Algoritmos baseados na meta-heurística VNS aplicados ao Problema de Escalonamento de Motoristas de Ônibus

Allainclair Flausino dos Santos
Universidade Estadual de Maringá
Avenida Colombo, 5790 – 87020-900 – Bloco C56 – Maringá – PR – Brasil
allainclair@gmail.com

Ademir Aparecido Constantino
Universidade Estadual de Maringá
Avenida Colombo, 5790 – 87020-900 – Bloco C56 – Maringá – PR – Brasil
aaconstantino@uem.br

Wesley Romão
Universidade Estadual de Maringá
Avenida Colombo, 5790 – 87020-900 – Bloco C56 – Maringá – PR – Brasil
wesley.uem@gmail.com

RESUMO

Para que o sistema de transporte público seja prestado com qualidade é necessário atacar inúmeros problemas, e um deles é o Problema de Escalonamento de Motoristas de Ônibus (PEMO), o qual é NP-difícil. Este trabalho apresenta quatro algoritmos baseados na meta-heurística VNS para o PEMO. Os quatro algoritmos possuem uma fase construtiva em comum e uma fase melhorativa que os diferem, tal que na fase melhorativa, os métodos Processo de Corte e Recombinação (PCR) e *k-swap* foram aplicados. Os algoritmos foram avaliados ao serem aplicados em instâncias da cidade de Maringá, PR, Brasil. Na avaliação dos algoritmos propostos, foi constatado que todos são competitivos, pois eles geraram soluções melhores em relação à função de custo ao serem comparados à uma outra abordagem recente, tal que o melhor algoritmo obteve melhores soluções na faixa de 11% a 15%. Em relação a um limite inferior o melhor algoritmo ficou na faixa de 11% a 13% acima desse limite.

PALAVRAS CHAVE. Escalonamento, Motoristas, Ônibus, VNS, meta-heurística.

Tópicos: L&T - Logística e Transportes, MH - Metaheurísticas, SE - PO em Serviços.

ABSTRACT

For the public transport system being provided with quality, it is necessary to tackle numerous problems and one of them is the Bus Driver Schedule Problem (BDSP) which is NP-hard. This work presents four algorithms based on the VNS meta-heuristic for the BDSP. The four algorithms have a constructive common phase and an improved phase that differs them, such that in the improved phase, the Cut and Combine Process and k-swap were applied. The algorithms were evaluated applying them on instances of a Brazilian city. In the algorithm evaluations was found that all are competitive when they were compared with another recent approach, because they generated better solutions with respect to the cost function, such that the best algorithm obtained better solutions in the range of 11% to 15%. With respect to a lower bound the best algorithm got the range of 11% to 13% over this bound.

KEYWORDS. Bus, Driver, Schedule, VNS, meta-heuristic.

Topics: L & T - Logistics and Transport, MH - Metaheuristics, SE - OR in Services.

1. Introdução

O Problema de Escalonamento de Motoristas de Ônibus (PEMO) está inserido no contexto de serviços de transportes públicos prestados à comunidade. Esses serviços atingem uma grande parcela da população, diretamente e indiretamente, principalmente em cidades com alto índice populacional, as quais possuem diversos serviços de transporte público.

Para que o transporte público seja prestado com qualidade, existe o Planejamento do Sistema de Transporte. Já no contexto de transporte por meio de ônibus, esse planejamento tem como primeira fase o Projeto da Rede de Atendimento, que define rotas a serem seguidas pelos veículos de forma a atender uma demanda previamente estipulada; e a última fase é o Planejamento do Rodízio de Motoristas, o qual consiste na determinação do plano de trabalho de um motorista ao longo de um período, seja uma semana ou um mês de trabalho, por exemplo [Prata 2010].

O Escalonamento de Motoristas (EMO) é uma fase intermediária, a qual determina jornadas de trabalho para os motoristas dos ônibus, geralmente durante o período de um dia de trabalho. Dessa forma, o PEMO consiste em determinar uma escala de motoristas de custo mínimo para cobrir uma escala de veículos [De Leone et al. 2011]. O custo da escala de motoristas está relacionado diretamente aos motoristas de ônibus, os quais estão distribuidos em estações (depósitos, garagens, etc.) e são alocados aos veículos ao se resolver o PEMO.

O PEMO é NP-difícil [Fischetti et al. 1987], mesmo quando existem apenas restrições relacionadas a jornada de trabalho. Isso faz com que a solução ótima para o problema em instâncias de tamanho razoável seja intratável de se encontrar. Dessa forma, diferentes abordagens são aplicadas para encontrar soluções mínimas (não necessariamente ótimas) que satisfaçam as restrições do problema e portanto sejam viáveis.

Dentre as abordagens para se resolver o PEMO, existem em geral duas mais comuns, a primeira é baseada em heurísticas e a segunda usa modelos de programação matemática. Uma abordagem heurística resolve o problema com uma abordagem intuitiva: o problema é interpretado, analisado e estruturado de forma que um método computacional razoável seja obtido por meio desse processo. Métodos heurísticos geralmente conseguem resolver instâncias de maior porte em tempo aceitável em comparação aos métodos de programação matemática, estes geralmente têm ineficiência de tempo de execução para instâncias grande de porte.

Este trabalho consiste em mostrar e avaliar quatro métodos computacionais baseados na meta-heurística VNS que resolvem o PEMO. Os quatro métodos têm em comum duas fases: fase construtiva e fase de melhoramento. Na fase construtiva é gerada a solução inicial que serve como entrada para a fase de melhoramento.

A fase construtiva é idêntica em todos os quatro métodos propostos, ela consiste em resolver vários Problemas de Atribuição (PA) [Hillier e Lieberman 2014] , no qual são atribuídas tarefas para jornadas de motoristas. As jornadas são formadas em um processo iterativo e cada tarefa do conjunto total de uma iteração é designada para uma dessas jornadas já formadas, ou caso não haja possibilidade de atribuir uma tarefa para uma jornada, cria-se uma nova jornada de motorista a partir da tarefa em questão.

A fase de melhoramento consiste em aplicar variações da meta-heurística *Variable Neighborhood Search* (VNS) [Hansen et al. 2009]. Cada variação da meta-heurística utiliza diversas estruturas de vizinhanças baseadas nos métodos de geração de vizinhos: Processo de Corte e Recombinação (PCR), o qual faz um corte em uma jornada, separando-a em dois segmentos (anterior e posterior ao corte), esses dois segmentos são recombinados com segmentos de outras jornadas (as quais são cortadas no mesmo "ponto") para formar novas jornadas; e, também, no método *k-swap*, o qual dado o parâmetro k, delimita um intervalo dentro de uma jornada, para que o segmento dentro desse intervalo seja recombinado com outros segmentos de outras jornadas no mesmo intervalo delimitado.

A avaliação dos quatro algoritmos foi feita ao se comparar os resultados obtidos com um método recente [Sakiyama et al. 2014], também baseado na méta-heurística VNS. Todos os

quatro métodos foram superiores na minimização de custo da escala de motoristas, em especial o Algoritmo VNS 4, o que indica a eficácia dos quatro métodos avaliados. Essa avaliação também foi feita com um limite inferior, tal que o melhor algoritmo ficou por volta de 11% a 13% acima desse limite.

O trabalho segue na Seção 2 com definições pertinentes ao PEMO; na Seção 3 são apresentadas heurísticas de buscas locais e os métodos de geração de vizinhos PCR e *k-swap*; na Seção 4 são apresentados a meta-heurística VNS e o método VND; na Seção 5 os quatro algoritmos propostos são definidos; na Seção 6 é descrito como os experimentos foram conduzidos e, também, são mostrados os resultados obtidos em comparação com uma abordagem recente [Sakiyama et al. 2014]; na Seção 7 conclui-se o presente trabalho e indica-se trabalhos futuros.

2. Definição do Problema

Uma $viagem$ possui um momento (temporal) e local (geográfico) de início, e um momento e local de término. Dessa forma é possível definir uma viagem como um par ordenado da forma: $viagem{=}((local, tempo)_i, (local, tempo)_t)$. Os índices i e t indicam o início e término da viagem respectivamente [Huisman et al. 2005].

Uma sequência de viagens atribuídas a um veículo definem um *quadro de execução* (também chamado de *bloco*), e um conjunto de quadros de execução definem uma *escala de veículos*. O Problema de Escalonamento de Veículos (PEV) tem como objetivo encontrar uma escala de veículos de custo mínimo [Pepin et al. 2008].

Os quadros de execução são divididos em *oportunidades de troca*, cada uma dessas oportunidades é definida por um par ordenado $(local, hora)$ que indica um lugar no tempo e espaço em que uma troca de motorista pode ocorrer. O $local$ é um lugar geográfico – geralmente uma estação ou depósito de veículos – que também é chamado de *ponto de troca* [Tóth e Krész 2012].

Uma *tarefa* é determinada por duas oportunidades de troca consecutivas, a tarefa é unidade mínima que pode ser atribuída a um motorista. Uma *peça de trabalho* é definida como a sequência de tarefas sobre um quadro de execução que pode ser executada por apenas um motorista sem interrupção [Huisman et al. 2005]. As tarefas que são atribuídas ao mesmo motorista definem uma *jornada de motorista*, logo um conjunto de jornadas de motoristas definem uma *escala de motoristas*. Na Figura 1 tem-se ilustrado os elementos que foram definidos. Sumariamente são dois quadros de execução nos quais uma jornada de motorista começa no primeiro quadro e termina no segundo, com uma parada intermediária com duração de uma hora e meia.

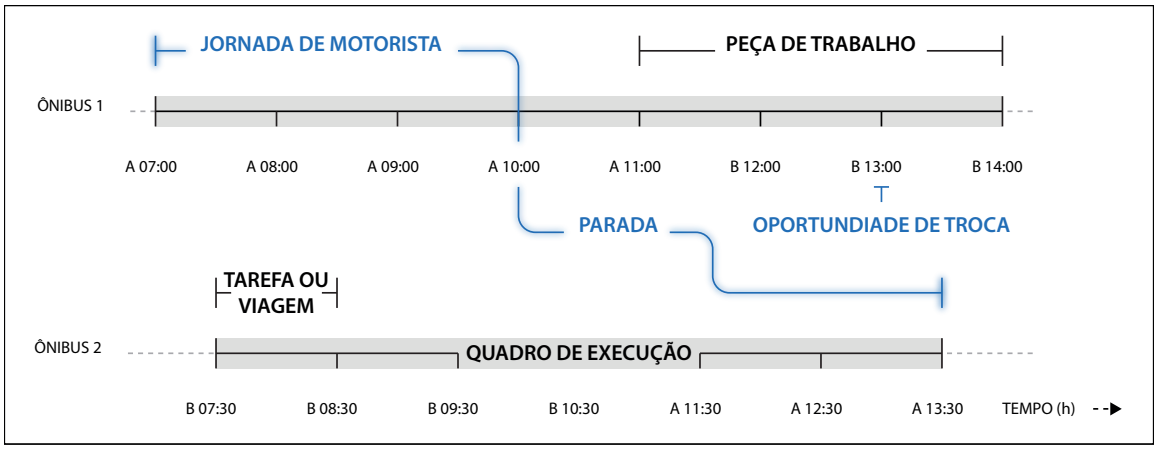

Figura 1: Alguns elementos fundamentais pertencentes ao contexto do PEMO.

Os métodos computacionais que resolvem o PEMO tem como função encontrar uma escala de motoristas de custo mínimo. Em uma abordagem de cobertura de conjuntos o PEMO pode ser formulado da seguinte forma:

$$\text{minimizar} \sum_{j=1}^{n} c_j x_j \tag{1}$$

$$\text{sujeito a} \sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1, 2, ..., m \tag{2}$$

Os significados dos elementos das Somatórias 1 e 2 são: n representa o número de jornadas de motoristas; m o número de tarefas; x_j é uma variável binária, se $x_j = 1$ a jornada j é selecionada na solução final, caso contrário $x_j = 0$; c_j representa o custo da jornada j; a_{ij} também é uma variável binária, caso $a_{ij} = 1$, a tarefa i é coberta pela jornada j, caso contrário $a_{ij} = 0$.

O custo envolvido em uma escala de motoristas é geralmente a combinação de custos fixos, como salários dos motoristas, em adição com custos variáveis, como pagamento de horas extras. Uma escala de motoristas é viável se cada tarefa é atribuída para apenas uma jornada de motorista, e cada jornada de motorista é uma sequência de tarefas as quais podem ser executadas por um motorista do ponto de vista físico e legal [Huisman et al. 2005], ou seja, as restrições impostas ao PEMO devem ser satisfeitas para que uma escala de motoristas seja viável.

De modo particular, isso indica que cada jornada de motorista deve satisfazer um conjunto de restrições espaciais e temporais. Algumas das restrições impostas ao PEMO são: jornada de trabalho, tempo máximo de horas extras, duração máxima de trabalho contínuo, intervalo concedido mínimo e máximo para descanso e tempo máximo da jornada de motorista o qual inclui o tempo trabalhado (pago) e intervalos para descanso.

3. Heurísticas de Buscas Locais, PCR e *k-swap*

Uma heurística de busca local consiste em escolher uma solução inicial x, encontrar uma direção de descida a partir de x dentro de uma vizinhança $N(x)$. Se não existe nenhuma direção de decida, a heurística para; caso contrário a mesma é iterada [Hansen et al. 2009]. Geralmente o processo *Best Improvement* é usado, o qual está descrito no Algoritmo 1. Uma alternativa é usar a heurística *First Improvement* (Algoritmo 2), pois a heurística *Best Improvement* pode consumir muito tempo.

Algorithm 1 best-improvement (x)

1: **repeat**
2: $\quad x' \leftarrow x$
3: $\quad x \leftarrow \textit{arg min}_{y \in N(x)} f(y)$
4: **until** $f(x) \geq f(x')$
5: **return** x

Algorithm 2 first-improvement (x)

1: **repeat**
2: $\quad x' \leftarrow x; i \leftarrow 0$
3: $\quad$ **repeat**
4: $\quad\quad i \leftarrow i + 1$
5: $\quad\quad x \leftarrow \textit{arg min}\{f(x), f(x_i)\}, x_i \in N(x)$
6: $\quad$ **until** $f(x) < f(x_i)$ **or** $i = |N(x)|$
7: **until** $f(x) \geq f(x')$
8: **return** x

O Procedimento de Corte e Recombinação (PCR) gera vizinhos a partir de cortes na solução de entrada. O corte pode acontecer em qualquer local da solução (desde que seja no intervalo de início e fim). Dado um local de corte, é feito a recombinação de todos os elementos

da solução nesse local. Para exemplificar, em um caso particular do EMO, a solução de entrada é uma escala de motoristas e o corte pode ser feito em um horário qualquer dentro do intervalo da escala. Dessa forma haverá uma recombinação das jornadas de motoristas naquele corte (geração de vizinhos) da seguinte forma: todos os segmentos das jornadas antes do horário de corte são recombinados com os segmentos das jornadas após o horário de corte, ou seja, forma-se uma vizinhança máxima de $|J| \times |J|$, tal que $|J|$ é a cardinalidade do conjunto de jornadas de motoristas da escala.

O procedimento *k-swap* é semelhante ao PCR, a diferença é que existem dois cortes: é feito um corte em um local da solução de entrada, e outro corte é feito a k posições em relação ao primeiro corte. O que é recombinado no *k-swap* são os segmentos internos delimitados pelos dois cortes. Um exemplo no contexto do EMO, semelhante ao PCR, um corte pode ser feito em um horário qualquer dentro do intervalo da escala, e outro corte pode ser feito a k minutos após esse primeiro corte. Dessa forma, os segmentos internos delimitados pelos cortes são recombinados entre as jornadas e a quantidade de vizinhos máximo também é de $|J| \times |J|$, dado que cada segmento pode ser recombinado com todas as jornadas de motoristas.

Existem duas formas de percorrer uma instância para aplicar os procedimentos PCR e *k-swap* (ou qualquer outro método compatível): a primeira é percorrer do início para o fim (*forward*) e a segunda é do fim para o início (*backward*).

4. VND e VNS

O método *Variable Neighbourhood Descent* (VND) tem como característica a mudança de vizinhaças, a qual é feita de maneira determinística. O Algoritmo 3 mostra o método VND.

A meta-heurística *Variable Neighborhood Search* (VNS) é um método de uso geral para resolução de problemas de otimização global e combinatória. A ideia básica consiste em mudar sistematicamente a vizinhança combinada com uma busca local [Hansen et al. 2009]. O Algoritmo 4 mostra o VNS básico, nele tem-se a função *shake* que a partir da solução corrente x é gerado uma nova solução x' em um processo que envolve aleatoriedade (diversificação). A função *busca-local* aplica um procedimento de busca local qualquer para gerar a nova solução x''. O Algoritmo 4 é a base para os algoritmos propostos.

Algorithm 3 vnd (x, k'_{max})

```
repeat
    k ← 1
    repeat
        x' ← arg min_{y∈N_k(x)} f(y) {encontre o melhor vizinho em N_k(x)}
        if x' < x then
            x ← x'
            k ← 1
        else
            k ← k + 1
        end if
    until k = k'_max
until nenhuma melhora é obtida
return x
```

5. Algoritmos Propostos

Os quatro algoritmos propostos são compostos por uma fase construtiva determinística, comum entre os quatro; e uma fase melhorativa, que os diferem. A fase construtiva gera uma solução inicial viável que serve de entrada para as quatro fases melhorativas, as quais são baseadas na meta-heurística VNS.

O primeiro passo da fase construtiva dos algoritmos propostos é um processo de geração de camadas, o qual tem como entrada um conjunto de tarefas T e retorna uma sequência de camadas

Algorithm 4 vns (x, k_{max})

$k \leftarrow 1$
repeat
  $x' \leftarrow$ *shake*(x, k)
  $x'' \leftarrow$ *busca-local*(x')
  if $x'' < x'$ **then**
    $x \leftarrow x''$
    $k \leftarrow 1$
  else
    $k \leftarrow k + 1$
  end if
until $k = k_{max}$
return x

C. O propósito do algoritmo é criar uma sequência de camadas contendo tarefas, essa sequência de camadas direciona a formação da escala de motoristas inicial. Para isso, as camadas são ordenadas para que tarefas em camadas posteriores possam ter a chance de ser sequenciadas com tarefas de camadas anteriores, se o sequenciamento não acontecer, uma nova jornada de motorista é criada pelo PA. Dessa forma, dada uma camada qualquer K, que é um elemento da sequência C, cada tarefa que está na camada $K + 1$ pode ser sequenciada com pelo menos uma tarefa da camada K. Esse processo é definido no Algoritmo 5 onde a função *não-pode-sequenciar* retorna verdadeiro se uma tarefa t não pode ser sequenciada com pelo menos uma tarefa da camada $C[i - 1]$ (que é a camada $K - 1$) indicada pelo algoritmo.

Algorithm 5 gerar-camadas (T)

$C \leftarrow$ *sequência vazia*
$K \leftarrow \emptyset$
insere-ao-fim(C, K)
while T *não está vazio* **do**
  $t \leftarrow$ *remover-mais-cedo-em*(T)
  $i \leftarrow$ *tamanho*(C)
  while $i > 0$ **and** *não-pode-sequenciar*$(C[i - 1], t)$ **do**
    $i \leftarrow i - 1$
  end while
  if $i = tamanho(C)$ **then**
    $K \leftarrow \emptyset$
    insere-ao-fim(C, K)
  end if
  $C[i] \leftarrow C[i] \cup t$
end while
return C

O segundo passo da fase construtiva é percorrer a sequência de camadas C para formar um conjunto de jornadas J, ou seja, a saída do Algoritmo 5 é a entrada para o Algoritmo 6 . Para isso, para cada camada K é resolvido um PA que atribui as tarefas da camada para uma jornada existente, ou caso haja inviabilidade de atribuição cria-se uma nova jornada. Cada elemento a_{ij} da matriz de custos $A = [a_{ij}]$ (a qual é a entrada para o PA), indica o custo de se atribuir a tarefa de índice j para a jornada de índice i (caso ambas existam), especialmente, essa matriz é formada pela composição de quatro blocos descrito a seguir.

Bloco 1 Para $i < |J|$ e $j < |K|$, $a_{ij} = g(i, j)$; caso haja inviabilidade de atribuição, $a_{ij} = \infty$.

$g(i, j)$ é uma função que calcula um valor dado o sequenciamento da jornada i com a tarefa j. $g(i, j)$ pode ser a própria função objetivo da nova jornada gerada ao se sequenciar a tarefa j à jornada i. Porém, modificações da função objetivo ao serem aplicada em $g(i, j)$ podem gerar soluções iniciais melhores. Posteriormente na Seção 6.1 é explicado qual valor exato $g(i, j)$ recebe no caso particular estudado nesse trabalho;

Bloco 2 Para $i < |J|$ e $j \geq |K|$, $a_{ij} = obj(i)$; caso haja inviabilidade de atribuição, $a_{ij} = \infty$.

$obj(i)$ é a função objetivo da jornada indexada por i, porém como explicado no Bloco 1, essa função pode ser modificada para encontrar melhores soluções;

Bloco 3 Para $i \geq |J|$ e $j < |K|$, $a_{ij} = c_{nj}$, (c_{nj} é o valor dado ao se criar uma jornada não vazia); caso seja **viável** designar a tarefa j para sua jornada correspondente do Bloco 1, $a_{ij} = \infty$. O índice da jornada correspondente em questão é obtido pela operação: $|J| - i$;

Bloco 3 Para $i \geq |J|$ e $j \geq |K|$, $a_{ij} = 0$, já que atribuir tarefas inexistentes para jornadas inexistentes não possuem custo.

Após a definição da matriz de custos um algoritmo para resolver o PA retorna uma solução, dessa forma tem-se para quais jornadas as tarefas de cada camada devem ser designadas. O método utilizado neste trabalho para resolver o PA foi o proposto por [Carpaneto e Toth 1987] e o proceso completo para geração da solução inicial é descrito no Algoritmo 6. A função *novas-jornadas* retorna as novas jornadas provenientes da resolução do PA, isto é, basicamente é verificado se o par designado i, j (jornada, tarefa) é do Bloco 1 (tarefa j designada para jornada i) ou do Bloco 3 (nova jornada); caso contrário não haverá modificação do conjunto corrente de jornadas J.

Algorithm 6 solução-inicial (C)

1: $J \leftarrow \emptyset$
2: **for** K *na ordem da sequência* C **do**
3: $M \leftarrow$ *construir-matriz-de-custos*(J, K)
4: $S \leftarrow$ *resolver-problema-de-atribuição*(M)
5: $J \leftarrow J \cup$ *novas-jornadas*(J, S)
6: **end for**
7: **return** J

Para a fase de melhoramento, quatro variações da meta-heurística VNS foram aplicadas no resultado gerado pela fase construtiva. O algoritmo base do VNS é mostrado no Algoritmo 4 e suas variações são descritas nas Seções 5.1 a 5.4 a seguir.

5.1. VNS 1

A primeira abordagem para a fase melhorativa é um VNS baseado no Algoritmo 4 que tem como argumento $k_{max} = 4$ para a função *shake*: para $k = 1$ o procedimento busca recombinar aleatoriamente com o procedimento PCR 15% de jornadas de motoristas relativas ao tamanho da escala de motoristas, o corte sempre é feito aleatoriamente dentro do intervalo da escala e são escolhidas apenas jornadas viáveis para compor a nova escala; para $k = 2$ o procedimento é análogo ao primeiro porém é usado o *k-swap*; para $k = 3$ o procedimento é análogo ao primeiro, porém busca-se a recombinação de 30% de jornadas de motoristas relativas ao tamanho da escala; e para $k = 4$ o procedimento é análogo ao segundo ($k = 2$), porém busca-se a recombinação de 30% de jornadas de motoristas relativas ao tamanho da escala.

Na função *busca-local* do Algoritmo 4 VNS, é aplicado o VND descrito no Algoritmo 3 com $k'_{max} = 36$. As 36 formas de gerar vizinhos são descritas a seguir.

Como descrito ao final da seção 3, existem duas formas de percorrer uma solução x', *forward* e *backward*, logo é possível percorrer uma escala de motoristas dessas duas formas. Além disso, é possível gerar vizinhos ao aplicar métodos semelhantes as heurísticas *First Improvement* e *Best Improvement* ao percorrer a escala de motoristas. Em adição, foi criado um método semelhante ao *First Improvement*, o qual percorre a escala, mas não para ao encontrar uma solução melhor que a corrente, ao invés de parar e retornar a primeira melhor solução, essa solução se torna a solução corrente e o processo continua até o final (como é feito no *Best Improvement*), esse método neste artigo é chamado de *Continuous Improvement*.

Em cada passo de recombinação de jornadas, seja por PCR ou *k-swap*, é formado uma matriz de custos $|J| \times |J|$. Essa matriz de custos é resolvida com o PA, tal que com a solução do PA é resolvido quais jornadas devem ser recombinadas e gerar uma nova instância de escala de motoristas.

Portanto, as 36 formas de gerar vizinhos ($k'_{max} = 36$) são a combinação das formas de percorrer a escala (*forward* e *backward*), regras para gerar vizinhos (*First*, *Best* e *Continuous Improvement*) e métodos de recombinar as jornadas (um PCR e 1,2,3,4,5-*swap*, aplicados nessa ordem). Logo temos $k'_{max} = 2 \times 3 \times 6 = 36$. A ordem de aplicação das regras para gerar vizinhos é *Continuous*, *Best* e *First Improvement* e a ordem para percorrer a vizinhança é *backward* e *forward*.

Portanto tem-se a ordem para k': 1. PCR, *Continuous Improvement*, *backward*; 2. PCR, *Continuous Improvement*, *forward*; 3. PCR, *Best Improvement*, *backward*; 4. PCR, *Best Improvement*, *forward*; 5. PCR, *First Improvement*, *backward*; 6. PCR, *First Improvement*, *forward*. O mesmo padrão segue para 1,2,3,4 e 5-*swap*.

5.2. VNS 2

A segunda abordagem para a fase melhorativa também é um VNS baseado no Algoritmo 4 e a função *shake* é a mesma descrita na Seção 5.1. Já a função *busca-local* usa uma abordagem diferente. São aplicadas as 36 formas de gerar vizinhos (descritas na Seção 5.1) e a forma que retornar o melhor resultado dentre os 36 é usada repetidas vezes até não encontrar mais melhoria (descida). Ao final desse processo de descida, o resultado é retornado da função *busca-local*.

5.3. VNS 3

A terceira abordagem segue uma linha semelhante ao que é descrito na Seção 5.2, a mudança está apenas em como a iteração da *busca-local* é feita: o melhor resultado dentre os 36 gerados pelas estruturas de vizinhança é tomado como solução para a próxima iteração, e esse processo para quando não existe melhora de uma iteração para outra.

5.4. VNS 4

A quarta abordagem segue uma linha semelhante ao que é descrito nas Seções 5.2 e 5.3, a mudança está também nas iterações feitas na *busca-local*: agora as 36 estruturas de vizinhanças são executadas iterativamente em paralelo até atingirem um mínimo local, ou seja, a solução da aplicação de uma estrutura de vizinhança, serve como entrada para ela mesma em uma nova iteração, até que esse processo não gere melhoria (processo de descida com apenas uma estrutura de vizinhança). Dessa forma, ao final, a melhor solução dentre os 36 mínimos locais é tomada como solução para uma próxima iteração. Quando não existir melhoria de uma iteração para outra, a função *busca-local* retorna a melhor solução encontrada.

6. Experimentos Computacionais

Os experimentos foram executados no sistema operacional Debian 8 em computadores Intel Xeon de 3.5GHz, com 30 GB de memória RAM. Existem quatro instâncias de escalas de veículos com dois pontos de troca (garagem e terminal). Os quatro algoritmos propostos e o algoritmo de comparação – o qual foi implementado – foram aplicados nessas instâncias tal que a maior instância possui 2313 viagens e a menor possui 1253 viagens. Como os algoritmos não são determinísticos, cada algoritmo executou trinta vezes cada uma das instâncias. A seguir os parâmetros, regras e resultados dos experimentos são descritos.

6.1. Parâmetros e Regras

Os algoritmos propostos foram implementados com as seguintes regras de trabalho:

1. A jornada de trabalho normal do motorista tem duração de 7 (sete) horas e 20 (vinte) minutos. Caso a jornada realizada seja menor, ainda assim é pago conforme o tempo mínimo pago por uma jornada de trabalho, ou seja, 7 (sete) horas e 20 (vinte) minutos;

2. O tempo máximo de horas extras trabalhadas não deve exceder 2 (duas) horas, ou seja, a jornada com horas extras poderá ter duração máxima de até 9 (nove) horas e 20 (vinte) minutos trabalhados;

3. O valor das horas extras trabalhadas recebe um adicional de 50% sobre o valor pago pela hora normal;

4. A duração máxima do trabalho contínuo é de 6 (seis) horas. Quando a duração do trabalho ultrapassar 6 (seis) horas, deve ser concedido um intervalo para descanso;

5. O intervalo para descanso deverá ter um tempo mínimo de 1 (uma) hora e 30 (trinta) minutos e máximo de 5 (cinco) horas;

6. A possibilidade de sequenciamento de duas tarefas t_i, t_j é definida por duas restrições: tempo de término t_i é menor que tempo de início de t_j e o local de término de t_i deve ser o mesmo que o local de início de t_j. Jornadas de motorista que não satisfaçam essas restrições são consideradas inviáveis;

7. A extensão máxima da jornada, ao considerar o tempo trabalhado mais o tempo de folga, deve ser no máximo 13 (treze) horas. Isso ocorre, pois o intervalo entre o fim da jornada de um dia e o início da jornada do outro dia deve ser de no mínimo 11 (onze) horas, pois é considerado que um motorista poderá executar a mesma jornada no dia seguinte, logo tem-se que a duração da jornada total não deve ser maior do que 13 (treze) horas;

8. A hora paga no período noturno, entre as 22 (vinte e duas) horas de um dia e às 5 (cinco) horas do dia seguinte, possui duração de 52,5 minutos;

9. Sobre as horas pagas no horário noturno haverá um adicional de 20% sobre o custo da jornada normal.

A função para obtenção do custo $c(x)$ de uma jornada x é descrita como segue:

$$c(x) = d(x) + n(x) + e(x) \tag{3}$$

As funções $d(x)$, $n(x)$ e $e(x)$ da Equação 3 retornam o tempo trabalhado (tempo ocioso está incluso) durante o dia, noite e adicional de horas extras respectivamente. Não é contabilizado as paradas de descanso pois não geram custo, diferente do tempo ocioso, onde o tempo de trabalho é contabilizado e o mesmo não está executando uma tarefa.

Os algoritmos propostos utilizam a seguinte função objetivo $f(x)$ para uma jornada x:

$$f(x) = c(x), \text{se as restrições 2, 4, 5, 6 e 7 são satisfeitas ou } f(x) = \infty, \text{caso contrário} \tag{4}$$

Para o Bloco 1 descrito na Seção 3, $g(i, j)$ é semelhante à $f(x)$, tal que i representa a jornada corrente e j a nova tarefa a ser sequenciada, porém $g(i, j)$ ignora a condição 1, ou seja, $g(i, j)$ pode ter valor menor que 440 minutos (sete horas e vinte minutos). O mesmo acontece para a função $obj(x)$ do Bloco 2 (Seção 3), isto é, $obj(x)$ é uma função que pode retornar um valor menor

que o mínimo de $f(x)$. Essas modificações foram escolhidas pois geraram uma solução incial (fase construtiva descrita na Seção 5) melhor que a aplicação da função objetivo $f(x)$ original.

Os locais de corte (nos quais as jornadas são recombinadas) são determinados pela demanda de viagens no horizonte de tempo determinado pelas viagens e, também, pela quantidade de camadas geradas na fase construtiva (Seção 5). Isto é, são gerados gerados sempre $|C|$ locais de corte e os cortes são feitos em intervalos de tempo menores quando a densidade de tarefas é maior em um determinado período do dia (na hora do *rush* por exemplo).

O limite inferior usado nesse trabalho é o mesmo usado em [Constantino et al. 2007].

6.2. Resultados

Os resultados obtidos dos algoritmos propostos estão dispostos nas Tabelas 1, 2 e 3. A Tabela 1 contêm a soma das funções de custo $c(x)$ e objetivo $f(x)$ das jornada de motorista (somas representadas for $C(x)$ e $F(x)$ respectivamente). A Tabela 2 contêm a cardinalidade da jornada ($|J|$) e o tempo de execução (tempo) resultantes da aplicação dos algoritmos, e a disposição de valores é a mesma que a Tabela 1. A Tabela 3 faz um comparativo da função de custo $C(x)$ de cada algoritmo com o limite inferior li, e a disposição de valores é a mesma que as Tabelas 1 e 2.

Existem três valores obtidos das execuções para cada instância: o valor mínimo (min), máximo (max), e a média de todos valores (med). As instâncias (coluna Inst) da Tabela 1 e 2 indicam quantas tarefas são necessárias para formar a escala de motoristas. Os quatro algoritmos descritos nas Seções 5.1, 5.2, 5.3 e 5.4 são indicados nas colunas VNS1, VNS2, VNS3 e VNS4 respectivamente, e a coluna SAK indica o algoritmo de comparação [Sakiyama et al. 2014].

De acordo com a Tabela 1, verifica-se que os melhores resultados de custo $C(x)$ são do Algortimo VNS 4, pois obteve índices de melhora na faixa de 11% à 15% ao ser comparado com o Algoritmo SAK [Sakiyama et al. 2014] e o Algoritmo VNS 4 também é melhor em relação ao limite inferior, o qual obteve valores na faixa de 11% a 13% acima desse limite, como pode ser visto na Tabela 3. Porém o mesmo tem os piores resultados no tempo de execução de acordo com a Tabela 2. Isso pode ser justificado pois o algoritmo precisa encontrar um mínimo local para cada uma das 36 estruturas de vizinhanças citadas na Seção 5.1, dessa forma, somente após esses mínimos locais serem encontrados os 36 resultados são comparados para que uma nova iteração da busca local inicie. Uma maneira simples (que não foi implementada nesse momento) de melhorar o tempo de execução dos algoritmos VNS 2, 3 e 4 é paralelizar os processos dentro da função *busca-local* (explanados nas Seções 5.2, 5.3 e 5.4), pois partes desses processos são independentes e dessa forma podem ser paralelizados.

Todos os quatro algoritmos propostos geram resultados de custo $C(x)$ melhor que o Algoritmo de comparação [Sakiyama et al. 2014] e o Algoritmo mais rápido é o VNS 1, o qual gera bons resultados, pois seus resultados comparados com o Algoritmo VNS 4 é cerca de 3% a 4% maior para a função $C(x)$.

Tabela 1: Resultados da somas da função de custo $c(x)$ e função objetivo $f(x)$ dos algoritmos propostos.

Inst		**VNS1**		**VNS2**		**VNS3**		**VNS4**		**SAK**	
		$C(x)$	$F(x)$	$C(x)$	$F(x)$	$C(x)$	$F(x)$	$C(x)$	$F(x)$	$C(x)$	$F(x)$
2313	min	154440	154440	155760	155760	153120	153120	**149745**	**149745**	165998	166824
	max	154440	154440	157960	157960	154440	154440	**150035**	**150035**	167223	167609
	med	154440	154440	157116	157116	153853	153853	**149960**	**149960**	166510	167116
2010	min	135080	135080	138600	138600	136840	136840	**130726**	**130726**	145735	146455
	max	135520	135520	139920	139920	138160	138160	**132130**	**132130**	147914	148460
	med	135476	135476	139141	139141	137328	137328	**131555**	**131555**	147170	147653
1517	min	103400	103400	103400	103400	103400	103400	**99160**	**99160**	111836	112260
	max	104280	104280	103400	103400	103400	103400	**100435**	**100435**	114049	114128
	med	103913	103913	103400	103400	103400	103400	**99851**	**99851**	112733	113230
1253	min	87120	87120	88880	88880	86680	86680	**81007**	**81007**	94210	94297
	max	88000	88000	89760	89760	87120	87120	**82003**	**82003**	94529	94529
	med	87725	87725	89408	89408	86973	86973	**81573**	**81573**	94469	94456

Tabela 2: Número de jornadas de motoristas e tempo de execução em minutos

Inst		**VNS1**		**VNS2**		**VNS3**		**VNS4**		**SAK**	
		$\|J\|$	tempo	$\|J\|$	tempo	$\|J\|$	tempo	$\|J\|$	tempo	$\|J\|$	tempo
2313	min	351	**265**	354	489	348	855	**340**	18737	377	16212
	max	351	**468**	359	2132	351	2774	**342**	20565	380	19987
	med	351	**333**	356	1039	349	1624	**341**	19610	378	18743
2010	min	307	**269**	315	503	311	776	**294**	7003	331	1299
	max	308	**516**	318	1331	314	1768	**296**	13078	336	6045
	med	308	**319**	316	942	312	1317	**295**	11570	334	4578
1517	min	235	**191**	235	343	235	601	**225**	4565	254	915
	max	237	**360**	235	1106	235	1430	**225**	12973	259	1937
	med	236	**246**	235	816	235	1259	**225**	9753	256	1275
1253	min	198	**82**	202	257	197	457	**186**	4078	214	617
	max	200	**154**	204	1078	198	1296	**186**	13450	215	1570
	med	199	**110**	203	647	198	1035	**186**	8792	214	1017

Tabela 3: Comparação dos Algoritmos com o limite inferior

Inst		**VNS1**	**VNS2**	**VNS3**	**VNS4**	**SAK**
		$C(x)/li - 1$	$C(x)/li - 1$	$C(x)/li - 1$	$C(x)/li - 1$	$C(x)/li - 1$
2313	min	0,1717	0,1817	0,1617	0,1356	0,2594
limite inferior:	max	0,1717	0,1817	0,1717	0,1383	0,2687
131800	med	0,1717	0,1817	0,1674	0,1377	0,2641
2010	min	0,1642	0,1946	0,1794	0,1267	0,2561
limite inferior:	max	0,1680	0,2060	0,1908	0,1388	0,2749
116019	med	0,1677	0,1994	0,1835	0,1339	0,2684
1517	min	0,1593	0,1593	0,1593	0,1117	0,2538
limite inferior:	max	0,1691	0,1593	0,1593	0,1260	0,2787
89191	med	0,1644	0,1593	0,1593	0,1189	0,2639
1253	min	0,2056	0,2299	0,1995	0,1210	0,3037
limite inferior:	max	0,2178	0,2421	0,2056	0,1348	0,3081
72261	med	0,2140	0,2372	0,2035	0,1288	0,3059

7. Conclusão

Este trabalho apresentou quatro algoritmos baseados na meta-heurística VNS para resolver o PEMO, e comparou os resultados obtidos com um algoritmo recente proposto para fins de comparação [Sakiyama et al. 2014]. Todos os quatro algoritmos propostos foram mais eficazes que o algoritmo de comparação, e dentre os quatro algoritmos, o Algoritmo VNS 4 mostrou-se o mais eficaz na obtenção de uma escala de motoristas de baixo custo. Já o Algoritmo VNS 1 é uma alternativa eficiente para encontrar uma boa solução dentre os quatro algoritmos propostos.

Com esses resultados é verificado indícios da eficácia, para o PEMO, de se usar a meta-heurística VNS em conjunto com o VND como busca local e, também, com as três abordagens de busca local apresentados neste trabalho, com ênfase no método VNS 4, o qual obteve os melhores valores de minimização de custo de escala de motoristas. Portanto, este trabalho contribui com soluções computacionais para um (PEMO) dos inúmeros problemas envolvidos no contexto dos serviços de transporte públicos que devem ser pesquisados.

É necessário aplicar esses quatro algoritmos em outras instâncias de escala de motoristas, além de compará-los com outras abordagens computacionais, para que os indícios da eficácia sejam corroborados por outros trabalhos.

Em trabalhos futuros, é possível diminuir a quantidade de estruturas de vizinhança sem perda de qualidade na função de custo, para que os algoritmos diminuam o tempo de execução (principalmente o Algoritmo 4). Para isso, pode ser feito um estudo de quais são as estruturas de vizinhanças mais promissoras em minimizar o custo da escala de motoristas. Uma outra ideia é aplicar as quatro buscas locais (ou variações delas) apresentas nesse trabalho em outras meta-heurísticas que suportem essas buscas-locais, como por exemplo um *Simulated Annealing* com uma busca em múltiplas vizinhanças.

Agradecimentos

Agradecemos à Fundação Araucária do Paraná, à CAPES e ao CNPq (processo 306754/2015-0) pelo suporte financeiro.

Referências

Carpaneto, G. e Toth, P. (1987). Primal-dual algrorithms for the assignment problem. *Discrete Applied Mathematics*, 18:137–153.

Constantino, A. A., Calvi, R., Araujo, S. A. d. e Neto, C. F. X. d. M. (2007). Algoritmo Baseado em Grafo Multipartido para Escalonamento de Pessoal em Empresa de Transporte. *Simpósio Brasileiro de Pesquisa Operacional*, 39:1759–1770.

De Leone, R., Festa, P. e Marchitto, E. (2011). A bus driver scheduling problem: a new mathematical model and a GRASP approximate solution. *International Transactions in Operational Research*, 17:707–727.

Fischetti, M., Martello, S. e Toth, P. (1987). The fixed job schedule problem with spread-time constraints. *Operations Research*, p. 849–858.

Hansen, P., Mladenović, N. e Moreno Pérez, J. A. (2009). Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175:367–407.

Hillier, F. S. e Lieberman, G. J. (2014). *Introduction to Operations Research*. McGraw-Hill Education, 10 edition.

Huisman, D., Freling, R. e Wagelmans, A. P. M. (2005). Multiple-Depot Integrated Vehicle and Crew Scheduling. *Transportation Science*, 39:491–502.

Pepin, A.-S., Desaulniers, G., Hertz, A. e Huisman, D. (2008). A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, 12:17–30.

Prata, B. D. A. (2010). Programação integrada de veículos e motoristas: uma visão geral. *Sistemas & Gestão*, 4:182–204.

Sakiyama, R. Z., Constantino, A. A. e Romão, W. (2014). Meta-heurística vns aplicada a um problema de planejamento operacional de transporte público. *Simpósio Brasileiro de Pesquisa Operacional*, 46:1632–1643.

Tóth, A. e Krész, M. (2012). An efficient solution approach for real-world driver scheduling problems in urban bus transportation. *Central European Journal of Operations Research*, 21:75–94.