UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ALLAINCLAIR FLAUSINO DOS SANTOS

# Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus

Maringá
2016

ALLAINCLAIR FLAUSINO DOS SANTOS

# Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Ademir Aparecido Constantino

Maringá
2016

Dados Internacionais de Catalogação na Publicação (CIP)
(Biblioteca Central - UEM, Maringá, PR, Brasil)

S237a

Santos, Allainclair Flausino dos
Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus / Allainclair Flausino dos Santos. -- Maringá, 2016.
89 f. : il. color., figs., tabs.

Orientador: Prof. Dr. Ademir Aparecido Constantino.
Coorientadora: Prof.ª Dr.ª nome da orientadora.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2016.

1. Escalonamento - Motoristas de ônibus - Meta-heurística. 2. Tripulação. 3. VNS. Problema de escalonamento de motoristas de ônibus.
I. Constantino, Ademir Aparecido, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD 21.ed.003.3

ECSL

# FOLHA DE APROVAÇÃO

ALLAINCLAIR FLAUSINO DOS SANTOS

**Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Ademir Aparecido Constantino
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Wesley Romão
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Silvio Alexandre de Araujo
Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP-SJRP

Aprovada em: 16 de dezembro de 2016.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

# AGRADECIMENTOS

Agradeço primeiramente toda minha família.

Meu orientador professor Dr. Ademir Aparecido Constantino por todo trajeto que foi iniciado em 2014.

Todos professores do Departamento de Informática – UEM, os quais já conheço desde 2010.

A banca examinadora composta pelos professores Dr. Wesley Romão e Dr. Silvio Alexandre de Araujo.

Inês, principalmente pelas informações e orientações sobre questões do Programa de Pós-graduação em Ciência da Computação.

## Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus


### RESUMO

Para que o sistema de transporte público seja prestado com qualidade é necessário investigar e resolver inúmeros problemas, e dentre os problemas computacionais, existe o Problema de Escalonamento de Motoristas de Ônibus (PEMO), o qual é NP-difícil. Este trabalho apresenta quatro novos algoritmos da meta-heurística VNS para o PEMO, um GVNS e três VNS denominados adaptativos. 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. 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 proposto obteve melhores soluções na faixa de 4% a 14%. E em relação a um limite inferior o melhor algoritmo ficou na faixa de 12% a 25% acima desse limite.



**Palavras-chave:** Problema de Escalonamento de Motoristas Ônibus. Problema de Escalonamento de Tripulação. VNS. Meta-heurística.

# Algorithms based on the VNS meta-heuristic applied on the Bus Driver Schedule Problem


## *ABSTRACT*

For the public transport system being provided with quality, it is need to tackle numerous problems and one of them is the Bus Driver Schedule Problem (BDSP) which is NP-hard. This work presents four new algorithms of the VNS meta-heuristic for the BDSP, one GVNS and three VNS named adaptatives. 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 Maringá, PR, Brazil 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 4% to 14%. With respect to a lower bound the best algorithm got the range of 12% to 25% over this bound.



***Keywords:*** Bus Driver Scheduling Problem. Crew Scheduling Problem. VNS. Metaheuristic

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE SIGLAS E ABREVIATURAS

**EMO**: Escalonamento de Motorista de Ônibus
**GRASP**: *Greedy Randomized Adaptive Search Procedure*
**GVNS**: *General Variable Neighbourhood Search*
**ILS**: *Iterated Local Serach*
**LI**: Limite Inferior
**PA**: Problema de Atribuição
**PCR**: Processo de Corte e Recombinação
**PEMO**: Problema de Escalonamento de Motorista de Ônibus
**PET**: Problema de Escalonamento Tripulações ou Tripulantes
**PEV**: Problema de Escalonamento de Veículos
**RVND**: *Reduced Variable Neighbourhood Descent*
**VLNS**: *Very Large-scale Neighbourhood Serach*
**VND**: *Variable Neighbourhood Descent*
**VNS**: *Variable Neighborhood Search*

# SUMÁRIO

# 1

# Introdução

O Problema de Escalonamento de Motoristas de Ônibus (PEMO), também conhecido como Problema de Escalonamento de Tripulação (PET), é pesquisado desde 1960 para que modelos computacionais e programas sejam desenvolvidos para resolver esse problema. O 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.

Uma das vantagens de se utilizar o transporte público é reduzir a quantidade de automóveis nas vias, pois o trânsito desses veículos em grande escala, principalmente em metrópoles, tornou-se um problema amplamente conhecido como congestionamento. Uma das causas desse problema pode ser explicada pelo fato dos carros ocuparem muito espaço para transportar poucos passageiros, este espaço chega a 70% das vias públicas, porém os carros transportam apenas 30% dos passageiros. Dessa forma, este trabalho aborda o serviço de Transporte Público por meio de ônibus, os quais são responsáveis por cerca de 87% das viagens diárias em comparação com todas as outras alternativas de transporte público no Brasil, segundo à Associação Nacional das Empresas de Transportes Urbanos (2016).

No Brasil, o poder público pode prestar o serviço de transporte público diretamente ou contratar empresas especializadas para realizá-lo por meio de concessão ou permissão. O poder público também precisa construir vias e terminais, organizar linhas e horários, implantar e manter pontos de parada, definir tarifas e fiscalizar as empresas que operam esse sistema. Algumas das responsabilidades das empresas contratadas são: manter os veículos conservados, cumprir as ordens das entidades públicas, contratar e capacitar

motoristas e cobradores. Os principais custos envolvidos, principalmente para essas empresas do sistema de transporte, são mão de obra e combustível, somente esses dois chegam a mais de 50% dos custos totais do serviço. Existem também os custos de impostos e veículos os quais juntos chegam a cerca de 40% do total, e assim restam cerca 7% de custos em outras atividades. Esses dados são apresentados na figura 1.1 (Associação Nacional das Empresas de Transportes Urbanos, 2016).

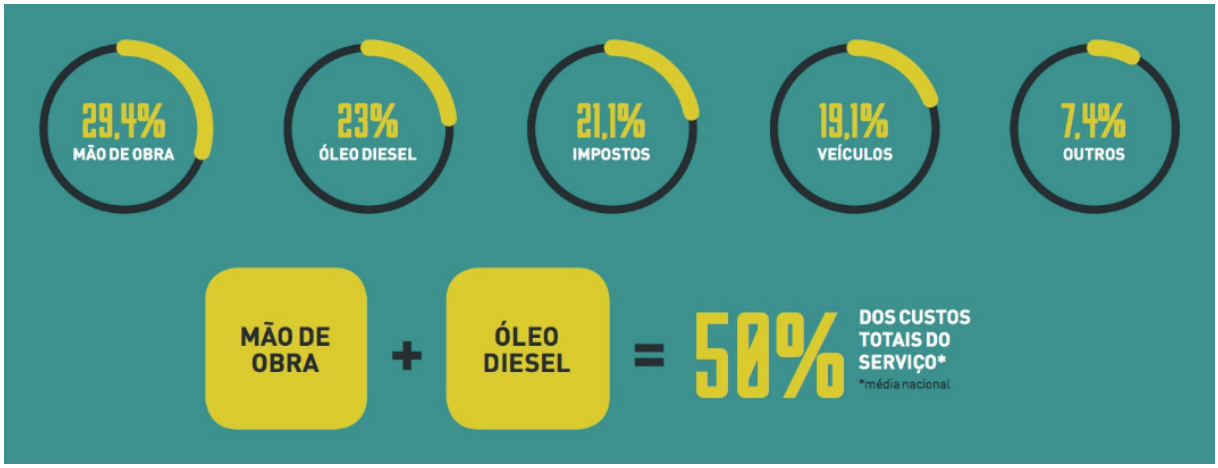


**Figura 1.1:** Principais custos envolvidos para uma empresa do sistema de transporte. Fonte: Associação Nacional das Empresas de Transportes Urbanos (2016).

Para que o transporte público seja prestado com qualidade, existe o planejamento do sistema de transporte. Esse planejamento é composto pelas fases a seguir.

**Projeto da rede de atendimento:** define rotas a serem seguidas pelos veículos de forma a atender uma demanda previamente estipulada;

**Planejamento das linhas:** consiste na definição dos veículos para as rotas, e também das respectivas frequências para percorrerem os trajetos com origem e destino, para o atendimento da demanda estipulada;

**Planejamento dos pontos de parada:** definem-se os pontos intermediários no trajeto de origem e destino o qual foi definido na fase de planejamento das linhas, os embarques e desembarques de passageiros são feitos nesses pontos intermediários;

**Tabela de horários:** transforma rotas e frequências em viagens com local e hora, referentes às origens e destinos;

**Escalonamento dos veículos:** consiste na atribuição de veículos para executarem as viagens definidas na tabela de horários;

**Escalonamento de motoristas:** consiste na alocação dos motoristas aos veículos, os quais irão efetuar as viagens planejadas, com isso gera-se uma escala de motoristas;

**Planejamento do rodízio de motoristas:** consiste na determinação do plano de trabalho de um motorista ao longo de um período, como por exemplo uma semana ou um mês de trabalho (Prata, 2010).

A figura 1.2 resume esse processo do planejamento do sistema público de transporte onde as setas tracejadas indicam um possível retorno (*feedback*) para a fase anterior ou entre fases quaisquer. O processo denotado na figura 1.2, apesar de ter uma característica sequencial, pode ter partes resolvidas de maneira integrada. A separação em partes ajuda a abstrair o sistema como um todo e também modularizar os problemas envolvidos.

No Escalonamento de Motorista de Ônibus (EMO) as jornadas de trabalho que são determinadas para os motoristas de ônibus, geralmente tem duração 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 (Leone et al., 2010). O custo da escala de motoristas está relacionado diretamente aos motoristas de ônibus – os quais estão distribuídos em estações, depósitos ou garagens – que 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 dessa forma sejam soluções 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, diferente dos métodos de programação matemática, os quais geralmente têm ineficiência de tempo de execução para instâncias de grande porte. Métodos baseados em programação matemática geralmente tem maior eficácia na redução de custos.

Antes de se resolver o PEMO, é necessário resolver o Problema de Escalonamento de Veículos (PEV). Esse problema tem como entrada um conjunto de viagens preestabelecidas e é necessário encontrar uma escala de veículos (ônibus) de custo mínimo, de modo que cada viagem deve ser executada por apenas um veículo (Pepin et al., 2008). A escala

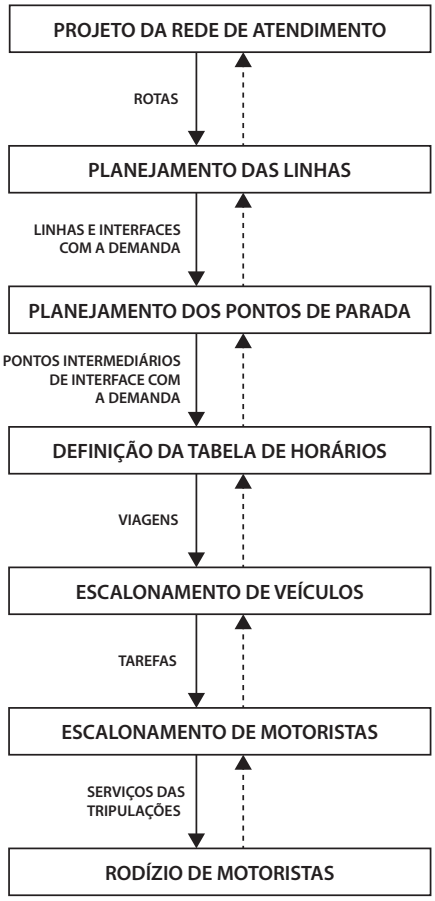


**Figura 1.2:** Planejamento do transporte público. Adaptado de Prata (2010).

de veículos é um conjunto finito de quadros de execução. Cada quadro é uma sequência de viagens que um ônibus deve executar, de forma que cada quadro é iniciado em um local geográfico e geralmente é finalizado ao retornar ao mesmo local de partida. Esse local pode ser um depósito, garagem ou algo do gênero em que o veículo deve permanecer.

## 1.1 Motivação e justificativa

Devido a importância na área da computação dos problemas de escalonamento no segmento de transporte público, salvo que essa importância vem em consequência da complexidade computacional de resolução, o desafio é encontrar metodologias computacionais (modelos e algoritmos) que permitam solucionar de forma efetiva o PEMO.

Ao encontrar-se essas metodologias, é possível gerar sistemas computacionais para a automação de problemas do mundo real.

A complexidade computacional está diretamente relacionada com a categorização de problemas no conjunto de problemas NP-Difíceis, o que impossibilita, do ponto de vista prático, encontrar soluções ótimas para instâncias de cardinalidade razoável. Por conseguinte, geralmente são construídos métodos heurísticos para resolver esses problemas em instâncias de maior porte, de modo que esses métodos têm o objetivo de encontrar a melhor solução possível (a mais próxima da solução ótima) em um tempo aceitável.

Além disso, existem os custos envolvidos com motoristas e ônibus para a empresa de transporte público, pois a mão de obra é um dos recursos mais custosos, como descrito anteriormente nesse capítulo. Portanto o PEMO deve ser investigado e resolvido de maneira efetiva para que esses custos sejam minimizados em tempo aceitável.

Não foram encontrados trabalhos na literatura em relação aos quatro métodos propostos aplicados no PEMO, o que indica ser uma importante lacuna que foi e deve ser explorada no contexto do PEMO. Com os resultados dos experimentos a partir dos algoritmos propostos, espera-se contribuir com novas metodologias computacionais para resolver o problema em questão com eficácia.

## 1.2 Objetivos e contribuições

Este trabalho consiste em mostrar e avaliar quatro novos algoritmos que usam a metaheurística VNS para resolver o PEMO. A partir de um desses algoritmos propostos, foram feitas duas variações para avaliar uma modificação na função objetivo que penaliza trocas de linhas.

Os quatro métodos e variações têm em comum duas fases: fase construtiva e fase de melhoramento. Na fase construtiva é gerada uma solução inicial viável, a qual serve de entrada para a Fase de Melhoramento.

### 1.2.1 Fase construtiva

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 existe um conjunto de tarefas, e cada tarefa desse conjunto deve ser designada para uma jornada de motorista.

As jornadas são formadas em um processo iterativo, de modo que cada tarefa do conjunto de uma iteração é designada para uma dessas jornadas já formadas; caso não

haja possibilidade de atribuir uma tarefa para uma jornada, cria-se uma nova jornada de motorista a partir da tarefa em questão.

Essa fase construtiva foi baseada no trabalho de Sakiyama et al. (2014) com modificações no algoritmo de geração de camadas e também na geração dos valores da matriz de custos do PA. A modificação proposta da fase construtiva e os resultados obtidos são mostrados nas Seções 4.1 e 5.3 respectivamente.

### 1.2.2 Fase de melhoramento

A fase de melhoramento consiste em aplicar a meta-heurística *Variable Neighborhood Search* (VNS) (Hansen et al., 2009) com variações no procedimento de busca local. Todas as variações de busca local utilizam os procedimentos PCR e *k-swap*.

**Processo de corte e recombinação (PCR):** faz um corte na escala de motoristas. Dado um momento temporal no horizonte de tempo da escala de motoristas, um corte é determinado pela secção das jornadas de motorista em dois conjuntos distintos nesse momento temporal. Ou seja, as jornadas de motorista são separadas em dois conjuntos de segmentos (anteriores e posteriores ao corte). Na medida que esses segmentos são recombinados entre si, pois são cortadas no mesmo "ponto", são formadas novas jornadas.

***k-swap*:** dado um parâmetro $k$ que delimita um intervalo dentro de uma escala de motoristas, é gerado um conjunto de segmentos delimitados pelo intervalo, tal que esses segmentos são recombinados entre si para gerar uma nova escala de motoristas.

As quatro variações de busca local do algoritmo VNS são:

**Variable Neighbourhood Descent (VND):** uma busca que muda a vizinhança de forma determinística. O VND como busca local do VNS é o algoritmo conhecido como *General Variable Neighbourhood Search*(GVNS);

**Busca adaptativa 1:** dada uma solução $x$ com um parâmetro $m$, $k'_{max}$ formas de explorar vizinhanças são executas iterativamente $m$ vezes sobre $x$. A melhor forma de exploração, dado seu resultado após as $m$ execuções, é escolhida para ser executada iterativamente até encontrar seu mínimo;

**Busca adaptativa 2:** dada uma solução $x$, $k'_{max}$ formas de explorar vizinhanças executam uma iteração sobre $x$ (como se $m = 1$ da busca adaptativa 1). A melhor solução encontrada é atribuída em $x$ e o processo é repetido até que não exista mais melhorias;

**Busca adaptativa 3:** dada uma solução $x$, $k'_{max}$ formas de explorar vizinhanças são executas até encontrarem as suas soluções mínimas respectivas. A menor solução dentre todas as soluções mínimas é atribuída em $x$ e uma nova iteração acontece. O processo para quando não existe mais melhoria.

As buscas locais VND e busca adaptativa 1 (chamada de VND-A), já foram aplicadas em outros problemas computacionais, em Geiger et al. (2011) é feito uma comparação dessas duas buscas, além de uma outra busca local que escolhe a forma de explorar vizinhança de forma aleatória (chamada de VND-R). Já as buscas adaptativas 2 e 3 são novas formas de exploração de vizinhança baseadas na ideia da busca adaptativa 1.

Com isso foram propostos quatro VNS que usam o VND, busca adaptativa 1, 2 e 3 respectivamente, os quais foram denominados de VNS 1, VNS 2, VNS 3 e VNS 4. Também foram propostas duas variações do algoritmo VNS 1, chamados de VNS 1 BI e VNS 1 BI P.

A avaliação desses quatro algoritmos propostos e as duas variações foi feita ao comparar-se os resultados obtidos neste trabalho com um método recente proposto que obteve bons resultado em Sakiyama et al. (2014), também baseado na meta-heurística VNS denominado de VNS-5-5.

Os quatro algoritmos propostos e as duas variações foram mais eficazes na minimização de custo da escala de motoristas, em especial o algoritmo VNS 4 (seção 4.4, VNS com busca adaptativa 3), o qual obteve os melhores resultados na minimização da função de custo. Isso indica a eficácia dos quatro métodos avaliados.

A melhora do algoritmo VNS 4 chegou a ser de 14% em comparação com o algoritmo de Sakiyama et al. (2014) para a função de custo. Apenas na menor instância o algoritmo VNS 4 foi inferior que o algoritmo de SAK. Em relação a um limite inferior, o algoritmo VNS 4 ficou na faixa de 11% (instâncias maiores) a 25% (instâncias menores) acima desse limite.

Já o algoritmo VNS 1 e sua variação VNS 1 BI mostraram-se como uma alternativa eficiente para o problema, pois dentre todas as propostas, em geral, esses dois algoritmos ficaram alternando a segunda melhor posição em relação a função de custo (atrás do VNS 4). Além disso, esses dois algoritmos são os mais eficientes dentre todos.

## 1.3 Organização do texto

O texto está organizado como segue: no capítulo 2 o PEMO é apresentado com elementos pertencente a esse contexto, e trabalhos relacionados são citados; no capítulo 3 é feita uma

revisão de literatura dos conceitos e métodos computacionais pertinentes a este trabalho; no capítulo 4 são mostrados e explicados todos algoritmos propostos desse trabalho; no capítulo 5 é mostrado como os experimentos foram conduzidos e os resultados obtidos após a aplicação dos algoritmos; no capítulo 6 este trabalho é sintetizado com informações sobre toda a pesquisa feita e sugestões para trabalhos futuros.

# 2

# O Problema de Escalonamento de Motoristas de Ônibus

Alguns elementos pertencentes ao contexto do PEMO são definidos a seguir.

Uma *viagem* possui um local (geográfico) e momento (temporal) de início, e um local e momento de término. Dessa forma é possível definir uma viagem como um par ordenado descrito na equação 2.1.

$$viagem = ((local, tempo)_i, (local, tempo)_t) \tag{2.1}$$

Os índices $i$ e $t$ da equação 2.1 indicam o início e término de uma viagem respectivamente (Huisman et al., 2005).

Uma sequência de viagens atribuídas a um veículo definem um *quadro de execução* (também conhecido como *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 $(local, tempo)$ que indica um lugar no espaço e tempo 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, 2013).

Uma *tarefa* é determinada pelo sequenciamento de duas oportunidades de troca consecutivas. Portanto, uma tarefa tem uma oportunidade de troca de início com tempo

$t_i$ e término com tempo $t_t$ (análogo à viagem), dessa forma, uma tarefa também é definida como um par ordenado descrito na equação 2.2.

$$tarefa = ((local, tempo)_i, (local, tempo)_t) \tag{2.2}$$

Assim como em uma viagem, os índices $i$ e $t$ da equação 2.2 indicam o início e término da tarefa respectivamente. A tarefa é a unidade mínima que pode ser atribuída a um motorista, assim como a viagem é a unidade mínima que pode ser atribuída a um veículo.

Uma *peça de trabalho* é definida como uma sequência de tarefas sobre um quadro de execução, sem paradas, que pode ser executada por apenas um motorista sem interrupção, de acordo com Huisman et al. (2005); porém Leone et al. (2010) define uma peça de trabalho de maneira diferente: uma peça de trabalho é basicamente uma tarefa como foi defina, e dentro dos limites de tempo dessa peça de trabalho, existe a possibilidade de existir paradas, caso o local onde o veículo está parado não seja uma oportunidade de troca. Um exemplo de parada dentro de uma peça de trabalho, é uma parada em um bairro, tal qual o motorista fica ocioso por um determinado tempo caso seja necessário, e o bairro em questão não é uma oportunidade de troca. Já uma *parada* pode ser paga se for um tempo ocioso; ou não é paga se uma parada é para descanso.

Uma sequência de tarefas que são atribuídas ao mesmo motorista formam uma *jornada de motorista*, logo um conjunto de jornadas de motorista formam uma *escala de motoristas*. Na figura 2.1 tem-se ilustrados os elementos que foram definidos para uma escala de veículos, e na figura 2.2 tem-se ilustrado os elementos que foram definidos para uma escala de motoristas. Sumariamente as duas figuras são dois quadros de execução nos quais existem três jornadas de motorista. Uma dessas jornadas começa no primeiro quadro e termina no segundo, com uma parada de descanso intermediária com duração de uma hora e meia.

Os métodos computacionais que resolvem o PEMO têm como objetivo encontrar uma escala de motoristas de custo mínimo. Em uma abordagem de cobertura de conjuntos o PEMO pode ser formulado da seguinte maneira.

$$\textbf{Minimizar} \quad \sum_{j=1}^{n} c_j x_j \tag{2.3}$$

$$\textbf{Sujeito a} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, 2, ..., m \tag{2.4}$$

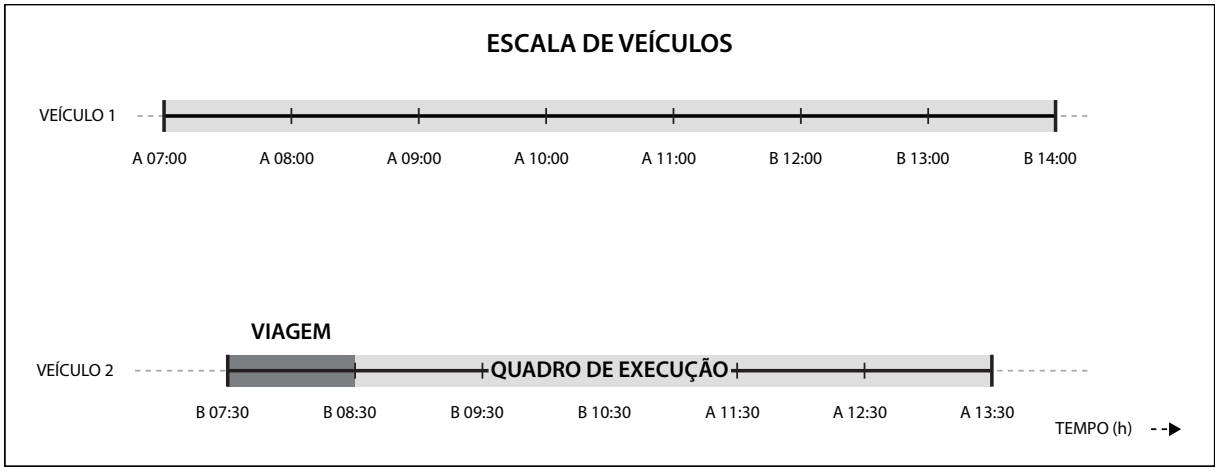


**Figura 2.1:** Alguns elementos fundamentais pertencentes à uma escala de veículos.

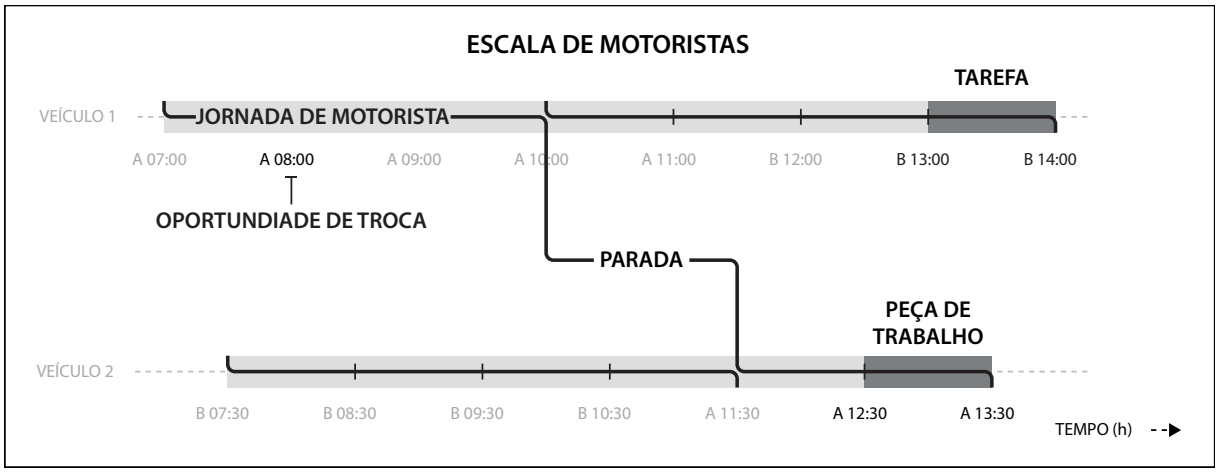


**Figura 2.2:** Alguns elementos fundamentais pertencentes à uma escala de motoristas.

Os significados dos termos dos somatórios 2.3 e 2.4 são: $n$ representa o número de jornadas de motorista, $m$ 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 mínimo e máximo para descanso e tempo máximo da jornada de motorista, que inclui o tempo trabalhado, tempo ocioso e intervalos para descanso.

Alguns trabalhos além de ter uma função objetivo para a minimização do custo total como a equação 2.3, também tem como objetivo minimizar a quantidade de jornadas de motorista, como é feito em Tóth e Krész (2013). Isto é, o parâmetro $n$ do somatório 2.3 também deve ser minimizado.

## 2.1 Métodos para resolver o PEMO

De modo geral, existem duas abordagens gerais para resolver o PEMO: programação matemática e heurística. Na programação matemática, o problema é modelado tal que sejam especificadas a função objetivo de minimização e restrições. Dessa forma, um resolvedor pode ler essas especificações como entrada, executar e gerar a solução exata. As abordagens mais usadas de programação matemática são a cobertura de conjuntos ou particionamento de conjuntos.

Na formulação de particionamento de conjuntos, cada tarefa é coberta apenas por uma jornada de motorista, já na formulação por cobertura de conjuntos é possível ter mais do que uma jornada de motorista cobrindo uma tarefa. Nesses dois modelos existem um conjunto de tarefas (linhas) que precisam ser cobertas, e um conjunto de jornadas de motorista viáveis predefinidas (colunas) que cobrem tarefas específicas (Portugal et al., 2008). A estratégia de geração de colunas é muito usada para resolver o problema, em Chen e Shen (2013) e Li et al. (2015) é usado essa abordagem.

A segunda forma de resolver do PEMO é por meio de métodos heurísticos, os quais são formulados a partir da análise do problema, e assim projetam-se algoritmos que resolvem esse problema de maneira determinística ou não determinística. Os métodos exatos de programação matemática e heurísticos para o PEMO também podem ser combinados para gerar métodos mistos.

Apesar da separação do PEV e PEMO para solucioná-los, esses dois podem ser resolvidos de maneira integrada. Dois trabalhos que modelam esses dois problemas de forma integrada são apresentados em Huisman et al. (2005) e Groot e Huisman (2008). Essa abordagem geralmente é ineficiente para instâncias de grande porte.

## 2.2 Trabalhos relacionados

No trabalho de Chen e Niu (2012) considera-se restrições de imparcialidade na criação de jornadas de motorista em relação ao tempo de execução de cada jornada. Isso é observado pois na criação de uma escala, jornadas com diferentes tamanhos podem ser criadas e, portanto alguns motoristas podem trabalhar mais do que outros. Logo, além de minimizar o custo das jornadas, nesse trabalho cria-se soluções com tempos homogêneos de trabalho para as jornadas de motorista. Nesse trabalho é apresentado uma Busca Tabu para resolver o PEMO.

Um trabalho que também usa o VNS para resolver o PEMO é apresentado em Ma et al. (2016). Esse trabalho usa como entrada quadros de execução com janelas de tempo. Uma *janela de tempo* é um período de tempo em que o motorista pode descansar, esse período de tempo está entre uma tarefa e outra, as quais pertencem a quadros de execução. Essa janela de tempo é dada em uma oportunidade de troca. É utilizado nesse trabalho um VNS básico com três estruturas de vizinhança chamadas de *Swap Two links*, *Swap Two pieces* e *Insert one piece*. Os experimentos foram conduzidos em instâncias de *Beijing Public Transport Group*.

Em Silva e Reis (2014) a meta-heurística *Iterated Local Search* (ILS) foi usada no PEMO. Essa proposta também foi comparada com trabalhos anteriores que usaram a meta-heurística VNS. As duas meta-heurísticas apresentadas (ILS e VNS) também usaram a técnica denominada *Very Large-scale Neighborhood Search* (VLNS) como procedimento de busca. As versões das meta-heurísticas foram aplicadas em dados reais de uma empresa de Belo Horizonte.

No trabalho de Leone et al. (2010) a meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) é usada em conjunto com estruturas de vizinhanças chamadas de *1-swap* e *Variable neighborhood swap*. Em De Leone et al. (2011) as meta-heurísticas GRASP, VNS e variações delas são aplicados no PEMO.

Em Calvi (2005) a proposta é separada em uma fase construtiva e uma fase melhorativa: na fase melhorativa, dois procedimentos denominados M1 e M2 são usados iterativamente. M1 é o PCR, tal que cada corte feito é modelado com uma matriz de custo do PA. Já o procedimento M2 faz cortes nas últimas tarefas de jornadas de motorista que estejam com horas extras, e assim resolver o PA para esse corte. A ideia desse procedimento é diminuir o custo da solução ao retirar tarefas das jornadas com horas extras e realocá-las em jornadas que estão com tempo ocioso.

# 3

# Fundamentação teórica

Neste capítulo é feito a revisão de alguns conceitos e métodos computacionais que são pertinentes ao trabalho, além disso, esses elementos servem de base para a proposta do trabalho ou são elementos pertencentes à proposta.

## 3.1 Problema de atribuição

O problema de tribuição (PA), também conhecido na literatura como problema de designação (*assignment problem*), é um tipo especial de problema de programação linear onde os atribuídos estão sendo indicados para realizar tarefas (Hillier e Lieberman, 2014). Alguns exemplos simples do PA são as situações de designar tarefas para máquinas, tarefas para funcionários ou funcionários para máquinas.

Para que essas aplicações citadas sejam válidas em um PA, é necessário que elas sejam formuladas de forma que sejam satisfeitas as seguintes suposições.

1. O número de atribuídos e o número de tarefas são os mesmos (denotado por $n$);
2. Cada atribuído é atribuído para apenas uma tarefa;
3. Cada tarefa é realizada por apenas um atribuído;
4. Existe um custo $c_{ij}$ associado com o atribuído $i$ $(i = 1, 2, ..., n)$ que realiza a tarefa $j$ $(j = 1, 2, ..., n)$;
5. O objetivo é determinar como todas $n$ atribuições deveriam ser feitas para minimizar o custo total.

As três primeiras suposições são bastante restritivas. Muitas aplicações em potencial geralmente não satisfazem essas restrições. Porém, frequentemente é possível reformular o problema para que as restrições sejam satisfeitas. Por exemplo, criar atribuídos fictícios ou tarefas fictícias, esses artifícios frequentemente podem ser o caminho para satisfazer essas restrições.

Qualquer problema que satisfaça todas essas suposições pode ser resolvido eficientemente por algoritmos específicos de PAs (Hillier e Lieberman, 2014).

O modelo matemático para o PA usa a seguinte variável de decisão:

$$x_{ij} = \begin{cases} 1 & \text{se o atribuído } i \text{ realiza a tarefa } j \\ 0 & \text{caso contrário} \end{cases} \tag{3.1}$$

Na definição da variável de decisão 3.1, $i = 1, 2, .., n$ e $j = 1, 2, .., n$, e como é notado, cada variável $x_{ij}$ é uma variável binária.

Seja $Z$ o custo total, o modelo do PA pode ser denotado como:

$$\textbf{Minimizar} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.2}$$

$$\textbf{Sujeito a} \quad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, ..., n \tag{3.3}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, ..., n \tag{3.4}$$

$$x_{ij} \geq 0, \quad \text{para todo } i \text{ e } j \tag{3.5}$$

$$x_{ij} \text{ é binária}, \quad \text{para todo } i \text{ e } j \tag{3.6}$$

As restrições 3.3 e 3.4 indicam que cada atribuído deve realizar apenas uma tarefa, enquanto que as restrições 3.5 e 3.6 indicam que cada tarefa deve ser realizada exatamente por um atribuído.

Um algoritmo conhecido na literatura para resolver o PA é o método húngaro desenvolvido por Kuhn (1955) na década de 50. Outro algoritmo – o qual é usado neste trabalho – para resolver o PA foi proposto por Carpaneto e Toth (1987), esse algoritmo tem complexidade $O(n^3)$, dado que a matriz de custo que representa o problema é de ordem $n$. De acordo com os resultados dos autores, esse algoritmo proposto se mostrou mais eficiente que o método húngaro na maioria das instâncias aplicadas.

## 3.2 Heurísticas de buscas locais

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 ela é iterada (Hansen et al., 2009). Geralmente $x' \in N(x)$ é obtido de $x$ ao executar uma mudança local, a qual pode ser chamado de *movimento* (dentro da solução $x$). Dessa forma, podemos definir uma estrutura de vizinhança a seguir.

**Definição 3.1 (Estrutura de Vizinhança)** *Seja $X$ o conjunto de todas as soluções viáveis, e ao considerar uma solução $x \in X$, uma estrutura de vizinhança associa a cada $x \in X$ uma vizinhança $N(x) \subseteq X$ da solução $x$ (Paula et al., 2006).*

Em problemas de otimização discreta, geralmente a definição 3.1 consiste de todas as soluções obtidas de $x$ por alguma modificação simples (Hansen e Mladenovic, 2003).

Vários critérios de parada para uma busca local podem ser definidos (Ma et al., 2016). Uma técnica simples de busca local pode ser descrita da seguinte forma:

1. Dado um conjunto $X$ de soluções viáveis, escolha uma solução inicial $x \in X$ e atribua $x_{best} = x$;
2. Obtenha uma solução $x' \in N(x)$ tal que $f(x') < f(x)$ para todas soluções pertencentes a $N(x)$;
3. Se $x'$ do passo 2 existir, então atribua $x = x'$ e $x_{best} = x'$ e volte ao passo 2, caso contrário pare o procedimento.

Geralmente o processo *best improvement* é usado (também conhecido como *Steepest Descent*) (Hansen e Mladenovic, 2003), 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.

**Algoritmo 1**

**function** BEST-IMPROVEMENT($x$)

```
1  repeat
2        x' ← x
3        x ← arg min_{y∈N(x)} f(y)
4  until f(x) ≥ f(x')
5  return  x
```

**Algoritmo 2**

**function** FIRST-IMPROVEMENT($x$)

```
1  repeat
2        x' ← x
3        i ← 0
4        repeat
5              i ← i + 1
6              x ← arg min{f(x), f(x_i)},    x_i ∈ N(x)
7        until f(x) < f(x_i) or i = |N(x)|
8  until f(x) ≥ f(x')
9  return  x
```

## 3.3 Processo de Corte e Recombinação (PCR)

Dado um índice $i$ de uma sequência $s$, um *corte* em $i$ nessa sequência determina dois segmentos distintos: $s_1$ anterior a $i$ e $s_2$ posterior a $i$. O elemento da sequência apontado por $i$ pode ser anexado no segmento anterior ou posterior, mas somente em um desses dois segmentos para que a consistência seja mantida.

Seja $S = \{s | s \text{ é uma } \textit{sequência}\}$ um conjunto finito. Dado um *corte* em $i$ que transpassa as sequências pertencentes a $S$, gerando dois conjuntos distintos $S_1$ e $S_2$, os quais contêm os segmentos das sequências originais de $S$, é possível recombinar $S_1$ e $S_2$ para gerar um novo conjunto $S'$.

Uma forma de se escolher uma das combinações para gerar $S'$ é resolver um PA, tal que cada linha $i$ da matriz de custos $C$ representa uma sequência $s_i \in S_1$ e cada coluna $j$ representa uma sequência de $s_j \in S_2$. Obtém-se uma matriz quadrada caso o corte gere uma bipartição perfeita do conjunto $S$. Caso a bipartição não seja perfeita, isto é, $|S_1| \neq |S_2|$, é possível criar tarefas ou atribuídos fictícios, como explicado na seção 3.1.

Cada elemento $c_{ij}$ da matriz de custos $C$ citada recebe um valor do tipo $c_{ij} = g(i, j)$, tal que $g(i, j)$ é um valor associado a um custo da recombinação da sequência apontada por $i$ com a sequência apontada por $j$. Caso o sequenciamento entre uma sequência $i$ com outra sequência $j$ não seja permitido (devido às restrições do problema, por exemplo), $c_{ij} = \infty$, para que $c_{ij}$ possivelmente não esteja na solução do PA. Esse processo é ilustrado na figura 3.1 e a estrutura de cada elemento $c_{ij}$ da matriz $C$ é mostrada na Equação 3.7.

$$c_{ij} = \begin{cases} \infty & \text{caso não exista possibilidade de sequenciamento da sequência } i \text{ com } j \\ g(i,j) & \text{caso contrário} \end{cases} \tag{3.7}$$

O processo de corte e recombinação (PCR) gera vizinhos a partir de $n$ cortes e recombinações sucessivas na solução de entrada $S$. Os locais de cortes podem ser feitos em qualquer local da solução (desde que seja no intervalo de início e fim), e o intervalo entre os cortes também pode variar de acordo com o valor de $n$.

Para exemplificar, em um caso particular no EMO, a solução de entrada $S$ é uma escala de motoristas, tal que cada $s \in S$ é uma jornada de motorista. Os cortes são dados uniformemente (com mesmos intervalos de tempo) do início da escala até o final dela.

Dessa forma, dado um dentre os $n$ cortes, nele haverá uma recombinação das jornadas de motorista 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 $n^2 = |J| \times |J|$, tal que $|J|$ é a cardinalidade do conjunto de jornadas de motorista da escala. $g(x)$ é uma função objetivo. Mais detalhes sobre parâmetros e algoritmos aplicados nesse trabalho para o PCR são mostrados na seção 4.3.1.

O algoritmo 3 define uma função PCR, mas esse algoritmo pode ser alterado para definir um critério de parada, caso a decisão seja não percorrer todos os locais de corte gerados, dessa forma, a solução corrente de quando o algoritmo decide parar é retornada. Também é possível definir se a solução $S$ deve ser substituída a cada iteração sobre $K$ (como está no algoritmo 3), ou se $S$ é mantido integro até o final, e dessa forma, são criadas soluções $S'$, tal que a melhor é retornada ao final das iterações, algo semelhante à abordagem heurística *best improvement*.

A função GERAR-LOCAIS-DE-CORTE($S$) do algoritmo 3 gera os locais $k \in K$ onde a solução corrente deve ser seccionada. os locais $k \in K$ são usados na função CORTAR($S, k$) para que seja gerado os dois conjuntos distintos $S_1$ e $S_2$ e assim a matriz de custos $C$

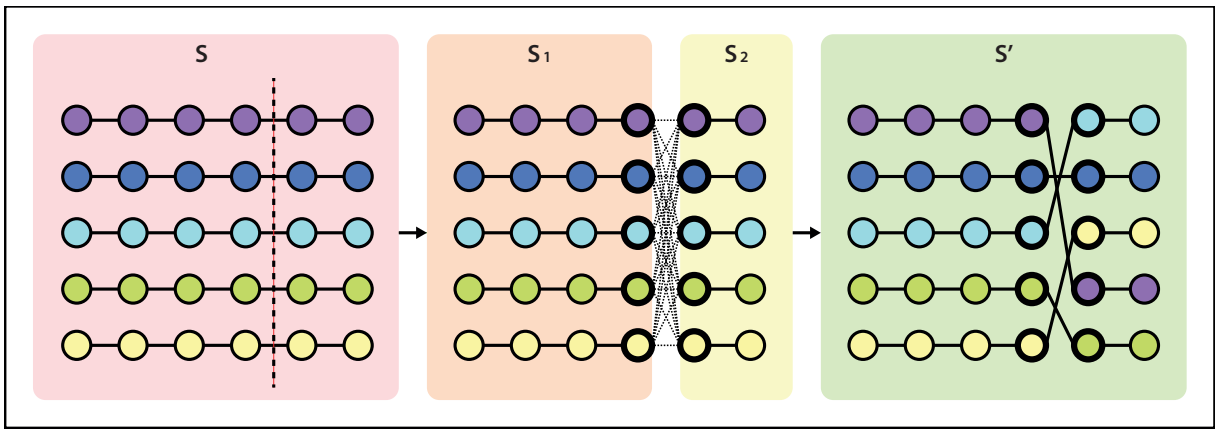


**Figura 3.1:** Ilustração de um corte do PCR, dado uma instância com cinco sequências para recombinar.

é criada e o PA resolvido. Ao final da iteração a resposta do PA é transformada em uma nova solução, e dessa forma a solução $S$ é atualizada. Quando todos elementos do conjunto $K$ são percorridos, o processo para e retorna a solução $S$ atualizada.

O trabalho de Constantino et al. (2007) utiliza o PCR referenciado como procedimento M1 e M2. O procedimento M1 é basicamente o PCR descrito nesta seção, já o procedimento M2 tem como propósito diminuir o custo da solução ao retirar tarefas (a última tarefa de cada jornada) das jornadas com horas extras e realocá-las em jornadas que estão com tempo ocioso.

Sakiyama et al. (2014) usa o PCR com dois critérios de parada: o primeiro percorre os cortes e guarda a melhor solução encontrada, retornando-a ao final do processo; o segundo percorre os cortes e substitui a solução corrente caso haja melhora, dessa forma uma nova solução corrente é atualizada caso haja melhora durante o percursos na vizinhança. Esses dois critérios são denominados de *first improvement* e *best improvement* nesse trabalho, porém este trabalho redefine esses dois critérios na seção 4.3.

**Algoritmo 3**

**function** PCR($S$)

```
1  K ← GERAR-LOCAIS-DE-CORTE(S)
2  for  k ∈ K  do
3       S1, S2 ← CORTAR(S, k)
4       C ← GERAR-MATRIZ-DE-CUSTOS(S1, S2)
5       sol ← RESOLVER-PROBLEMA-ATRIBUIÇÃO(C)
6       S ← GERAR-NOVA-SOLUÇÃO(S, sol)
7  return  S
```

## 3.4 K-swap

Seja $S = \{s|s$ é uma *sequência*$\}$ um conjunto finito. Dado dois cortes distintos, que transpassam as sequências pertencentes a $S$ e, portanto, delimitam um intervalo. Dessa forma é gerado dois conjuntos distintos $S_1$ e $S_2$, tal que $S_1$ contém os segmentos das sequências externas ao intervalo delimitado pelos dois cortes, e $S_2$ contém os segmentos das sequências internas ao intervalo delimitado pelos dois cortes. Com isso, é possível recombinar $S_1$ e $S_2$ para gerar um novo conjunto $S'$.

Semelhante ao formato do PCR descrito na seção 3.3, uma maneira de se escolher uma das combinações para gerar $S'$ é resolver um PA, tal que cada linha $i$ da matriz de custos $C$ representa os dois segmentos de sequência $s_i \in S_1$, tal que $s_i = (s_{i1}, s_{i2})$. E cada coluna $j$ representa uma sequência $s_j \in S_2$. Agora existem dois segmentos associados $(s_{i1}, s_{i2})$ a uma sequência em $s_i \in S_1$, pois diferente do PCR, existem dois cortes que delimitaram esses dois segmentos para cada $s_i$. Já no conjunto $S_2$ não é necessário segmentar as suas respectivas sequências, pois as sequências são geradas pelos intervalos delimitados pelos dois cortes. As figura 3.2, figura 3.3 e figura 3.4 ilustram como são feitos esses dois cortes, os conjuntos $S_1$ e $S_2$ e as recombinações durante o processo *k-swap*.

Cada elemento $c_{ij}$ da matriz de custos $C$ do PA recebe um valor $g(i, j)$, tal que $g(i, j)$ é um valor associado a um custo da recombinação de cada elemento $s_i \in S_1$, apontado por $i$, com cada elemento $s_j \in S_2$ apontada por $j$. Caso a recombinação entre um elemento $s_i \in S_1$ com outro elemento $s_j \in S_2$ não seja permitido (devido às restrições do problema, por exemplo), $c_{ij} = \infty$, para que $c_{ij}$ possivelmente não esteja na solução do PA. Esse processo é ilustrado nas figura 3.2, figura 3.3 e figura 3.4 em três etapas, e a estrutura de cada elemento $c_{ij}$ da matriz $C$ é mostrada na equação 3.8.

$$c_{ij} = \begin{cases} \infty & \text{caso não exista possibilidade de combinação do elemento } s_i \text{ com } s_j \\ g(i,j) & \text{caso contrário} \end{cases} \tag{3.8}$$

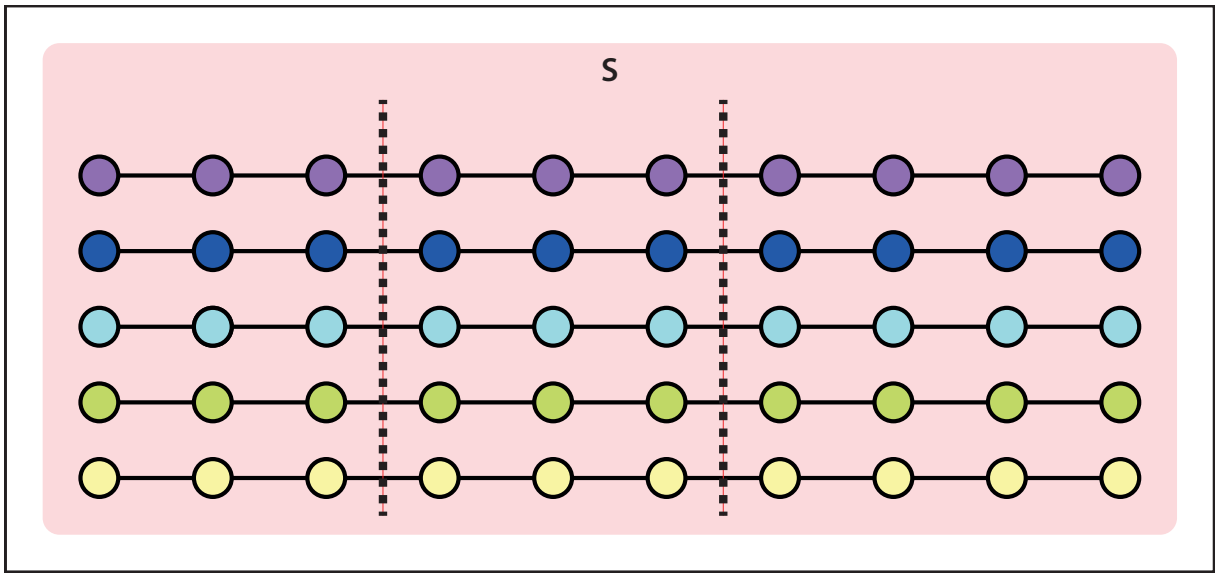


**Figura 3.2:** Primeira etapa: ilustração de dois locais cortes do procedimento *k-swap* com $k = 3$, dado uma instância com cinco sequências para recombinar.

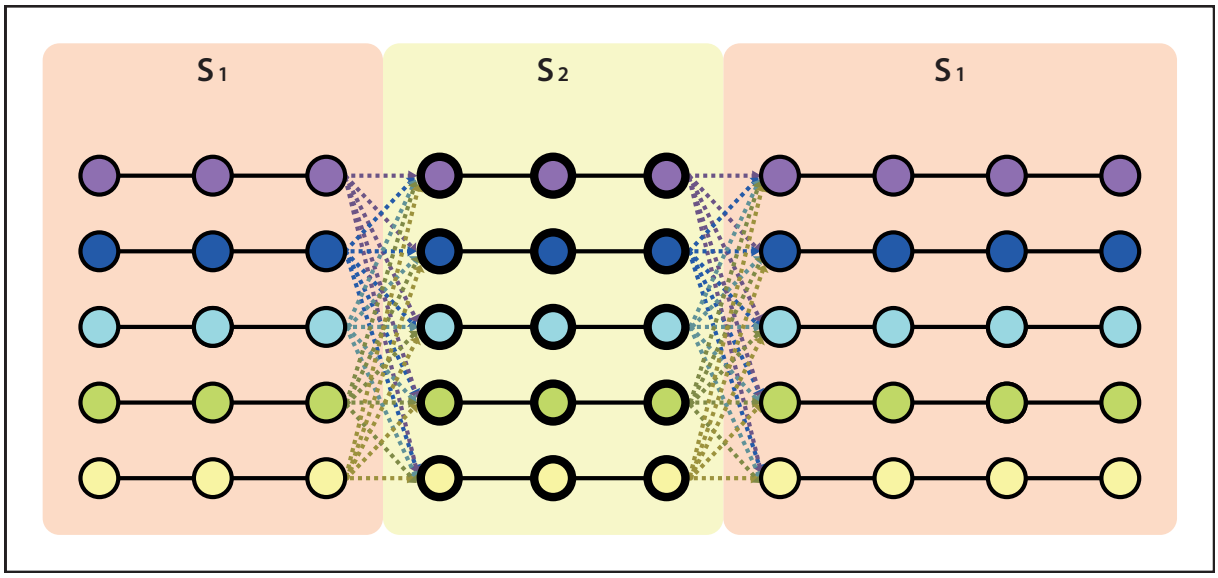


**Figura 3.3:** Segunda etapa: Possibilidades de recombinação entre as sequências nos locais de corte do procedimento *k-swap*.

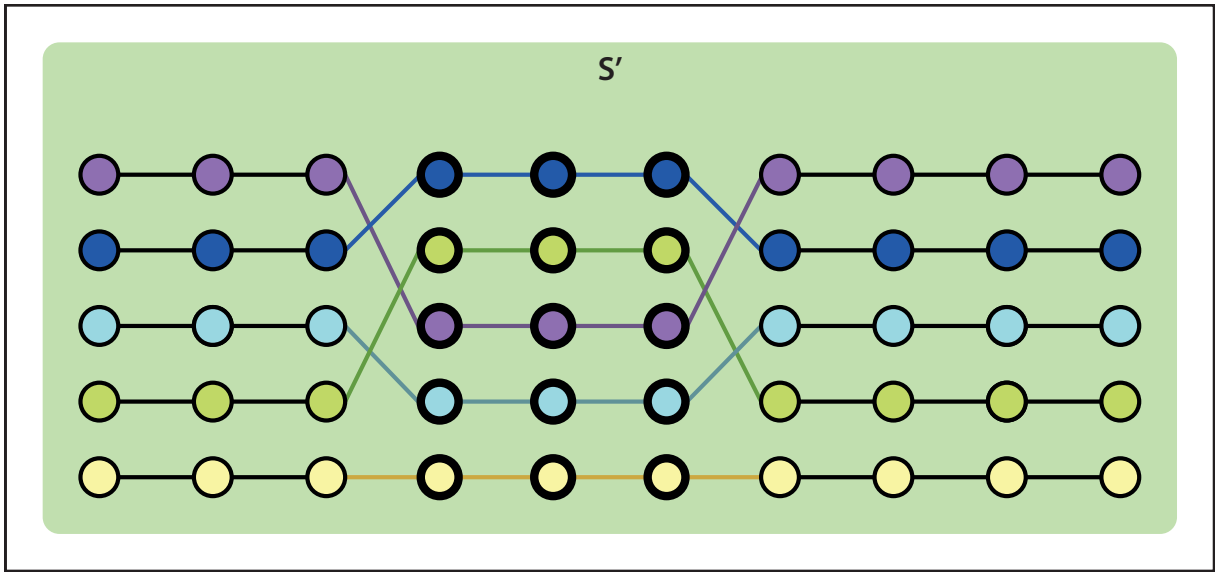


**Figura 3.4:** Terceira etapa: Finalização das recombinações entre as sequências após resolver o PA do procedimento *k-swap*.

O procedimento *k-swap* é semelhante ao PCR, a geração de vizinhos continua sendo de $n$ cortes na solução de entrada, porém são feitas $n-k$ recombinações sucessivas nesses cortes. Os dois cortes precisam ser distintos e obedecer ao valor $k$, ou seja, o segundo corte precisa ser dado a $k$ posições após o primeiro corte. Os dois cortes podem ser feitos em qualquer local da solução (desde que seja no intervalo de início e fim) e o intervalo entre os cortes também pode variar de acordo com o valor $n$.

Para exemplificar, em um caso particular no EMO, a solução de entrada é uma escala de motoristas e os dois cortes são dados uniformemente, com mesmos intervalos de tempo, do início da escala até o final dela, tal que o segundo corte está a $k$ cortes do primeiro corte (todos os locais de cortes podem ser gerados previamente, como descrito no algoritmo 4).

Um segmento de jornada de motorista é definido como uma sequência de tarefas contíguas que pertencem à essa jornada. Dessa forma, dado os dois cortes, neles ocorrem uma recombinação de jornadas de motorista da seguinte forma: todos os segmentos das jornadas externas ao segmento delimitado pelos dois cortes são recombinados com os segmentos das jornadas internas delimitados pelos dois cortes. Ou seja, forma-se uma vizinhança máxima de $n^2 = |J| \times |J|$, tal que $|J|$ é a cardinalidade do conjunto de jornadas de motorista da escala. Da mesma forma que no PCR, $g(x)$ é uma função objetivo. Mais detalhes sobre os parâmetros e algoritmos aplicados nesse trabalho para o *k-swap* são mostrados na seção 4.3.1.

O algoritmo 4 define uma função *k-swap*, da mesma forma que no PCR, esse algoritmo pode ser alterado para definir um critério de parada, caso a decisão seja não percorrer todas as camadas geradas, dessa forma, a solução corrente de quando o algoritmo para é retornada.

A função GERAR-LOCAIS-DE-CORTE$(S, k)$ do algoritmo 4 gera os locais onde a solução corrente deve ser seccionada. Diferente do PCR, agora um conjunto de tuplas $(q_i, q_j)$ é necessário para fazer dois cortes na solução $S$. Os locais de corte gerados são usados na função CORTAR$(S, q_1, q_2)$, de forma que seja gerado os dois conjuntos distintos $S_1$ e $S_2$, assim a matriz de custos $C$ é criada e o PA resolvido. Ao final da iteração, a resposta do PA é transformada em uma nova solução, por conseguinte a solução $S$ é atualizada com essa nova solução. Quando todos elementos $(q_1, q_2) \in Q$ são percorridos, o processo para e retorna a solução $S$ atualizada.

**Algoritmo 4**

**function** KSWAP($S, k$)

```
1  Q ← GERAR-LOCAIS-DE-CORTE(S, k)
2  for (q1, q2) ∈ Q do
3      S1, S2 ← CORTAR(S, q1, q2)
4      C ← GERAR-MATRIZ-DE-CUSTOS(S1, S2)
5      sol ← RESOLVER-PROBLEMA-ATRIBUIÇÃO(C)
6      S ← GERAR-NOVA-SOLUÇÃO(S, sol)
7  return S
```

## 3.5 Critérios para percorrer as camadas

Os conjuntos $K$ e $Q$ do algoritmo 3 e algoritmo 4 respectivamente podem ser transformados ou criados como sequências para direcionar uma ordem de percurso no conjunto. Dessa forma, existem em geral duas formas de percorrer uma instância para aplicar os procedimentos PCR ou *k-swap* nessas sequências (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*). A quantidade de cortes para formar a vizinhança também pode variar, porém deve-se equilibrar a quantidade de cortes com o tempo de execução do algoritmo proposto, pois quanto mais cortes, mais tempo é necessário para calcular as recombinações desses cortes. Consequentemente, a eficiência do algoritmo é comprometida, e além disso, a eficácia do algoritmo com mais cortes pode não ser muito melhor do que um algoritmo com uma quantidade menor de cortes. A quantidade de cortes ($n$) deste trabalho é definida na seção 4.3.1.

Também é possível criar critérios para parar a execução dos algoritmo 3 e algoritmo 4. Um critério geralmente usado é a parada na primeira melhora (*repeat* interno do algoritmo 2 FIRST-IMPROVEMENT), ou seja, o algoritmo não percorre todas as camadas, entretanto ele para quando encontra a primeira melhora $S'$ em relação à solução $S$ e a retorna.

## 3.6 Variable Neighbourhood Descent (VND)

O método *Variable Neighbourhood Descent* (VND) tem como característica a mudança de vizinhanças de maneira determinística. O algoritmo 5 mostra o método VND. Ele procura um mínimo local dado $k'_{max}$ vizinhanças, pois as chances de se encontrar um mínimo global são maiores do que procurar apenas em uma vizinhança. Além disso, é possível usar uma

estratégia aninhada, dado $k'_{max} = 3$, duas vizinhanças podem ser usadas em uma solução $x'$ após a aplicação de uma terceira, ou seja $x' \in N_3(x)$ (Hansen et al., 2009).

**Algoritmo 5**

**function** VND$(x, k'_{max})$

```
1  repeat
2      k ← 1
3      repeat
4          x' ← arg min_{y∈N_k(x)} f(y) // encontre o melhor vizinho em N_k(x)
5          if f(x') < f(x) then
6              x ← x'
7              k ← 1
8          else
9              k ← k + 1
10     until k = k'_max
11 until nenhuma melhora obtida
12 return x
```

## 3.7 Reduced Variable Neighbourhood Descent (RVND)

O método *Reduced Variable Neighbourhood Descent* (RVND) é obtido se pontos aleatórios são selecionados de $N_k(x)$ e nenhuma decida é feita. Ao invés disso, os valores desses pontos são comparados com a solução corrente, a qual é atualizada em caso de melhora. Um critério de parada deve ser selecionado, como tempo máximo de processamento, ou número máximo de iterações. O algoritmo 6 mostra o RVND, nele o critério de parada está definido como tempo máximo de processamento $t_{max}$. A função SHAKE$(x, k)$ da linha 4 gera uma solução aleatória $x'$ da vizinhança $k$ selecionada, isto é, $x' \in N_k(x)$ (Hansen et al., 2009).

O RVND é útil para instâncias de tamanho muito grande, para as quais a busca local é custosa. Também foi observado, de acordo com Hansen et al. (2009), que o melhor parâmetro para $k_{max}$ do algoritmo 6 é geralmente 2.

**Algoritmo 6**

**function** RVNS($x, k_{max}, t_{max}$)

```
1  repeat
2      k ← 1
3      repeat
4          x' ← SHAKE(x, k)
5          if  f(x') < f(x)  then
6              x ← x'
7              k ← 1
8          else
9              k ← k + 1
10     until k = k'_max
11     t ← CPU-TIME()
12 until t > t_max
13 return  x
```

## 3.8 Variable Neighbourhood Search (VNS)

A meta-heurística *Variable Neighborhood Search* (VNS) vem sendo aplicada em diversos problemas computacionais, dois exemplos são: coloração de grafos (Avanthay et al., 2003) e no Problema de Roteamento de Veículos (*Vehicle Routing Problem* - VRP) (Polacek et al., 2004). Essas e outras aplicações indicam que o VNS é um método efetivo para resolver esses tipos de problemas e problemas relacionados na área de otimização combinatória.

O 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 com a combinação de uma busca local (Hansen et al., 2009). O algoritmo 7 mostra o VNS básico, nele tem-se a função SHAKE$(x, k)$ que a partir da solução corrente $x$ e um parâmetro $k$ é gerado uma nova solução $x'$ em um processo que envolve aleatoriedade (diversificação). A função BUSCA-LOCAL$(x')$ aplica um processo de busca local qualquer para gerar uma nova solução $x''$. O algoritmo 7 é a base para os algoritmos propostos.

Caso a função BUSCA-LOCAL$(x')$ do algoritmo 7 seja o algoritmo 5 VND, tem-se o VNS Geral – *General VNS* (GVNS).

**Algoritmo 7**

**function** VNS($x, k_{max}$)

```
1  k ← 1
2  repeat
3        x' ← SHAKE(x, k)
4        x'' ← BUSCA-LOCAL(x')
5        if   f(x'') < f(x')   then
6              x ← x''
7              k ← 1
8        else
9              k ← k + 1
10 until k = k_max
11 return   x
```

## 3.9 Problema de Escalonamento de Trabalhadores em Local Fixo

Constantino et al. (2007) utilizaram uma abordagem de Raff (1983) para obter uma estimativa de motoristas requeridos para o PEMO – tratado em Constantino et al. (2007) como Problema de Escalonamento de Tripulantes (PET). Nessa abordagem, considerações espaciais do PEMO são desconsideradas para obter-se uma figura clara das considerações temporais (Constantino et al., 2007).

Para esse problema é considerado a divisão de um dia de trabalho em uma sequência $T$ de intervalos de tempo, e uma demanda de trabalhadores $d_t$ associada com cada intervalo de tempo $t \in T$. O problema de escalonamento de trabalhadores consiste em encontrar um conjunto de jornadas diárias que cubra todo o trabalho requerido, ou seja, cada $d_t$ associado a um tempo $t$.

Para usar esse modelo proposto por Raff (1983) no cálculo de um limite inferior para o PEMO, define-se $d_t$ como sendo o número de veículos que operam durante todo o intervalo de tempo $t$. Dessa forma, $d_t$ é um limite inferior para o número de jornadas de motoristas requeridas durante o período de tempo $t$. Para obter esse limite inferior usado por Constantino et al. (2007) foi considerado que cada $t \in T$ tem duração de um minuto.

# 4

# Algoritmos propostos

Os quatro algoritmos propostos são compostos por uma fase construtiva determinística que é comum entre esses algoritmos; e uma fase melhorativa, que diferem esses algoritmos. A fase construtiva, descrita na seção 4.1, gera uma solução inicial viável que serve de entrada para as quatro fases melhorativas.

Todas as fases melhorativas têm como algoritmo proposto a meta-heurística VNS, descrito no algoritmo 7 da seção 3.8. O que difere as quatro propostas é a função BUSCA-LOCAL desse algoritmo.

Portanto, a primeira proposta é um GVNS (VNS com VND em sua busca local, seção 4.4). A segunda, terceira e quarta propostas são algoritmos VNS com buscas locais diferentes denominadas adaptativas, as quais são descritas nas seções 4.5, 4.6 e 4.7. As fases melhorativas tem critérios e métodos para exploração de vizinhança nas suas respectivas funções BUSCA-LOCAL, isso está proposto na seção 4.3.

Os algoritmos usam o termo *sequência* (que podem ser vetores), essas sequências têm como primeiro índice o valor 0 e terminam em $n - 1$, tal que $n$ representa o tamanho da sequência.

## 4.1 Fase construtiva

Como argumentado em alguns trabalhos pesquisados (Leone et al., 2010), (Silva e Reis, 2014), uma boa fase construtiva é importante para a fase melhorativa, pois auxilia na eficiência da convergência durante a fase melhorativa. Embora tenha sido observado que vários trabalhos usam métodos mais simples na criação de uma solução inicial sem se

preocupar e analisar com mais profundidade a solução inicial gerada, este trabalho propõe um algoritmo para criar uma solução inicial mais elaborada para o PEMO, baseado no trabalho de Sakiyama et al. (2014).

### 4.1.1 Geração de camadas

O primeiro passo da fase construtiva é um processo de geração de camadas, o qual é descrito no algoritmo 8. Esse primeiro algoritmo tem como entrada um conjunto de tarefas $T$, tal que esse conjunto pode ser transformado em uma sequência, e as tarefas dessa sequência são ordenadas pelos seus respectivos tempos de oportunidade de troca de início ($t_i$). Dessa forma é facilitado o trabalho da função REMOVER-MAIS-CEDO($T$) que faz a remoção da tarefa que tem o menor tempo $t_i$. Esse primeiro passo retorna uma sequência de camadas $C$.

O propósito do algoritmo 8 é criar uma sequência de camadas $C$, tal que cada camada $K_i$ da sequência $C$ contenha tarefas. Essa sequência de camadas $C$ direciona a formação de uma escala de motoristas inicial viável no segundo passo da construção da solução inicial.

As camadas da sequência $C$ são geradas de forma que as tarefas em camadas posteriores possam ter a chance de ser sequenciadas com tarefas de camadas anteriores. Isto é, dada uma camada qualquer $K_{i-1}$ (anterior), que é um elemento da sequência $C$, cada tarefa que está na camada $K_i$ (posterior) pode ser sequenciada com pelo menos uma tarefa da camada $K_{i-1}$. O processo de geração de camadas é definido no algoritmo 8 onde a função NÃO-PODE-SEQUENCIAR($K, t$) retorna verdadeiro se uma tarefa $t$ não pode ser sequenciada com pelo menos uma tarefa da camada $C[i-1]$ (que é a camada $K_{i-1}$) indicada pelo algoritmo.

A figura 4.1 ilustra a geração de camadas a partir de seis quadros de execução ($\{Q_1, Q_2, ..., Q_6\}$) em uma hipotética escala de veículos que inicia no começo da manhã às 5:00h. Um detalhe nessa geração de camadas é que na camada 1 ($K_1$) estão inclusas as primeiras tarefas dos quadros de execução $Q_1$ e $Q_2$, pois essas tarefas estão sendo iniciadas na garagem, mesmo que essas duas tarefas estejam começando após as três primeiras tarefas dos quadros $Q_4$, $Q_5$ e $Q_6$. Outro detalhe relacionado, está no quadro de execução $Q_3$, a sua primeira tarefa está na camada 2 ($K_2$), já que é possível sequenciar essa tarefa com pelo menos uma tarefa da camada 1 ($K_1$). Esses detalhes explanados estão relacionados com a função NÃO-PODE-SEQUENCIAR($K, t$), a qual contempla as restrições do problema, e no caso da figura 4.1, o sequenciamento de tarefas que iniciam em diferentes oportunidades de troca não é permitido.

**Algoritmo 8**

```
function GERAR-CAMADAS(T)
 1  C ← SEQUÊNCIA VAZIA
 2  K ← ∅
 3  INSERE-AO-FIM(C, K)
 4  while T ≠ ∅ do
 5        t ← REMOVER-MAIS-CEDO(T)
 6        i ← TAMANHO(C)
 7        while i > 0 and NÃO-PODE-SEQUENCIAR(C[i − 1], t) do
 8              i ← i − 1
 9        if i = TAMANHO(C) then
10              K ← ∅
11              INSERE-AO-FIM(C, K)
12        C[i] ← C[i] ∪ t
13  return C
```

### 4.1.2 Solução inicial

O segundo passo da fase construtiva é percorrer a sequência de camadas $C$ para formar um conjunto de jornadas $J$, o qual é a escala de motoristas inicial viável. Logo, a saída do algoritmo 8 é a entrada para o algoritmo 9.

Para isso, para cada camada $K_i$ de $C$ é resolvido um PA que atribui as tarefas da camada $K_i$ para uma jornada existente em $J$; ou caso haja inviabilidade de atribuição, cria-se uma nova jornada em $J$.

Cada elemento $a_{ij}$ da matriz de custos $A_{|J|+|K|} = [a_{ij}]$, que é a entrada para o PA citado, 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 e resumida na matriz 4.1. Os índices $i$ e $j$ que percorrem a matriz são iniciados em 0 e têm valores máximos de $(|J| + |K|) - 1$.

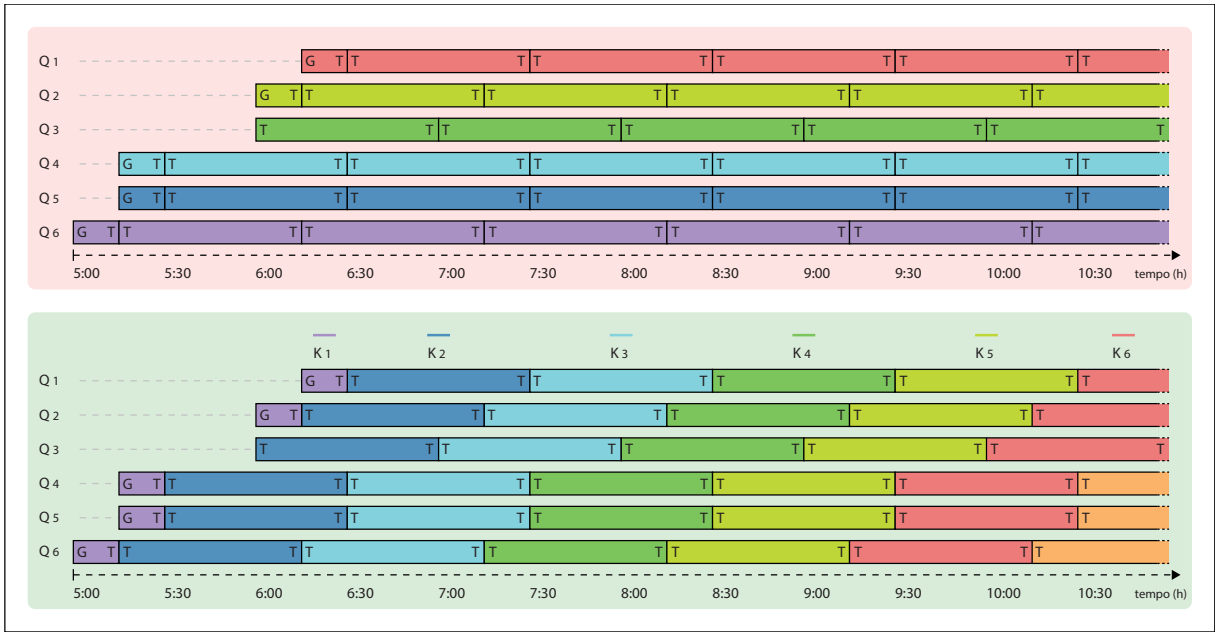


**Figura 4.1:** Geração das camadas (quadro inferior) a partir de seis quadros de execução $\{Q_1, Q_2, ..., Q_6\}$ (quadro superior).

$$\begin{bmatrix} \textit{Bloco 1} & \textit{Bloco 2} \\ \textit{se } i < |J| \textit{ e } j < |K| & \textit{se } i < |J| \textit{ e } j \geq |K| \\ a_{ij} = g(i,j) \textit{ ou } a_{ij} = \infty & a_{ij} = obj(i) \textit{ ou } a_{ij} = \infty \\ \\ \textit{Bloco 3} & \textit{Bloco 4} \\ \textit{se } i \geq |J| \textit{ e } j < |K| & \textit{se } i \geq |J| \textit{ e } j \geq |K| \\ a_{ij} = c_{nj} \textit{ ou } a_{ij} = \infty & a_{ij} = 0 \end{bmatrix} \tag{4.1}$$

**Bloco 1** Para $i < |J|$ e $j < |K|$, $a_{ij} = g(i,j)$; caso haja inviabilidade de atribuição, $a_{ij} = \infty$. Ou seja, nesse bloco são tratados os casos de tarefas reais atribuídas à jornadas reais.

$g(i,j)$ é uma função que calcula um valor para a matriz de custos do PA, 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 usadas em $g(i,j)$ podem gerar soluções iniciais melhores. Posteriormente na seção 5.2 é explicado qual valor exato $g(i,j)$ recebe neste trabalho;

**Bloco 2** Para $i < |J|$ e $j \geq |K|$, $a_{ij} = obj(i)$; caso haja inviabilidade de atribuição, $a_{ij} = \infty$. Ou seja, nesse bloco são tratados os casos de tarefas fictícias atribuídas às jornadas reais, logo, isso indica uma *parada* (um intervalo de descanso por exemplo).

$obj(i)$ é a função objetivo da jornada indexada por $i$, porém como explicado no Bloco 1, essa função pode ser modificada para encontrar melhores soluções. Essa função também pode adicionar penalidades para penalizar paradas como é feito em Sakiyama et al. (2014);

**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 $|J| - i$. Isso é feito para forçar tarefas serem designadas para jornadas já existentes, ao invés de criar novas jornadas;

**Bloco 4** 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 proposto por Carpaneto e Toth (1987) e o processo completo para geração da solução inicial é descrito no algoritmo 9. A função NOVAS-JORNADAS$(J, S)$ 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 $J$, pois pares designados $i, j$ dos blocos 2 e 4 são de tarefas fictícias.

**Algoritmo 9**

**function** SOLUÇÃO-INICIAL$(C)$

```
1  J ← ∅
2  for  K na ordem da sequência C  do
3       M ← CONSTRUIR-MATRIZ-DE-CUSTOS(J, K)
4       S ← RESOLVER-PROBLEMA-DE-ATRIBUIÇÃO(M)
5       J ← J ∪ NOVAS-JORNADAS(J, S)
6  return  J
```

Para a fase de melhoramento, quatro variações da meta-heurística VNS foram aplicadas no resultado gerado pela fase construtiva. O algoritmo base das quatro propostas

está formulado no algoritmo 7 e suas variações são descritas nas seções 4.4 a 4.7. Antes disso, na seção 4.2 é mostrado como a função SHAKE foi aplicada em todos algoritmos propostos e na seção 4.3 é explicado como são feitos os percursos e critérios de parada nos procedimentos PCR e *k-swap*.

## 4.2 Shake

A função SHAKE, descrita no algoritmo 10, também é usada em todos os quatro algoritmos propostos, a qual tem a função de diversificar a solução $x$ de entrada e o parâmetro $k$ direciona qual método será usado para a diversificação. Além disso, existem duas variáveis que controlam a quantidade de recombinações que devem ser feitas na solução $x$, as quais são *fat1* e *fat2*, que indicam que 15% e 30% das jornadas da solução $x$ devem ser recombinadas respectivamente.

A função *shake1slice*, escolhe aleatoriamente um tempo dentro do intervalo de começo e final da jornada $x$ e faz um corte na solução (da mesma forma que um corte no PCR da seção 3.3). Com isso, recombina-se as jornadas nesse corte com apenas o critério de que os segmentos de jornadas gerados do corte possam ser sequenciados, para que a recombinação não gere soluções inviáveis. Ou seja, dado um corte na solução $x$, procura-se recombinar segmentos de jornadas que gerem novas jornadas viáveis sem se preocupar com a função objetivo.

A função *shake2slice* é semelhante, mas nesse caso são feitos dois cortes próximos (da mesma forma que dois cortes no processo *k-swap* da seção 3.4). Dessa forma, semelhante ao *k-swap*, os segmentos delimitados pelos dois cortes são recombinados na solução $x$ com o mesmo critério da função *shake1slice*, ou seja, procura-se recombinar segmentos de jornadas com o mesmo intuito de gerar diversificação na solução $x$.

Caso os cortes aleatórios não gerem a percentagem de recombinações desejadas (15% e 30% da cardinalidade das jornadas), um novo corte é feito, e esse processo se repete até que o número de recombinações seja alcançado.

**Algoritmo 10**

**function** SHAKE($x, k$)

```
 1  n ← LENGTH(x)
 2  fat1 ← 0.15
 3  fat2 ← 0.30
 4  if  k = 1  then
 5      return  SHAKE2SLICE(x, n × fat1)
 6  else if  k = 2  then
 7      return  SHAKE1SLICE(x, n × fat1)
 8  else if  k = 3  then
 9      return  SHAKE1SLICE(x, n × fat2)
10  else if  k = 4  then
11      return  SHAKE2SLICE(x, n × fat2)
```

## 4.3 Percurso, critérios de parada e atualização da solução nas estruturas de vizinhança

Foram escolhidas duas formas de percurso nos procedimentos PCR e *k-swap*:

***Forward*:** percorre a vizinhança gerada na ordem crescente de tempo;

***Backward*:** percorre a vizinhança gerada na ordem decrescente de tempo.

Os critérios de parada e de atualização da solução corrente do percurso escolhidos são três:

***First improvement*:** o procedimento para quando a primeira solução encontrada $s'$ é melhor que a solução de entrada $s$, dessa forma retorna-se a solução $s'$. Esse critério é descrito no algoritmo 2, nas linhas 4 até 7;

***Best improvement*:** o procedimento continua até o final do percurso na vizinhança e retorna a melhor solução encontrada $s'$ dentre todas, dada a solução de entrada $s$. Esse critério é usado em Sakiyama et al. (2014) e Calvi (2005);

***Continuous improvement*:** durante o percurso, o procedimento atualiza a solução $s$ por $s'$, caso $s'$ seja melhor que $s$ (ou seja, $f(s') < f(s)$); caso contrário a solução $s$ é mantida. Esse procedimento continua até o final do percurso na vizinhança. Esse critério é usado em Sakiyama et al. (2014) e Calvi (2005) com a denominação de *first improvement*.

Com isso são feitas combinações entre as duas formas de percurso (*forward* e *backward*) com os critérios de parada e atualização da solução corrente (*First*, *Best* e *Continuous improvement*) para explorar as vizinhanças que façam parte da função BUSCA-LOCAL do algoritmo 7.

A figura 4.2 ilustra os três critérios de parada e atualização da solução que deve ser retornada. No primeiro caso (*Continuous improvement*), a solução $s$ é sempre atualizada por $s'$ quando esta é melhor que $s$. No segundo caso (*best improvement*), a solução $s$ é mantida, e a melhor $s'$ é retornada no final da busca. No terceiro caso, a primeira melhora $s'$ sobre $s$ é retornada.

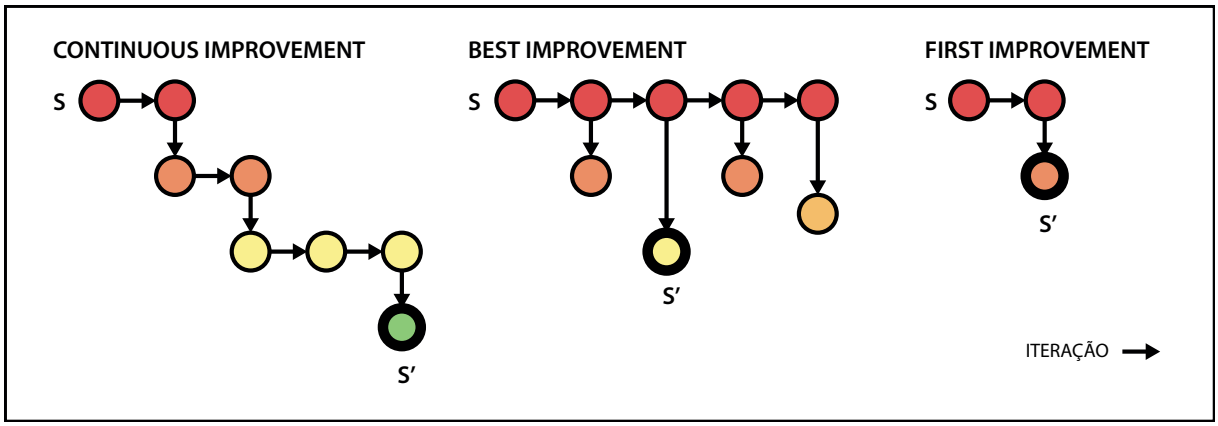


**Figura 4.2:** Critérios de parada durante o percurso na vizinhança, dada uma solução $s$ de entrada e a obtenção de uma solução $s'$ como resultado.

### 4.3.1 Geração dos locais de corte nos procedimentos PCR e kswap propostos

Neste trabalho, o conjunto $K$ do algoritmo 3 é visto como uma sequência de elementos que representam uma ordem de tempo crescente ou decrescente para os percursos *forward* ou *backward* respectivamente. Cada elemento $k_i$ da sequência $K$ representa o tempo em que a função CORTAR$(S, k)$ (linha 3) irá fazer o corte na solução $S$.

A função GERAR-LOCAIS-DE-CORTE$(S)$ – usada no algoritmo 3 e redefinida no algoritmo 11 – recebe dois novos parâmetros: $n = |C|$, que é retornado pelo algoritmo 8, e $T$, que neste trabalho é a sequência ordenada de tarefas por oportunidade de troca de início $(t_i)$ do conjunto $T$ do algoritmo 8. Essa função tem duas variáveis $step = |T|/n$ e $i = [1...n+1]$. Dessa forma, cada elemento $k_i$ da sequência $K$ recebe o tempo $t_i$ da tarefa $T_{step \times i}$ e assim são criados $n$ tempos para que os cortes sejam feitos.

Esse método de geração da sequência de tempos $K$ foi proposto para dar prioridades em períodos de tempo em que haja uma densidade de tarefas maior em relação a outros períodos de tempo. Logo, haverá mais cortes em períodos de tempo de pico, por exemplo: no início da manhã e no final da tarde. O algoritmo 11 descreve a função GERAR-LOCAIS-DE-CORTE$(S, n, T)$ e a função $t_i(t)$ desse algoritmo retorna o valor de tempo $t_i$ da tarefa $t$.

A figura 4.3 ilustra como os cortes são feitos em uma instância real de 3478 tarefas (tabela 5.1). Foi contado quantas tarefas existem a cada minuto e mostrado no gráfico, e as linhas verticais que cortam o gráfico são os tempos de corte criados pelo algoritmo 11. É possível observar onde existem mais tarefas por minutos a distância entre os cortes das linhas verticais é menor.

**Algoritmo 11**

**function** GERAR-LOCAIS-DE-CORTE$(S, n, T)$

```
1  K ← SEQUÊNCIA-VAZIA
2  step ← |T|/n
3  for  i ← 1 to n + 1  do
4       t ← T[step × i]
5       t ← t_i(t)
6       INSERE-AO-FIM(K, t)
7  return  K
```

Como o algoritmo 11 gera a sequência $K$ na ordem crescente de tempo, ele é usado no percurso *forward*; com isso, para o percurso *backward* basta inverter a sequência $K$ (ou percorrer essa sequência na ordem inversa).

A função GERAR-LOCAIS-DE-CORTE$(S, k)$ que gera o conjunto $Q$ do procedimento *k-swap* descrito na seção 3.4 e no algoritmo 4 é semelhante ao algoritmo 11. Porém, como o *k-swap* requer dois cortes, é necessário usar o parâmetro $k$ para gerar um par de tempos $(m_1, m_2)$, tal que $m_2$ seja um tempo a $k$ posições de $m_1$.

O algoritmo 12 usa o algoritmo 11 para gerar os locais preliminares de corte representado pela sequência $M$ (linha 1). Em seguida são criadas duas sequências $M_1$ e $M_2$ a partir da sequência $M$ (linhas 2 e 3), porém nessas cópias, são excluídos os $k$ últimos elementos de $M$ na sequência $M_1$; e os $k$ primeiros elementos de $M$ na sequência $M_2$. Com isso a sequência $Q$ recebe os pares $(m_1, m_2)$ provenientes das sequências $M_1$ e $M_2$. Esse processo deve ser feito para gerar $n - k$ locais de corte.

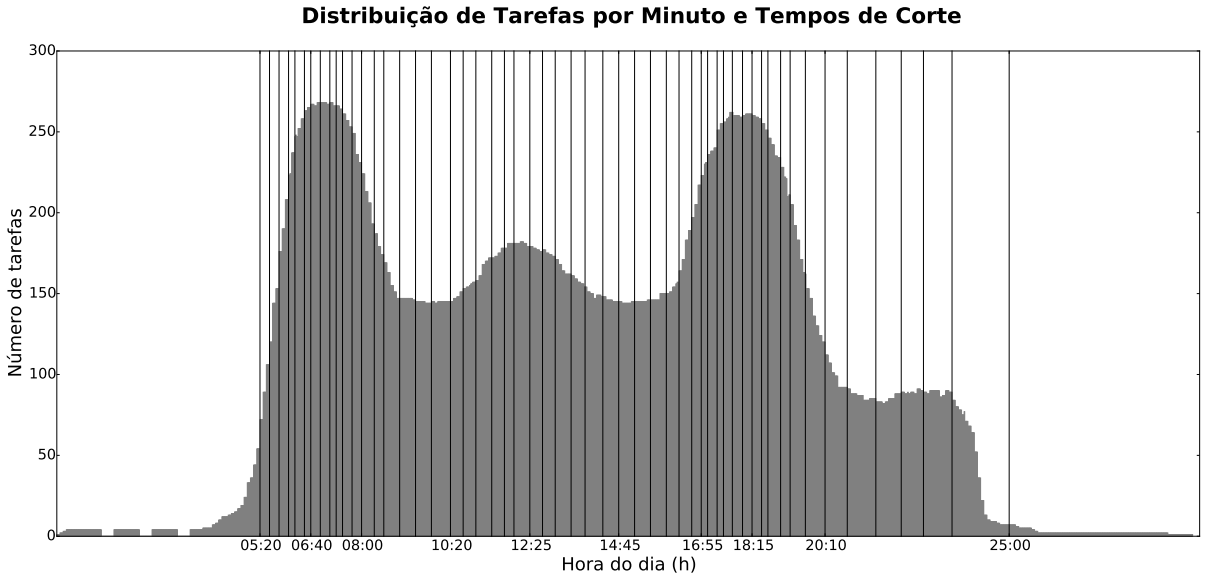


**Figura 4.3:** Distribuição de tarefas por minutos e tempos de corte (linhas verticais) em uma instância real de 3478 tarefas (tabela 5.1). O eixo $x$ pode ultrapassar 24:00h pois o horizonte da escala de motoristas ultrapassa 24:00h.

Como o algoritmo 12 gera a sequência $Q$ na ordem crescente de tempo, ele é usado no percurso *forward*; com isso, para o percurso *backward* basta inverter a sequência $Q$ (ou percorrer essa sequência na ordem inversa).

### 4.3.2 Proposta de critérios de parada e atualização da solução nos procedimentos PCR e kswap

Para o PCR, após gerar os locais de cortes descritos na seção 4.3.1, é necessário percorrer esses locais para atualizar a solução $S$ como descrito na seção 3.3 e no algoritmo 3. A

**Algoritmo 12**

**function** GERAR-LOCAIS-DE-CORTE-KSWAP$(S, n, T, k)$

1 $M \leftarrow$ GERAR-LOCAIS-DE-CORTE$(S, n, T)$
2 $M_1 \leftarrow copiar(M, 0, |M| - k)$
3 $M_2 \leftarrow copiar(M, k, |M|)$
4 $Q \leftarrow$ SEQUÊNCIA-VAZIA
5 **for** $(m_1, m_2) \leftarrow (M_1, M_2)$ **do**
6 $\quad Q \leftarrow$ INSERE-AO-FIM$((m_1, m_2))$
7 **return** $Q$

matriz de custos $C$ é formada da recombinação dos segmentos de jornadas como descrito na seção 3.3. A função $g(x)$ descrita na seção 3.3, usada neste trabalho para formar a matriz $C$ de entrada para o PA, nesse contexto é a *função objetivo* $f(x)$ descrita na seção 5.2.

Dessa forma, para cada $c_{ij}$ calcula-se a função objetivo $f(x)$ após o sequenciamento da jornada parcial $e_i$ à esquerda do corte e da jornada parcial à direita do corte $d_j$ para obter $f(x) = f(e_i + d_j)$. Deve ser lembrado que caso não haja possibilidade de sequenciamento entre as duas jornadas parciais $e_i$ e $d_j$, $f(x) = \infty$.

As três propostas de algoritmo para o PCR são descritas nos algoritmos 13, 14 e 15. A função GERAR-LOCAIS-DE-CORTE$(S, n, T)$ está definida na seção 4.3.1 anterior. O novo parâmetro *inverte* é um valor booleano que indica se a sequência $K$ deve ser invertida ou não. Isso é feito para indicar os percursos *forward* ou *backward*.

É importante notar que no algoritmo 13, se não é encontrada nenhuma solução melhor que $S$, a mesma é retornada (linha 11). E o algoritmo 15 substitui $S$ caso ocorra alguma melhoria (linha 9) em relação a solução $S$.

**Algoritmo 13**

**function** PCR-FIRST-IMPROVEMENT$(S, n, T, inverte)$

```
 1  K ← GERAR-LOCAIS-DE-CORTE(S, n, T)
 2  if inverte then
 3       INVERTER(K)
 4  for k na ordem da sequência K do
 5       S1, S2 ← CORTAR(S, k)
 6       C ← GERAR-MATRIZ-DE-CUSTOS(S1, S2)
 7       sol ← RESOLVER-PROBLEMA-ATRIBUIÇÃO(C)
 8       S' ← GERAR-NOVA-SOLUÇÃO(sol, T)
 9       if f(S') < f(S) then
10            return S'
11  return S
```

As propostas dos algoritmos do *k-swap* são análogos às propostas do PCR, a diferença é que existe o parâmetro $k$ do algoritmo 12 para indicar onde os dois cortes devem ser feitos.

**Algoritmo 14**

```
function PCR-BEST-IMPROVEMENT(S, n, T, inverte)
 1  K ← GERAR-LOCAIS-DE-CORTE(S, n, T)
 2  if inverte then
 3       INVERTER(K)
 4  best ← S
 5  for k na ordem da sequência K do
 6       S_1, S_2 ← CORTAR(S, k)
 7       C ← GERAR-MATRIZ-DE-CUSTOS(S_1, S_2)
 8       sol ← RESOLVER-PROBLEMA-ATRIBUIÇÃO(C)
 9       S' ← GERAR-NOVA-SOLUÇÃO(sol, T)
10       if f(S') < f(best) then
11            best ← S'
12  return best
```

**Algoritmo 15**

```
function PCR-CONTINUOUS-IMPROVEMENT(S, n, T, inverte)
 1  K ← GERAR-LOCAIS-DE-CORTE(S, n, T)
 2  if inverte then
 3       INVERTER(K)
 4  for k na ordem da sequência K do
 5       S_1, S_2 ← CORTAR(S, k)
 6       C ← GERAR-MATRIZ-DE-CUSTOS(S_1, S_2)
 7       sol ← RESOLVER-PROBLEMA-ATRIBUIÇÃO(C)
 8       S' ← GERAR-NOVA-SOLUÇÃO(sol, T)
 9       if f(S') < f(S) then
10            S ← S' // solução S atualizada
11  return S
```

## 4.4 VNS 1

A primeira abordagem para a fase melhorativa é um VNS como descrito no algoritmo 7 da seção 3.8. Esse algoritmo tem o argumento $k_{max} = 4$ pois a função *shake* proposta tem quatro formas diferentes de fazer diversificação (como descrito na seção 4.2).

Na função BUSCA-LOCAL do algoritmo 7 (linha 4) é aplicado o VND descrito no algoritmo 5, ou seja, este algoritmo proposto é um GVNS (seção 3.8).

As formas de explorar os vizinhos e selecioná-los no algoritmo VND (linha 4 do algoritmo 5) são aplicações dos procedimentos PCR e *k-swap* descritos nas seções 3.3

e 3.4 com seus respectivos algoritmos 13, 15 e 14 e algoritmos análogos para o *k-swap*. O parâmetro $k'_{max} = 36$, portanto a proposta VNS 1 ($GVNS$) é formulado pelo algoritmo 17 e tem como função interna o VND, formulado pelo algoritmo 16.

**Algoritmo 16**

**function** VND$(S, k'_{max}, n, T)$

```
 1  repeat
 2      k ← 1
 3      repeat
 4          if  k = 1  then
 5              S' ← PCR-CONTINUOUS-IMPROVEMENT(S, n, T, true)
 6          else if  k = 2  then
 7              S' ← PCR-CONTINUOUS-IMPROVEMENT(S, n, T, false)
 8          else if  k = 3  then
 9              S' ← PCR-BEST-IMPROVEMENT(S, n, T, true)
10          else if  k = 4  then
11              S' ← PCR-BEST-IMPROVEMENT(S, n, T, false)
12              ...
13          else if  k = 36  then
14              S' ← KSWAP-FIRST-IMPROVEMENT(S, n, T, 5, false)
15          if  f(S') < f(S)  then
16              S ← S'
17              k ← 1
18          else
19              k ← k + 1
20      until k = k'_max
21  until nenhuma melhoria obtida
22  return  S
```

Os dois novos parâmetros $n$ (tamanho do conjunto de camadas da solução inicial) e $T$ (sequência de tarefas), dos algoritmo 16 e 17, são os mesmos descritos na seção 4.3.1. O quarto parâmetro da função KSWAP-FIRST-IMPROVEMENT$(S, n, T, k, inverte)$, é o valor de $k$ do *kswap*, que é o mesmo parâmetro do algoritmo 12 para indicar onde os dois cortes devem ser feitos. A linha 12 do algoritmo 16 indica a continuidade do algoritmo para chegar até o valor $k = 36$, e as 36 formas de explorar a solução $S'$ são descritas na tabela 4.1.

Portanto, as 36 formas de explorar as vizinhanças ($k'_{max} = 36$) são as propostas de percurso e critérios de parada e atualização da solução descritas na seção 4.3. Dessa forma, tem-se os percursos *forward* e *backward* combinados com os *first*, *best* e *continuous*

**Algoritmo 17**

**function** GVNS($S, k_{max}, k'_{max}, n, T$)

```
1  k ← 1
2  repeat
3       S' ← SHAKE(S, k)
4       S'' ← VND(S', k'_max, n, T)
5       if   f(S'') < f(S')   then
6            S ← S''
7            k ← 1
8       else
9            k ← k + 1
10 until k = k_max
11 return   S
```

*improvement* e as estruturas de vizinhança PCR, 1, 2, 3, 4 e 5-*swap*. Logo, tem-se $k'_{max} = 2 \times 3 \times 6 = 36$. A ordem de aplicação é dada na tabela 4.1.

É necessário comentar que ordens diferentes na aplicação das formas de explorar vizinhos podem levar a resultados diferentes, como foi constatado em Hansen et al. (2009). Logo a tabela 4.1 poderia ter ordens diferentes para os valores de $k'$ com intuito de avaliar uma ordem que gere soluções melhores. Da mesma forma é sugerido que a quantidade de estruturas de vizinhança seja menor, em torno de 3. Porém, para este trabalho, notou-se que existe uma quantidade grande de possíveis vizinhanças e não foi descartada nenhuma delas.

### 4.4.1 VNS 1 best improvement e com penalização de trocas de linhas

Duas variações da proposta VNS 1 foram feitas para comparar uma abordagem com uma função objetivo que penaliza trocas de linhas nas jornadas de motorista durante a execução do algoritmo.

O primeiro algoritmo, denominado VNS 1 BI (*best improvement*) utiliza somente as formas de explorar vizinhos (tabela 4.1) que usam as abordagens *best improvement* e *forward*, logo a formas de explorar vizinhos caem para 6 ($k'_{max} = 6$): PCR-BI-*forward*, 1, 2, 3, 4 e 5-*swap*-BI-*forward*. O restante do algoritmo é idêntico ao VNS 1.

O segundo algoritmo, denominado VNS 1 BI P (*best improvement* com penalização) utiliza as mesmas formas de explorar vizinhos do algoritmo VNS 1 BI, o que muda é a função objetivo que penaliza trocas de linhas nas recombinações de jornadas de motorista.

**Tabela 4.1:** Sequência dos valores de $k$ para o algoritmo VNS 1 apresentado na seção 4.4. CI, BI e FI significam *continuous improvement*, *best improvement* e *first improvement* respectivamente.

| $k$ | **processo** | $k$ | **processo** | $k$ | **processo** |
|---|---|---|---|---|---|
| 1 | PCR-CI-backward | 13 | 2-swap-CI-backward | 25 | 4-swap-CI-backward |
| 2 | PCR-CI-forward | 14 | 2-swap-CI-forward | 26 | 4-swap-CI-forward |
| 3 | PCR-BI-backward | 15 | 2-swap-BI-backward | 27 | 4-swap-BI-backward |
| 4 | PCR-BI-forward | 16 | 2-swap-BI-forward | 28 | 4-swap-BI-forward |
| 5 | PCR-FI-backward | 17 | 2-swap-FI-backward | 29 | 4-swap-FI-backward |
| 6 | PCR-FI-forward | 18 | 2-swap-FI-forward | 30 | 4-swap-FI-forward |
| 7 | 1-swap-CI-backward | 19 | 3-swap-CI-backward | 31 | 5-swap-CI-backward |
| 8 | 1-swap-CI-forward | 20 | 3-swap-CI-forward | 32 | 5-swap-CI-forward |
| 9 | 1-swap-BI-backward | 21 | 3-swap-BI-backward | 33 | 5-swap-BI-backward |
| 10 | 1-swap-BI-forward | 22 | 3-swap-BI-forward | 34 | 5-swap-BI-forward |
| 11 | 1-swap-FI-backward | 23 | 3-swap-FI-backward | 35 | 5-swap-FI-backward |
| 12 | 1-swap-FI-forward | 24 | 3-swap-FI-forward | 36 | 5-swap-FI-forward |

Isto é, durante a resolução do PA é adicionado uma penalidade para recombinações de jornadas de motorista que fazem trocas de linhas.

A penalidade é de 100 minutos para uma troca de linha, 1000 minutos para duas trocas de linhas e 10000 minutos para três ou mais trocas de linhas. Isto é, esse valor é adicionado na função objetivo, caso existam um desses 3 casos; caso contrário (sem trocas de linhas) nada é adicionado. A função objetivo é explanada com mais detalhes na seção 5.2.

## 4.5 VNS 2

A segunda abordagem para a fase melhorativa também é um VNS baseado no algoritmo 7 e a função *shake* é a mesma descrita na seção 4.4. Porém, a função BUSCA-LOCAL usa uma abordagem diferente. Todas as 36 formas de explorar vizinhos (descritas na seção 4.4) são aplicadas na solução $S'$ até um valor $m$ de vezes, e a forma que retornar o melhor resultado depois das $m$ execuções é usada iterativamente até não encontrar mais melhoria. Ao final desse processo de descida, o resultado é retornado da função BUSCA-LOCAL.

Esta proposta busca encontrar uma forma de explorar vizinhança promissora dentre todas as outras e na sequência usar essa forma até encontrar o seu respectivo mínimo. Esta proposta é de Geiger et al. (2011), ela é aplicada em outros problemas computacionais e é chamada de VND-A (*adaptative*), pois é um VND com uma abordagem de escolha de exploração de vizinhança diferente do VND tradicional.

**Algoritmo 18**

**function** BUSCA-ADAP-1$(x, m, n, T)$

```
 1  best ← x
 2  repeat
 3       S ← ∅
 4       S' ← COPIAR(best)
 5       for i ← 1 to m do
 6            S' ← PCR-CONTINUOUS-IMPROVEMENT(S', n, T, true)
 7       S ← S ∪ (S'', 1)
 8       S' ← COPIAR(best)
 9       for i ← 1 to m do
10            S' ← PCR-CONTINUOUS-IMPROVEMENT(S', n, T, false)
11       S ← S ∪ (S', 2)
12       S' ← COPIAR(best)
13       for i ← 1 to m do
14            S' ← PCR-BEST-IMPROVEMENT(S', n, T, true)
15       S ← S ∪ (S', 3)
16       S' ← COPIAR(best)
17       for i ← 1 to m do
18            S' ← PCR-BEST-IMPROVEMENT(S', n, T, false)
19       S ← S ∪ (S', 4)
20       ...
21       S' ← COPIAR(best)
22       for i ← 1 to m do
23            S' ← KSWAP-FIRST-IMPROVEMENT(S', n, T, 5, false)
24       S ← S ∪ (S', 36)
25       // Dentre todas as soluções S, aplica-se a melhor busca que obteve esse resultado
26       S' ← APLICAR-MELHOR-BUSCA(S)
27       if f(S') < f(best) then
28            best ← S'
29  until nenhuma melhoria obtida para best
30  return best
```

O algoritmo 18 redefine a função BUSCA-LOCAL do algoritmo VNS original. O novo parâmetro $m$ é a quantidade de vezes que cada forma de explorar vizinhos deve ser aplicada, neste trabalho o valor de $m = 1$, porém é possível que $m > 1$, dessa forma é necessário manter o laço. O conjunto $S$ guarda as soluções encontradas junto com um índice que aponta para a forma de explorar a vizinhança. A linha 19 do algoritmo 18 indica a continuidade do algoritmo para completar a aplicação das 36 formas de explorar vizinhos. A função APLICAR-MELHOR-BUSCA$(S)$ (linha 26) resume a aplicação da melhor

**Algoritmo 19**

```
function VNS2(S, m, n, T)
 1  k ← 1
 2  repeat
 3       S' ← SHAKE(S, k)
 4       S'' ← BUSCA-ADAP-1(S', m, n, T)
 5       if   f(S'') < f(S')   then
 6            S ← S''
 7            k ← 1
 8       else
 9            k ← k + 1
10  until k = k_max
11  return   S
```

forma de explorar vizinhos dada como entrada a sua respectiva solução $S'$ concebida anteriormente, essa função procede até que o mínimo seja atingido. Ao final o algoritmo retorna a solução $S$ encontrada.

O algoritmo 19 apenas substitui a função original BUSCA-LOCAL pela nova busca local BUSCA-ADAP-1$(S', m, n, T)$, que é o algoritmo 18, e o parâmetro $m$ precisa ser adicionado.

## 4.6 VNS 3

A terceira abordagem segue uma linha semelhante ao que é descrito na seção 4.5, a mudança está apenas em como o algoritmo 18 é feito. Nesta abordagem, o algoritmo de busca local aplica uma vez todas as formas de explorar vizinhos e a melhor solução dentre as 36 geradas é tomada como solução para a próxima iteração. Esse processo para quando não existe melhoria de uma iteração para outra.

Dessa forma, o algoritmo 20 define essa nova função BUSCA-LOCAL. A variável $melhor_{f(x)}$ salva o melhor valor de função objetivo, a variável $S$ salva todas soluções de uma iteração para serem comparadas na linha 17, caso haja uma solução melhor que a atual, a solução corrente $x$ é atualizada na linha 18 e o processo se repete, caso contrário $x$ será retornado. O algoritmo 21 define a proposta do VNS 3, a única mudança é a função BUSCA-ADAP-2$(S', n, T)$ (linha 4) em relação ao algoritmo 19, e a remoção do parâmetro $m$.

Esta proposta segue um caminho diferente em relação a proposta da seção 4.5 pois não executa exaustivamente uma forma de explorar a vizinhança na solução $x$. Ao invés

**Algoritmo 20**

**function** BUSCA-ADAP-2$(x, n, T)$

```
1  melhor_{f(x)} ← ∞
2  while  f(x) < melhor_{f(x)}  do
3        S ← ∅
4        melhor ← x
5        melhor_{f(x)} ← f(x)
6        x' ← PCR-CONTINUOUS-IMPROVEMENT(x, n, T, true)
7        S ← S ∪ x'
8        x' ← PCR-CONTINUOUS-IMPROVEMENT(x, n, T, false)
9        S ← S ∪ x'
10       x' ← PCR-BEST-IMPROVEMENT(x, n, T, true)
11       S ← S ∪ x'
12       x' ← PCR-BEST-IMPROVEMENT(x, n, T, false)
13       S ← S ∪ x'
14       ...
15       x' ← KSWAP-FIRST-IMPROVEMENT(x, n, T, 5, false)
16       S ← S ∪ x'
17       if  f(MELHOR-SOLUÇÃO(S)) < melhor_{f(x)}  then
18             x ← melhor(S)
19 return  x
```

**Algoritmo 21**

**function** VNS3$(S, n, T)$

```
1  k ← 1
2  repeat
3        S' ← SHAKE(S, k)
4        S'' ← BUSCA-ADAP-2(S', n, T)
5        if  f(S'') < f(S)'  then
6              S ← S''
7              k ← 1
8        else
9              k ← k + 1
10 until k = k_max
11 return  S
```

disso é escolhida a melhor solução da iteração corrente e na próxima iteração todas as formas de explorar vizinhanças são aplicadas novamente.

## 4.7 VNS 4

**Algoritmo 22**

**function** BUSCA-ADAP-3$(x, n, T)$

```
 1  melhor_{f(x)} ← ∞
 2  while  f(x) < melhor_{f(x)}  do
 3        S ← ∅
 4        melhor ← x
 5        melhor_{f(x)} ← f(x)
 6        x' ← COPIAR(x)
 7        repeat
 8              x' ← PCR-CONTINUOUS-IMPROVEMENT(x', n, T, true)
 9        until nenhuma melhoria para x'
10        S ← S ∪ x'
11        x' ← COPIAR(x)
12        repeat
13              x' ← PCR-CONTINUOUS-IMPROVEMENT(x', n, T, false)
14        until nenhuma melhoria para x'
15        S ← S ∪ x'
16        x' ← COPIAR(x)
17        repeat
18              x' ← PCR-BEST-IMPROVEMENT(x', n, T, true)
19        until nenhuma melhoria para x'
20        S ← S ∪ x'
21        x' ← COPIAR(x)
22        repeat
23              x' ← PCR-BEST-IMPROVEMENT(x', n, T, false)
24        until nenhuma melhoria para x'
25        S ← S ∪ x'
26        ...
27        x' ← COPIAR(x)
28        repeat
29              x' ← KSWAP-FIRST-IMPROVEMENT(x', n, T, 5, false)
30        until nenhuma melhoria para x'
31        S ← S ∪ x'
32        if  f(MELHOR-SOLUÇÃO(S)) < melhor_{f(x)}  then
33              x ← melhor(S)
34  return  x
```

A quarta abordagem segue uma linha semelhante ao que é descrito nas seções 4.5 e 4.6, a mudança está nas iterações feitas na função BUSCA-LOCAL: agora as 36 formas de

**Algoritmo 23**

**function** VNS4$(S, n, T)$

```
1  k ← 1
2  repeat
3        S' ← SHAKE(S, k)
4        S'' ← BUSCA-ADAP-3(S', n, T)
5        if   f(S'') < f(S')   then
6              S ← S''
7              k ← 1
8        else
9              k ← k + 1
10 until k = k_max
11 return   S
```

explorar vizinhança são executadas até que atinjam um mínimo local, ou seja, a solução da aplicação de uma forma de explorar vizinhos serve como entrada para ela mesma em um processo iterativo até que esse processo não gere melhoria (processo de descida com apenas uma forma de gerar vizinhos). Dessa forma, ao final, a melhor solução dentre os 36 mínimos locais é tomada como solução para uma próxima iteração da busca local, e o processo se repete. Quando não existir melhoria de uma iteração para outra, o processo retorna a melhor solução encontrada.

O algoritmo 22 define essa busca local. A diferença entre esse algoritmo e o algoritmo 20 é que a forma de explorar vizinhos faz um processo de descida até que não haja melhoria. O algoritmo 23 define a proposta do VNS 4, a única mudança é a função BUSCA-ADAP-3$(S', n, T)$ (linha 4) em relação ao algoritmo 21.

## 4.8 Ilustração das três buscas adaptativas

A figura 4.4 ilustra como os três algoritmos adaptativos propostos do VNS (VNS 2, VNS 3 e VNS 4) executam as suas respectivas funções BUSCA-ADAP.

Dada uma solução corrente $x$, a BUSCA-ADAP-1 (usada no VNS 2 e referenciada na figura 4.4 como adap 1) executa até $m$ vezes iterativamente utilizando a mesma forma de explorar vizinhança, e isso é feito para todas as formas – nesse trabalho $m$ foi atribuído com valor unitário como explicado na seção 4.5. Dentre todas as soluções dessas execuções, a formar de explorar vizinhança escolhida é a que obteve a melhor solução respectiva, assim

o algoritmo continua a usar essa forma até que não haja mais melhoria e retorna $x'$ da busca local.

A função BUSCA-ADAP-2 (usada no VNS 3 e referenciada na figura 4.4 como adap 2), aplica também todas as formas de explorar vizinhanças na solução corrente $x$, porém isso é feito apenas uma vez, diferente da função BUSCA-ADAP-1 que executa $m$ vezes. Com isso, a forma que obtiver a melhor solução é escolhida como a próxima solução corrente $x$, e o processo se repete iterativamente até que não haja mais melhoria e $x'$ é a nova solução da busca local.

Já a função BUSCA-ADAP-3 (usada no VNS 4 e referenciada na figura 4.4 como adap 3), executa iterativamente todas as formas de explorar vizinhança e salva suas soluções finais com o mesmo critério de parada, isto é, até que não haja mais melhoria nas soluções. A melhor solução de uma iteração é usada como a nova solução corrente $x$ na próxima iteração. O processo para quando não é encontrada melhoria e $x'$ é retornada da busca local.

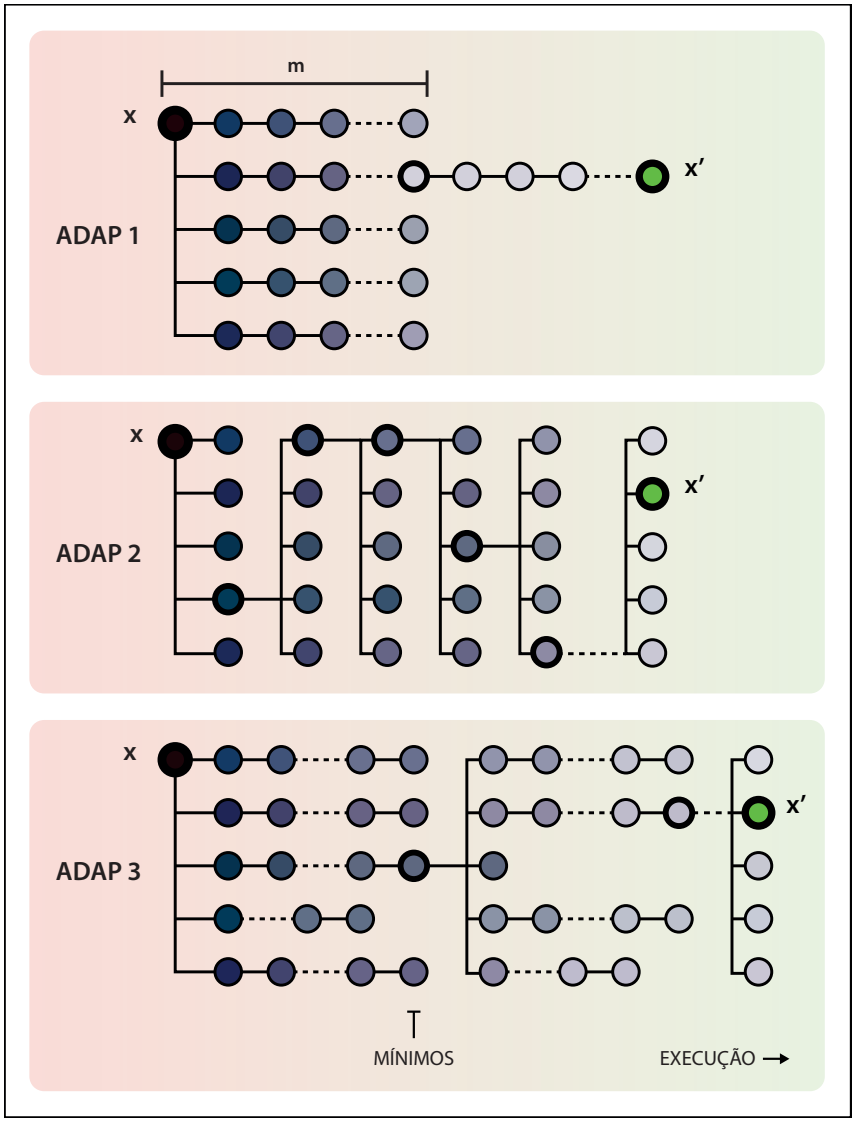


**Figura 4.4:** Três métodos adaptativos usados nos VNS 2, 3 e 4 respectivamente.

# 5

# Experimentos computacionais

Para a validação dos algoritmos propostos, os experimentos foram executados no sistema operacional Debian 8 em computadores Intel Xeon de 3.5 GHz, com 30 GB de memória RAM.

A linguagem de programação de implementação foi Python, com execução dos códigos no interpretador PyPy na versão 4.0. PyPy foi utilizado para melhorar a eficiência dos códigos implementados.

Existe um conjunto de instâncias de escalas de veículos com dois pontos de troca (garagem e terminal). Os quatro algoritmos propostos e o algoritmo de comparação foram aplicados nessas instâncias tal que a maior instância possui 3478 tarefas e a menor possui 130 tarefas. Em especial, a instância com 3478 tarefas possui três pontos de troca, também é possível fazer a troca em um determinado bairro.

Como os algoritmos não são determinísticos, cada programa referente ao algoritmo executou trinta vezes cada uma das instâncias. A descrição das instâncias, parâmetros, regras e resultados dos experimentos são descritos a seguir.

## 5.1 Instâncias utilizadas

O conjunto de instâncias utilizadas para execução dos algoritmos é o mesmo que o conjunto utilizado por Sakiyama et al. (2014), com exceção de uma instância. Esse conjunto contém duas instâncias reais e oito instâncias geradas aleatoriamente por Calvi (2005) baseadas em uma instância real. A instância RE3478 foi gerada recentemente com tarefas reais atualizadas de uma empresa que presta serviço na cidade de Maringá–PR.

**Tabela 5.1:** Conjunto de instâncias utilizadas para execução dos algoritmos propostos.

| **Rótulo** | **Tarefas** | **Quadros** | **Veículos** | **Tipo** |
|---|---|---|---|---|
| AL130 | 130 | 11 | 10 | Gerada aleatoriamente |
| AL251 | 251 | 36 | 23 | Gerada aleatoriamente |
| RE412 | 412 | 69 | 42 | Real |
| AL512 | 512 | 80 | 47 | Gerada aleatoriamente |
| AL761 | 761 | 113 | 70 | Gerada aleatoriamente |
| AL1000 | 1000 | 166 | 89 | Gerada aleatoriamente |
| AL1253 | 1253 | 207 | 112 | Gerada aleatoriamente |
| AL1512 | 1512 | 247 | 132 | Gerada aleatoriamente |
| AL2010 | 2010 | 311 | 166 | Gerada aleatoriamente |
| RE2313 | 2313 | 359 | 188 | Real |
| RE3478 | 3478 | 334 | 271 | Real |

A tabela 5.1 apresenta o conjunto de instâncias: na primeira coluna é a denominação da instância, a segunda coluna é a quantidade de tarefas, a terceira coluna indica a quantidade de quadros de execução da instância, a quarta coluna indica a quantidade de veículos necessários para cobrir os quadros de execução e a quinta coluna indica o tipo da instância.

Para se calcular a quantidade de veículos necessários para cobrir os quadros de execução, é preciso ordená-los por ordem crescente de início $t_i$ e avaliar se é possível sequenciá-los ao percorrer essa sequência. Com isso é possível agrupá-los em conjuntos de quadros de execução que podem ser sequenciados.

## 5.2 Parâmetros e regras

Os algoritmos propostos foram implementados com as seguintes regras:

1. A jornada de trabalho normal de 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. Tempos que ultrapassam às 7:20h são pagos em horas extras;

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: o tempo de término $t_i$ deve ser menor que o tempo de início de $t_j$, e ponto de troca de término de $t_i$ deve ser o mesmo que o ponto de troca de início de $t_j$. Jornadas de motorista que não satisfaçam essas duas restrições são consideradas inviáveis;

7. A extensão máxima da jornada, ao considerar o tempo trabalhado mais o tempo de descanso, 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;

10. As jornadas de motorista podem começar e terminar em pontos de troca distintos.

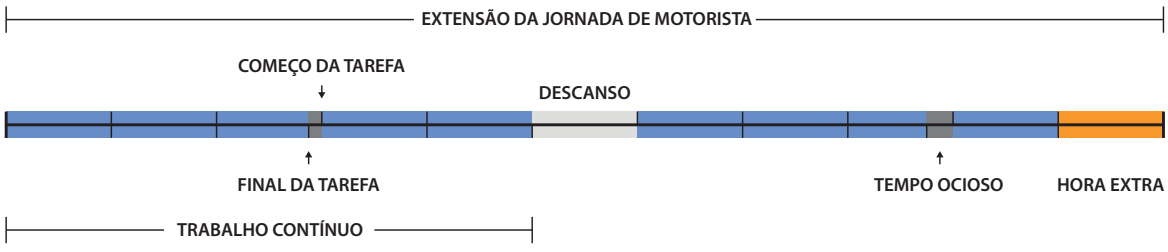


**Figura 5.1:** Ilustração das regras de uma jornada de motorista.

A partir disso, é possível mostrar algumas equações para gerar a função final de custo $C(x)$ de uma jornada de motorista $x$ e também a função objetivo $f(x)$ usada neste

trabalho. A equação 5.1 é a de transformação do tempo noturno $t_n(x)$ em $T_n(x)$ que tem duração de 52,5 minutos. $t_n(x)$ representa o tempo noturno de uma jornada $x$ sem a devida transformação.

$$T_n(x) = t_n(x).60/52,5 \tag{5.1}$$

Com isso é obtido a equação 5.2 a qual é o tempo total $T(x)$ de uma jornada de motorista, a qual é a soma do tempo noturno $T_n(x)$ e o tempo executado durante o dia $t_d(x)$.

$$T(x) = T_n(x) + t_d(x) \tag{5.2}$$

Um custo preliminar $c(x)$ é calculado conforme a equação 5.3, a qual atribui o adicional de trabalho noturno ($T_n(x) \times 1,2$) e soma-se esse valor com o tempo de trabalho durante o dia.

$$c(x) = T_n(x).1,2 + t_d(x) \tag{5.3}$$

Dessa forma é obtido a função de custo total $C(x)$ de uma jornada $x$, conforme a equação 5.4, tal que $t_{max}$ é o tempo máximo de 7 horas e 20 minutos da jornada de trabalho. É importante notar que caso a parcela $(T(x) - t_{max}) \times 1,5 < 0$, essa parcela é ignorada da equação $C(x)$, isto é, soma-se 0, e caso $C(x) < t_{max}$, $C(x) = 440$ minutos, pois é o custo mínimo de uma jornada.

$$C(x) = c(x) + (T(x) - t_{max}).1,5 \tag{5.4}$$

Portanto a equação 5.4 retorna o custo de trabalho (tempo ocioso está incluso) durante o dia, noite e adicional de horas extras após feitas as transformações necessárias. 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 5.5 $f(x)$ para uma jornada $x$:

$$f(x) = \begin{cases} C(x) & \text{se as restrições 2, 4, 5, 6 e 7 são satisfeitas} \\ \infty & \text{caso contrário} \end{cases} \tag{5.5}$$

Como explicado na seção 4.4.1, a função objetivo $f(x)$ do algoritmo **VNS 1 BI P** deve acrescentar a condição para penalizar trocas de linhas em jornadas de motorista.

Logo $f(x)$ é acrescida de 100 minutos caso ocorra uma troca de linha, 1000 minutos caso ocorra duas trocas de linhas e 10000 minutos caso ocorra três ou mais trocas de linhas.

Para o bloco 1 descrito na seção 4.1, $g(i, j)$ 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 4.1), 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 inicial (fase construtiva descrita na seção 4.1) 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 4.1). Isto é, são 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). Isto é explicado com mais detalhes na seção 4.3.1.

## 5.3 Resultados da fase construtiva

Como descrito na seção 4.1, a fase construtiva deve criar uma solução inicial viável para servir de entrada a todos algoritmos propostos. Portanto, é preciso aplicar o algoritmo 8 gerador de camadas e na sequência o algoritmo 9 para gerar a solução inicial.

Os gráficos indicados pelas figura 5.2 e figura 5.3 mostram como estão distribuídas as tarefas nas camadas após a aplicação do algoritmo 8 gerador de camadas. Os dois gráficos são das duas maiores instâncias – as quais são reais – com 3475 e 2313 tarefas respectivamente. É possível perceber que as camadas não têm uma distribuição homogênea de tarefas.

Os resultados das soluções iniciais são mostrados na tabela 5.2. Esses resultados são obtidos da aplicação do algoritmo 9 em todas as instâncias citadas na tabela 5.1. A tabela 5.2 indica o valor da soma da função de custo $C(x)$ de todas as jornadas da escala de motoristas, e também o tamanho da escala de motoristas inicial $|J|$ gerada. A coluna SAK representa os resultados da aplicação do algoritmo de solução inicial proposto por Sakiyama et al. (2014) e a coluna GAP é a razão dos resultados entre os dois algoritmos transformados em percentagem pela seguinte fórmula: $GAP = (SAK/Proposta - 1) \times 100$.

É possível notar que o algoritmo de solução inicial proposto obteve melhores resultados em dez das onze instâncias aplicadas. Os valores numéricos da função de custo $C(x)$ que foram inferiores a proposta de Sakiyama et al. (2014) estão com percentuais na faixa de

3% a 15%, e em relação ao tamanho da escala de motoristas $|J|$, os valores estão na faixa de 6% a 16%.

Esses dois algoritmos foram executados apenas uma vez pois são determinísticos.

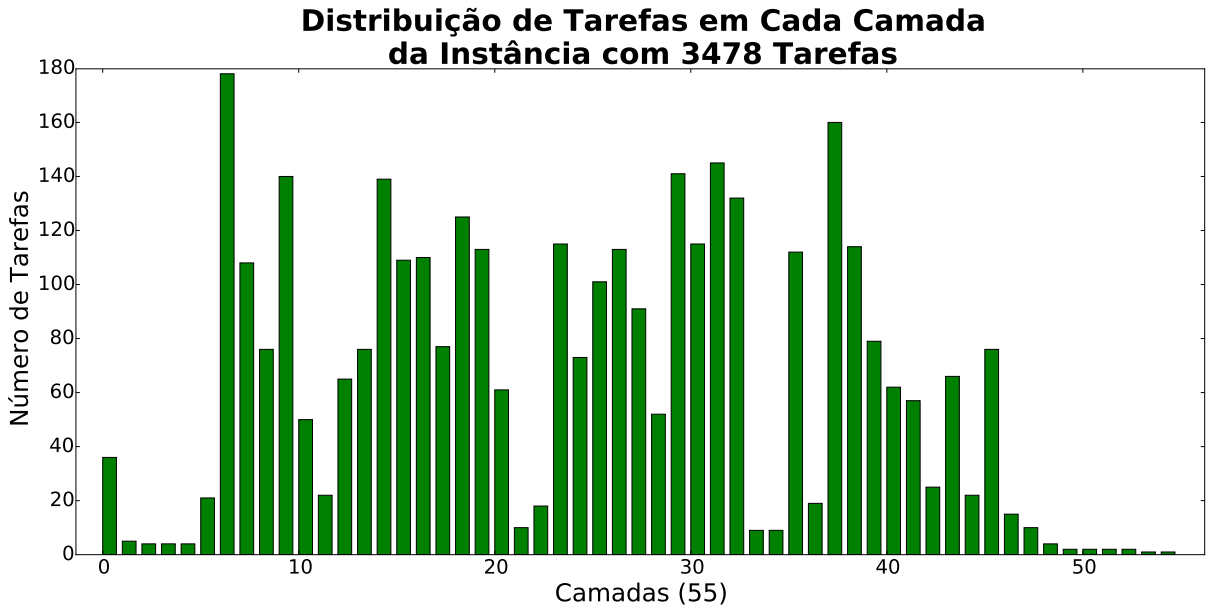


**Figura 5.2:** Distribuição de tarefas em camadas após a aplicação do algoritmo 8 para a instância de 3478 tarefas.

**Tabela 5.2:** Resultados do algoritmo de criação de solução inicial

| **Inst** | **solução-inicial** | | **SAK** | | **GAP** | |
|---|---|---|---|---|---|---|
| | $C(x)$ | $\|J\|$ | $C(x)$ | $\|J\|$ | $C(x)$ | $\|J\|$ |
| 3478 | **277885** | **552** | 321195 | 643 | 15,59 | 16,49 |
| 2313 | **190014** | **381** | 214092 | 424 | 12,67 | 11,29 |
| 2010 | **171096** | **336** | 189344 | 375 | 10,67 | 11,61 |
| 1517 | **131261** | **260** | 149634 | 297 | 14,00 | 14,23 |
| 1253 | **114665** | **224** | 124473 | 250 | 8,55 | 11,61 |
| 1000 | **91313** | **176** | 97485 | 196 | 6,76 | 11,36 |
| 761 | **66338** | **130** | 71760 | 148 | 8,17 | 13,85 |
| 512 | **45374** | **90** | 50417 | 102 | 11,11 | 13,33 |
| 412 | **41247** | **80** | 42817 | 85 | 3,81 | 6,25 |
| 251 | **22997** | **46** | 24926 | 53 | 8,39 | 15,22 |
| 130 | 12021 | **23** | ***11283** | **23** | -6,14 | 0,00 |

* Essa solução $x$ contém uma jornada de motorista que ultrapassa o tempo do limite máximo de trabalho contínuo de 6h.

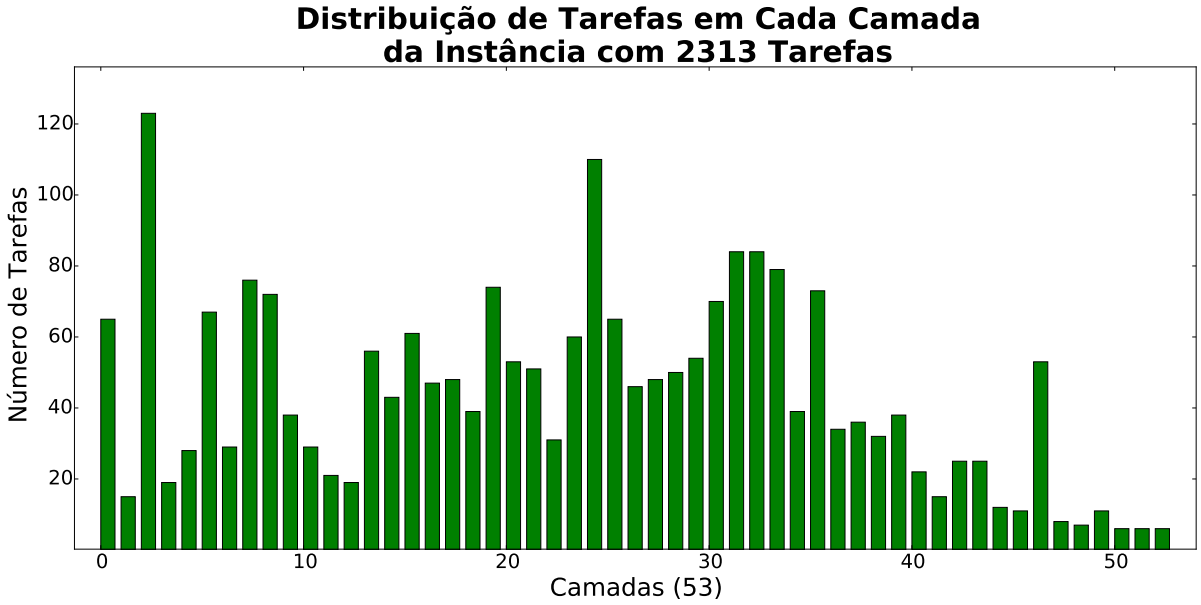


**Figura 5.3:** Distribuição de tarefas em camadas após a aplicação do algoritmo 8 para a instância de 2313 tarefas.

## 5.4 Resultados da fase melhorativa

Os resultados obtidos dos algoritmos propostos estão dispostos nas tabela 5.3, tabela 5.4 e tabela 5.5, junto com os resultados obtidos da implementação do algoritmo de comparação SAK. A tabela 5.3 contêm as somas das funções de custo $C(x)$ e as somas das funções objetivo $f(x)$ das jornadas de motorista (somas representadas for $C(x)$ e $F(x)$ respectivamente). A tabela 5.5 contêm a cardinalidade da jornada ($|J|$) e o tempo de execução (t) resultantes da aplicação dos algoritmos. A tabela 5.4 indica o $GAP$ entre os algoritmos propostos comparados com o algoritmo SAK, e esse $GAP$ é calculado da seguinte forma: $GAP = (SAK/Proposta - 1) \times 100$.

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 5.3, tabela 5.5 e tabela 5.4 indicam a quantidade de tarefas da instância, as quais estão na tabela 5.1. Os quatro algoritmos descritos nas seções 4.4, 4.5, 4.6 e 4.7 são indicados nas colunas VNS 1, VNS 2, VNS 3 e VNS 4 respectivamente, e a coluna SAK indica o algoritmo de comparação (Sakiyama et al., 2014). As variações do algoritmo VNS 1 (seção 4.4.1) são rotulados por VNS 1 BI e VNS 1 BI P.

De acordo com a tabela 5.3 e tabela 5.4, verifica-se que os melhores resultados de custo $C(x)$ são do algoritmo VNS 4, pois obteve índices de melhora na faixa de 4% à 14% ao

ser comparado com o algoritmo SAK (Sakiyama et al., 2014), exceto na menor instância. Porém o VNS 4 tem os piores resultados no tempo de execução de acordo com a tabela 5.5. Isso pode ser justificado pois o algoritmo precisa encontrar um mínimo local para cada uma das 36 estruturas de vizinhanças citadas na seção 4.4. 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 4.5, 4.6 e 4.7), 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 BI, o qual gera bons resultados junto com o VNS 1, pois seus resultados em relação a função de custo ficam atrás apenas do algoritmo VNS 4.

Os valores de $F(x)$ para os algoritmos VNS 1, VNS 2, VNS 3 e VNS 4 na tabela 5.3 foram suprimidos pois eram idênticos aos seus respectivos valores para $C(x)$, isso ocorre pois a função objetivo reflete diretamente a função de custo caso não haja nenhuma violação de regras. Como nenhuma regra foi violada, essas duas funções tem os mesmos valores.

E em relação ao algoritmo VNS 1 BI P que penaliza trocas de linhas, assim como o algoritmo SAK, foi notado uma melhora com o $GAP$ de 2% a 7% em relação a função de custo, como pode ser constatado na tabela 5.4.

**Tabela 5.3:** Resultados da somas da função de custo $C(x)$ e função objetivo $f(x)$ dos algoritmos propostos e do algoritmo de comparação (SAK).

| **Inst** | | **VNS 1** | **VNS 1 BI** | **VNS 1 BI P** | | **VNS 2** | **VNS 3** | **VNS 4** | **SAK** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $C(x)$ | $C(x)$ | $C(x)$ | $F(x)$ | $C(x)$ | $C(x)$ | $C(x)$ | $C(x)$ | $F(x)$ |
| 3478 | min | 222590 | 223444 | 226041 | 231123 | 230460 | 224880 | **217506** | 241440 | 242477 |
| | max | 223890 | 226328 | 228909 | 233865 | 232305 | 227300 | **218650** | 246536 | 245613 |
| | med | 223332 | 224894 | 227395 | 232502 | 231458 | 225694 | **218076** | 243794 | 243984 |
| 2313 | min | 154440 | 154440 | 155940 | 157539 | 155760 | 153120 | **149450** | 166822 | 167103 |
| | max | 154440 | 154440 | 157610 | 159080 | 157960 | 154440 | **149920** | 168683 | 169294 |
| | med | 154440 | 154440 | 156833 | 158413 | 156933 | 153940 | **149655** | 167805 | 168117 |
| 2010 | min | 135080 | 135080 | 136990 | 139452 | 138550 | 137005 | **130200** | 145275 | 145872 |
| | max | 135520 | 135080 | 138425 | 140168 | 139950 | 138504 | **131500** | 147863 | 148009 |
| | med | 135476 | 135080 | 137594 | 139830 | 139197 | 137840 | **130834** | 146561 | 146996 |
| 1517 | min | 103400 | 102080 | 105664 | 107064 | 102750 | 103400 | **99160** | 111836 | 112260 |
| | max | 104280 | 104000 | 106710 | 108429 | 104450 | 104605 | **100540** | 114049 | 114128 |
| | med | 103855 | 102986 | 106216 | 107660 | 103446 | 103999 | **99871** | 112793 | 113230 |
| 1253 | min | 87120 | 82041 | 89345 | 89945 | 88440 | 86240 | **81922** | 93800 | 94297 |
| | max | 88000 | **82669** | 90737 | 92137 | 90200 | 87120 | 82805 | 94655 | 94529 |
| | med | 87725 | **82256** | 90041 | 91117 | 89400 | 86663 | 82461 | 94213 | 94383 |
| 1000 | min | 67760 | 68200 | 69191 | 70991 | 71720 | 68200 | **64729** | 73990 | 75042 |
| | max | 68640 | 68900 | 69710 | 72067 | 72305 | 69080 | **65304** | 74897 | 76500 |
| | med | 68204 | 68497 | 69495 | 71516 | 71913 | 68674 | **65026** | 74391 | 75724 |
| 761 | min | 51920 | 50160 | 53240 | 54140 | 51920 | 51040 | **49720** | 55532 | 56066 |
| | max | 52360 | 50160 | 53750 | 54752 | 52800 | 51480 | **49720** | 56369 | 57073 |
| | med | 52219 | 50160 | 53380 | 54392 | 52414 | 51326 | **49720** | 55976 | 56605 |
| 512 | min | 35200 | **34760** | 36960 | 37597 | 35640 | 35200 | **34760** | 38323 | 38824 |
| | max | 35200 | **35041** | 37914 | 38314 | 35900 | 35640 | 35100 | 39201 | 39460 |
| | med | 35200 | **34844** | 37469 | 37930 | 35760 | 35450 | 34878 | 38668 | 39130 |
| 412 | min | 33000 | 33000 | 33880 | 33880 | 33440 | 33440 | **31680** | 34760 | 34909 |
| | max | 33000 | 33000 | 34006 | 34106 | 33880 | 33880 | **32000** | 35649 | 35662 |
| | med | 33000 | 33000 | 33921 | 33995 | 33730 | 33717 | **31788** | 35393 | 35502 |
| 251 | min | **18920** | **18920** | 19360 | 19560 | **18920** | **18920** | **18920** | 19802 | 20055 |
| | max | **18920** | **18920** | 19800 | 20300 | **18920** | **18920** | **18920** | 20680 | 20740 |
| | med | **18920** | **18920** | 19536 | 19939 | **18920** | **18920** | **18920** | 20297 | 20333 |
| 130 | min | 10120 | 10120 | 10120 | 10220 | 10120 | 10120 | 10120 | **9681** | 9733 |
| | max | **10120** | **10120** | **10120** | 10220 | **10120** | **10120** | **10120** | 10133 | 10180 |
| | med | 10120 | 10120 | 10120 | 10220 | 10120 | 10120 | 10120 | **9917** | 9999 |

**Tabela 5.4:** GAP entre os algoritmos propostos e o algoritmo de comparação SAK

| **Inst** | | **VNS 1** GAP | **VNS 1 BI** GAP | **VNS 1 BI P** GAP | **VNS 2** GAP | **VNS 3** GAP | **VNS 4** GAP |
|---|---|---|---|---|---|---|---|
| 3478 | min | 8,47 | 8,05 | 6,81 | 4,76 | 7,36 | 11,00 |
| | max | 10,11 | 8,93 | 7,70 | 6,13 | 8,46 | 12,75 |
| | med | 9,16 | 8,40 | 7,21 | 5,33 | 8,02 | 11,79 |
| 2313 | min | 8,02 | 8,02 | 6,98 | 7,10 | 8,95 | 11,62 |
| | max | 9,22 | 9,22 | 7,03 | 6,79 | 9,22 | 12,52 |
| | med | 8,65 | 8,65 | 7,00 | 6,93 | 9,01 | 12,13 |
| 2010 | min | 7,55 | 7,55 | 6,05 | 4,85 | 6,04 | 11,58 |
| | max | 9,11 | 9,46 | 6,82 | 5,65 | 6,76 | 12,44 |
| | med | 8,18 | 8,50 | 6,52 | 5,29 | 6,33 | 12,02 |
| 1517 | min | 8,16 | 9,56 | 5,84 | 8,84 | 8,16 | 12,78 |
| | max | 9,37 | 9,66 | 6,88 | 9,19 | 9,03 | 13,44 |
| | med | 8,61 | 9,52 | 6,19 | 9,04 | 8,46 | 12,94 |
| 1253 | min | 7,67 | 14,33 | 4,99 | 6,06 | 8,77 | 14,50 |
| | max | 7,56 | 14,50 | 4,32 | 4,94 | 8,65 | 14,31 |
| | med | 7,40 | 14,54 | 4,63 | 5,38 | 8,71 | 14,25 |
| 1000 | min | 9,19 | 8,49 | 6,94 | 3,17 | 8,49 | 14,31 |
| | max | 9,12 | 8,70 | 7,44 | 3,58 | 8,42 | 14,69 |
| | med | 9,07 | 8,60 | 7,05 | 3,45 | 8,32 | 14,40 |
| 761 | min | 6,96 | 10,71 | 4,31 | 6,96 | 8,80 | 11,69 |
| | max | 7,66 | 12,38 | 4,87 | 6,76 | 9,50 | 13,37 |
| | med | 7,19 | 11,59 | 4,86 | 6,80 | 9,06 | 12,58 |
| 512 | min | 8,87 | 10,25 | 3,69 | 7,53 | 8,87 | 10,25 |
| | max | 11,37 | 11,87 | 3,39 | 9,19 | 9,99 | 11,68 |
| | med | 9,85 | 10,97 | 3,20 | 8,13 | 9,08 | 10,87 |
| 412 | min | 5,33 | 5,33 | 2,60 | 3,95 | 3,95 | 9,72 |
| | max | 8,03 | 8,03 | 4,83 | 5,22 | 5,22 | 11,40 |
| | med | 7,25 | 7,25 | 4,34 | 4,93 | 4,97 | 11,34 |
| 251 | min | 4,66 | 4,66 | 2,28 | 4,66 | 4,66 | 4,66 |
| | max | 9,30 | 9,30 | 4,44 | 9,30 | 9,30 | 9,30 |
| | med | 7,28 | 7,28 | 3,90 | 7,28 | 7,28 | 7,28 |
| 130 | min | -4,34 | -4,34 | -4,34 | -4,34 | -4,34 | -4,34 |
| | max | 0,13 | 0,13 | 0,13 | 0,13 | 0,13 | 0,13 |
| | med | -2,01 | -2,01 | -2,01 | -2,01 | -2,01 | -2,01 |

**Tabela 5.5:** Número de jornadas de motorista ($|J|$) e tempo de execução em minutos ($t$)

| **Inst** | | **VNS 1** | | **VNS 1 BI** | | **VNS 1 BI P** | | **VNS 2** | | **VNS 3** | | **VNS 4** | | **SAK** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\|J\|$ | t | $\|J\|$ | t | $\|J\|$ | t | $\|J\|$ | t | $\|J\|$ | t | $\|J\|$ | t | $\|J\|$ | t |
| 3478 | min | 506 | 987 | 508 | **800** | 516 | 1454 | 523 | 1295 | 511 | 2540 | **495** | 15657 | 551 | 5048 |
| | max | 509 | 1997 | 514 | **1501** | 520 | 1654 | 528 | 2510 | 516 | 3168 | **498** | 27731 | 558 | 10963 |
| | med | 507 | 1505 | 511 | **1116** | 518 | 1490 | 526 | 1806 | 512 | 2967 | **496** | 21635 | 554 | 7305 |
| 2313 | min | 351 | 180 | 351 | **150** | 353 | 980 | 354 | 489 | 348 | 657 | **340** | 19419 | 379 | 512 |
| | max | 351 | 468 | 351 | **210** | 354 | 1140 | 359 | 1589 | 351 | 1898 | **342** | 23681 | 383 | 3486 |
| | med | 351 | 294 | 351 | **176** | 353 | 1014 | 356 | 948 | 349 | 1146 | **341** | 20154 | 381 | 1503 |
| 2010 | min | 307 | 150 | 307 | **92** | 310 | 673 | 315 | 500 | 312 | 543 | **294** | 9500 | 330 | 391 |
| | max | 308 | 382 | 307 | **151** | 314 | 720 | 318 | 1174 | 313 | 758 | **294** | 11382 | 336 | 2513 |
| | med | 307 | 287 | 307 | **124** | 312 | 686 | 316 | 820 | 312 | 646 | **294** | 10216 | 333 | 1072 |
| 1517 | min | 235 | 191 | 232 | **75** | 240 | 175 | 234 | 267 | 235 | 354 | **225** | 3214 | 254 | 301 |
| | max | 237 | 360 | 232 | **89** | 242 | 465 | 235 | 364 | 237 | 481 | **226** | 3658 | 259 | 1047 |
| | med | 236 | 246 | 232 | **80** | 241 | 315 | 234 | 297 | 235 | 423 | **226** | 3320 | 256 | 658 |
| 1253 | min | 198 | **50** | **186** | **50** | 203 | 86 | 201 | 154 | 196 | 194 | **186** | 2541 | 214 | 240 |
| | max | 200 | 154 | **188** | **65** | 206 | 243 | 205 | 262 | 198 | 294 | 189 | 2985 | 214 | 846 |
| | med | 199 | 92 | **187** | **58** | 205 | 148 | 203 | 196 | 197 | 204 | 188 | 2647 | 214 | 347 |
| 1000 | min | 154 | **39** | 155 | 40 | 157 | 75 | 163 | 74 | 163 | 74 | **147** | 1005 | 168 | 43 |
| | max | 156 | 73 | 156 | **48** | 158 | 104 | 163 | 81 | 165 | 81 | **148** | 1874 | 170 | 266 |
| | med | 155 | 48 | 155 | **43** | 157 | 91 | 163 | 78 | 164 | 78 | **147** | 1450 | 169 | 169 |
| 761 | min | 118 | 23 | 114 | **20** | 121 | 28 | 118 | 39 | 116 | 64 | **113** | 504 | 126 | 24 |
| | max | 119 | 31 | 114 | **22** | 121 | 48 | 120 | 78 | 117 | 113 | **113** | 954 | 128 | 66 |
| | med | 118 | 25 | 114 | **20** | 121 | 33 | 119 | 56 | 116 | 74 | **113** | 658 | 127 | 42 |
| 512 | min | 80 | 7 | **79** | **3** | 84 | 8 | 81 | 12 | 80 | 17 | **79** | 414 | 87 | 10 |
| | max | 80 | 7 | **79** | **4** | 86 | 14 | 82 | 19 | 81 | 28 | 80 | 757 | 89 | 28 |
| | med | 80 | 7 | **79** | **3,8** | 85 | 10 | 81 | 14 | 80 | 20 | **79** | 547 | 87 | 20 |
| 412 | min | 75 | 4 | 75 | **1** | 77 | 3 | 76 | 6 | 76 | 13 | **72** | 324 | 79 | 7 |
| | max | 75 | 5 | 75 | **2** | 77 | 7 | 77 | 14 | 77 | 22 | **73** | 457 | 81 | 11 |
| | med | 75 | 4,8 | 75 | **1,4** | 77 | 5 | 76 | 8 | 76 | 15 | **73** | 369 | 80 | 8 |
| 251 | min | **43** | 1 | **43** | **0,4** | 44 | 1 | **43** | 3 | **43** | 4 | **43** | 125 | 45 | 1 |
| | max | **43** | 2 | **43** | **1** | 45 | 3 | **43** | 3 | **43** | 5 | **43** | 214 | 47 | 2 |
| | med | **43** | 1,8 | **43** | **0,54** | 45 | 1,8 | **43** | 3 | **43** | 5 | **43** | 136 | 46 | 1,6 |
| 130 | min | 23 | 0,45 | 23 | **0,13** | 23 | 0,18 | 23 | 0,53 | 23 | 1 | 23 | 95 | **22** | 0,25 |
| | max | **23** | 0,49 | **23** | **0,14** | **23** | 0,31 | **23** | 0,53 | 23 | 1 | **23** | 136 | **23** | 1 |
| | med | 23 | 0,47 | 23 | **0,13** | 23 | 0,21 | 23 | 0,53 | 23 | 1 | 23 | 115 | **22** | 0,45 |

As figuras a seguir mostram a distribuição de custo da escala de motoristas, isto é, a soma de $C(x)$ de todas as jornadas de motorista da escala. Esses custos são provenientes da execução dos algoritmos propostos e do algoritmo Sakiyama et al. (2014) (SAK) que foram implementados. Cada algoritmo foi executado 30 vezes e esses valores são representados nos gráficos a seguir (figura 5.4, figura 5.5, figura 5.6, figura 5.7, figura 5.8, figura 5.9, figura 5.10, figura 5.11, figura 5.12, figura 5.13 e figura 5.14).

Os gráficos são *box plots* (Bussab e Morettin, 2013), em que no eixo das ordenadas tem-se o valor de custo das escalas de motorista gerado, e no eixo das abscissas tem-se as denominações dos algoritmos, as quais são as mesmas das tabelas anteriores. *Box plots* são diagramas que contêm retângulos onde estão representados a mediana e os quartis.

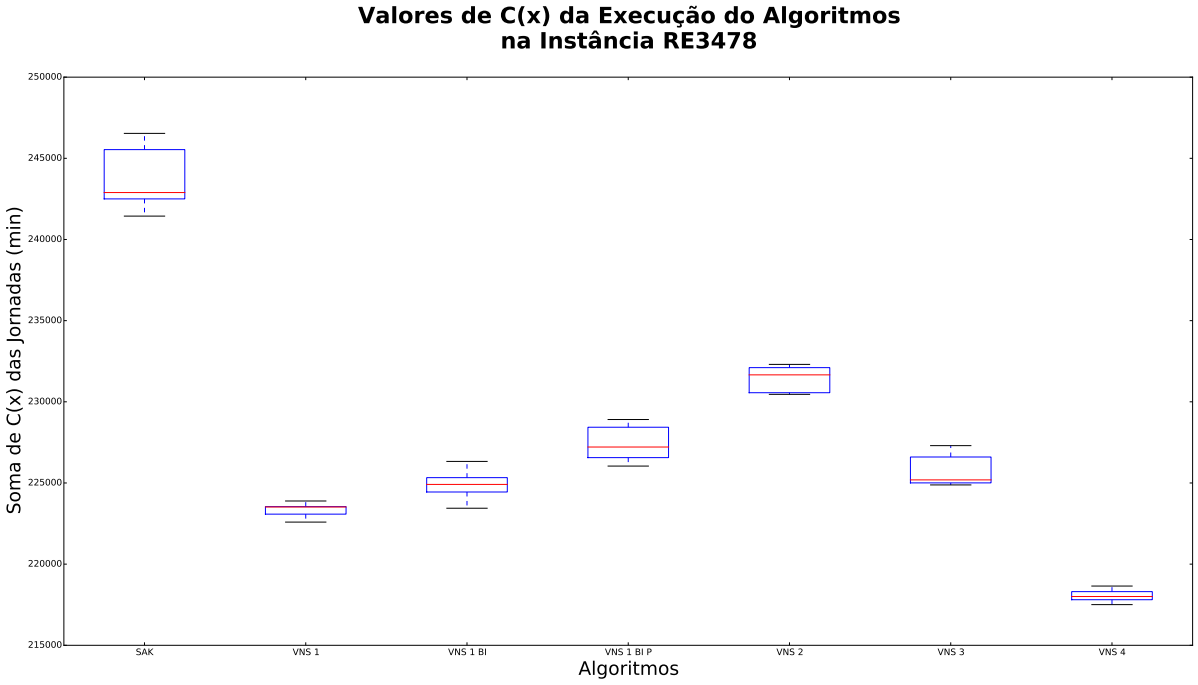


**Figura 5.4:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância RE3478.

Em geral, o algoritmo VNS 4 obteve os melhores resultados para a soma da função de custo $C(x)$ de acordo com os gráficos, exceto nas figura 5.8 e figura 5.14, os algoritmos VNS 1 BI e SAK obtiveram melhores resultados respectivamente. Todas as seis propostas foram melhores que o algoritmo de comparação SAK, exceto na última instância AL130 (figura 5.14).

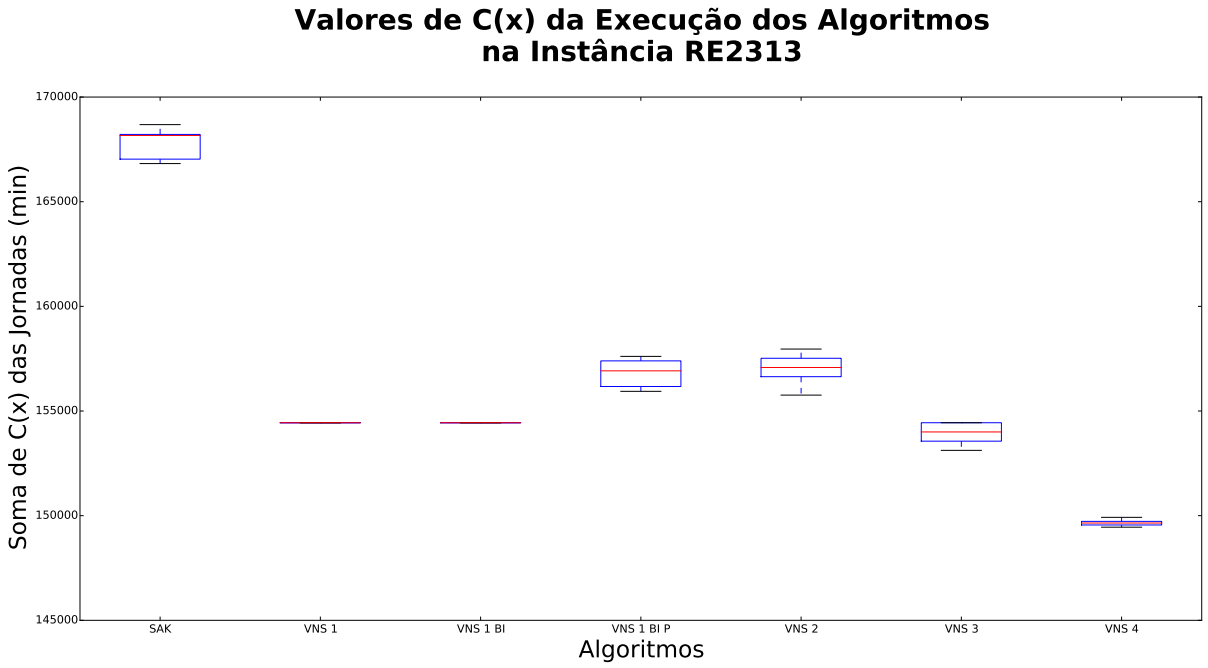


**Figura 5.5:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância RE2313.

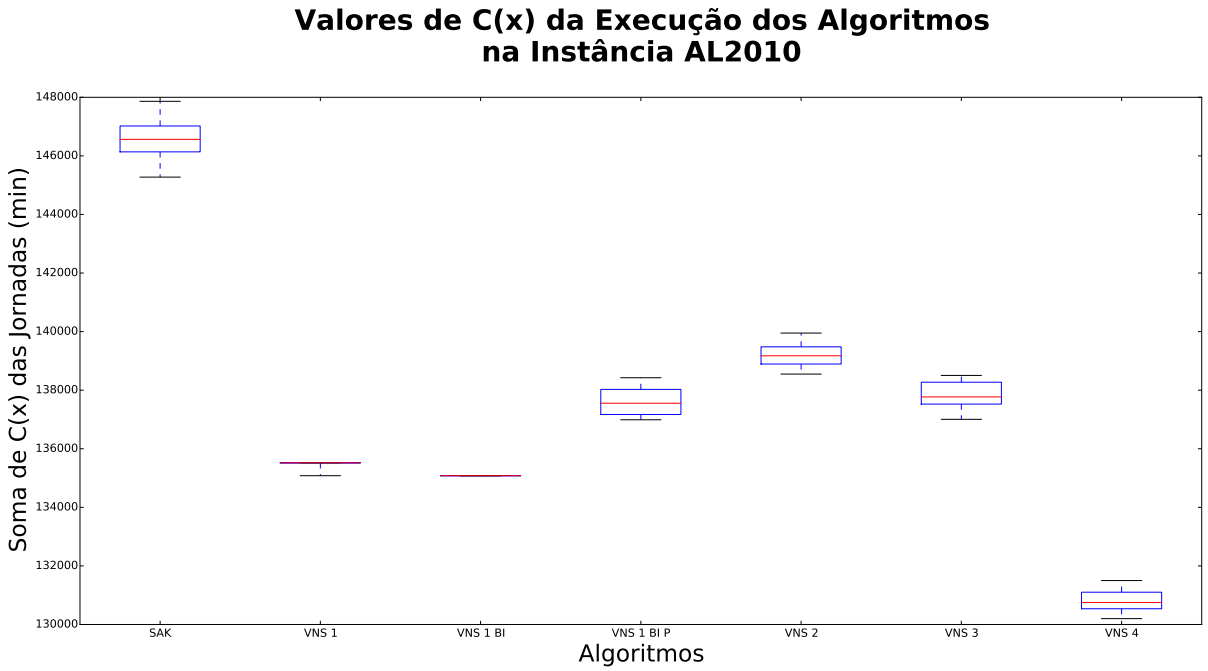


**Figura 5.6:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL2010.

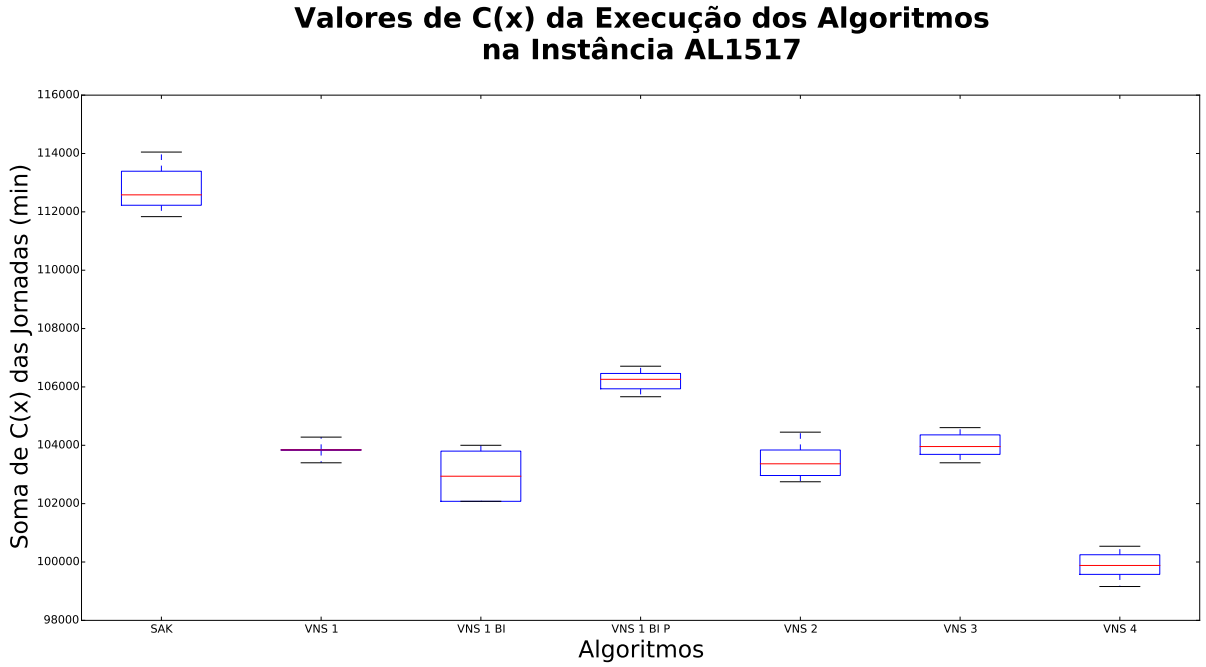


**Figura 5.7:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL1517.

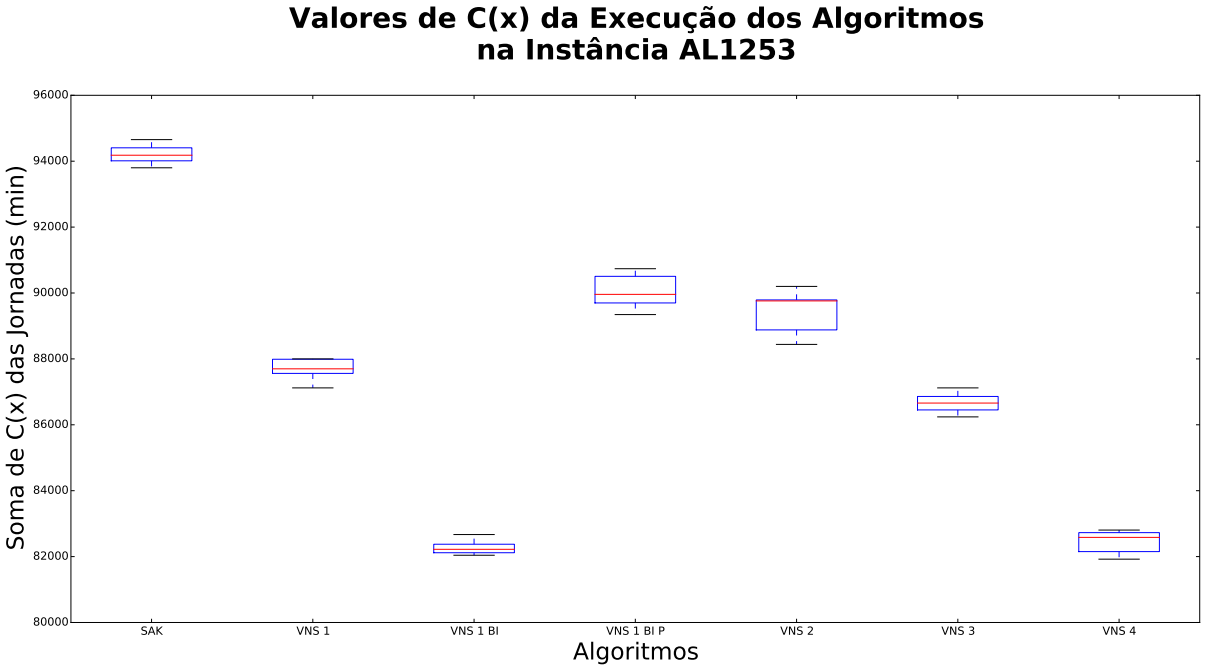


**Figura 5.8:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL1253.

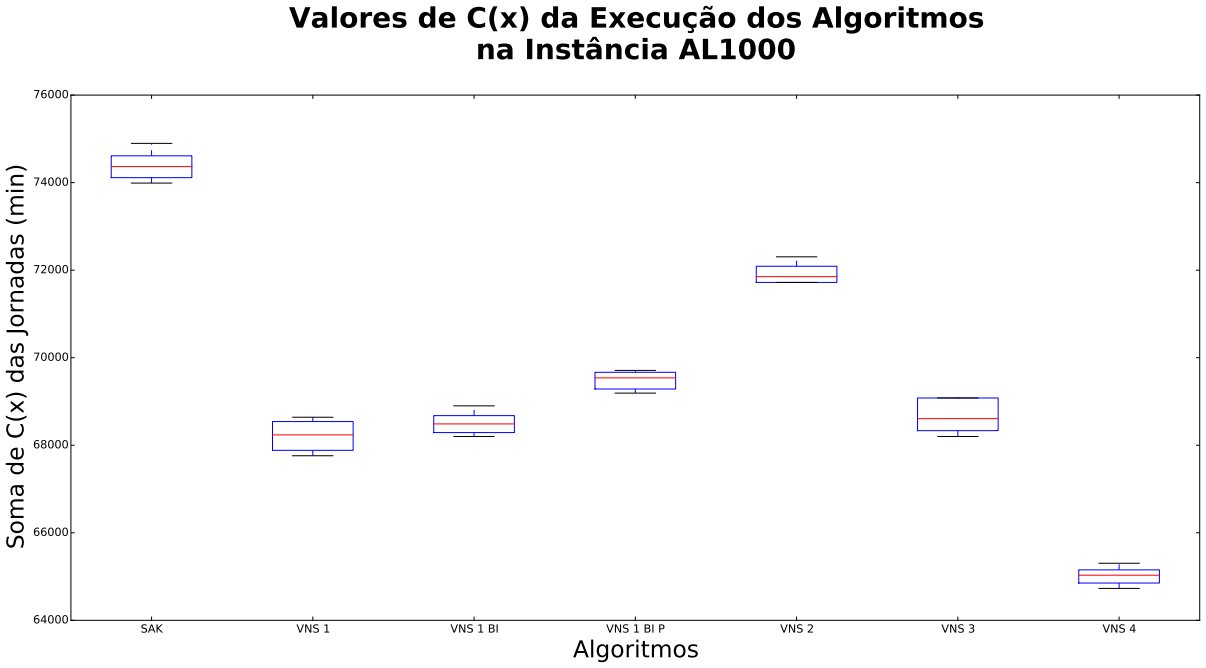


**Figura 5.9:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL1000.

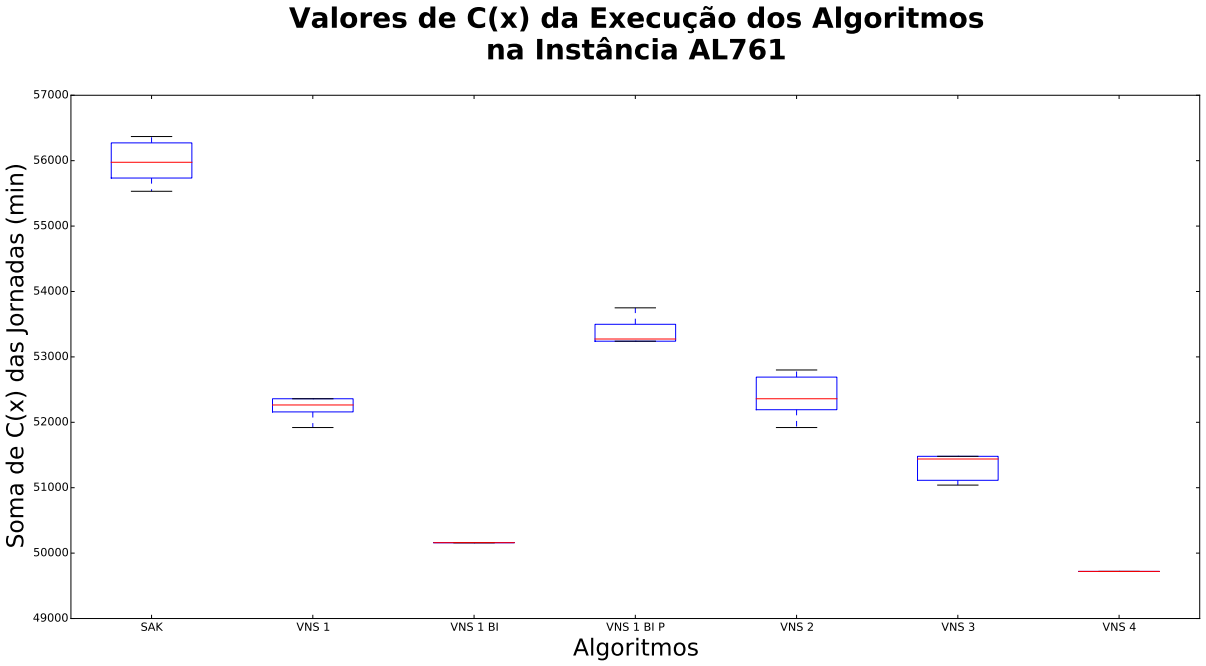


**Figura 5.10:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL761.

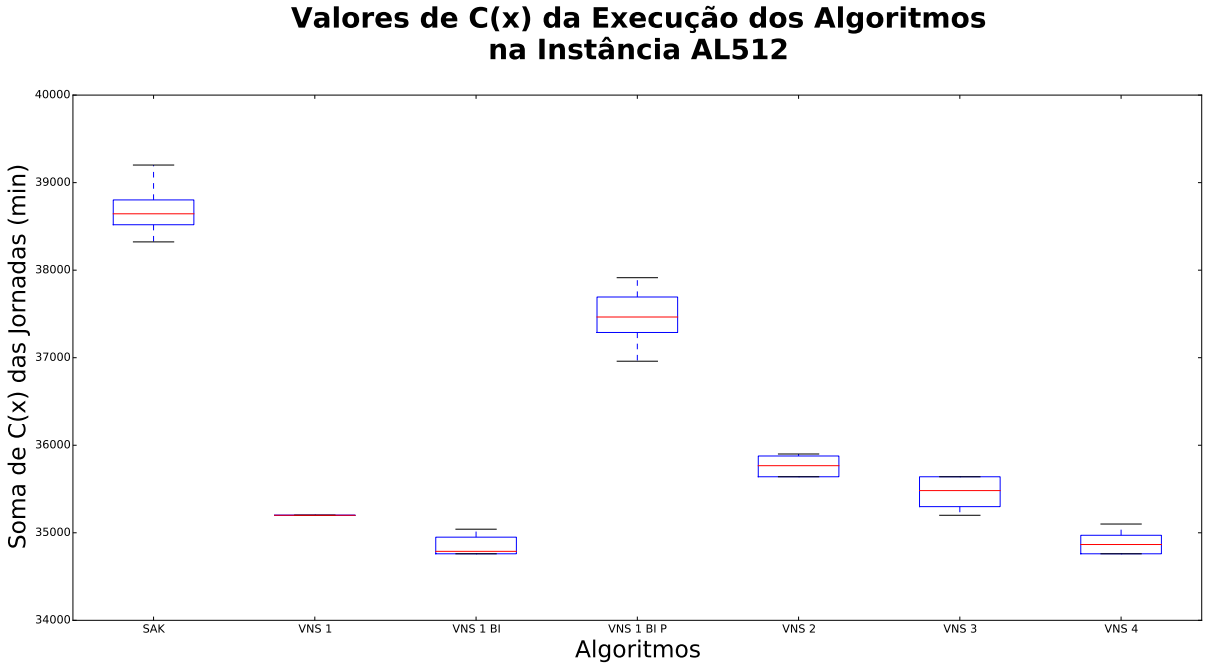


**Figura 5.11:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL512.

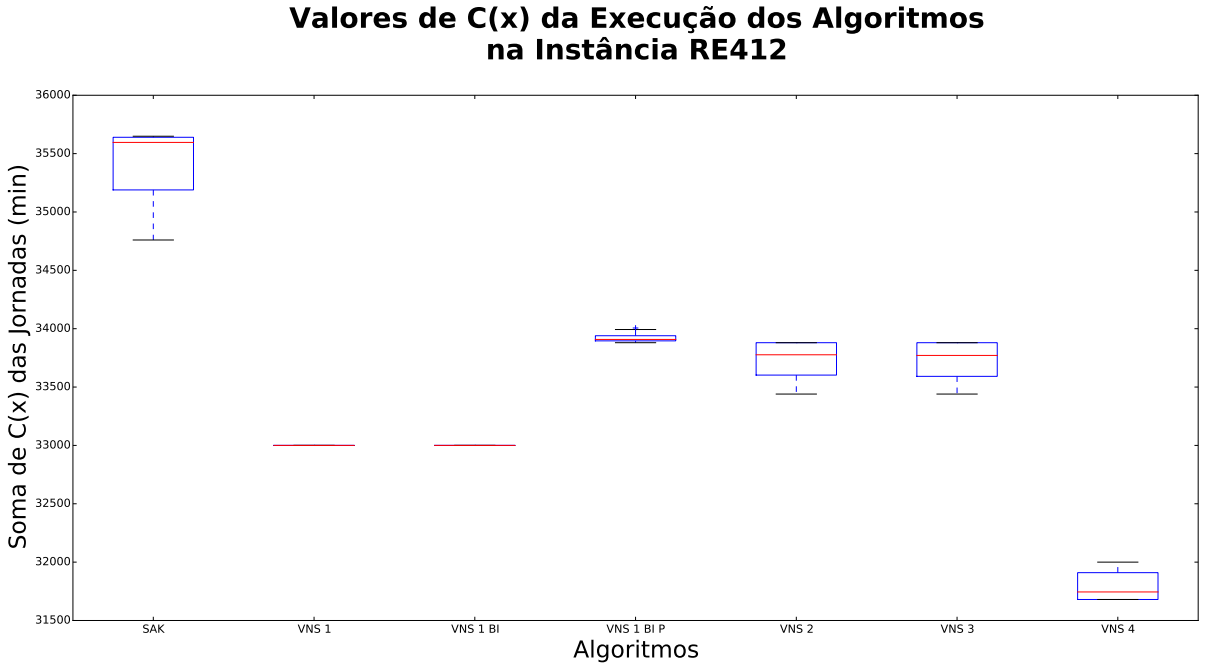


**Figura 5.12:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância RE412.

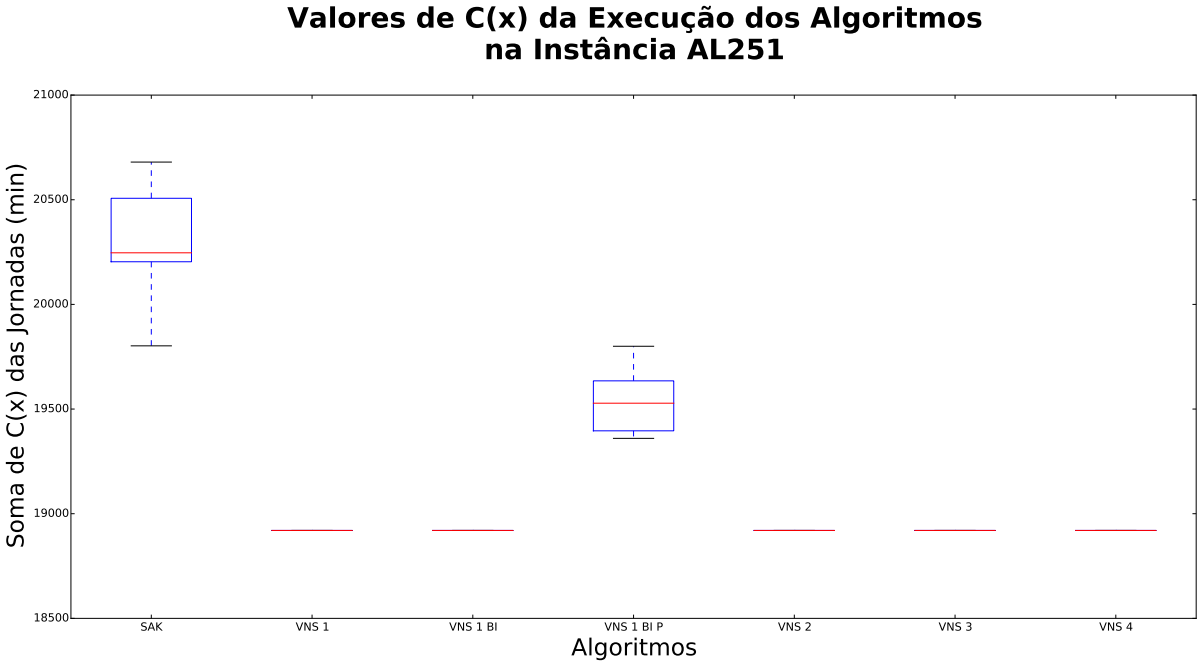


**Figura 5.13:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL251.

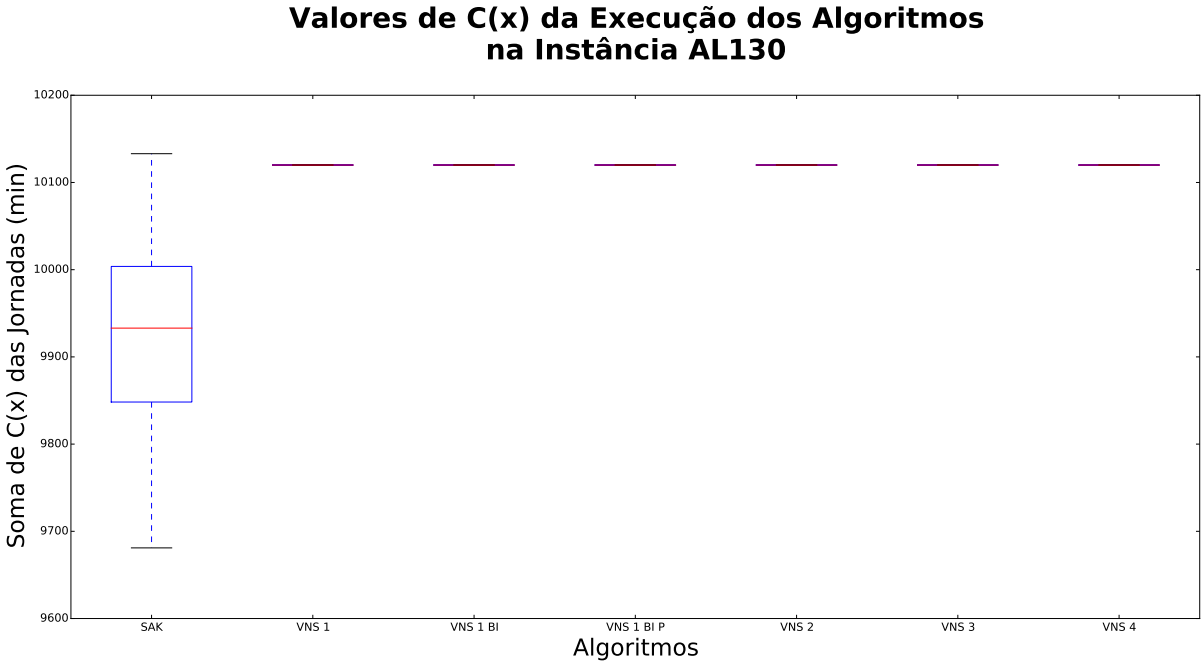


**Figura 5.14:** Resultados obtidos da aplicação dos algoritmos propostos e algoritmo de comparação (SAK) na instância AL130.

O algoritmo VNS 1 BI P, que fez a penalização de trocas de linhas na função objetivo, ficou alternando sua posição em relação aos algoritmos VNS 2: nas figura 5.4, figura 5.5, figura 5.6 e figura 5.9 esse algoritmo foi melhor que o algoritmo VNS 2.

O algoritmo VNS 1 BI chegou a ser melhor que o algoritmo VNS 1 em alguns casos (figura 5.7, figura 5.8, figura 5.10, figura 5.11), isso pode ser mais um indício que muitas formas para explorar vizinhos no algoritmo VND não necessariamente ajudam na redução da função objetivo, já que o algoritmo VNS 1 tem $k = 36$ e o algoritmo VNS 1 BI tem $k = 12$.

## 5.5 Comparação dos resultados com um limite inferior

A tabela 5.6 faz uma comparação dos resultados obtidos de custo da escala de motoristas (somatório de $C(x)$ para cada jornada de motorista) com um limite inferior (LI) calculado por Calvi (2005).

Para a comparação é usado o $GAP = (C(x)/LI - 1) \times 100$, que representa uma percentagem de quão distante cada algoritmo proposto está distante do LI. Os valores min, max e med da tabela 5.6, são os mesmos valores da tabela 5.3 para os valores de $C(x)$. Isto é, são comparados os valores mínimos, médios e máximos de $C(x)$ da execução dos algoritmos propostos com os LIs que estão na primeira coluna da tabela 5.6.

Com esses valores de $GAP$, percebe-se que o algoritmo VNS 4 é o que está mais próximo dos LIs, principalmente nas instâncias de maior porte. Em geral o valor de $GAP$ varia entre 12% até 25% para o algoritmo VNS 4. Em Alguns casos (Instância AL1253 e AL512) o algoritmo VNS 1 BI P é o que está mais próximo dos LIs.

**Tabela 5.6:** Comparação da soma de custos $C(x)$ com um limite inferior (LI).

| **Inst** | | **VNS 1** *GAP* | **VNS 1 BI** *GAP* | **VNS 1 BI P** *GAP* | **VNS 2** *GAP* | **VNS 3** *GAP* | **VNS 4** *GAP* |
|---|---|---|---|---|---|---|---|
| 2313 | min | 17,18 | 17,18 | 18,32 | 18,18 | 16,18 | **13,39** |
| LI: | max | 17,18 | 17,18 | 19,58 | 19,85 | 17,18 | **13,75** |
| 131800 | med | 17,18 | 17,18 | 18,99 | 19,07 | 16,80 | **13,55** |
| 2010 | min | 16,43 | 16,43 | 18,08 | 19,42 | 18,09 | **12,22** |
| LI: | max | 16,81 | 16,43 | 19,31 | 20,63 | 19,38 | **13,34** |
| 116019 | med | 16,77 | 16,43 | 18,60 | 19,98 | 18,81 | **12,77** |
| 1517 | min | 15,93 | 14,45 | 18,47 | 15,20 | 15,93 | **11,18** |
| LI: | max | 16,92 | 16,60 | 19,64 | 17,11 | 17,28 | **12,72** |
| 89191 | med | 16,44 | 15,47 | 19,09 | 15,98 | 16,60 | **11,97** |
| 1253 | min | 20,56 | 13,53 | 23,64 | 22,39 | 19,35 | **13,37** |
| LI: | max | 21,78 | **14,40** | 25,57 | 24,83 | 20,56 | 14,59 |
| 72261 | med | 21,40 | **13,83** | 24,61 | 23,72 | 19,93 | 14,12 |
| 1000 | min | 18,88 | 19,65 | 21,39 | 25,82 | 19,65 | **13,56** |
| LI: | max | 20,42 | 20,88 | 22,30 | 26,85 | 21,19 | **14,57** |
| 57000 | med | 19,66 | 20,17 | 21,92 | 26,16 | 20,48 | **14,08** |
| 761 | min | 20,72 | 16,62 | 23,79 | 20,72 | 18,67 | **15,60** |
| LI: | max | 21,74 | 16,62 | 24,97 | 22,76 | 19,69 | **15,60** |
| 43010 | med | 21,41 | 16,62 | 24,11 | 21,86 | 19,34 | **15,60** |
| 512 | min | 14,07 | **12,65** | 19,77 | 15,50 | 14,07 | **12,65** |
| LI: | max | 14,07 | **13,56** | 22,87 | 16,34 | 15,50 | 13,75 |
| 30858 | med | 14,07 | **12,92** | 21,42 | 15,89 | 14,88 | 13,03 |
| 412 | min | 29,78 | 29,78 | 33,24 | 31,51 | 31,51 | **24,59** |
| LI: | max | 29,78 | 29,78 | 33,74 | 33,24 | 33,24 | **25,85** |
| 25427 | med | 29,78 | 29,78 | 33,41 | 32,65 | 32,60 | **25,02** |
| 251 | min | **20,86** | **20,86** | 23,67 | **20,86** | **20,86** | **20,86** |
| LI: | max | **20,86** | **20,86** | 26,48 | **20,86** | **20,86** | **20,86** |
| 15655 | med | **20,86** | **20,86** | 24,79 | **20,86** | **20,86** | **20,86** |
| 130 | min | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** |
| LI: | max | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** |
| 8057 | med | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** | **25,61** |

## 5.6 Trocas de linhas nas jornadas de motorista

Como os algoritmos propostos não consideraram as penalizações por trocas de linhas, houve uma variação considerável nesse valor de trocas de linhas (exceto o algoritmo VNS 1 BI P). A tabela 5.7 mostra os valores mínimos, máximos e médios na quantidade de trocas de linhas por jornada de motorista. O algoritmo SAK ao penalizar trocas de linhas obteve no máximo duas trocas por jornada de motorista.

Já o algoritmo VNS 1 BI P proposto ao penalizar trocas de linhas, obteve no máximo três trocas por jornada de motorista na instância RE3478, no restante das instâncias essa proposta foi igual ao algoritmo SAK, ou seja, obteve no máximo duas trocas por jornada.

Nesses dois algoritmos em que as trocas de linhas foram baixas, a linha da média (med) não foi preenchida.

**Tabela 5.7:** Valores de trocas de linhas nas jornadas de motorista

| **Inst** | | **VNS 1** trocas | **VNS 1 BI** trocas | **VNS 2** trocas | **VNS 3** trocas | **VNS 4** trocas | **VNS 1 BI P** trocas | **SAK** trocas |
|---|---|---|---|---|---|---|---|---|
| 3478 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 11 | 11 | 10 | 10 | 10 | 3 | 2 |
| | med | 5,42 | 5,24 | 5,30 | 5,41 | 5,43 | | |
| 2313 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 10 | 9 | 9 | 9 | 9 | 2 | 2 |
| | med | 5,21 | 5,15 | 5,11 | 5,35 | 5,34 | | |
| 2010 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 9 | 9 | 9 | 10 | 10 | 2 | 2 |
| | med | 5,01 | 5,08 | 4,97 | 5,19 | 5,30 | | |
| 1517 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 9 | 8 | 9 | 9 | 8 | 2 | 2 |
| | med | 4,94 | 5,05 | 4,92 | 4,99 | 5,11 | | |
| 1253 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 9 | 9 | 8 | 8 | 8 | 2 | 2 |
| | med | 4,89 | 4,96 | 4,87 | 4,95 | 4,95 | | |
| 1000 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 8 | 7 | 8 | 8 | 9 | 2 | 2 |
| | med | 4,85 | 4,61 | 4,85 | 4,96 | 4,86 | | |
| 761 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 8 | 8 | 8 | 8 | 8 | 2 | 2 |
| | med | 4,7 | 4,63 | 4,89 | 4,95 | 4,69 | | |
| 512 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 7 | 7 | 8 | 7 | 7 | 2 | 2 |
| | med | 4,67 | 4,37 | 4,66 | 4,49 | 4,43 | | |
| 412 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 6 | 6 | 7 | 7 | 6 | 2 | 2 |
| | med | 3,50 | 3,28 | 3,71 | 3,61 | 3,69 | | |
| 251 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 6 | 6 | 7 | 6 | 7 | 2 | 2 |
| | med | 3,48 | 3,37 | 3,65 | 3,51 | 3,65 | | |
| 130 | min | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | max | 5 | 6 | 4 | 5 | 6 | 2 | 2 |
| | med | 3,00 | 2,39 | 2,26 | 2,82 | 2,13 | | |

## 5.7 Convergência dos algoritmos propostos

As figura 5.15 e figura 5.16 mostram a convergência dos algoritmos propostos em uma execução da instância com 3478 tarefas. Os gráficos foram separados para uma melhor visualização da convergência dos algoritmos VNS 1, VNS 2 e VNS 3. As linhas horizontais presentes no gráfico apenas auxiliam na visualização do custo no momento em que os algoritmos param.

É possível notar nos gráficos que os algoritmos VNS 1 e VNS 2 convergem mais rápido ao se comparar com os algoritmos VNS 3 e VNS 4, isso é um indício do projeto desses dois algoritmos.

O algoritmo VNS 1 é um GNVS, logo em sua função BUSCA-LOCAL, se existe uma melhoria na solução corrente proveniente de uma forma de explorar a vizinhança, a solução corrente é atualizada por essa solução. Dessa forma, não é necessário executar as outras formas de explorar vizinhança na solução corrente, como é feito nos algoritmos VNS 2, 3 e 4. Isso pode ser visto como um método guloso de escolher uma forma de explorar a vizinhança, o que faz com que a convergência seja mais rápida.

Já o algoritmo VNS 2, em sua função BUSCA-LOCAL, depois de executar todas as formas de explorar vizinhanças $m$ vezes, é escolhido a melhor dentre todas as formas e esta é executada iterativamente até não existir melhoria. Essa escolha também determina uma convergência acentuada, pois a partir desse momento nenhuma outra forma de explorar vizinhança é usada na função BUSCA-LOCAL.

Nos algoritmos VNS 3 e VNS 4, em suas funções BUSCA-LOCAL constantemente são executadas todas as formas de explorar vizinhanças, isso faz com que a convergência seja lenta. Em especial, o algoritmo VNS 4 executa exaustivamente todas as formas de explorar vizinhança, o que leva a sua ineficiência.

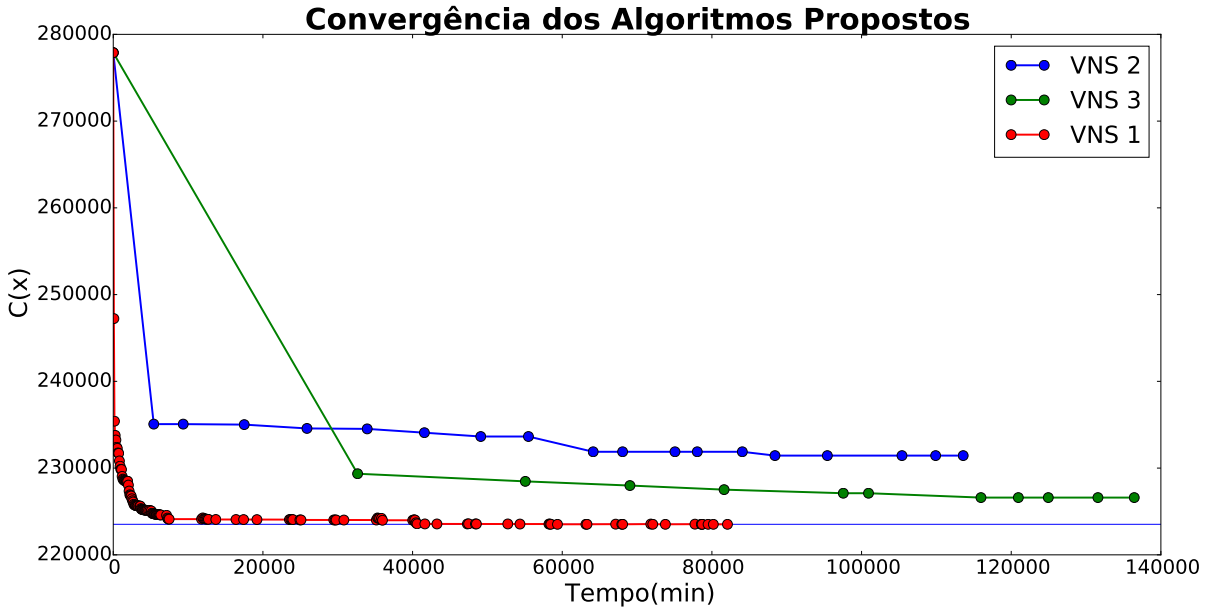


**Figura 5.15:** Convergência dos algoritmos VNS 1, VNS 2 e VNS 3. O custo da escala está representado no eixo vertical e o tempo está representado no eixo horizontal.

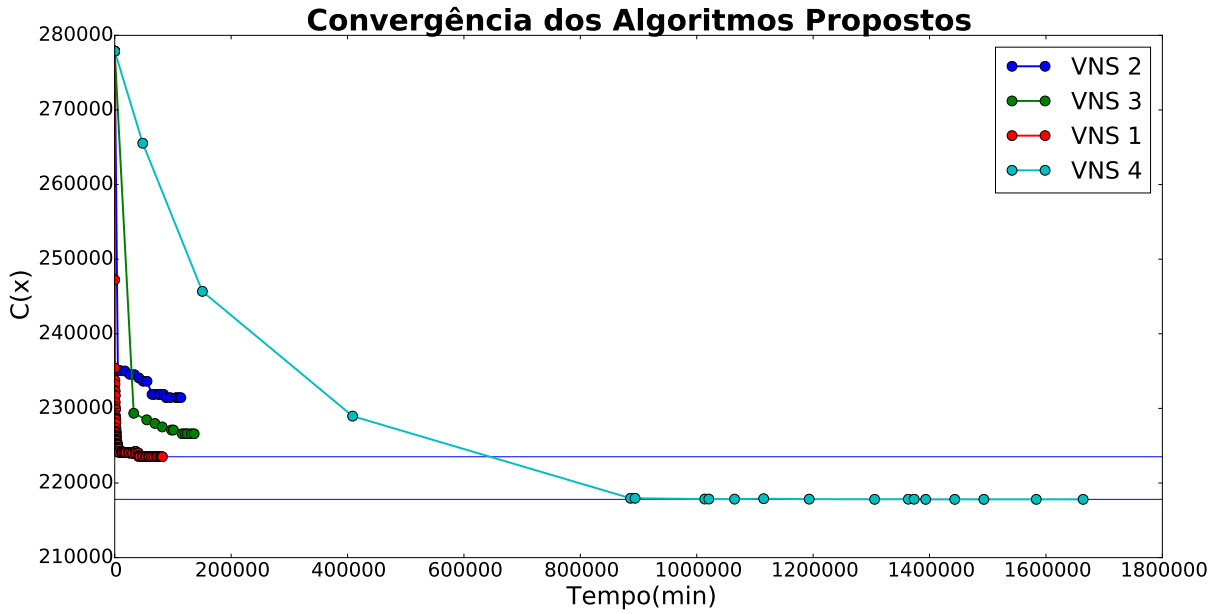


**Figura 5.16:** Convergência dos algoritmos VNS 1, VNS 2, VNS 3 e VNS 4. O custo da escala está representado no eixo vertical e o tempo está representado no eixo horizontal.

# 6

# Conclusão

Este trabalho apresentou quatro algoritmos baseados na meta-heurística VNS para resolver o PEMO, bem como duas variações do algoritmo VNS 1. Os algoritmos obtiveram resultados que foram comparados com um algoritmo recente proposto por Sakiyama et al. (2014). Os experimentos foram executados sobre um conjunto de 11 instâncias, sendo que a maior instância foi gerada nesse trabalho baseada em dados recentes de uma empresa de transporte de Maringá–PR. Todos os quatro algoritmos principais e variações tiveram um valor da função de custo reduzido em relação ao algoritmo de Sakiyama et al. (2014) (exceto na menor instância).

Os procedimentos PCR e *k-swap* usados nos algoritmos propostos, tiveram variações em relação as abordagens *first improvement*, *best improvement* e *continuous improvement*. Nesses procedimentos também foram utilizadas as sequências de corte denominadas *forward* e *backward* e esses cortes foram feitos de acordo com a demanda no horizonte de tempo das escalas de motoristas (seções 4.3 e 4.3.1).

O algoritmo VNS 1 (seção 4.4) e a suas duas variações (VNS 1 BI e VNS 1 BI P seção 4.4.1) usam o VND clássico como busca local, isso faz com que a escolha da forma para explorar a vizinhança seja de certa forma gulosa, pois sempre que existe uma melhoria na solução corrente para um valor de $k$, $k$ volta a receber 1. Embora isso aconteça, os resultados para o VNS 1 e variações foram muito satisfatórias, pois mostram-se os mais eficientes devido a essa lógica na troca do valor de $k$, além de serem eficazes: em geral o VNS 1 e VNS 1 BI estavam situados na segunda posição em relação a função de custo das escalas de motoristas geradas (seção 5.4). Isso mostra evidências que a abordagem VNS em conjunto com o VND é eficaz e eficiente quando aplicados no PEMO.

A função BUSCA-LOCAL do algoritmo VNS 4 (seção 4.7) usa o processo de descida em todas as formas de explorar vizinhança até que não haja melhoria, e ao final escolhe a melhor solução dentre todas para iniciar uma nova iteração. Devido a essa estratégia, esse método é o mais caro computacionalmente, mas foi o algoritmo – em geral – mais eficaz comparado aos outros (seção 5.4). Os resultados provenientes das experimentações desse algoritmo indicam evidências que esse método é competitivo na busca por escalas de baixo custo para o PEMO.

O algoritmo VNS 2 e VNS 3 (seção 4.5 e 4.6) usam uma lógica "adaptativa"na escolha das formas de exploração da vizinhança. Esses algoritmos buscam uma forma promissora de explorar vizinhos, para isso, todas as formas devem ser executadas uma quantidade $m$ de vezes antes de se escolher a melhor forma (VNS 2); ou escolher a melhor solução dada uma execução de cada forma de explorar vizinhos (VNS 3). Em relação aos resultados dessas duas abordagens, elas foram mais eficazes que o algoritmo de comparação SAK (seção 5.4).

Dessa forma, este trabalho contribui com quatro novos algoritmos principais para um (PEMO) dos problemas envolvidos no contexto dos serviços de transporte públicos que devem ser pesquisados. Esses novos algoritmos foram baseados principalmente nos trabalhos de Sakiyama et al. (2014), Hansen et al. (2009), Geiger et al. (2011) e Calvi (2005).

É indicado aplicar todos esses algoritmos (ou novas propostas derivadas dessas) em outras instâncias de escalas de veículos para que novas evidências no valor de custo da jornada de motorista sejam encontradas.

Também é sugerido a comparação das propostas deste trabalho com outras abordagens computacionais aplicadas no PEMO. Isso pode ser feito para que as comparações entre algoritmos – principalmente em relação a função de custo – evidenciem as propostas mais eficazes de resolução do problema.

Em trabalhos futuros, é sugerido fazer uma comparação com um limite inferior melhorado para avaliar o quão próximo da melhor solução os algoritmos propostos, algoritmos derivados e novas propostas estão.

Também é sugerido que seja feito um estudo para diminuir a quantidade de formas de explorar a vizinhança sem perda considerável de qualidade na função de custo. Isso deve ser feito para que os algoritmos diminuam o tempo de execução e não percam a qualidade da solução (principalmente o algoritmo 4). Para isso, pode ser feito um estudo de quais são as formas (ou combinações de formas) mais promissoras de explorar vizinhança. A comparação entre o algoritmo VNS 1 e algoritmo VNS 1 BI indicam esse caminho, ou

seja, para o VND um exagero na quantidade de formas de explorar a vizinhança não necessariamente gera melhores soluções.

As funções BUSCA-LOCAL dos algoritmos VNS 2, VNS 3 e VNS 4 podem ter trechos paralelizados para que esses algoritmos tenham uma melhor eficiência, logo é sugerido que isso seja feito em futuros trabalhos.

Uma outra ideia é aplicar as quatro buscas locais (ou variações delas) apresentadas nesse trabalho em outras meta-heurísticas que suportem essas buscas-locais, como por exemplo um *Simulated Annealing* com uma busca em múltiplas vizinhanças. Em relação as buscas denominadas adaptativas usadas no VNS 2, 3 e 4, sugere-se a aplicação em outros problemas computacionais em que elas possam ser usadas.

Por fim, o problema de escalonamento de motorista de ônibus (PEMO) deve ser continuamente pesquisado para que novos métodos eficazes sejam criados. Isso deve ser feito devido a sua complexidade computacional de resolução, e também devido a importância no planejamento de transporte público, devido principalmente aos custos envolvidos com recursos humanos.

# REFERÊNCIAS

Associação Nacional das Empresas de Transportes Urbanos Como Funciona o Transporte Público - NTU: Associação Nacional das Empresas de Transportes Urbanos. 2016.
Disponível em transportepublico.org.br

Avanthay, C.; Hertz, A.; Zufferey, N. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, v. 151, n. 2, p. 379–388, 2003.
Disponível em http://www.sciencedirect.com/science/article/pii/S0377221702008329

Bussab, W. d. O.; Morettin, P. A. *Estatística Básica*. 8 ed. Editora Saraiva, 2013.

Calvi, R. *Um Algoritmo para o Problema de Escalonamento de Tripulação em Empresas de Ônibus*. Dissertação de Mestrado, Unversidade Estadual de Maringá, 2005.

Carpaneto, G.; Toth, P. Primal-dual algrorithms for the assignment problem. *Discrete Applied Mathematics*, v. 18, n. 2, p. 137–153, 1987.
Disponível em http://www.sciencedirect.com/science/article/pii/0166218X87900163

Chen, M.; Niu, H. Research on the Scheduling Problem of Urban Bus Crew Based on Impartiality. *Procedia - Social and Behavioral Sciences*, v. 43, p. 503–511, 2012.
Disponível em http://www.sciencedirect.com/science/article/pii/S187704281201004X

Chen, S.; Shen, Y. An improved column generation algorithm for crew scheduling problems. *ResearchGate*, v. 10, n. 1, p. 175–183, 2013.
Disponível em https://www.researchgate.net/publication/279699645_An_improved_column_generation_algorithm_for_crew_scheduling_problems

Constantino, A. A.; Calvi, R.; Araujo, S. A. d.; Neto, C. F. X. d. M. Algoritmo Baseado em Grafo Multipartido para Escalonamento de Pessoal em Empresa de Transporte. *Simpósio Brasileiro de Pesquisa Operacional*, v. 39, p. 1759–1770, 2007.
Disponível em http://www.din.uem.br/sbpo/sbpo2007/pdf/arq0174.pdf

De Leone, R.; Festa, P.; Marchitto, E. Solving a bus driver scheduling problem with randomized multistart heuristics. *International Transactions in Operational Research*, v. 18, n. 6, p. 707–727, 2011.
Disponível em http://onlinelibrary.wiley.com/doi/10.1111/j.1475-3995.2011.00827.x/abstract

Fischetti, M.; Martello, S.; Toth, P. The Fixed Job Schedule Problem with Spread-time Constraints. *Oper. Res.*, v. 35, n. 6, p. 849–858, 1987.
Disponível em http://dx.doi.org/10.1287/opre.35.6.849

Geiger, M. J.; Sevaux, M.; Voss, S. Neigborhood Selection in Variable Neighborhood Search. *arXiv:1109.3313 [cs]*, arXiv: 1109.3313, 2011.
Disponível em http://arxiv.org/abs/1109.3313

Groot, S. W. d.; Huisman, D. Vehicle and Crew Scheduling: Solving Large Real-World Instances with an Integrated Approach. In: Hickman, P. M.; Mirchandani, P. P.; Voss, P. D. S., eds. *Computer-aided Systems in Public Transport*, n. 600 in Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, p. 43–56, dOI: 10.1007/978-3-540-73312-6_3, 2008.
Disponível em http://link.springer.com/chapter/10.1007/978-3-540-73312-6_3

Hansen, P.; Mladenovic, N. *A tutorial on variable neighborhood search.* Groupe d'études et de recherche en analyse des décisions, HEC Montréal, 2003.

Hansen, P.; Mladenović, N.; Moreno Pérez, J. A. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, v. 175, n. 1, p. 367–407, 2009.
Disponível em http://link.springer.com/article/10.1007/s10479-009-0657-6

Hillier, F. S.; Lieberman, G. J. *Introduction to Operations Research with Access Card for Premium Content.* 10 edition ed. McGraw-Hill Education, 2014.

Huisman, D.; Freling, R.; Wagelmans, A. P. M. Multiple-Depot Integrated Vehicle and Crew Scheduling. *Transportation Science*, v. 39, n. 4, p. 491–502, 2005.
Disponível em http://dx.doi.org/10.1287/trsc.1040.0104

KUHN, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, v. 2, n. 1-2, p. 83–97, 1955.
Disponível em `http://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109/abstract`

LEONE, R. D.; FESTA, P.; MARCHITTO, E. A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics*, v. 17, n. 4, p. 441–466, 2010.
Disponível em `http://link.springer.com/article/10.1007/s10732-010-9141-3`

LI, H.; WANG, Y.; LI, S.; LI, S. A Column Generation Based Hyper-Heuristic to the Bus Driver Scheduling Problem. *Discrete Dynamics in Nature and Society*, v. 2015, p. e638104, 2015.
Disponível em `http://www.hindawi.com/journals/ddns/2015/638104/abs/`

MA, J.; CEDER, A. A.; YANG, Y.; LIU, T.; GUAN, W. A case study of Beijing bus crew scheduling: a variable neighborhood-based approach. *Journal of Advanced Transportation*, v. 50, n. 4, p. 434–445, 2016.
Disponível em `http://onlinelibrary.wiley.com/doi/10.1002/atr.1333/abstract`

PAULA, M. R. D.; RAVETTI, M. G.; PARDALOS, P. M. Abordagem Variable Neighborhood Search para o Problema de Seqüenciamento com Máquinas Paralelas e Tempos de Preparação Dependentes da Seqüência. *XXXVIII Simpósio Brasileiro de Pesquisa Operacional*, v. 38, p. 1265–1276, 2006.

PEPIN, A.-S.; DESAULNIERS, G.; HERTZ, A.; HUISMAN, D. A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of Scheduling*, v. 12, n. 1, p. 17–30, 2008.
Disponível em `http://link.springer.com/article/10.1007/s10951-008-0072-x`

POLACEK, M.; HARTL, R. F.; DOERNER, K.; REIMANN, M. A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, v. 10, n. 6, p. 613–627, 2004.
Disponível em `http://link.springer.com/article/10.1007/s10732-005-5432-5`

PORTUGAL, R.; LOURENÇO, H. R.; PAIXÃO, J. P. Driver scheduling problem modelling. *Public Transport*, v. 1, n. 2, p. 103–120, 2008.
Disponível em `http://link.springer.com/article/10.1007/s12469-008-0007-0`

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

Raff, S. Routing and Scheduling of Vehicles and Crews. The State of the ArtRouting and scheduling of vehicles and crews. *Computers & Operations Research*, v. 10, n. 2, p. 63–211, 1983.
Disponível em http://www.sciencedirect.com/science/article/pii/0305054883900308

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

Silva, G. P.; Reis, A. F. d. S. A study of different metaheuristics to solve the urban transit crew scheduling problem. *Journal of Transport Literature*, v. 8, n. 4, p. 227–251, 2014.
Disponível em http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2238-10312014000400010&lng=en&nrm=iso&tlng=en

Tóth, A.; Krész, M. An efficient solution approach for real-world driver scheduling problems in urban bus transportation. *Central European Journal of Operations Research*, v. 21, n. 1, p. 75–94, 2013.
Disponível em http://link.springer.com/article/10.1007/s10100-012-0274-3