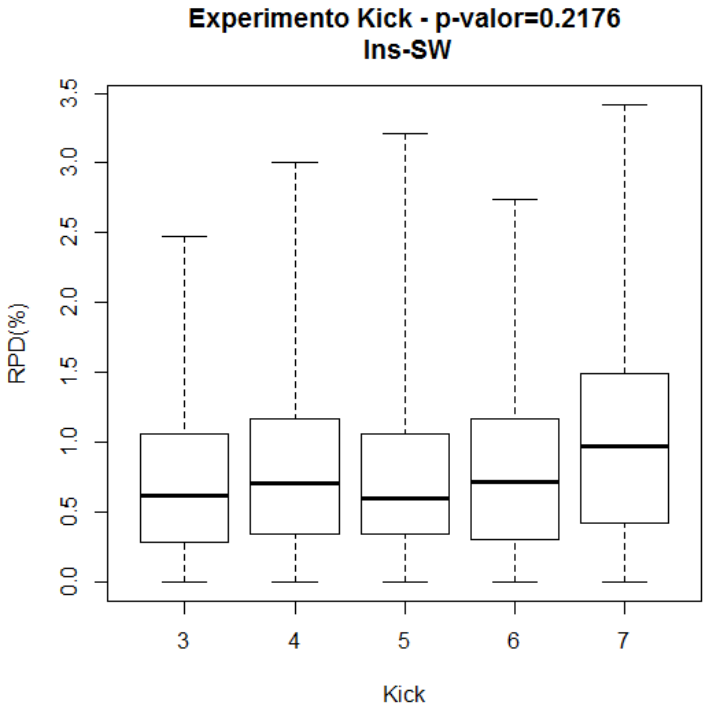

Figura 12 – Boxplots VNS INS-SW

Kick	min	max	mean	median	std
3	0.0000000	2.4786073	0.7634025	0.6137974	0.6259472
4	0.0000000	2.9999016	0.8229976	0.7008709	0.6501155
5	0.0000000	3.2110092	0.7961083	0.5945965	0.6894727
6	0.0000000	2.7367401	0.7831497	0.7154179	0.5942874
7	0.0000000	3.4162240	1.0422298	0.9668162	0.8107179

Tabela 9 – Valores DRP(%) dos resultados de VNS INS-SW

tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 9 apresenta os resultados detalhados do gráfico.

Na figura 13 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Inserção e Melhor-Inversão. O seu p-valor é menor que 5% o tornando significativo, portando o kick utilizado será o de 4, com o p-valor de 0.002076.

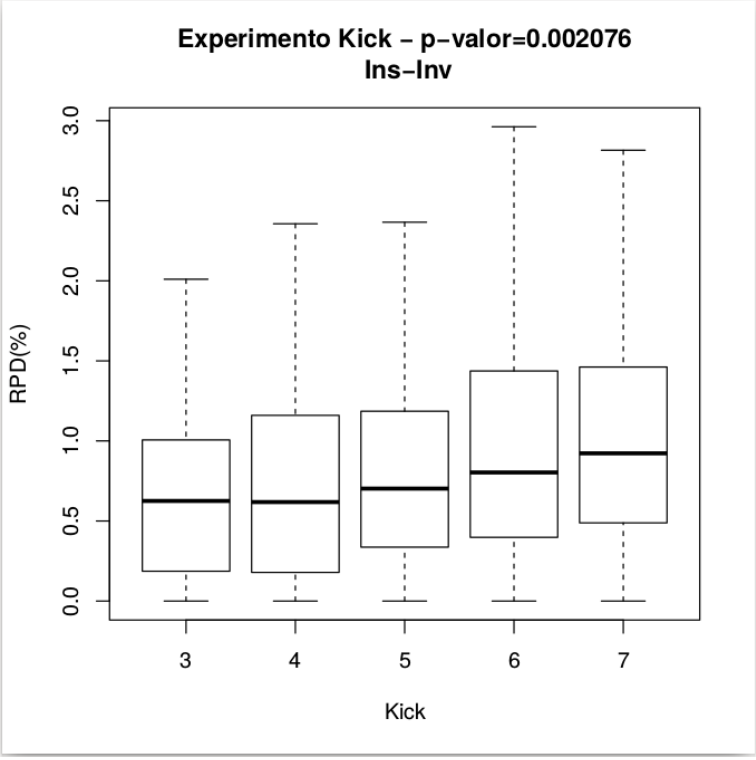

Figura 13 – Boxplots VNS INS-INV

Kick	min	max	mean	median	std
3	0.0000000	2.0098548	0.6324311	0.6252468	0.4886784
4	0.0000000	2.3560465	0.7204039	0.6180142	0.5890175
5	0.0000000	2.3658227	0.7726269	0.7022282	0.5352805
6	0.0000000	2.9621664	0.9725751	0.8030128	0.6979156
7	0.0000000	2.8155245	1.0079184	0.9226290	0.6659627

Tabela 10 – Valores DRP(%) dos resultados de VNS INS-INV

A tabela 10 apresenta os resultados detalhados do gráfico.

Na figura 14 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Inversão e Melhor-Troca. O seu p-valor é menor que 5% o tornando significativo, como resultado o kick utilizado será o de 3, com o p-valor de 0.007977. A tabela 11 apresenta os resultados detalhados do gráfico.

Na figura 15 apresenta o boxplots do VNS que utiliza o VND com duas buscas locais na ordem: Melhor-Inversão e Melhor-Inserção. O seu p-valor é maior que 5% o tornando insignificativo, portando o kick utilizado será o de 3, como regra padrão. A tabela 12 apresenta os resultados detalhados do gráfico.

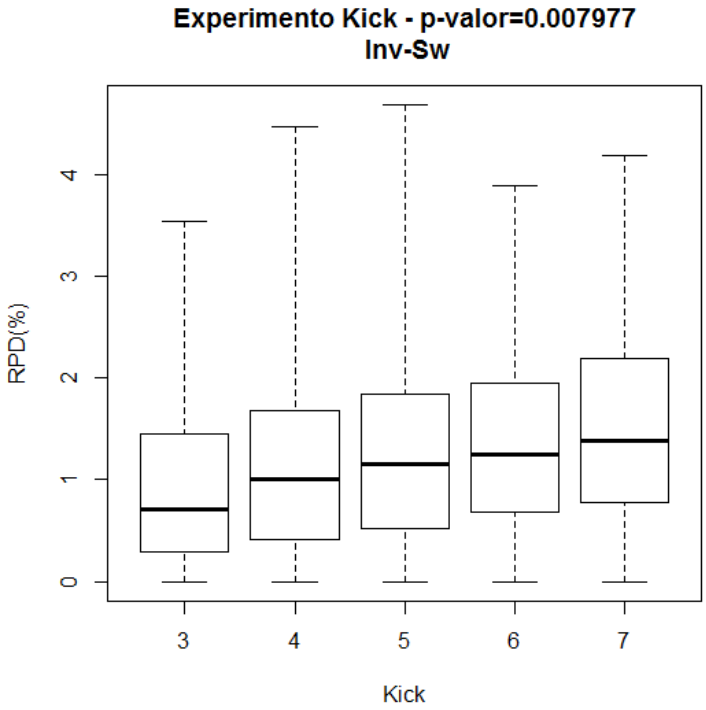

Figura 14 – Boxplots VNS INV-SW

Kick	min	max	mean	median	std
3	0.0000000	3.5403256	0.9730996	0.7058669	0.8579298
4	0.0000000	4.4717331	1.1338901	1.0004917	0.8942266
5	0.0000000	4.6802595	1.2733653	1.1484580	0.9671792
6	0.0000000	3.8930051	1.3516876	1.2417254	0.9157134
7	0.0000000	4.182113	1.508955	1.384268	1.034762

Tabela 11 – Valores DRP(%) dos resultados de VNS INV-SW

Kick	min	max	mean	median	std
3	0.0000000	3.4036068	1.0416965	0.8746317	0.8449003
4	0.0000000	3.6068072	1.1763916	1.0227332	0.9523702
5	0.0000000	3.6068072	1.1967284	1.1583074	0.8429292
6	0.0000000	3.8820455	1.2232310	1.0294560	0.9426279
7	0.0000000	3.8633819	1.3913490	1.3788740	0.8823738

Tabela 12 – Valores DRP(%) dos resultados de VNS INV-INS

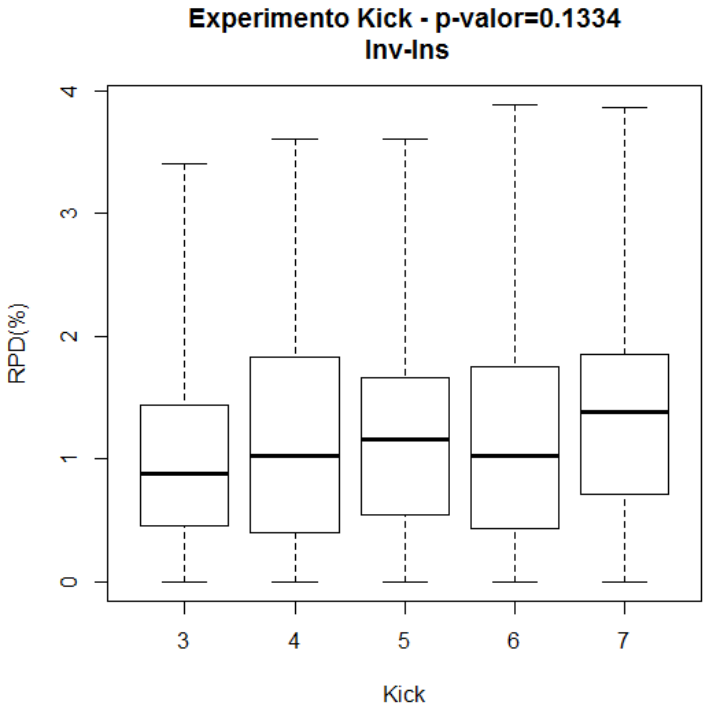

Figura 15 – Boxplots VNS INV-INS

4 Experimento Final

Todos os algoritmos de VNS e VND foram implementados em C++, utilizando o sistema GNU/Linux e copilado por GNU C++ Compiler. O experimento foi executado em 72 instâncias de Ruben Ruiz (SDST10, 100, 125, 50 - 32, 33, 34, 42, 43, 44, 52, 53, 54, 62, 63, 64, 72, 73, 74, 82, 83, 84). Cinco execuções independentes sobre cada instância. Foi utilizado a equação de DRP levando em conta os resultados obtidos de cada VNS com as instâncias. O critério de parada utilizado foi o tempo limite de processamento, utilizando o cálculo usado em Dubois-Lacoste, Pagnozzi e Stützle (2016), onde o tempo limite fica: $k \times (m/2) \times 60$. Utilizado como valor do kick o menor valor mediano para cada VNS, caso o valor tenha sido insignificativo, setou-se o valor do kick para 3. Foi se utilizado os testes de Kruskal-Wallis diante dos resultados testes de distribuição normal e assim identificação de melhor VNS com critério de makespan. Sendo exibido através do gráfico 16, se observa que o melhor VNS, com a menor mediana, foi o VNS_SW-INS, com um p-valor de $2,2 \times 10^{-16}$, onde o mesmo utiliza o VND com duas buscas locais: Melhor-Troca e Melhor-Inserção. Através da tabela 13 se pode observar os resultados dos testes de cada VNS, mostrando seu mínimo, máximo, média, mediana e o desvio

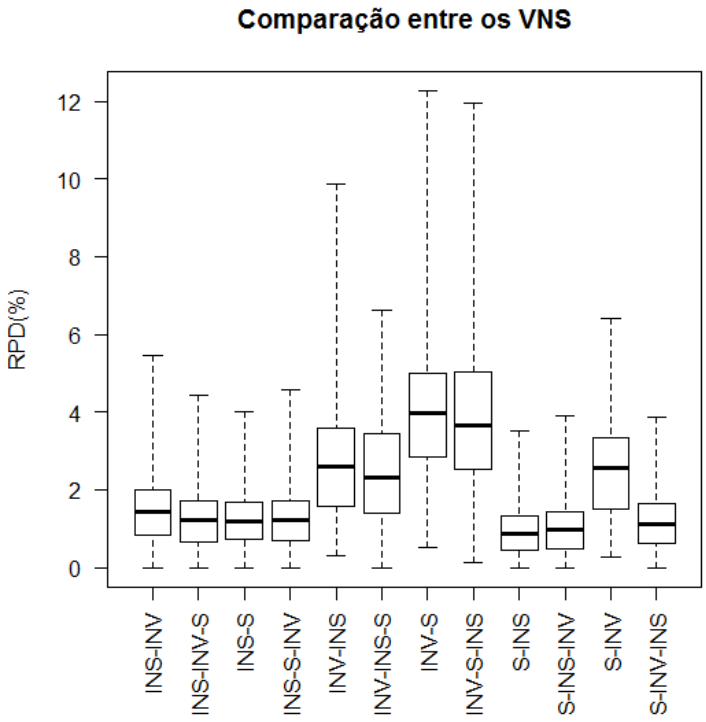

Figura 16 – Boxplots de todos os VNS

DRP%	min	max	mean	median	std
INS-INV	0.000000	5.474268	1.521996	1.446160	0.905044
INS-INV-SW	0.000000	4.4399596	1.2762721	1.2105846	0.8126209
INS-SW	0.000000	4.0181863	1.2456520	1.1786189	0.7392949
INS-SW-INV	0.000000	4.5711477	1.2676753	1.2270988	0.8075847
INV-INS	0.3071253	9.8795773	2.6877141	2.5999460	1.4313673
INV-INS-SW	0.000000	6.636501	2.483594	2.325742	1.378628
INV-SW	0.5266854	12.2634554	4.0336733	3.9641136	1.7708932
INV-SW-INS	0.1363327	11.9479312	3.9193244	3.6690670	1.9191712
SW-INS	0.000000	3.5065590	0.9289756	0.8688159	0.6363031
SW-INS-INV	0.000000	3.9075940	1.0238038	0.9590901	0.6867982
SW-INV	0.2620545	6.4266405	2.6114308	2.5752380	1.3149127
SW-INV-INS	0.000000	3.8849647	1.1869190	1.1017289	0.7325047

Tabela 13 – Valores DRP(%) de cada VNS, utilizando o Kruskal-Wallis

padrão

5 Conclusões Finais

Neste trabalho foi apresentado um estudo estrutural do problema de programação em ambiente flowshop permutacional com tempos de preparação(setup) das tarefas em cada máquina separados do tempo de processamento. Também apresentou-se as propriedades da meta-heurística VNS (*Variable Neighbourhood Search*) e sua variação o VND (*Variable Neighbourhood Descent*) para estudo e aplicação em proposta.

Novos algoritmos foram propostos para o referido problema, com base nas propriedades das meta-heurísticas mencionadas, com o critério de minimizar o makespan. Para as avaliações de adequação dos algoritmos e parâmetros foi efetuado experimentos computacionais e testes de Kruskal-Wallis sobre instâncias com n = 20, 50, 100 e 200 tarefas de Ruben Ruiz[1], sendo cinco execuções independentes para cada instância.

Como conclusão, utilizando-se da equação de DRP(%) sobre cada resultado de novo algoritmo proposto, se obteve valores significativos, no qual os mesmos mostram-se comprometedores e adequados para ambiente em questão. Validando assim a proposta para aceitação.

Em destaque, diante dos resultados dos algoritmos, está o VNS com o VND de duas buscas locais sequencialmente: Melhor-Troca e Melhor-Inserção, que obteve menor mediana diante dos outros algoritmos propostos, sendo o mais adequado entre os outros.

Outros estudos envolvendo o VNS_SW-INS serão desenvolvidos, podendo levar à métodos híbridos, e mais testes abrangidos.

Lembrando que a solução proposta é uma solução aproximada (heurística) do problema de programação da produção, no ambiente flowshop.

[1] (http://soa.iti.es)

Referências

BRIMBERG, J. et al. Improvements and comparison of heuristics for solving the multisource weber problem. *Operations Research*, n. 444–460, 2000. Citado na página 24.

CAMPBELL, H. G.; DUDEK, R. A.; SMITH, M. L. A heuristic algorithm for n job, m machine sequencing problem. *Management Science*, p. 630–637, 1970. Citado na página 13.

CHENG, T. C. E.; GUPTA, J. N. D.; WANG, G. A review of flowshop scheduling research with setup times. *Production and Operations Management*, Wiley-Blackwell, v. 9, n. 3, p. 262–282, set. 2000. Citado na página 17.

CONOVER, W. J. Statistics of the kolmogorov-smirnov type. *Practical nonparametric statistics*, p. 428–473, 1999. Citado na página 36.

COOPIN, B. *ARTIFICIAL INTELLIGENCE ILLUMINATED*. [S.l.]: Jones & Bartlett Publishers, Inc., 2004. I. Citado na página 22.

DANNENBRING, D. G. An evaluation of flow-shop sequencing heuristics. *Management Science*, v. 23, p. 1174–1182, 1977. Citado na página 13.

DONG, X.; HUANG, H.; CHEN, P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *School of Computer and Information Technology*, 2008. Citado 2 vezes nas páginas 15 e 37.

DOWDY, S.; WEARDEN, S.; CHILKO, D. *Statistics for Research*. [S.l.]: John Wiley & Sons, Inc., 2004. Citado na página 37.

DUBOIS-LACOSTE, J.; PAGNOZZI, F.; STüTZLE, T. An iterate d gree dy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Elsevier Ltd.*, 2016. Citado 2 vezes nas páginas 35 e 49.

GAREY, M. R.; JOHNSON, D. S.; SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 1, n. 2, p. 117–129, may 1976. Citado na página 13.

GLOVER, F.; KOCHENBERGER, G.; GARY, A. *Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science*. [S.l.]: Kluwer Academic Publishers, 2003. Citado na página 21.

GUPTA, J. N. D. A review of flowshop scheduling research. *Disaggregation: Problems in Manufacturing and Service Organizations*, p. 359 –364, 1979. Citado na página 13.

HANSEN, P.; MLADENOVIć, N. J-means: a new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognition*, v. 34, n. 405–413, 2001. Citado na página 24.

HANSEN, P.; MLADENOVIć, N.; PéREZ, J. Variable neighbourhood search: methods and applications. *Springer Science+Business Media, LLC*, v. 41, n. 19, p. 192003, out. 2009. Citado 4 vezes nas páginas 21, 22, 23 e 32.

HEJAZI, S. R.; SAGHAFIAN, S. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, v. 43, p. 2895–2929, 2005. Citado na página 13.

JOHNSON, S. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1954. Citado na página 13.

MCGILL, R.; TUKEY, J. W.; LARSEN, W. A. Variations of box plots. *The American Statistician*, 1978. Citado na página 36.

MLADENOVIć, N. A variable neighborhood algorithm—a new metaheuristic for combinatorial optimization. *Optimization days*, n. 112, 1995. Citado na página 21.

OSMAN, I.; POTTS, C. Simulated annealing for permutation flow-shop scheduling. *Elsevier*, 1989. Citado na página 13.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2017. Disponível em: <https://www.R-project.org/>. Citado na página 36.

RUIZA, R.; STüTZLEB, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Elsevier*, v. 117, n. 3, p. 2033–2049, mar. 2007. Citado na página 36.

TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, v. 64, p. 278–285, 1993. Citado na página 13.