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

MARIANE AFFONSO MEDEIROS

VNS_BLS, um algoritmo com busca locais simultâneas para a resolução do Problema de Escalonamento de Motorista de Transporte Público

Maringá
2018

MARIANE AFFONSO MEDEIROS

VNS_BLS, um algoritmo com busca locais simultâneas para a resolução do Problema de Escalonamento de Motorista de Transporte Público

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
2018

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

Medeiros, Mariane Affonso

M488v VNS BLS, um algoritmo com busca locais simultâneas para a resolução do Problema de Escalonamento de Motorista de transporte público/ Mariane Affonso Medeiros. -- Maringá, 2018.
80 f. il. : figs., color., tabs.

Orientador: Prof. Dr. Ademir Aparecido Constantino.

Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2018.

1. Problemas de escalonamento de motorista de ônibus. 2. Transporte público. 3. Variable Neighborhood Search. 4. Buscas locais. 5. Pesquisa operacional. I. Constantino, Ademir Aparecido, orient. II. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Informática. Programa de Pós-Graduação em Computação. III Título.

CDD 22. ED. 003.3
Jane Lessa Monção CRB9 1173

FOLHA DE APROVAÇÃO

MARIANE AFFONSO MEDEIROS

VNS_BLS, um algoritmo com busca locais simultâneas para a resolução do problema de escalonamento de motorista de transporte público

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

Profa. Dra. Gislaine Camila Lapasini Leal
Universidade Estadual de Maringá – DEP/UEM

Prof. Dr. Luiz Satoru Ochi
Universidade Federal Fluminense – IC/UFF

Aprovada em: 05 de dezembro de 2018.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

AGRADECIMENTOS

Agradeço e dedico esta dissertação a minha família maravilhosa sem os quais eu não teria chegado até aqui. Vô, mãe, vó, tia, primo essa conquista tem muito de vocês. Obrigada mãe, por ter me obrigado a estudar em tantos momentos que eu não compreendia a necessidade. Minha força vem de você, você é uma mulher brilhante. Agradeço todos os meus amigos, os que fiz durante este período e aos antigos. Sem amigos não aguentamos muito da caminhada, sou grata por ter os melhores. Gostaria de dizer muito obrigada a todos os professores que já passaram em minha vida, todos sem exceção são uma inspiração para mim. Obrigada UTFPR, por ter me ensinado muito mais que coisas profissionais, sou uma pessoa melhor por ter passado por este lugar. Inês, nossa secretária incrível, obrigada pela paciência de responder repetidas vezes as mesmas perguntas e por sempre saber absolutamente tudo de papéis e burocracias que temos que assinar, você é sensacional. Obrigada professor Ademir Constantino pela oportunidade que você me deu, aprendi muito com você ao longo destes dois anos. Obrigada professor Anderson Faustino pela extrema dedicação, você é uma pessoa em quem me inspiro. Professores Ademir e Anderson obrigada por este período de convívio, pela compreensão, dedicação, colaboração e principalmente paciência. E finalmente muito obrigada a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido a este trabalho.

VNS_BLS, um algoritmo com busca locais simultâneas para a resolução do Problema de Escalonamento de Motorista de Transporte Público

RESUMO

O Problema de Escalonamento de Motoristas de Transporte Público consiste em construir escalas de trabalho para motoristas de uma empresa de transporte público, de forma que as escalas geradas respeitem restrições impostas e minimizem custos operacionais. Este trabalho aborda instâncias de dados de grande escala oriundas de casos reais de empresa de transporte público. A utilização de meta-heurísticas para resolução deste problema é bastante explorado, pois as meta-heurísticas são capazes de encontrar boas soluções em um tempo computacional razoável. NO entanto dependendo do tamanho das instâncias do problema o tempo computacional consumido para estes casos pode ser elevado mesmo utilizando técnicas heurísticas. Entretanto, com o crescente desenvolvimento de novos hardwares, há cada vez mais computadores com multinúcleos. O que segure o desenvolvimento e estudo de novas técnicas heurísticas que explore estes recursos de processamento. Sendo assim, este trabalho propõe o VNS_BLS, uma meta-heurística baseada em VNS que utiliza 36 buscas locais diferentes executadas de forma simultânea. O VNS_BLS possui quatro versões que se diferem em questão de comunicação com memória compartilhada e ociosidade. O algoritmo foi testado com instâncias de dados reais com 412 até 3478 tarefas. Os resultados obtidos pelas versões foram comparados entre elas e com outras abordagens da literatura. Com base nos experimentos realizados foi possível perceber que nem todas as 36 buscas locais do algoritmo são de fato efetivas na melhoria da solução. Notou-se, também, que comparado com outras propostas o VNS_BLS se mostra competitivo, pois alcança bons resultados em menor tempo computacional.

Palavras-chave: Problema de Escalonamento de Motoristas. *Variable Neighborhood Search*. Buscas Locais.

VNS_BLS an algorithm with simultaneous local search to resolution of Bus Driver Scheduling Problem

ABSTRACT

The Bus Driver Scheduling Problem consists of construct work schedules for drivers of a public transport company, that the scales generated respect imposed constraints and minimize operation costs. This paper addresses large-scale data instances from real cases of a public transport company. The use of metaheuristics to solve this problem is well explored, since metaheuristics are able to find good solutions in a reasonable computational time. However depending on the size of instances of the problem the computational time consumed for these cases can be high, even using heuristic techniques. However, with the increasing development of new hardware, there are more computers with multi-cores. This increase the need to development and study of new heuristic techniques that exploits these processing resources. Thus, this work proposes VNS_BLS, a VNS-based metaheuristic that uses 36 different local searches performed simultaneously. VNS_BLS has four versions that differ in terms of communication with shared memory and idleness. The algorithm was tested with real data instances with 412 to 3478 tasks. The results obtained by the versions were compared between them and with other approaches in the literature. Based on the experiments was possible to realize that not all 36 local searches of algorithm are effective in improving the solution. It was also noticed that, compared to other proposals, the VNS_BLS is competitive because achieves good results in less computational time.

Keywords: Bus Driver Scheduling Problem. Variable Neighborhood Search. Local Search.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

PEM: Problema de Escalonamento de Motoristas.
PET: Problema de Escalonamento de Tripulação.
VNS: Variable Neighborhood Search.
PA: Problema de Atribuição.
PCR: Processo de Corte e Recombinação.
BT: Busca Tabu.
ILS: *Iterated Local Search*.
RVND: *Randomized Variable Neighborhood Descent*.
VNS_BLS: VNS com Buscas Locais Simultâneas.
VNS_BLS_SC_CO: VNS_BLS Sem Comunicação Com Memória Compartilhada e Com Ociosidade.
VNS_BLS_CC_CO: VNS_BLS Com Comunicação Com Memória Compartilhada e Com Ociosidade.
VNS_BLS_SC_SO: VNS_BLS Sem Comunicação Com Memória Compartilhada e Sem Ociosidade.
VNS_BLS_CC_SO: VNS_BLS Com Comunicação Com Memória Compartilhada e Sem Ociosidade.
BL.PCR.F.FI: Busca Local Com Problema de Corte e Recombinações Foward First Improvement.
BL.PCR.B.FI: Busca Local Com Problema de Corte e Recombinações Backward First Improvement.
BL.PCR.F.CN: Busca Local Com Problema de Corte e Recombinações Foward Continuous Improvement.
BL.PCR.B.CN: Busca Local Com Problema de Corte e Recombinações Backward Continuous Improvement.
BL.PCR.F.BS: Busca Local Com Problema de Corte e Recombinações Foward Best Improvement.
BL.PCR.B.BS: Busca Local Com Problema de Corte e Recombinações Backward Best Improvement.
BL.2SWAP.F.FI: Busca Local Com 2-swap Foward First Improvement.
BL.2SWAP.B.FI: Busca Local Com 2-swap Backward First Improvement.
BL.3SWAP.F.FI: Busca Local Com 3-swap Foward First Improvement.
BL.3SWAP.B.FI: Busca Local Com 3-swap Backward First Improvement.
BL.4SWAP.F.BS: Busca Local Com 4-swap Foward Best Improvement.

BL.4SWAP.B.BS: Busca Local Com 4-swap Backward Best Improvement.
BL.1SWAP.F.CN: Busca Local Com 1-swap Foward Continuous Improvement.
BL.1SWAP.B.CN: Busca Local Com 1-swap Backward Continuous Improvement.
BL.2SWAP.F.CN: Busca Local Com 2-swap Foward Continuous Improvement.
BL.2SWAP.B.CN: Busca Local Com 2-swap Backward Continuous Improvement.
BL.3SWAP.F.CN: Busca Local Com 3-swap Foward Continuous Improvement.
BL.3SWAP.B.CN: Busca Local Com 3-swap Backward Continuous Improvement.
BL.4SWAP.F.CN: Busca Local Com 4-swap Foward Continuous Improvement.
BL.4SWAP.B.CN: Busca Local Com 4-swap Backward Continuous Improvement.
BL.5SWAP.F.CN: Busca Local Com 5-swap Foward Continuous Improvement.
BL.5SWAP.B.CN: Busca Local Com 5-swap Backward Continuous Improvement.

SUMÁRIO

1

Introdução

O transporte coletivo é um serviço essencial fornecido para a sociedade, pois além de empregar diversas pessoas, grande parte da população depende deste tipo de serviço (Calvi, 2005) para se locomover. Uma pesquisa feita em 2014, pela revista CNI-IBOPE, mostrou que 24% da população brasileira usa o transporte público como principal meio de locomoção (CNI-IBOPE, 2014). Sendo assim observa-se que este serviço é essencial para a população como um todo.

Os veículos e motoristas são os principais recursos de uma companhia de transporte público, portanto a forma como são alocados tem impacto direto na qualidade e no custo do serviço (Silva e da Cunha, 2010). Uma boa alocação pode resultar na redução de custos, beneficiar funcionários, proporcionar jornadas de trabalho menos exaustivas, e consequentemente aumentar a qualidade do serviço.

Dessa maneira, o planejamento operacional do transporte coletivo é uma área de pesquisa promissora e vem sendo estudada sob diversas maneiras ao longo do tempo. Este planejamento pode ser dividido em algumas atividades para facilitar o entendimento e diminuir a complexidade. As atividade são: programação de horários, programação de veículos, escalonamento e rotação de motoristas. Todas as atividades em conjunto compõem o planejamento operacional completo que uma empresa de transporte público precisa fazer. Cada uma destas atividades pode ser estudada em separado, pois caracterizam um tipo de problema específico. Este trabalho tem como foco a escalonamento de motoristas, chamado de Problema de Escalonamento de Tripulação (PET), Escalonamento de Condutores ou ainda Problema de Escalonamento de Motoristas (PEM).

O PEM consiste na geração de uma escala de trabalho para um conjunto de motoristas, de forma a respeitar determinadas restrições e otimizar uma função objetivo pré definida. Esta função objetivo pode representar, por exemplo, custos operacionais da empresa. Ao longo dos anos diversas pesquisas vêm se concentrando na resolução deste problema, tendo inclusive conferências especializadas como *Conference of Advanced System of Public Transport* [1]. Ainda assim, esta é uma área de pesquisa em progresso e há diversas possíveis estratégias surgindo para obter soluções do problema.

Há na literatura diversas propostas para resolução do PEM, que se dividem em abordagens que utilizam programação matemática e métodos heurísticos. As abordagens matemáticas foram as primeiras a serem investigadas e encontradas na literatura. As mais comuns são: o problema de cobertura de conjuntos (Fores, 1996) e partição de conjuntos (Portugal *et al.*, 2009). A ideia dessas abordagens é criar um conjunto de jornadas factíveis, que não infrinjam restrições estabelecidas. Estas abordagens não testam todas as possibilidades de combinações, no entanto garantem matematicamente que as combinações não testadas não são boas dada o método de avaliação considerada.

Devido ao tempo computacional exigido pelas abordagens matemáticas, existem pesquisas concentradas na utilização de meta-heurísticas para resolução do PEM. Estas abordagens não garantem encontrar a solução ótima, ao invés disso utilizam características do problema para concentrar a busca em determinadas soluções do espaço de busca. Dessa maneira, tendem a consumir um esforço computacional menor e chegar a soluções promissoras. Algumas meta-heurísticas que podem ser encontrados na literatura são: Busca Tabu (Marinho *et al.*, 2004), *Simulated Annealing* (Silva *et al.*, 2002), *Variable Neighborhood Search* (Dos Santos, 2016) e Algoritmo Genético (Dias *et al.*, 2002).

Ao longo dos anos, houve um fenômeno de crescimento em relação a dimensão do problema de escalonamento de motoristas, devido a dois fatores: a rápida expansão das linhas de ônibus e o aumento de pessoas que utilizam transporte público (Li, 2013). Sendo assim, além das diversas restrições a serem consideradas, deve-se levar em consideração a dimensão que as instâncias do problema podem atingir.

Sendo assim, com o impulsionamento que o potencial de hardware vem sofrendo ao longo dos anos, com computadores que oferecem diversos núcleos para processamento, fica evidente a possibilidade de explorar mais o hardware a fim de melhorar a qualidade das soluções pois as meta-heurísticas acabam por utilizar um núcleo por vez em suas buscas, devido a forma de implementação. Desperdiçando assim recursos que as máquinas podem oferecer.

[1] www.caspt.org

Portanto a exploração do potencial de hardware, através de implementações que consigam utilizar os recursos oferecidos se demonstra útil na prática para resolver várias aplicações e problemas da vida real que podem exigir dias ou semanas de tempo computacional para serem resolvidos. Este cenário é comum quando consideramos problemas da classe NP-Difíceis. Problemas marcados por uma grande quantidade de possíveis combinações e um espaço de solução amplo. É importante ressaltar que a exploração dos recursos não significa que o tempo de execução para obtenção das soluções será sempre baixo, pois devido a complexidade do problema o tempo pode ainda ser significativo.

Observa-se então uma lacuna de pesquisa relevante, a implementação de meta-heurísticas que se aproveitem por completo dos recursos de hardware oferecidos, como quantidade de núcleos por exemplo. A fim de de reduzir o tempo de execução dos algoritmos e possivelmente, gerar soluções melhores que as apresentadas na literatura. Neste contexto, a proposta deste trabalho é explorar a meta-heurística VNS, utilizando diferentes buscas simultâneas no espaço de solução, a fim de analisar a qualidade das soluções e a relevância das diferentes busca locais na obtenção das soluções.

O objetivo geral deste trabalho é propor um algoritmo com buscas simultâneas, para resolução do PEM. Os objetivos específicos são:

- Avaliar o desempenho do algoritmo proposto, considerando a qualidade da solução e tempo de execução.
- Avaliar as diferentes versões do algoritmo proposto.
- Avaliar quais buscas locais colaboram mais na melhora da solução.

Os próximos capítulos estão organizados como segue: Capítulo 2 traz definições do problema, lista trabalhos encontrados na literatura que tratam do PEM e descreve os algoritmos que serão utilizados neste trabalho. O Capítulo 3 define o algoritmo proposto. Capítulo 4 apresenta os resultados obtidos pelo algoritmo proposto. Por fim o Capítulo 5 traz as conclusões e trabalhos futuros.

2

Referencial Teórico

Este capítulo apresenta um referencial teórico a fim de embasar a construção deste trabalho. A Seção 2.1 trata sobre conceitos e características do problema de escalonamento de motoristas. Na Seção 2.2 é abordado o conceito de meta-heurísticas, apresentando definições e técnicas utilizadas por esta abordagem. A Seção **??** apresenta meta-heurísticas com conceitos de execução simultânea. A Seção 2.3 apresenta técnicas e algoritmos que serão utilizados na construção da proposta deste trabalho. Por fim a Seção 2.4 apresenta uma revisão da literatura sobre o PEM.

2.1 Problema de Escalonamento de Motorista

Produzir escalas eficientes não melhora apenas o serviço e condições de trabalho, mas também tem impacto direto nos custos operacionais gastos pelas empresas de transporte público (Blais *et al.*, 1990). Dessa maneira Problema de Escalonamento de Motoristas consiste em determinar a escala diária dos motoristas de forma a obedecer restrições, minimizar custos e cumprir uma escala de veículos para uma tabela de horários (Sakayma, 2014).

Devido ao grande número de combinações possíveis para a resolução do PEM e a quantidade de restrições que podem ser consideradas, o problema é classificado como NP-Difícil (De Leone *et al.*, 2011). Sendo assim, há um crescente esforço na utilização de meta-heurísticas para a resolução do problema, dado que abordagens exatas tendem a solucionar instâncias que consideram poucas tarefas para combinação, como pode ser visto em De Leone *et al.* (2011) e Yunes *et al.* (2005).

Existem atividades envolvidas em todo o planejamento operacional de transporte público que são divididas em quatro programações diferentes. As atividades envolvidas neste planejamento são ilustradas na Figura - 2.1.

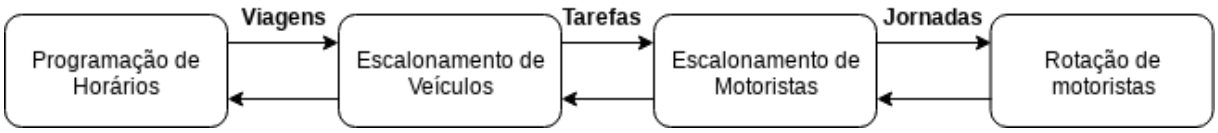


Figura 2.1: Planejamento de rede de transporte de transportes públicos.

As etapas descritas na Figura - 2.1 são:

- **Programação de Horários**: é composta por linhas de ônibus que correspondem a uma viagem entre duas localidades. Esta programação tem como resultado viagens que possuem horário de início e fim e terminal de saída e chegada.
- **Escalonamento de Veículos**: processo no qual é feita a atribuição de veículos às viagens geradas pela Programação de Horários. Como resultado da Programação de Veículos há um conjunto de tarefas que são compostas por viagens designadas aos seus respectivos veículos.
- **Escalonamento de Motoristas**: agrupa tarefas geradas pela etapa de Escalonamento de Veículos, formando jornadas que posteriormente serão atribuídas às tripulações. Este agrupamento pode ser feito de forma a minimizar custos.
- **Rotação de motoristas**: a partir das jornadas geradas pela etapa Escalonamento de Motoristas, a Rotação distribui as jornadas de trabalho aos motoristas compondo assim, escalas de trabalho dado um período de tempo, que podem ser semanal ou mensal. Esta etapa leva em consideração dias de folgas, feriados, recessos, férias e restrições legais (Gonçalves, 2010).

2.1.1 Terminologia

É importante definir alguns termos que serão citados ao longo do texto e que são utilizados ao se definir o PEM.

- Oportunidade de Troca: tupla composta por horário e local onde é permitida que um motorista faça uma troca de veículo.

- Tarefa: percurso entre duas oportunidades de troca. É a mínima porção de trabalho que pode ser atribuída a um motorista (Constantino *et al.*, 2017). Uma tarefa é composta por um par ordenado, $tarefa = ((origem, saida), (destino, chegada))$, a primeira representa o ponto de partida e o horário de saída da tarefa e a segunda tupla representa o ponto de destino e o horário de chegada (término) da tarefa.

- Bloco: uma sequência de viagens que podem ser executadas em um dia por um veículo x.

- Linha de Ônibus: definida por uma rota que um veículo x percorre.

- Equipe: conjunto de trabalhadores composto por motoristas e em alguns casos cobradores. A atribuição de uma tarefa t a uma equipe e, significa que a equipe e está trabalhando na linha de ônibus correspondente a tarefa t.

- *Peace Of Work*: sequência de tarefas consecutivas que são executadas na mesma linha de ônibus e atribuídas ao mesmo motorista.

- Tempo de Parada: é definido por uma parada não paga pela empresa de transporte. Consiste nos horários que o motorista fica parado entre duas tarefas a ele atribuídas.

- Tempo de Descanso: é uma parada paga pela empresa de transporte. O tempo de descanso é os horários atribuídos aos motoristas entre tarefas consecutivas caso seu tempo de trabalho contínuo tenha ultrapassado o máximo imposto.

- Jornada de trabalho: é uma sequência ordenada de tarefas que podem ser atribuídas a um motorista em um determinado dia.

A Figura - 2.2 ilustra um exemplo de duas linhas de ônibus.

Linha de Ônibus 1

Ônibus 1

	Origem	Saída	Destino	Chegada
Tarefa 1 →	G	04:30	T1	05:00
Tarefa 2 →	T1	05:00	T2	07:00
Tarefa 3 →	T2	07:10	T3	07:30
Tarefa 4 →	T3	07:40	T1	08:00

Linha de Ônibus 2

Ônibus 2

	Origem	Saída	Destino	Chegada
Tarefa 5 →	T1	08:00	T3	08:30
Tarefa 6 →	T3	09:00	T2	10:00

Figura 2.2: Exemplo de duas linhas de ônibus.

- A primeira linha possui quatro tarefas, a tarefa 1 sai do ponto G às 04:30 da manhã e termina às 08:00 no terminal 1 (T1);
- A segunda linha possui duas tarefas, a primeira saindo de T1 às 08:00 e terminando em T2 às 10:00. As tarefas das linhas de ônibus podem ser combinadas em jornadas, para posteriormente serem atribuídas a um motorista.

Duas possíveis jornadas baseadas nas linhas de ônibus da Figura - 2.2 são apresentadas na Figura - 2.3.

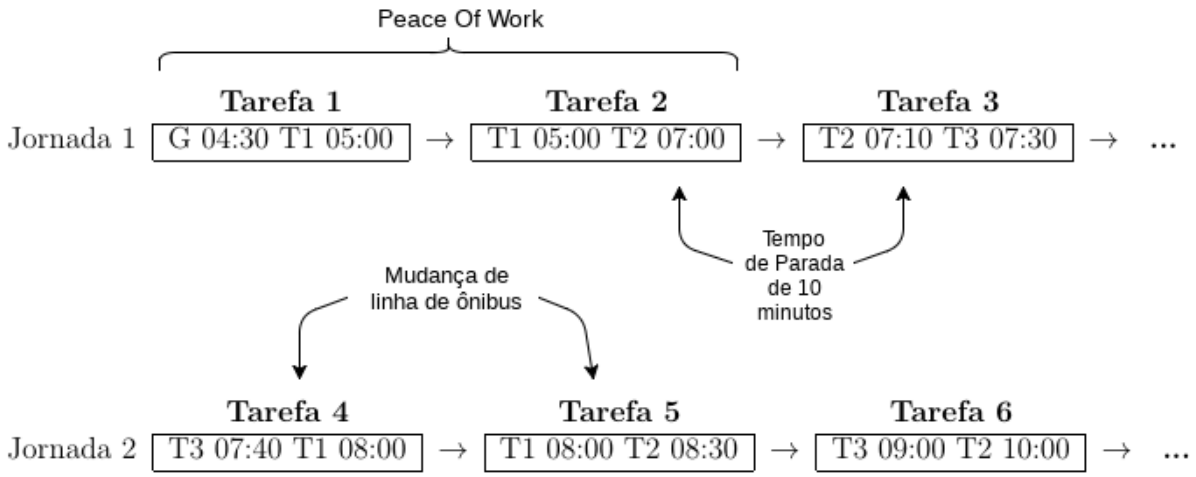


Figura 2.3: Exemplo de duas possíveis jornadas.

A Jornada 1 possui três tarefas (Tarefa 1, 2 e 3), a Tarefa 1 e 2 formam um *Peace of Work*, pois são consecutivas, estão na mesma linha de ônibus e podem ser atribuídas ao mesmo motorista. Entre a Tarefa 2 e 3 há um Tempo de Parada de 10 minutos. Na Jornada 2 há três tarefas e uma troca de linha entre a Tarefa 2 e 3. Estas jornadas indicam possíveis combinações das tarefas.

2.2 Meta-heurísticas

Existem problemas computacionais que podem ser representados por um conjunto de soluções, uma função associada e um conjunto de regras que especificam como as soluções devem ser construídas (Crainic e Toulouse, 2010), estes problemas são chamados de problemas de otimização combinatória. O objetivo destes problemas é selecionar um subconjunto de soluções que satisfaçam as regras de forma que o valor da função determinada seja maximizado ou minimizado, de acordo com o problema em questão, entre todas as combinações possíveis.

Problemas de otimização combinatória muitas vezes são formulados como problemas de otimização de inteiros. Em diversos casos este tipo de formulação não consegue resolver instâncias de problemas com dimensões reais, pois a quantidade de possíveis soluções cresce exponencialmente de acordo com a quantidade de soluções no conjunto inicial (Crainic e Toulouse, 2010). Dessa maneira, diversos métodos são estudados para que seja possível reformular estes problemas de maneira a reduzir o esforço para encontrar a solução ótima. Em geral, os métodos propõem a exploração de apenas algumas regiões do espaço de solução, como por exemplo as meta-heurísticas.

Meta-heurísticas são formalmente definidas como processos iterativos que guiam uma heurística subordinada, combinando diferentes conceitos para a exploração do espaço de busca (Osman e Laporte, 1996). Elas utilizam estratégias de aprendizado para estruturar informações a fim de encontrar soluções o mais próximo do ótimo possível (Osman e Laporte, 1996).

Estas estratégias de aprendizado auxiliam as meta-heurísticas a obter em cada iteração boas (do ponto de vista do método de avaliação) soluções encontradas na vizinhança da solução corrente (Crainic e Toulouse, 2010). Ao contrário de heurísticas de buscas locais, as meta-heurísticas podem construir soluções que não são necessariamente melhores que a anterior, esta característica constitui o principal mecanismo para evitar a parada em ótimos locais.

Devido a movimentação no espaço de solução feita pelas meta-heurísticas, elas não garantem a exploração sistemática de todo o espaço como os métodos exatos. No entanto, as meta-heurísticas examinam partes do espaço de soluções de acordo com algum critério que defina que aquela região pode ser promissora para encontrar soluções ótimas dada a função de avaliação. Meta-heurísticas bem estruturadas podem evitar cair em ótimos locais, soluções já visitadas ou ainda podem observar quais regiões promissoras do espaço de solução ainda não foram visitadas (Crainic e Toulouse, 2010).

Este trabalho utiliza a meta-heurística de trajetória *Variable Neighborhood Search* (VNS) em conjunto com estratégias de buscas locais. É utilizado várias buscas simultâneas para exploração do espaço de solução. Meta-heurísticas de trajetória podem ser vistas como extensões inteligentes de algoritmos de busca local (Blum e Roli, 2003). O objetivo desse tipo de meta-heurística é escapar dos ótimos locais para prosseguir na exploração do espaço de solução. Alguns exemplos deste tipo de meta-heurística são: Busca Tabu (BT), *Iterated Local Search* (ILS), *Variable Neighborhood Search*, *GRASP* e *Simulated Annealing*.

2.2.1 Técnicas de Busca Local

Dada uma solução S uma heurística de busca local tem como objetivo mover a solução S para uma solução S' com valor de função objetivo maior ou menor dependendo do problema em questão. Hansen *et al.* (2008) descrevem duas técnicas que podem ser utilizadas para a construção de heurísticas de busca local *First Improvement* e *Best Improvement*.

O *First Improvement* move-se para o primeiro vizinho com valor de função objetivo menor que a solução corrente, sem explorar toda a vizinhança, um conjunto de possíveis soluções geradas a partir de uma modificação em S. Vizinho é uma solução S' gerada a partir de uma solução S através de alguma modificação. A partir de uma solução S a cada iteração, S é substituída pela primeira solução S' na vizinhança que satisfaça a condição $f(S') < f(S)$. A Figura - 2.4, ilustra o funcionamento da técnica *First Improvement*, o primeiro S' gerado não satisfaz a condição $f(S') < f(S)$, portanto S não é substituído por S', o próximo vizinho S' satisfaz a condição, então S é atualizado. A Figura - 2.4 apresenta uma vizinhança que não chega a ser explorada, pois o vizinho anterior satisfez a condição e a busca irá continuar a partir da solução atualizada.

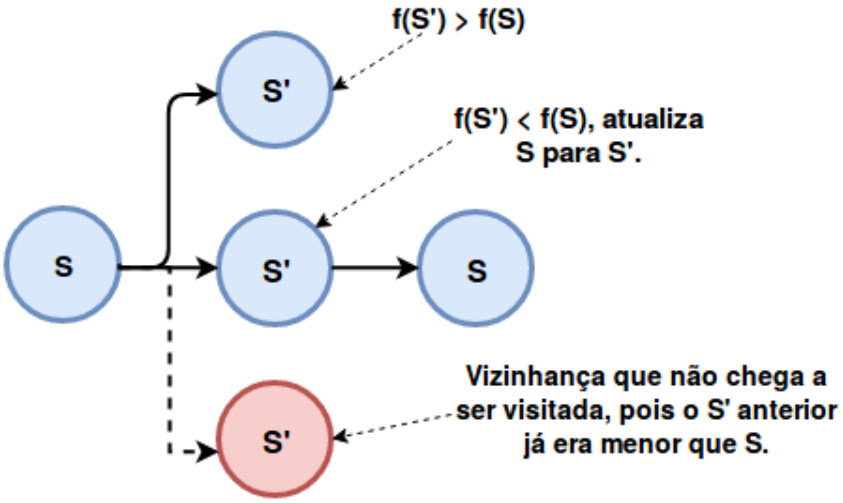


Figura 2.4: *First Improvement.*

A técnica de *Best Improvement* percorre toda a vizinhança a partir da solução corrente S e atualiza S para a solução S', de menor valor de função objetivo de toda a vizinhança. A Figura - 2.5 exemplifica a técnica *Best Improvement*, onde a vizinhança de S é percorrida e S é atualizado para a solução S' de menor valor da função objetivo.

Sakayma (2014) e Dos Santos (2016) utilizaram uma técnica chamada *Continuous Improvement* que é inspirada nas técnicas *First* e *Best Improvement*. Esta técnica atualiza

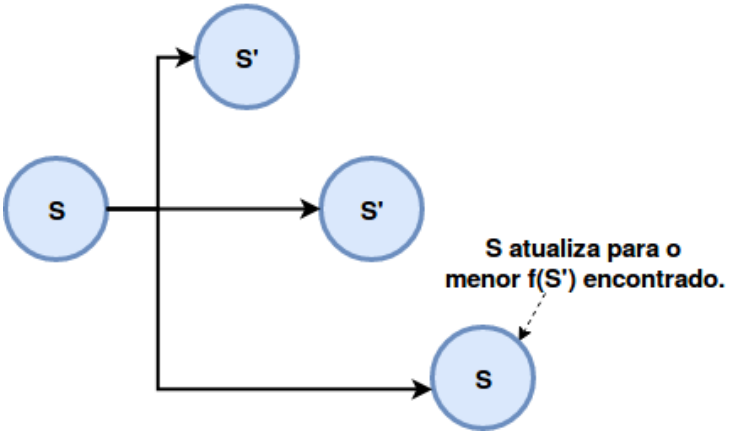


Figura 2.5: *Best Improvement.*

a solução S para S' , caso S' o valor da função objetivo seja menor que de S. Esse procedimento continua até o final do percurso da vizinhança. A Figura - 2.6 demonstra o funcionamento desta técnica, onde a solução S é atualizada para S', e continua a explorar os vizinhos da solução S atualizada.

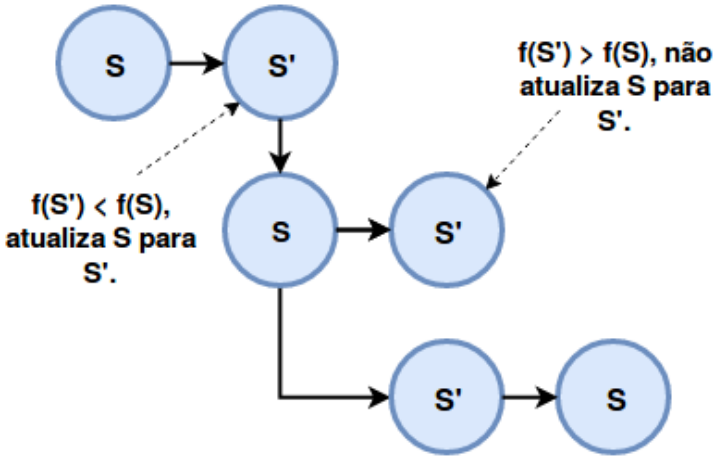


Figura 2.6: *Continuous Improvement.*

2.2.2 Variable Neighbourhood Search

A meta-heurística VNS foi proposta por Mladenovic (1995) e consiste na troca sistemática de vizinhos de uma solução. O design desta meta-heurística permite diversas modificações em sua estrutura podendo assim obter diferentes versões do algoritmo (Blum e Roli, 2003). O VNS busca explorar vizinhos cada vez mais distantes (diferentes) da solução corrente. O VNS é composto basicamente pelas seguintes etapas: *shake*, busca local e movimentação.

No *shake* um vizinho S_0 na vizinhança z é escolhido de forma aleatória. Este S_0 se torna a solução de partida para a busca local. A busca local intensifica a procura através de modificações na solução S_0.

O Algoritmo 1 ilustra o funcionamento do VNS, que recebe por parâmetro uma solução S_0 considerada a solução inicial e z_{max} que é o número máximo de vizinhanças que serão exploradas.

Algoritmo 1 Procedimento VNS

```
1:  function VNS(S_0, z_max)
2:      S ← S_0
3:      while critério de parada não satisfeito do
4:          z ← 1
5:          while z ≤ z_max do
6:              encontre um vizinho S' ∈ N_(z)(S)
7:              S'' ← busca_local(S')
8:              if f(S'') < f(S) then
9:                  S ← S''
10:                 z ← 1
11:             else
12:                 z ← z + 1
        return S
```

O VNS gera uma solução S' vizinha a S, através de modificações na solução S. A solução S' é utilizada pela busca local a fim de encontrar soluções melhores baseada em S'. Verifica-se então se a solução S'', obtida pela busca local, possui valor de função objetivo menor que S, se sim, S é atualiza para S'' e z recebe 1, para iniciar a busca na vizinhança da nova solução. Caso contrário, incrementa-se o valor de z e busca em outra vizinhança por uma solução melhor.

2.3 Algoritmos para o Problema de Escalonamento de Motorista

Esta seção apresenta algoritmos que serão usados neste trabalho: problema de atribuição, pcr e k-swap. O Problema de Atribuição é utilizado pelo a fim de determinar qual combinação de tarefas resulta em uma escala com menor custo, os algoritmos PCR e k-swap são utilizados para gerar modificações nas soluções construídas durante a busca. Estes algoritmo são utilizados em conjunto pelo algoritmo proposto neste trabalho.

2.3.1 Problema de Atribuição

O Problema de Atribuição (PA) ou Problema de Designação, é um problema de otimização combinatória utilizada na área de pesquisa operacional e um tipo especial de problema de programação linear no qual designados são indicados para a realização de tarefas (Hillier e Lieberman, 2013). Os designados são as jornadas realizadas pelas tripulações, dessa forma é analisado o custo de uma tarefa a uma jornada. O PA é utilizado neste trabalho a fim de determinar entre um conjunto de possíveis atribuições de tarefas à escalas, qual atribuição possui o menor custo associado. Dada uma matriz de custos com dimensões $n \times n$, o problema consiste em associar cada linha i a uma coluna j sob um custo c_{ij} , de modo que a soma dos custos seja minimizada.

A Equação 2.1 minimiza o custo da soma das atribuições entre linhas e colunas; a Equação 2.2 exige que para cada linha tenha uma coluna associada; a Equação 2.3 garante que para cada coluna seja designada uma linha; a Equação 2.4 garante que as variáveis envolvidas assumam apenas os valores de decisão 0 e 1. Este problema pode ser resolvido em tempo polinomial (Carpaneto e Toth, 1987).

$$\min z = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}.x_{ij} \tag{2.1}$$

$$\text{sujeito a: } \sum_{i=1}^{n} x_{ij} = 1, (j = 1, ..., n) \tag{2.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, (i = 1, ..., n) \tag{2.3}$$

$$x_{ij} \in 0, 1 (i = 1, ..., n; j = 1, ..., n) \tag{2.4}$$

Para que o PA possa ser aplicado a um problema, é necessário que o mesmo seja modelado em uma matriz quadrada (Pentico, 2007), para isto são adicionadas tarefas e/ou jornadas fictícias para que a matriz de custo fique quadrada.

2.3.2 Operador PCR

O PEM, pode ser representado como um grafo multipartido. Um grafo multipartido G é um grafo onde os vértices $V_n = \{V_1, V_2, ..., V_n\}$ podem ser particionados em q conjuntos disjuntos onde n é o número de partições. Um corte na solução é representado por um

conjunto de arestas entre duas partições consecutivas de G. Sendo assim, os operadores PCR e Kswap utilizam desta representação para modificar a solução.

Um operador é uma forma de modificar uma solução S, gerando uma solução S'. O operador PCR, também chamado de Operador de Cortes e Recombinações (PCR) é um operador de soluções que visa gerar novas soluções a partir de uma solução inicial S, uma solução é uma escala de trabalho quando consideramos o PEM. Esse procedimento garante que a solução gerada nunca será pior que a solução inicial S, sempre será igual ou melhor (Constantino *et al.*, 2014). O PCR é utilizado em diversos trabalhos de escalonamento, de Melo *et al.* (2010), Rizzato *et al.* (2010) e Constantino *et al.* (2014) foi utilizado para otimização de escalas de enfermeiros. Sakayma (2014) e Dos Santos (2016) utilizam este operador para otimização de escalas de motoristas de transporte público.

No PCR dado um índice i de uma sequência s, um corte i em s determina dois segmentos distintos, $S1$ anterior a i e $S2$ posterior a i. Seja $S = \{s \mid s \text{ é uma sequência}\}$ um conjunto finito e dado um corte em i que corta as sequências pertencentes a S, gera-se dois conjuntos distintos $S1$ e $S2$ que contém os segmentos das sequências originais de S (Figura - 2.7). É possível, então, recombinar $S1$ e $S2$ para gerar um novo conjunto S'. A Figura - 2.7 exemplifica um possível local de corte na solução S.

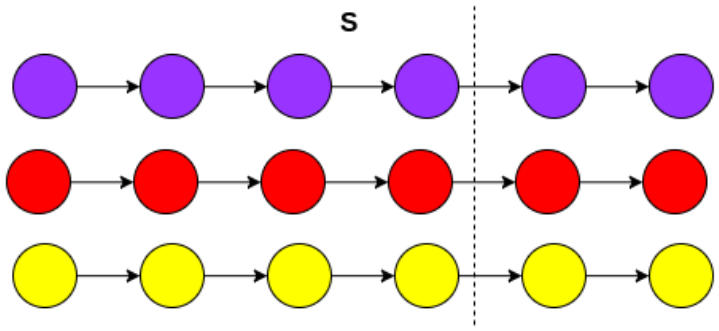


Figura 2.7: Exemplo de possíveis locais de cortes na solução S.

O processo de recombinar $S1$ e $S2$ gera uma matriz de custos C, que contém o custo de cada recombinação entre as tarefas de $S1$ e $S2$. Através de resolução do PA, é possível obter uma solução S', onde, cada linha da matriz de custos C representa uma sequência $s_i \in S_1$ e, cada coluna j representa uma sequência $s_j \in S_2$. Caso o corte gere uma bipartição perfeita do conjunto S, obtém-se uma matriz quadrada, se a bipartição não for perfeita, isto é, $|s_1| \neq |s_2|$, cria-se tarefas fictícias para obter uma matriz quadrada. O funcionamento do PCR implementado neste trabalho é demonstrado no Algoritmo 2.

O Algoritmo 2 percorre os W possíveis locais de corte da solução S dada como entrada. A função $cortar(S, w)$ faz o corte da solução S, este procedimento de corte tem

Algoritmo 2 PCR

```
1: function PCR(S)
2:     for w ∈ W do
3:         S1 , S2 ← cortar(S,w)
4:         C ← gerar-matriz-custos(S1,S2)
5:         sol ← PA(C)
6:         S' ← gera-nova-solução(S, sol)
       return S'
```

como objetivo separar a solução em duas. O processo de corte irá ter como resultado duas soluções $S1$ e $S2$ (Figura - 2.8). A função *gerar-matriz-custos*(S_1, S_2), calcula o custo das possíveis recombinações existentes entre $S1$ e $S2$ e gera uma matriz de custos C, com os valores destas recombinações. A matriz C será utilizada pelo $PA(C)$, problema de atribuição, a fim de encontrar a recombinação de menor custo associado. O retorno da função $PA(C)$ é um vetor que representa a nova solução S'. A função *gera-nova-solucao*(S, sol), constrói uma nova solução S' a partir do vetor *sol* que armazena a melhor recombinação de tarefas no corte w.

A Figura - 2.8 é um exemplo de um possível corte em S. Após o corte realizado, obtém-se duas jornadas parciais, uma a direita e outra a esquerda. Os segmentos de jornada a esquerda são recombinados com os segmentados da direita, como ilustra as setas na Figura - 2.8. Monta-se então uma matriz de custos C e resolve-se um PA para esta matriz.

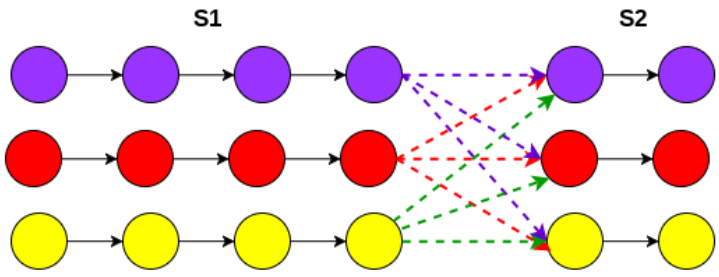


Figura 2.8: Possíveis recombinações para a solução S.

A saída do PA representa uma possível recombinação de menor custo associado (Figura - 2.9). Há diversas combinações possíveis, porém o PA retorna a recombinação que possui custo igual ou menor que o da solução de entrada.

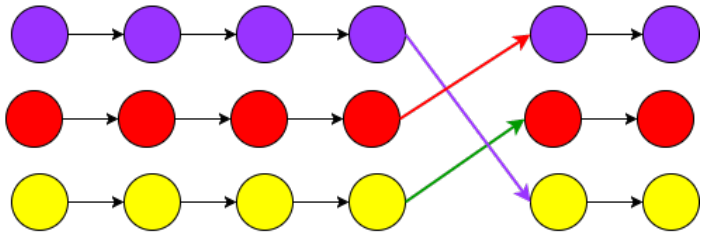


Figura 2.9: Solução gerada a partir as recombinações de uma solução inicial S.

2.3.3 Operador k-swap

O k-swap é um operador de soluções para gerar estruturas de vizinhanças, seu funcionamento é semelhante ao PCR, porém ao invés de fazer um corte na solução, o k-swap faz dois cortes, de forma a manter k vértices de distância entre os cortes. Este operador é utilizado em trabalhos de escalonamento de motorista de transporte público, como Sakayma (2014) e Dos Santos (2016).

Dado um conjunto finito S de tarefas em sequência, considera-se dois cortes distintos que cortam as sequências pertencentes a S delimitando um intervalo. Estes corte irão gerar 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 os segmentos das sequências internas ao intervalo (Conjunto S_1 e S_2 da Figura - 2.10).

Dada duas soluções geradas a partir do corte feito na solução, recombina-se S_1 e S_2 a fim de gerar um novo conjunto S'. Uma maneira de determinar uma das combinações para gerar S' é por meio da resolução do PA. A matriz de custos C é gerada através das possíveis recombinações de S_1 e S_2, tal que cada linha i da matriz C represente dois segmentos de sequência $s_i \in S_1$ e cada coluna j representa uma sequência $s_j \in S_2$. No operador k-swap é gerado duas matrizes de custo, uma em cada corte da solução S.

Cada elemento c_{ij} da matriz de custos C gerada para resolução do PA, recebe um valor $g(i, j)$, tal que $g(i, j)$ é uma um custo associado a recombinação de cada elemento $s_i \in S_1$, para cada elemento $s_j \in S_2$. O valor de c_{ij} pode ser infinito caso a recombinação não seja permitida. A Equação 2.5 apresenta como se dá o valor atribuído a cada posição da matriz de custos, esta equação é a mesma usada pelo PCR.

$$c_{ij} = \begin{cases} g(i,j) & \text{caso seja possível a combinação entre o elemento } S_i \text{ e } S_j \\ \infty & \text{caso contrário} \end{cases} \tag{2.5}$$

O Algoritmo 3 ilustra como é o funcionamento do k-swap implementado neste trabalho.

Algoritmo 3 Procedimento k-swap

```
1: function K-SWAP(S, k)
2:     for c ∈ C do
3:         S_1 , S_2 ← cortar(S,k)
4:         C1, C2 ← gerar-matriz-custos(S_1,S_2)
5:         sol ← PA(C1, C2)
6:         S' ← gera-nova-solução(S, sol)
       return S'
```

A função $cortar(S, k)$, corta a solução S em duas arestas distintas (que representam a ligação entre duas tarefas), e retorna S_1 e S_2, como mostra a Figura - 2.10. O valor k, é dado por parâmetro e representa a distância que o segundo corte deve ser feito. Os cortes gerados no procedimento k-swap devem ser distintos e obedecer ao valor k, ou seja, o segundo corte c_2 deve ser à k posições do primeiro corte c_1. Os cortes c_1 e c_2 podem ser feitos em qualquer local da solução. Em $gerar\text{-}matriz\text{-}custos(S_1, S_2)$ é feita as recombinações com o lado esquerdo de S_2 com o direito da S_1, e das tarefas do lado direito de S_2 com os do lado esquerdo de S_1 (Figura - 2.10). A função retorna duas matrizes de custo, cada uma correspondente a um corte. No $PA(C1, C2)$ é resolvido cada uma das matrizes de custo e retorna a recombinação com menor custo associado. A função $gera\text{-}nova\text{-}solucao(S, sol)$ é igual a descrita no PCR (Seção 2.3.2).

Na Figura - 2.10 é possível ver os cortes realizados na solução S, considerando $k = 2$ (2-swap). A Figura - 2.10 ilustra os segmentos das jornadas entre entre o primeiro corte e o segundo, que são recombinados com os seguimentos da esquerda do primeiro corte e à direita do segundo corte. Como no PCR, o custo as recombinações é calculado por um PA, no k-swap, serão dois PA resolvidos, um em cada corte.

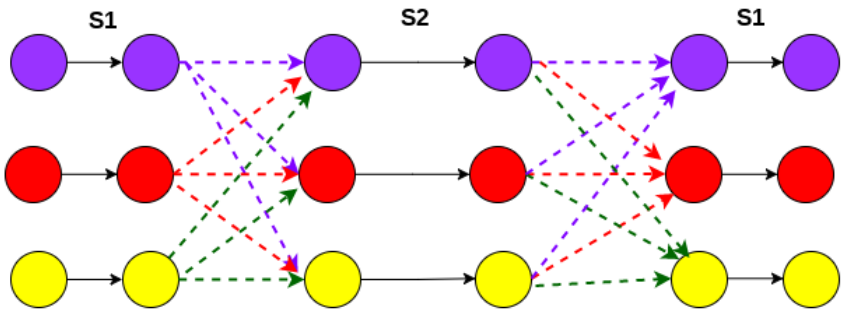


Figura 2.10: Locais de cortes na solução e possíveis recombinações dos segmentos formados.

A Figura - 2.11 demonstra uma possível recombinação gerada.

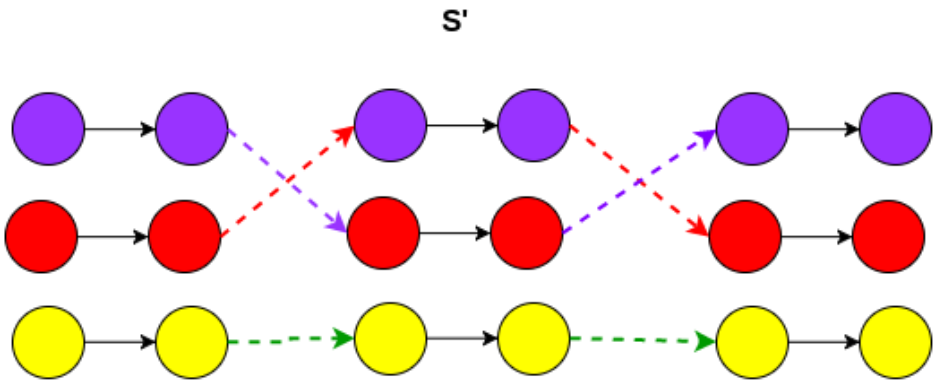


Figura 2.11: Recombinação final da solução S.

2.4 Trabalhos Relacionados

Esta seção trata dos trabalhos relacionados encontrados na literatura e está dividida em duas subseções: a primeira irá tratar dos trabalhos encontrados que resolvem o PEM, enquanto a segunda aborda os trabalhos da literatura que utilizam estratégias simultâneas em meta-heurísticas.

2.4.1 Resolução do PEM

Silva *et al.* (2004) formula o PEM como o problema de partição de conjuntos utilizando o método Simplex para resolvê-lo. A fim de validar a proposta, os autores realizaram teste em instâncias reais, que contém 11 linhas de 11 com 87 viagens. O método proposto mostrou-se capaz de encontrar soluções ótimas para o problema quando considerado cada linha de ônibus separadamente. Os autores consideram as linhas separadamente pois notaram que resolver o problema para todas as linhas tem um custo de tempo alto.

Em Yunes *et al.* (2005) é utilizada a abordagem baseada no problema de cobertura de conjuntos e geração de colunas com o objetivo de minimizar os números de motoristas. Para validar a proposta os autores usaram duas instâncias reais, compostas por duas linhas, com 125 e 246 viagens cada uma. Através da análise dos resultados os autores notaram que é possível encontrar soluções com boa qualidade (do ponto de vista da função objetivo considerada).

No geral as propostas de métodos exatos para resolução do PEM, não utilizam instâncias expressivas quanto ao tamanho para validação da proposta. Dessa forma uso de meta-heurísticas para resolução do PEM é uma opção interessante para que seja possível

a validação das abordagens utilizando instâncias de larga escala e para verificar a redução de tempo computacional.

Na literatura é possível encontrar propostas que utilizam diferentes meta-heurísticas. Marinho *et al.* (2004) utiliza a meta-heurística Busca Tabu (BT) para resolver o PEM. Para fazer modificações na solução é utilizado duas estruturas de vizinhança geradas através de movimentos diferentes. O primeiro movimento consiste na realocação de uma tarefa de uma jornada para outra. No segundo movimento é feito a troca de tarefas entre duas jornadas. O conjunto das soluções geradas pelos movimentos geram duas vizinhanças distintas.

Gonçalves (2010) utiliza as meta-heurísticas Busca Tabu e *Iterated Local Search* (ILS) para resolução do PEM. Para validar a proposta, o autor utiliza dados reais de uma empresa de Belo Horizonte, onde a instâncias possuem entre 27 e 515 viagens. A geração da vizinhança é feita através de dois métodos de realocação e troca de tarefas.

Sakayma (2014) utiliza o VNS para resolução do PEM e os procedimentos PCR e k-swap para gerar soluções vizinhas. Dos Santos (2016), baseia-se no trabalho de Sakayma (2014) e utiliza VNS, em conjuntos com os procedimentos PCR e k-swap para resolução do PEM. No entanto o autor, faz a proposta de quatro variações do VNS. As instâncias usadas pelo autor variam de 130 a 3478 viagens.

A Tabela - 2.1 sumariza os trabalhos citados, separando em técnica matemática e heurística. A informações presentes na tabela mostram as técnicas usadas, o ano e o tamanho das instâncias de teste que os autores utilizaram para validar suas propostas.

Tabela 2.1: Técnicas encontradas na literatura para resolução do PEM.

Técnica	Autor	Ano	Número de Viagens	Método
Matemática	(Silva *et al.*, 2004)	2004	87	Método Simplex
	(Yunes *et al.*, 2005)	2005	246	Geração de Colunas
Heurística	(Marinho *et al.*, 2004)	2004	–	Busca Tabu
	(Gonçalves, 2010)	2010	515	Busca Tabu e *Iterated Local Search*
	(Dos Santos, 2016)	2016	3478	VNS

2.4.2 VNS com abordagens de execuções simultâneas

Meta-heurísticas são essenciais para resolução de problemas de otimização combinatória e muitas vezes representam a única abordagem prática para resolução de alguns problemas complexos com dimensões reais (Crainic e Toulouse, 2010). No entanto existem problemas que não podem ser resolvidos apropriadamente em um período de tempo razoável, devido à sua complexidade ou ao tamanho das instâncias de teste.

Dessa forma, uma questão importante no projeto e calibração de meta-heurística é torná-las robustas, no sentido de oferecer um alto nível de desempenho e manter qualidade das soluções (Crainic e Toulouse, 2010). A modificação de meta-heurística para que sejam executadas de forma simultânea, visa abordar este problema. O principal objetivo deste tipo de modificação é a possibilidade de resolver instâncias de problemas reais de forma a utilizar um tempo computacional razoável. Em alguns contextos tal estratégia se mostra mais robusta do que versões sequenciais (Crainic e Toulouse, 2010).

A principal ideia de fazer uso de computação simultânea é que vários processos possam funcionar simultaneamente em diferentes processadores resolvendo um problema (Crainic e Toulouse, 2010). Sendo assim, pode-se considerar que há uma decomposição da carga computacional e distribuição de tarefas aos processadores disponíveis. Esta decomposição pode ser do próprio algoritmo, dos dados da instância do problema ou da estrutura do problema (Crainic *et al.*, 2009a,b).

A utilização de abordagens que permitem a execução simultânea de algumas partes das meta-heurísticas é uma abordagem promissora para aumentar a eficiência de métodos heurísticos e meta-heurísticos. Na literatura é possível encontrar uma quantidade significativa de trabalhos que propõem este tipo de modificação. Estas modificações vão desde realizar a distribuição de cálculos elementares entre processadores até a busca *multi-thread* cooperativa (Alba, 2005; Azencott, 1992; Badeau *et al.*, 1997). Portanto, é possível notar uma vertente de pesquisa interessante, que procura utilizar todo o poder de processamento de hardware para melhorar algoritmos já propostos.

Em Garca-Lopez *et al.* (2002) foi proposto três estratégias para execução simultânea do VNS para executar instâncias de grande porte do problema p-mediana. A primeira estratégia faz o desmembrmento da busca local de um VNS sequencial, a fim de que as partes da busca executem de forma simultânea. A segunda estratégia envolveu execuções independentes de vários procedimentos sequenciais do VNS, com a melhor solução sendo selecionada no final. A terceira estratégia utilizou a abordagem mestre-escravo, onde um processador principal executa o VNS sequencial e as etapas de *Shake* e Busca Local são executadas simultaneamente por processadores escravos. As duas últimas estratégias foram as que obtiveram melhores resultados.

Em Sevkli e Aydin (2007) é proposto uma abordagem de execução simultânea do VNS para solução do problema *Job Shop Scheduling*. Nesse trabalho foram considerados quatro abordagens:

1. Estratégia cooperativa sincronizada proposta em Garca-Lopez *et al.* (2002);

2. Método assíncrono com coordenação centralizada proposta em Crainic *et al.* (2004);

3. Abordagem não centralizada utilizando topologia de anel unidirecional;

4. Abordagem não centralizada utilizando topologia de anel bidirecional.

Os métodos 3 e 4 são propostas novas dos autores. No método 3 cada processador executa um *shake* e uma busca local, enviando o resultado obtido ao próximo processador e coletando o resultado do processador anterior. A solução recém-chegada torna-se um ponto inicial para o próximo ciclo de execução independente da sua qualidade. O método 4 considera como ponto inicial para o próximo ciclo de execução a melhor de três soluções, a obtida pela sua própria execução e duas soluções recebidas dos processadores adjacentes. A investigação experimental mostrou que o método 3 superou os demais em relação à qualidade da solução.

Em Knausz (2008) é feito a combinação de uma busca local de tempo restrito com a meta-heurística *Randomized Variable Neighborhood Descent* (RVND). Diversas iterações da busca local são executas em diferentes vizinhanças. Então os processos são sincronizados e a melhor solução encontrada é propagada para a próxima fase da busca local. Os testes computacionais mostraram uma redução do tempo de computação. O VNS proposto neste trabalho resolve o Problema de Sequenciamento de Carros.

Eskandarpour *et al.* (2013) desenvolveu um modelo de rede pós-venda considerando decisões estratégicas e táticas, bem como conflitos objetivos para empresas de vendas. Para resolver este problema, é proposto uma VNS Paralelo Multi-objetivo, ou seja, o VNS considera mais de uma função objetivo para avaliar as soluções. O algoritmo proposto executa n instâncias do VNS simultaneamente e de forma independente. Uma instância, chamada de instância central, é utilizada para coordenação da busca e controle. A melhor solução de cada instância do VNS é enviada para a instância central no final de cada iteração. Então a instância central seleciona a melhor solução entre todas as recebidas (n soluções) e passa para as n instâncias de execução do VNS como uma solução inicial para a próxima iteração. Este procedimento continua até atingir um critério de parada, predefinido por um número iterações.

Djenic *et al.* (2016), apresentam um VNS para resolução do Problema de Locação de Terminal de Ônibus. Neste artigo é feito uma análise de qual parte do VNS tem maior consumo de processamento, para então definir qual etapa será executada de forma simultânea. Os autores paralelizam duas funções internas da busca local. Uma função calcula o primeiro e o segundo terminal mais próximo para cada estação disponível. E a segunda função calcula qual terminal é o melhor a ser fechado se o terminal t for estabelecido.

No trabalho de Sanchez-Oro *et al.* (2015), é proposto um VNS paralelo síncrono para resolver o problema de alocação dinâmica de memória em sistemas embarcados. O VNS proposto paraleliza a busca local da meta-heurística. No primeiro passo o algoritmo gera uma solução inicial aleatória, itera sobre ela a fim de atingir uma vizinhança máxima e então faz uma pertubação da solução. Em seguida, o algoritmo prossegue com a busca local paralela. Para cada processador, o VNS estabelece a região que a busca local irá melhorar e inicia a instância correspondente.

Em Davidovic e Crainic (2015) propõem um algoritmo para escalonar tarefas de comunicação em processadores. Para isso é proposto uma abordagem que faz a decomposição do espaço de soluções. Decomposição de vizinhança significa a divisão da vizinhança e a exploração das regiões em paralelo, cada região é explorada por um processador diferente. Cada processador examina os vizinhos da região associada dentro de uma única iteração de uma busca local.

O VNS_BLS proposto neste artigo difere do trabalho de Eskandarpour *et al.* (2013), pois considera-se uma abordagem mono objetiva e a abordagem para execução simultânea adotada é diferente, pois a busca local é executada de forma simultânea e não considerada várias execuções independentes do VNS. Já o trabalho de Djenic *et al.* (2016), propõem execução em paralelo de duas funções específicas da busca local, o que difere do VNS_BLS, que propõe a execução de várias buscas locais em simultâneo. Considerando o trabalho de Sanchez-Oro *et al.* (2015), as abordagens se assemelham pois propõem a execução simultânea da busca local, porém o VNS_BLS propõe 36 diferentes buscas locais. Em Davidovic e Crainic (2015), a abordagem utilizada é diferente pois usa-se a ideia de decomposição do espaço de solução, para que várias busca locais executem cada uma em um trecho do espaço de busca.

O VNS_BLS utiliza de 36 diferentes buscas locais que partem de uma mesma solução para tentar encontrar soluções com valor de função objetivo menor que a solução inicial. O VNS_BLS faz ainda uma comunicação entre as buscas independentes, utilizando a memória compartilhada que é responsável por armazenar a melhor solução obtida pelas buscas.

2.5 Considerações Finais

O objetivo desde capítulo foi descrever os algoritmos que serão utilizados nesse trabalho, como PA, PCR, k-swap e VNS, além de retratar trabalhos que abordam o assunto deste texto. De acordo com a literatura, nota-se que é possível encontrar diversas abordagens de VNS Paralelizados, no entanto nenhum utilizando buscas locais diferentes. Percebe-se

também que não há abordagens com execução simultânea para resolução do PEM. Com base nas informações recuperadas na revisão de literatura, podemos observar que até o presente momento não há trabalhos que utilizem um VNS com buscas locais simultâneas para resolução do PEM. O próximo capítulo apresenta a proposta deste trabalho.

3

VNS com Buscas Locais Simultâneas

Este capítulo descreve o VNS_BLS proposto neste trabalho e suas variações. Este algoritmo possui a estrutura básica de um VNS sequencial, porém possui diversas buscas locais executadas de forma simultânea. O capítulo tem como objetivo detalhar o VNS_BLS, bem como suas características e processos utilizados em sua implementação.

Em um VNS, explora-se apenas uma solução do espaço de busca por vez (considerando que espaço de busca é o conjunto de todas as soluções possíveis do problema), ou seja o algoritmo encontra uma solução e investiga a vizinhança desta. Devido ao fato do espaço de busca ser inviável de ser explorado inteiro, é natural que meta-heurísticas investiguem determinadas soluções por vez, utilizando uma função objetivo para determinar quais são as soluções mais promissoras de serem exploradas.

No entanto, devido há recursos computacionais como, disponibilidade de diversos núcleos para processamento que permitem executar diferentes instâncias de forma simultânea, surge a possibilidade de utilizar estes recursos para definir um algoritmo que execute buscas locais simultâneas com diferentes formas de modificar a solução. Ampliando assim o potencial de exploração no espaço de busca, com essa ideia, foi proposto o algoritmo VNS_BLS.

O VNS_BLS possui estrutura do VNS sequencial até a busca local. Neste trecho, ao invés de apresentar apenas uma busca local o VNS_BLS executa múltiplas buscas que investigam diferentes soluções do espaço de busca. Os resultados obtidos por cada busca local podem ser compartilhados entre elas, para que haja influência das buscas que obtém melhores resultados a fim de convergir para um mínimo local. A Figura - 3.1 apresenta a estrutura geral do VNS_BLS.

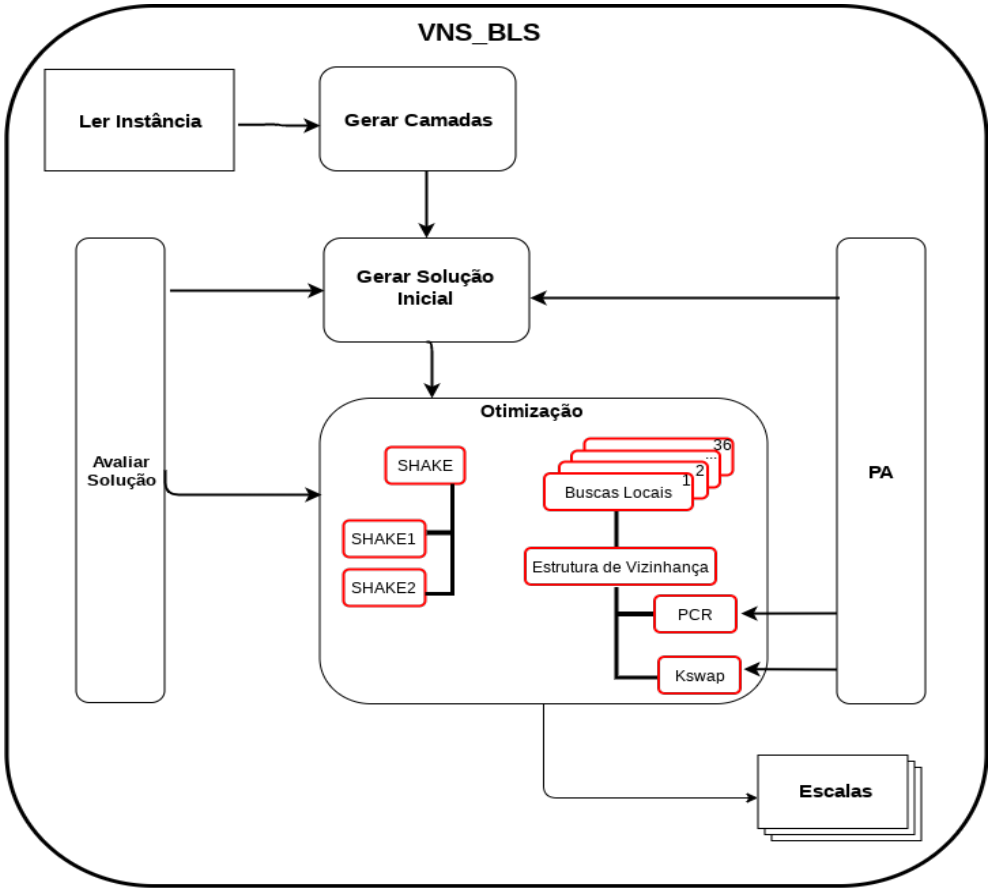


Figura 3.1: Estrutura geral do VNS_BLS.

Os módulos se dividem em: Ler Instância, Gerar Camadas, Gerar Solução Inicial, Avaliar Solução, PA, Otimização e Escalas. O módulo Otimização é composto por submódulos: *Shake*, que possui duas variações e Buscas Locais, que é composto por diferentes buscas locais. O módulo Buscas Locais, utiliza dois tipos de Estrutura de Vizinhança, que são PCR e k-swap.

O módulo Instância é responsável pela leitura dos arquivos e estruturação das informações. Este módulo recebe um arquivo de entrada e converte cada linha do arquivo em tarefas que possuem hora de início e término e local de partida e chegada.

Dada as tarefas como entrada, o módulo Gerar Camadas tem como objetivo separar este conjunto de tarefas em camadas. Cada camada deve conter tarefas que não podem ser sequenciadas entre si, dessa forma tarefas de diferentes camadas podem ser sequenciadas.

Em Gerar Solução Inicial combina-se cada tarefa de camadas diferentes e verifica-se qual combinação resulta em um menor valor de função objetivo. A combinação que obtiver

uma escala de trabalho com menor valor de função objetivo é escolhida. Os módulos PA e Avaliar Solução são transversais pois são utilizados nos módulos Gerar Solução Inicial e Otimização, ambos auxiliam no processo destes dois módulos.

O módulo Otimização, compreende o VNS que otimiza a solução inicial. O VNS_BLS implementado neste trabalho utiliza 36 diferentes busca locais a fim de intensificar a busca no espaço de soluções. Para avaliar as soluções construídas em um processo de otimização é utilizada uma função objetivo, que deve avaliar a qualidade da solução de acordo com características do problema. A função objetivo utilizada neste trabalho atribui um valor a uma solução S caso ela seja viável, caso contrário, o valor da função objetivo será ∞.

3.1 Gerar Camadas

O processo de gerar camadas tem como entrada um conjunto T de tarefas que são reorganizadas em camadas, sub-conjuntos, que podem ser sequenciais entre si. Na Figura - 3.2 existem 9 tarefas distribuídas ao longo das cinco da manhã até o meio dia. As tarefas estão separadas em 3 camadas, cada uma contendo 3 tarefas. As tarefas da camada 1 podem ser sequenciadas com as da camada 2, que são sequenciadas com as da camada 3. As tarefas contidas em cada camada, não podem ser sequenciadas entre si.

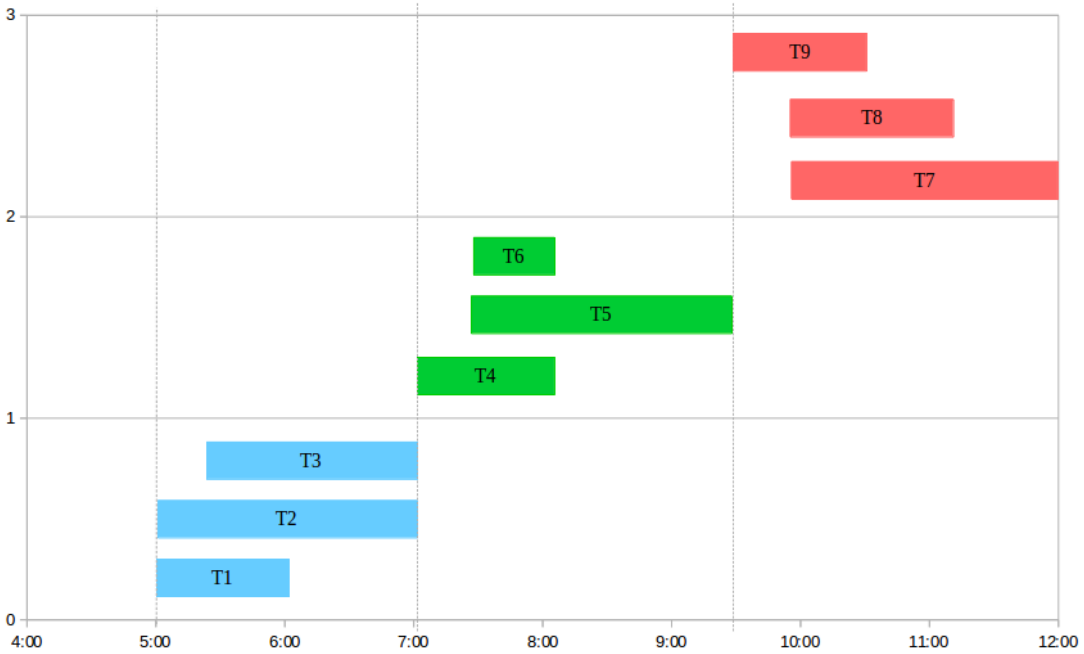


Figura 3.2: Divisão das camadas feitas pelo VNS_BLS.

Considerando uma sequência de camadas C e uma variável i como índice dessa sequência, temos que C é gerada de forma que as tarefas contidas na camada $C[i]$ possam ser sequenciadas com pelo menos uma tarefa da camada $C[i+1]$. Isso significa que as tarefas que estão em uma camada não podem ser sequenciadas entre elas. O Algoritmo 4 ilustra o processo de geração de camadas, onde a saída do método é uma sequência C de camadas.

Algoritmo 4 Procedimento de geração de camadas

```
1: function GERAR-CAMADAS(T)
2:     C ← {}
3:     while T ≠ ∅ do
4:         t ← remover-mais-cedo(T)
5:         i ← |C|
6:         while i > 0  &  não-pode-sequenciar(C[i − 1], t) do
7:             i ← i - 1
8:         C[i] ← C[i] ∪ t
9: return C
```

Os métodos do algoritmo tem as seguintes finalidades:

- *remover_mais_cedo*: remove da lista de tarefas T a tarefa que inicia mais cedo.
- *nao_pode_sequenciar*: verifica se a tarefa t não é sequenciada com pelo menos uma tarefa de $C[i-1]$ (Dos Santos, 2016).

3.2 Gerar Solução Inicial

Para gerar a solução inicial, inicia com um conjunto vazio de jornadas J e percorre-se a sequência de camadas C, gerada pelo procedimento geração de camadas, a fim de formar o conjunto de jornadas J. Para formar o conjunto J, resolve-se um PA para cada camada $C[i]$ da sequência C. O PA deve verificar o custo de atribuir as tarefas da camada $C[i]$ a uma jornada existente em J, caso haja inviabilidade de atribuição, cria-se uma nova jornada em J.

Cada elemento a_{tj} da matriz de custos $A_{|T|+|J|} = [a_{ij}]$, indica o custo de atribuição da tarefa do índice t para uma jornada j. A Matriz 3.1 ilustra a divisão da matriz de custos, $|J|$ representa a quantidade de jornadas e $|T|$ a quantidade de tarefas.

$$M = \begin{bmatrix} \text{Bloco 1} & \text{Bloco 2} \\ \text{se } t < |T| \text{ e } j < |J| & \text{se } t < |T| \text{ e } j \geq |J| \\ a_{tj} = \text{g(t,j) ou } a_{tj} = \infty & a_{tj} = \text{g(j) ou } a_{tj} = \infty \\ \text{Bloco 3} & \text{Bloco 4} \\ \text{se } t \geq |T| \text{ e } j < |J| & \text{se } t \geq |T| \text{ e } j \geq |J| \\ a_{tj} = c_t \text{ ou } a_{tj} = \infty & a_{tj} = 0 \end{bmatrix} \tag{3.1}$$

- **Bloco 1**: trata as atribuições de tarefas reais às jornadas reais. Caso haja viabilidade da atribuição, a posição $a_{t,j}$ assume o valor da função $g(t, j)$. Sendo $g(t, j)$ a função objetivo da nova jornada gerada ao sequenciar a tarefa t à jornada j. Caso haja, inviabilidade da atribuição, a posição a_{tj} assume valor infinito.

- **Bloco 2**: trata casos de atribuição de tarefas fictícias às jornadas reais, estas tarefas representam possíveis tempos de parada na jornada do motorista. Sendo $g(nulo, j)$ é a função objetivo da jornada j, passando *nulo* no parâmetro da tarefa que deveria ser atribuída a jornada. Caso haja inviabilidade de atribuição $a_{tj} = \infty$.

- **Bloco 3**: trata os casos de atribuição de tarefas reais às jornadas fictícias. Seja $a_{tj} = c_t$, onde c_t um custo mínimo atribuído ao se se criar uma jornada não vazia; caso seja viável designar a tarefa t para sua jornada correspondente do Bloco 1, $a_{tj} = \infty$. Isso é feito para que tarefas sejam designadas para jornadas existentes, ao invés de criar novas jornadas;

- **Bloco 4**: trata a atribuição de tarefas inexistentes para jornadas inexistentes portanto, não possuem custo.

Dada uma matriz de custos A, ela é então designada para um algoritmo que resolve PA, explicado na Seção 2.3.1, para selecionar a combinação com menor custo associado. O processo para gerar a solução inicial é ilustrado no algoritmo 5.

Algoritmo 5 Procedimento de geração da solução inicial (Dos Santos, 2016)

```
1: function SOLUÇÃO-INICIAL(C)
2:     S ← ∅
3:     for i = 0 ate |C| faca do
4:         A ← construir-matriz-de-custos(S, C[i])
5:         J ← problema-de-atribuição(A)
6:         S ← S ∪ novas-jornadas(S, J)
       return J
```

O Algoritmo 5 recebe como entrada um conjunto de camadas C. A função $construir-matriz-de-custos(S, C[i])$ constrói a matriz de custos inserindo as tarefas da camada $C[i]$ nas jornadas do conjunto S, ou seja o método verifica a possibilidade de combinação das tarefas em cada jornada para designar a combinação com menor custo. Em seguida, o método *problema-de-atribuição*(A) é executado a fim de determinar a combinação que gerou uma jornada de menor custo. O método $novas-jornadas(S, J)$ insere a combinação que resultou menor custo total à S.

3.3 Otimização

O módulo Otimização tem como objetivo melhorar a solução inicial, operando sucessivas modificações na solução. Para efetuar estas modificações são utilizadas as estruturas de vizinhanças PCR e k-swap, que foram explicado nas Seções 2.3.2, 2.3.3.

Considerando que os dados do problema são representados por um grafo multipartido, sendo que cada partição de vértices são divididas por informações temporais das tarefas, hora de partida e chegada, então é possível estabelecer uma ordem destas partições, para que sejam percorridas. Sendo assim há duas formas de percurso:

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

A Tabela - 3.1 apresenta as possíveis combinações para percorrer a vizinhança com os critérios de parada dos operadores, os critérios foram explicados na Seção 2.2.1.

Tabela 3.1: Variações das buscas locais utilizadas pelo VNS_BLS.

Nome	Variações	Exploração de Vizinhos	Critério Parada	Total
PCR	-	Forward (F) Backward (B)	First (FI) Best (BS) Continuous (CN)	6
k-swap	k = 1,...,5	Forward (F) Backward (B)	First (FI) Best (BS) Continuous (CN)	30

A coluna *variações* indica se existe variação do operador, como pode ser observado, k-swap possui 5 variações considerando de 1 até 5 cortes de distância na linha do tempo. A coluna *exploração de vizinhos* indica como a vizinhança será percorrida. Por fim, *critério de parada* indica qual dos critérios será utilizado. Estes três fatores combinados, resultam

em 36 possíveis formar de modificar uma solução, que são combinados em uma busca local, originando assim 36 buscas locais diferentes. O módulo Buscas Locais, evidencia que o VNS_BLS possui 36 buscas locais, onde cada busca utiliza uma forma de movimentar a solução.

O módulo Otimização possui um submódulo chamado *Shake*. A função *shake* tem como objetivo diversificar a solução corrente a fim de evitar que a busca caia em mínimos ou máximos locais. O *shake* utilizado neste trabalho controla a quantidade de recombinações que devem ser feitas em uma solução, elas indicam sucessivamente 15% e 30% de recombinações em uma solução, ou seja, 15% e 30% das jornadas da solução devem ser recombinadas (Dos Santos, 2016). Há duas funções de *shake* implementadas:

- *Shake1Slice*: escolhe de forma aleatória uma jornada j de uma solução S e faz um corte aleatório em x, onde este corte, consiste na segmentação da solução definida dentro do intervalo tempo de início e fim de j. Recombina-se então as jornadas presentes nestes dois subconjuntos gerados pelo corte na solução S, onde o critério para recombinação consiste apenas na viabilidade da combinação (Seção 4.2.1).

- *Shake2Slice*: nesta versão é feita dois cortes na solução S, similar ao procedimento k-swap. Os segmentos delimitados pelos dois cortes são recombinados com o mesmo critério da função *Shake1Slice*.

Caso os cortes aleatórios não gerem a percentagem de recombinações indicadas, ou seja não atinjam 15% ou 30% das jornadas, um novo corte é feito e o processo se repete até que o número de recombinações seja alcançado.

A estratégia utilizada neste trabalho considera um VNS sequencial com diferentes buscas locais executadas de forma simultânea. Um processador se torna mestre e executa as etapas básicas do VNS sequencial até a etapa da busca local. Nesta etapa, várias buscas são executada em simultâneo por processadores escravos. O Algoritmo 6 apresenta o algoritmo de busca local utilizado pelo VNS_BLS. Seja, $MV_l(S)$, $l = 1, 2, ..., 36$ o conjunto de possíveis operações a serem feitas para modificar S.

As buscas locais são executadas simultaneamente a cada iteração do VNS_BLS e armazenam suas soluções em uma memória compartilhada ($atualizaMemoriaCompartilhada$). Se a solução gerada pela busca for melhor que a solução existente na memória compartilhada, esta é atualizada, caso contrário nada é feito. Ao término da execução das buscas, a memória compartilhada terá a melhor solução obtida pelas 36 buscas locais.

Como são diferentes buscas executadas em cada processador é possível que algumas consumam mais tempo de execução que outras. Dessa forma, há a possibilidade de

Algoritmo 6 Procedimento de Busca Local

```
1: function BUSCALOCAL(S, l)
2:     while melhora S do
3:         S' ← MV_l(S)
4:         atualizaMemoriaCompartilhada(S')
5:         if f(S') < f(S) then
6:             S ← S'
       return S
```

que uma busca termine sua execução e ainda tenha outras sendo executadas. A busca que terminou mais cedo fica esperando que todas acabem para que o algoritmo siga o fluxo normal. No entanto, há a possibilidade de alterar o algoritmo para que a busca que terminou continue executando algumas iterações como forma de aproveitamento da ociosidade dos processadores. Gerando assim dois comportamentos:

- Com ociosidade: quando uma busca acabar sua execução ela fica aguardando que todas as buscas acabem para que siga o fluxo do VNS_BLS.
- Sem ociosidade: quando uma busca acabar sua execução ela verifica se todas as buscas já acabaram, caso nem todas tenham terminado de executar, é feita a execução de mais uma iteração da busca até que todas terminem.

Como há uma memória compartilhada entre todas as buscas, cada vez que uma busca obtém uma solução ela consulta a memória compartilhada e caso a solução gerada seja melhor que a presente na memória compartilhada esta é então atualizada. Há dois comportamentos possíveis para o algoritmo nesta situação:

- Comunicação Unidirecional: a busca local apenas deposita sua solução na memória compartilhada caso a solução corrente da busca seja melhor que a solução da memória compartilhada.
- Comunicação Bidirecional: a busca local deposita e resgata a solução da memória compartilhada caso esta seja melhor que a solução corrente da busca.

Sendo assim é possível propor 4 variações do VNS_BLS:

- **VNS_BLS_SC_CO**: Nesta versão as buscas locais depositam suas soluções na memória compartilhada caso a solução gerada seja melhor que a solução atual da memória compartilhada. A cada iteração do VNS_BLS_SC_CO a solução inicial das buscas é atualizada pela solução que esta na memória compartilhada. Quando uma

busca termina, ela fica ociosa esperando que todas acabem para que a próxima iteração do VNS possa acontecer. Então esta é uma busca com comunicação unidirecional e com ociosidade.

- **VNS_BLS_CC_CO**: Nesta versão as buscas locais depositam suas soluções na memória compartilhada e, também, resgatam a solução da memória compartilhada caso esta seja melhor que a solução da busca. Nesta versão há a característica de intensificação da busca em regiões que tem boas soluções. Busca com comunicação bidirecional e com ociosidade.

- **VNS_BLS_SC_SO**: A principal característica desta versão do VNS_BLS é que quando uma busca local termina, ela observa se todas já terminaram, caso não a busca em questão irá iterar mais uma vez até que todas as buscas tenham terminado. Dessa maneira as buscas que até então ficavam ociosas, têm a oportunidade de explorar mais ao espaço de solução. A condição de parada da busca nesta versão, será também a quantidade de buscas locais finalizadas. Quando uma solução gerada não é melhor que a solução atual (S') é então incrementado as buscas finalizadas e se faz a verificação se todas as buscas já finalizaram. Caso não, então é feito um novo *shake* e a busca continua. Busca com comunicação unidirecional e sem ociosidade.

- **VNS_BLS_CC_SO**: possui ambas as características, tanto de continuar a busca local enquanto todas as buscas não tenham terminado, como de inserir e resgatar soluções na memória compartilhada. Busca com comunicação bidirecional e sem ociosidade.

O Algoritmo 7 descreve o VNS_BLS e suas variações.

O VNS_BLS recebe por parâmetro uma solução S, que é a solução obtida pelo módulo Gerar Solução Inicial, o parâmetro z_{max} é o número máximo de buscas locais que serão utilizadas, o conjunto MV com as variações de modificações para serem efetuadas na solução durante a busca local, l_{max} indica a quantidade de operações possíveis, para este trabalho 36 e *op* para representar qual versão do VNS_BLS será executada. O método $Shake(S, z)$ gera uma solução S' vizinha a S, através de modificações na solução S. A solução S' é utilizada pelas buscas locais. Na linha 6 está laço de repetição que demonstra as buscas que irão ocorrer em simultâneo. As 36 buscas locais serão executadas simultaneamente de acordo com o variação da modificação da solução indicada por MV_lS'. Entre as linhas 11 a 26, é descrito as variações do VNS_BLS. Após a execução das buscas, verifica-se (Linha 8) se a solução S'', obtida pela busca local, possui função objetivo menor que S, se sim, S é atualiza para S'' e z recebe 1, para iniciar a busca nas vizinhanças da

Algoritmo 7 Algoritmo do VNS_BLS

```
1: function VNS_BLS(S, z_max, l_max, MV, op)
2:     z ← 1
3:     solucaoCompartilhada = {}
4:     while z ≤ z_max do
5:         S' ← Shake(S, z)
6:         for all l = 1atel_max do
7:             while melhora S' do
8:                 S'' ← MV_l(S')
9:                 atualizaMemoriaCompartilhada(S'')
10:                switch op do
11:                    case VNS_BLS_SC_CO
12:                        if f(S'') < f(S') then
13:                            S' ← S''
14:                    case VNS_BLS_CC_CO
15:                        if f(solucaoCompartilhada) < f(S'') then
16:                            S'' ← solucaoCompartilhada
17:                        if f(S'') < f(S') then
18:                            S' ← S''
19:                    case VNS_BLS_SC_SO
20:                        if f(S'') < f(S') then
21:                            S' ← S''
22:                        else
23:                            buscas_finalizadas = buscas_finalizadas + 1
24:                            if buscas_finalizadas < total_buscas then
25:                                S' = shake(S')
26:                    case VNS_BLS_CC_SO
27:                        if f(solucaoCompartilhada) < f(S'') then
28:                            S'' ← solucaoCompartilhada
29:                        if f(S'') < f(S') then
30:                            S' ← S''
31:                        else
32:                            buscas_finalizadas = buscas_finalizadas + 1
33:                            if buscas_finalizadas < z_max then
34:                                S' = shake(S')
35:        if f(solucaoCompartilhada) < f(S) then
36:            S ← solucaoCompartilhada
37:            z ← 1
38:        else
39:            z ← z + 1
           return S
```

nova solução. Caso contrário, o valor de z é incrementado e para que a buscas vá para outra região do espaço de solução.

4

Resultados Computacionais

Este capítulo descreve os resultados obtidos pela abordagem proposta neste trabalho. O Capítulo encontra-se divido da seguinte maneira: Seção 4.1 trata das questões de pesquisa abordadas neste trabalho. A Seção 4.2 mostra qual foi o ambiente experimental utilizado para execução dos experimentos realizados, as instâncias utilizadas, configurações das máquinas onde foram executados os experimentos, especificações do problema e a função de objetivo da solução. A Seção 4.3 traz uma análise dos resultados computacionais com base na evolução da qualidade da solução quando comparada com a solução inicial. Na Seção 4.4 é analisado a influência de cada busca local na construção das soluções. A comparação entre as diferentes versões do algoritmo é tratada na Seção 4.5. Nesta seção analisamos as versões com relação a qualidade da solução obtida e ao tempo computacional, com objetivo de determinar qual versão possui melhor desempenho. A Seção 4.6 traz a comparação da melhor versão do VNS_BLS com algumas propostas encontradas na literatura. Na seção 4.7 é mostrado versões do VNS_BLS com algumas buscas locais retiradas.

4.1 Questões de pesquisa

As questões de pesquisa tem como objetivo guiar a análise de resultados computacionais obtidos pela proposta do trabalho. As questões elaboradas neste trabalho são:

- Qual o desempenho computacional de cada versão do VNS_BLS com relação a melhora da solução inicial?

- Quais buscas locais são mais eficientes para melhorar a solução em cada versão do VNS_BLS?
- Qual versão do VNS_BLS tem melhor desempenho com relação a qualidade da solução e tempo computacional?
- Qual o desempenho da versão do VNS_BLS que obtiver melhores resultados comparado com outros algoritmos da literatura?

4.2 Ambiente Experimental

Os experimentos realizados neste trabalho foram executados no seguinte ambiente experimental:

- Plataforma de Hardware: Os experimentos foram executados em um servidor com sistema operacional Ubuntu 16.04.3 LTS, 31 GB de memória e processador Intel(R) Xeon(R) CPU E7-4860 v2 @ 2.60GHz com 60 núcleos.
- Platarforma de Software: O algoritmo proposto foi desenvolvido na linguagem C e o compilador usado foi GNU Compiler Conection (GCC) 5.4. Para implementar as buscas simultâneas foi utilizado a biblioteca POSIX Threads Library (Pthread).
- Dados Experimentais: Cada versão do VNS_BLS foi executada 5 vezes para cada instância e os resultados apresentados representam a média entre as 5 execuções.
- Instâncias: Foram utilizadas 3 instâncias reais, baseadas nas linhas de transporte rodoviário urbano da cidade de Maringá, Paraná para a realização dos experimentos [1]. As instâncias que serão utilizadas e a quantidade de tarefas de cada instância são demonstradas na Tabela - 4.1. Os experimentos executados neste trabalho levaram cerca de 60 dias.

Tabela 4.1: Instâncias utilizadas para execução dos algoritmos propostos.

Nome	**Tarefas**	**Tipo**
Inst412	412	Real
Inst2313	2313	Real
Inst3478	3478	Real

A Tabela - 4.2 apresenta o valor da função objetivo e a quantidade de jornadas das soluções iniciais, obtidas pelo VNS_BLS.

[1] www.gpea.uem.br/benchmark.html

Tabela 4.2: Soluções iniciais.

Nome	Tarefas	F(x)	$\|J\|$
Inst412	412	38.453	79
Inst2313	2313	194.867	389
Inst3478	3478	283.508	566

Como a construção da solução inicial é determinística (Seção 3.2), as soluções inicias serão sempre iguais para todas as versões do VNS_BLS.

4.2.1 Parâmetros do Problema

Uma escala viável deve respeitar especificações legais estabelecidas pelas empresas de transporte público, que podem mudar de acordo com a empresa e estado considerado. As especificações consideradas neste trabalho, são condizentes a empresa responsável pelo transporte público da cidade de Maringá, Paraná:

1. A jornada de trabalho normal de motorista tem duração de 7 horas e 20 minutos (440 minutos). Caso a jornada realizada seja menor, paga-se pelo tempo mínimo de trabalho. Para jornadas que ultrapassam o tempo máximo, são pagas horas extras pelo tempo excedido;

2. O tempo máximo de horas extras trabalhadas não deve exceder 2 horas (120 minutos), então uma jornada, com horas extras, poderá ter duração máxima de 9 horas e 20 minutos (560 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 horas (360 minutos). Quando a duração da jornada ultrapassar 6 horas, deve ser concedido um intervalo para descanso;

5. O intervalo para descanso deverá ter duração mínima de 1 hora e 30 minutos (90 minutos) e máxima de 5 horas (300 minutos);

6. A possibilidade de sequenciamento de duas tarefas t_i e t_j é definida por duas restrições:

    - O tempo de término t_i deve ser menor ou igual ao tempo de início de t_j;

- O ponto de troca de término de t_i deve ser o mesmo que o ponto de troca de início de t_j;

7. A extensão máxima da jornada, levando em consideração tempo trabalhado e tempo de descanso, deve ser no máximo 13 horas (780 minutos).

8. A hora paga no período noturno, entre as 22 horas de um dia e às 5 horas da manhã, possui duração de 52,5 minutos;

9. As horas trabalhados no horário noturno têm um adicional de 20% sobre o custo da jornada normal;

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

Uma solução deste problema é composta por um conjunto de jornadas, que reúne todas as tarefas de um determinado dia, de forma que o custo total das jornadas e o número de mudanças de veículos sejam minimizados, consequentemente minimizando o número de jornadas (Constantino *et al.*, 2017).

4.2.2 Função Objetivo

Em um problema de otimização a função objetivo deve avaliar a solução gerada de acordo com características do problema. A função objetivo utilizada neste trabalho atribui um valor de custo a uma solução S se ela for viável, caso contrário, o valor da função objetivo é ∞.

A função objetivo é montada com base na especificação do problema e é calculada com base no custo de cada jornada da solução. O valor de custo da solução é obtida pela função $C(x)$. Portanto a função objetivo utilizada neste trabalho é definida da seguinte forma, como mostra a Equação 4.1:

$$F(x) = \begin{cases} C(x) & \text{se as restrições 2, 4, 5, 6 e 7 são satisfeitas} \\ \infty & \text{caso contrário} \end{cases} \tag{4.1}$$

A Equação 4.2 define $C(x)$. Como a hora paga entre 22 horas e 5 da manhã, possui 52.5 minutos $T_n(x)$ faz a transformação do tempo noturno $t_n(x)$ em duração de 52.5. A Equação 4.3 define $T_n(x)$. Sejam $t_n(x)$ e $t_d(x)$ o tempo noturno e diurno de uma jornada x que um motorista executa. Seja t_{max} o tempo máximo que uma jornada pode ter (7 horas e 20 minutos). A multiplicação por 1,2 representa o adicional de 20% nas corridas

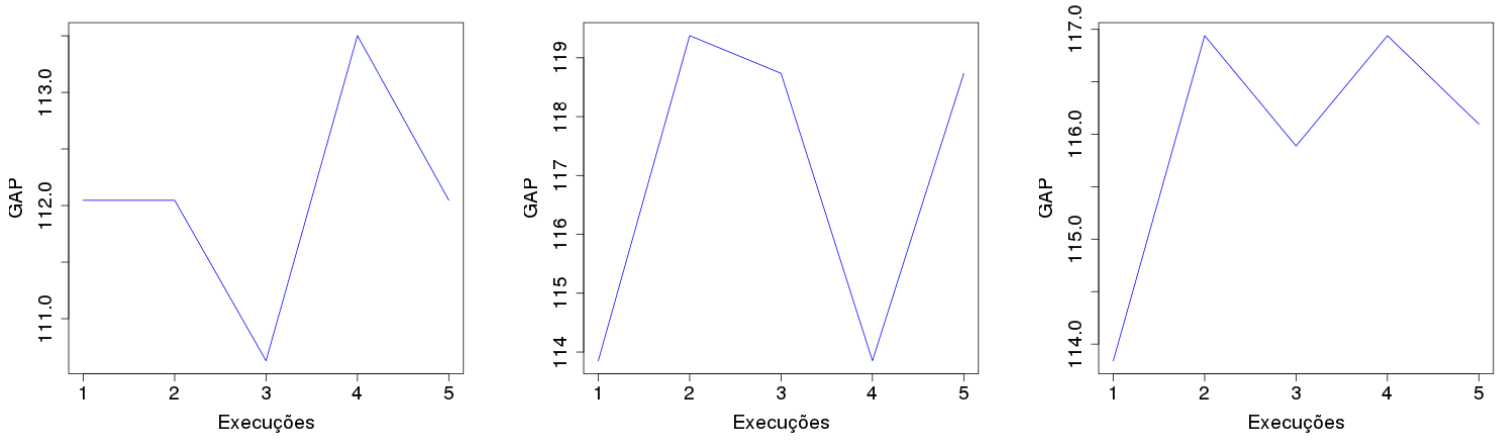
noturnas e a multiplicação por 1,5 representa o pagamento de 50% por horas extras. Esta função objetivo foi definida em Dos Santos (2016).

$$C(x) = 1,2.T_n(x) + t_d(x) + 1,5.(T_n(x) + t_d(x) - t_{max}) \tag{4.2}$$

$$T_n(x) = t_n(x).60/52.5 \tag{4.3}$$

4.3 Desempenho das versões do VNS_BLS com relação a melhora da solução inicial

O objetivo desta seção é analisar como o algoritmo se comporta em relação a melhora da solução inicial, para isso, foi comparado para cada execução das versões do VNS_BLS qual o GAP entre a solução inicial e a solução final. O GAP é a razão dos resultados entre duas soluções transformados em percentagem, $GAP = ((PropostaComparada/PropostaNova) - 1) * 100$. A Figura - 4.1 ilustra o GAP das soluções obtidas pela versão VNS_BLS_SC_CO (Seção 3.3) em relação a solução inicial.



(a) Inst412 (b) Inst2313 (c) Inst3478

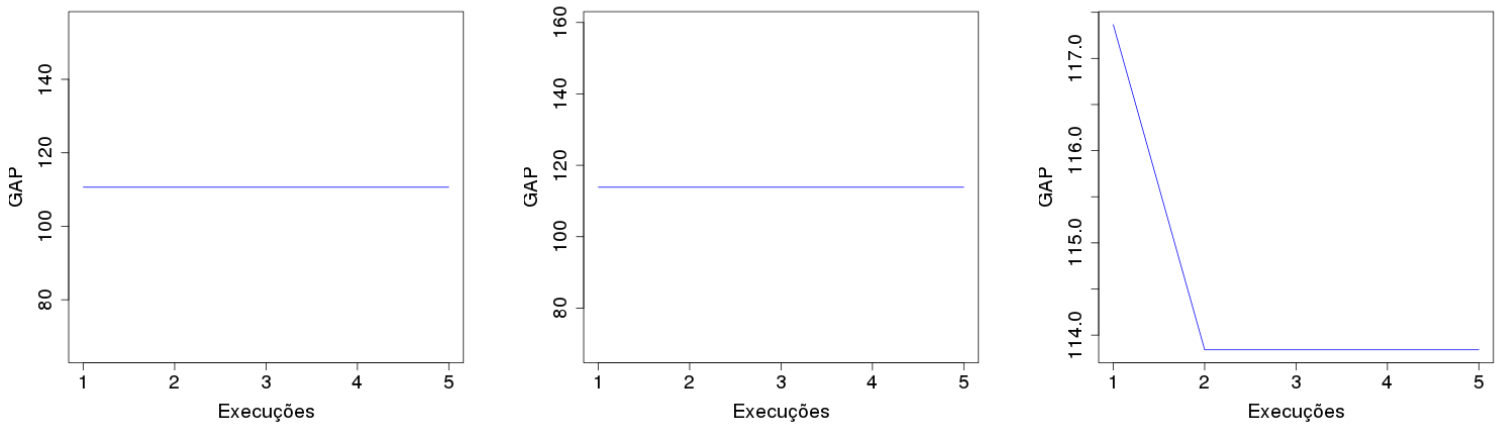
Figura 4.1: Melhoria da função objetivo obtida em relação a solução inicial versão VNS_BLS_SC_CO.

Na Figura - 4.1(a) nota-se que quatro das cinco iterações do algoritmo obtiveram GAP maior que 112%. Já a iteração 3, obteve menos de 110%, apesar de ser bem inferior

ás outras ainda é uma solução valida pois otimizou a solução inicial, não houve grande variação do GAP das soluções entre as iteração.

A Figura - 4.1(b) mostra o GAP da instância Inst2313. A versão VNS_BLS_SC_CO obteve um GAP maior que Inst412, cerca de 119%. A iteração 1 e 3 foram as que tiveram menor GAP, porém ainda assim maior que 100%.

Para Inst3478, Figura - 4.1(c), o GAP variou de 114% á 117%. Nota-se que para todas as instâncias o VNS_BLS_SC_CO obteve GAP em relação a solução inicial acima de 100%. Existem mais execuções que produzem bons resultados do que produzem maus resultados, considerando que maus resultados são os que não possuem GAP alto. Um GAP baixo significa que a diferença entra a solução inicial e final é pequena, portanto não houve melhora da solução. O GAP da versão VNS_BLS_CC_CO é apresentado na Figura - 4.2.



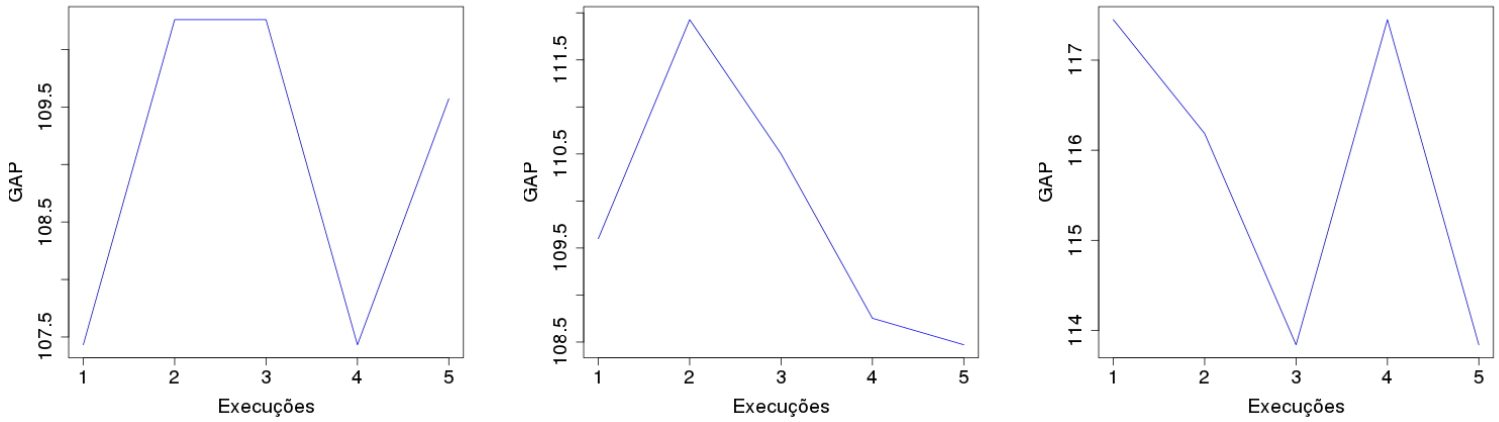
(a) Inst412 (b) Inst2313 (c) Inst3478

Figura 4.2: Melhoria da função objetivo obtida em relação a solução inicial versão VNS_BLS_CC_CO.

Observa-se que, o algoritmo manteve a mesma solução para quase todas as instâncias, resultando em um GAP constante. Em todas as execuções é encontrado uma solução com o mesmo valor de função objetivo. É importante lembrar que por mais que o valor de função objetivo seja igual nas diferentes execuções do algoritmo, não necessariamente a solução é a mesma, pois pode haver diferentes combinações que resultam no mesmo valor de função objetivo. Como esta versão do algoritmo possui comunicação bidirecional (Seção 3.3), ou seja ela pode armazenar e recuperar solução na memória compartilhada, conclui-se que há uma intensificação das buscas em torno de uma solução consequentemente havendo

pouca diversificação. Fazendo assim, com que as buscas não explorem o espaço de soluções. No entanto ainda assim, a VNS_BLS_CC_CO obteve GAP acima de 100% em todas as execuções de todas as instâncias.

A Figura - 4.3 apresenta o desempenho da versão VNS_BLS_SC_SO. Para todas as instâncias o algoritmo obteve GAP acima de 100%. O menor GAP obtido foi para a Inst412 na iteração 4, que obteve 107%, seguida da 4 iteração da Inst2313 que obteve 108%.

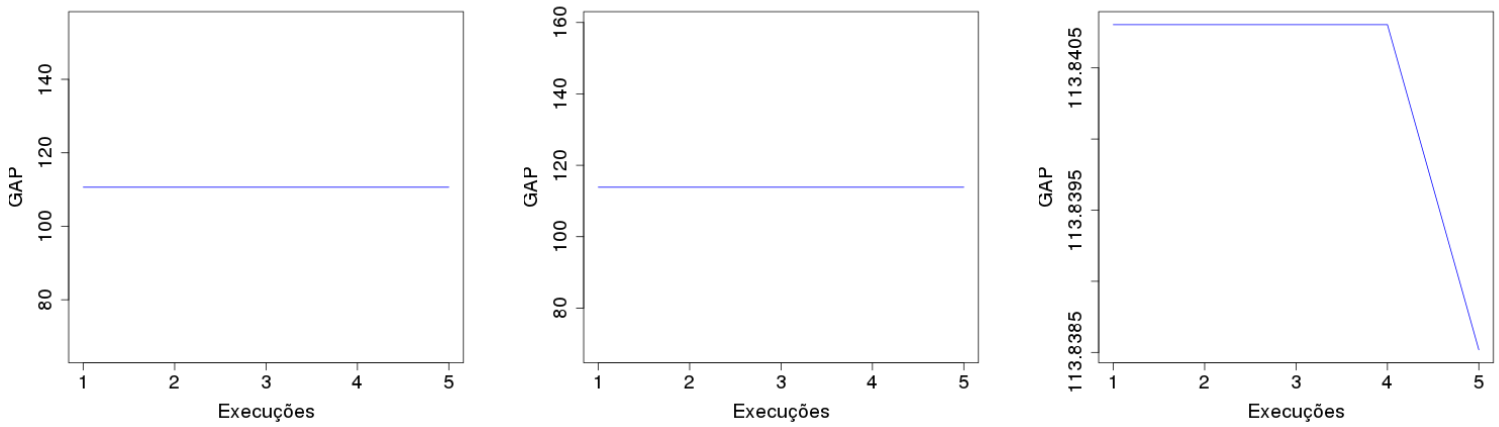


(a) Inst412 (b) Inst2313 (c) Inst3478

Figura 4.3: Melhoria da função objetivo obtida em relação a solução inicial versão VNS_BLS_SC_SO.

A versão VNS_BLS_CC_SO obteve o mesmo comportamento da versão VNS_BLS_CC_CO, onde para duas instâncias, Inst412 e Inst2313, o GAP foi constante para todas as execuções do algoritmo, como pode ser visualizado em Figura - 4.4(a) e Figura - 4.4(b). Ins3478 obteve GAP de 113.8405% para todas as iterações, exceto a 5 execução onde o GAP foi de 113.8385%. O GAP para esta instância variou pouco da melhor para a pior iteração, ficando praticamente constante. A partir destes resultados é possível concluir que quando há comunicação bidirecional entre a memória compartilhada o GAP tente a ser constante, devido a característica de intensificação da busca. Portanto, esta característica faz com que a meta-heurística não investigue outras regiões do espaço.

Nota-se que algumas execuções exploram regiões não promissoras do espaço de solução, o que resulta em resultados finais ruins, com um GAP em relação a solução inicial baixo, significando que, a diferença entre o valor de função objetivo entre a solução inicial e

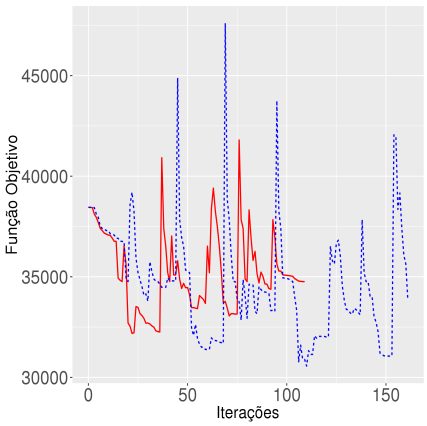


(a) Inst412 (b) Inst2313 (c) Inst3478

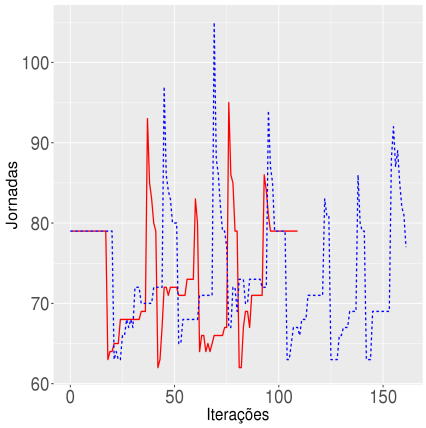
Figura 4.4: Melhoria da função objetivo obtida em relação a solução inicial versão VNS_BLS_CC_SO.

final é pequeno. Outro fato a ser destacado, é que não há necessidade de exatas 5 execuções do VNS_BLS para que se encontre a melhor solução. Pelas figuras percebe-se que normalmente até a terceira execução já foi obtido a solução com menor valor de função objetivo, mostrando que o algoritmo se estabiliza em um mínimo local até a terceira para a maioria dos casos.

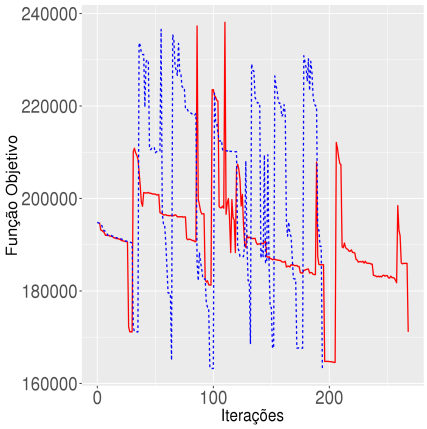
A Figura - 4.5, Figura - 4.6, Figura - 4.7, Figura - 4.8 apresentam a evolução da função objetivo e do número de jornadas, da melhor (linha azul) e da pior solução obtida (linha vermelha) ao longo das execuções.



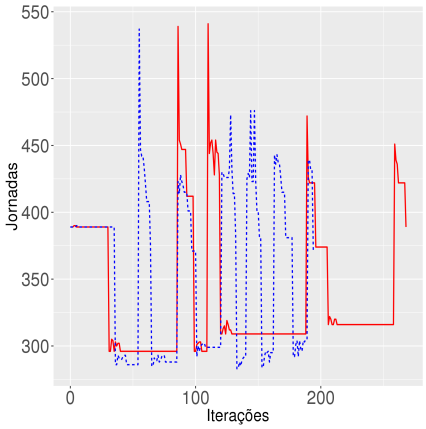
(a) Inst412 - Função Objetivo



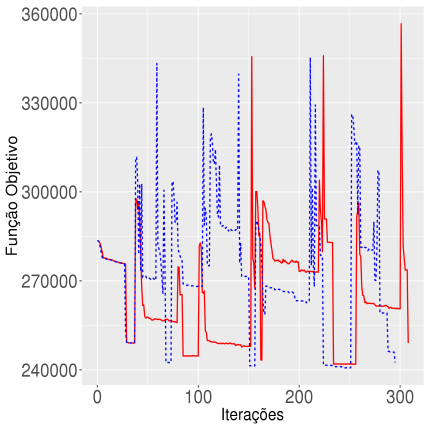
(b) Inst412 - Jornadas



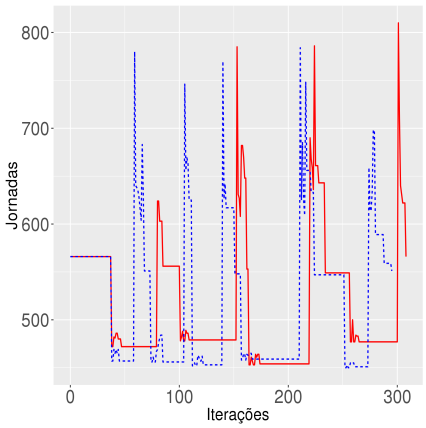
(c) Inst2313 - Função Objetivo



(d) Inst2313 - Jornadas

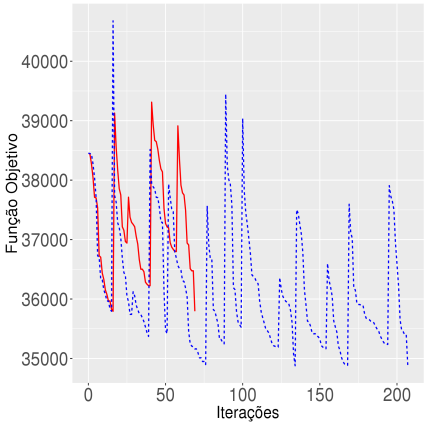


(e) Inst3478 - Função Objetivo

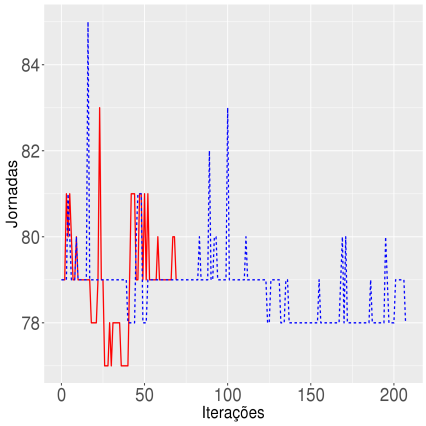


(f) Inst3478 - Jornadas

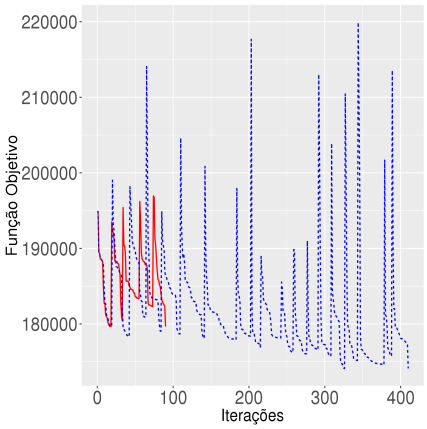
Figura 4.5: Evolução da função objetivo e número de jornadas ao longo das iterações da versão VNS_BLS_SC_CO. Linha vermelha - pior solução; Linha azul - melhor solução



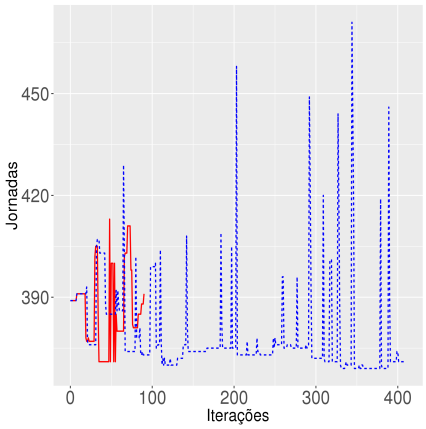
(a) Inst412 - Função Objetivo



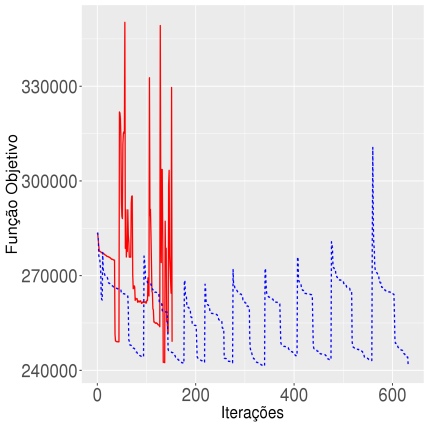
(b) Inst412 - Jornadas



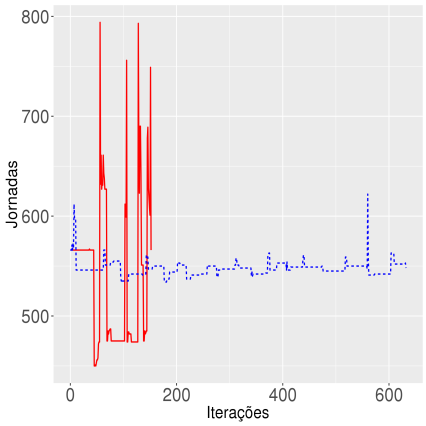
(c) Inst2313 - Função Objetivo



(d) Inst2313 - Jornadas

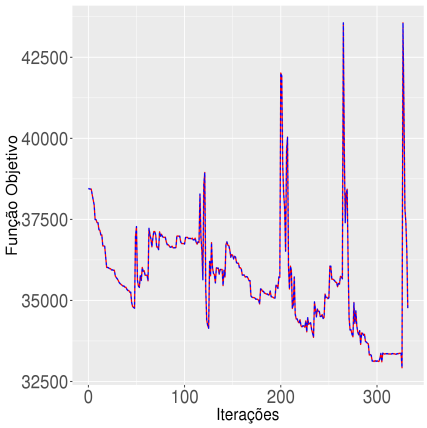


(e) Inst3478 - Função Objetivo

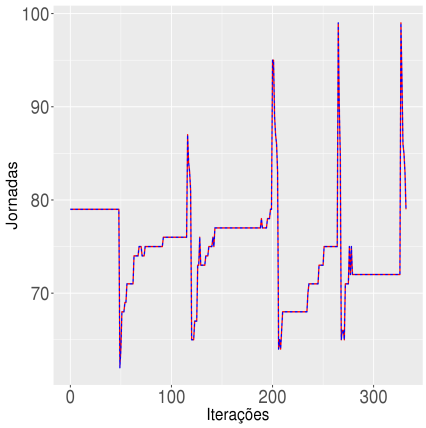


(f) Inst3478 - Jornadas

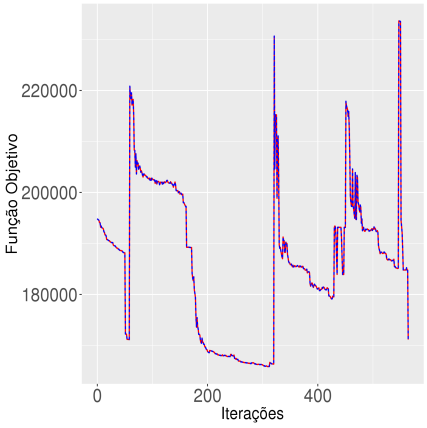
Figura 4.6: Evolução da função objetivo e número de jornadas ao longo das iterações da versão VNS_BLS_SC_SO. Linha vermelha - pior solução; Linha azul - melhor solução



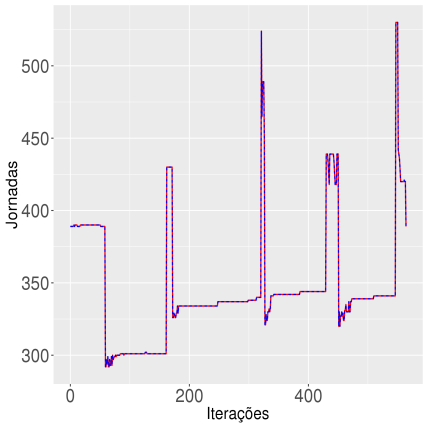
(a) Inst412 - Função Objetivo



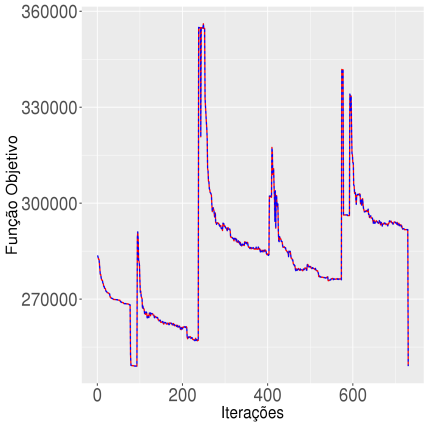
(b) Inst412 - Jornadas



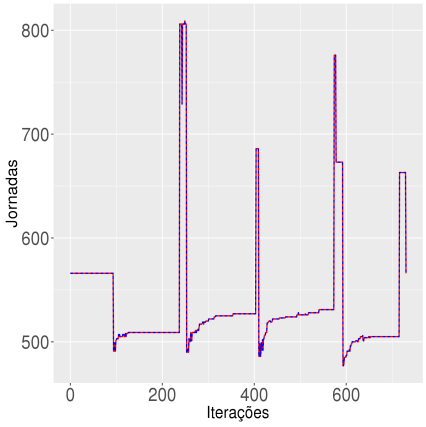
(c) Inst2313 - Função Objetivo



(d) Inst2313 - Jornadas

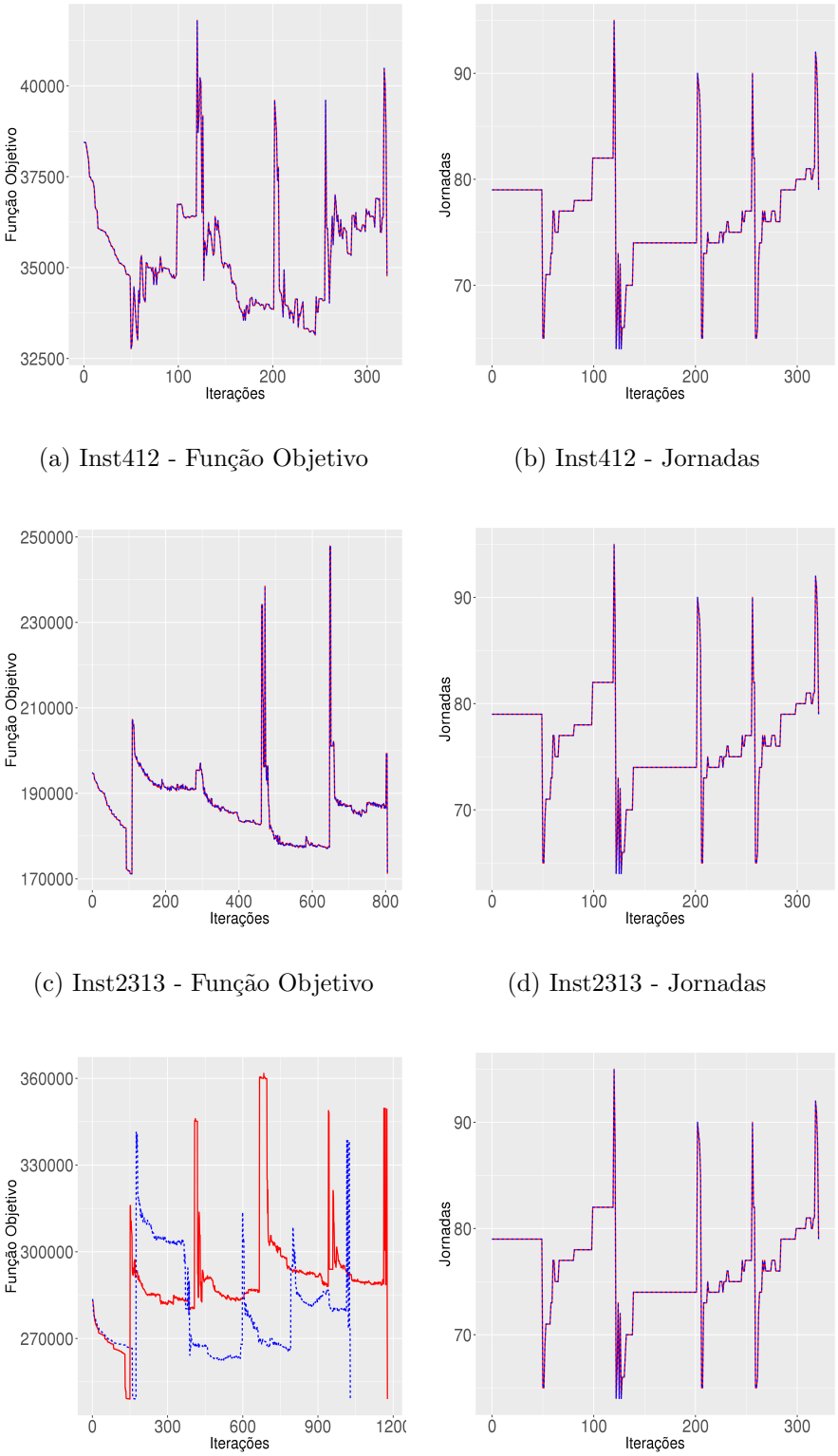


(e) Inst3478 - Função Objetivo



(f) Inst3478 - Jornadas

Figura 4.7: Evolução da função objetivo e número de jornadas ao longo das iterações da versão VNS_BLS_CC_CO. Linha vermelha - pior solução; Linha azul - melhor solução



(a) Inst412 - Função Objetivo

(b) Inst412 - Jornadas

(c) Inst2313 - Função Objetivo

(d) Inst2313 - Jornadas

(e) Inst3478 - Função Objetivo

(f) Inst3478 - Jornadas

Figura 4.8: Evolução da função objetivo e número de jornadas ao longo das iterações da versão VNS_BLS_CC_SO. Linha vermelha - pior solução; Linha azul - melhor solução

As figuras detalham que a quantidade de jornadas esta diretamente relacionada ao valor da função objetivo. Quanto menor a quantidade de jornadas menor tende a ser o valor da função. A quantidade de jornadas tende a se estabilizar antes do valor da função objetivo, embora a quantidade de jornadas se estabilize, o VNS_BLS faz diferentes combinações que tendem a diminuir o valor da função.

O valor da função objetivo da melhor solução, tente a ser mais estável ao longo das iterações, enquanto a pior solução é mais instável, variando de iteração para iteração, reforçando a tentativa do algoritmo em buscar regiões promissoras no espaço de busca. Nota-se que a melhor solução passa várias iterações com o valor da função objetivo igual, ou seja, passa muitas iterações sem ter melhora na solução. No entanto apesar de serem valores de função objetivo iguais, podem ser soluções diferentes (com combinações de tarefas diferentes), que em determinada iteração do algoritmo acabam encontrando um valor de função objetivo menor.

A melhor e pior solução das versões que possuem comunicação bidirecional com a memória compartilhada (VNS_BLS_CC_CO, VNS_BLS_CC_SO), tiveram o mesmo comportamento ao longo das iterações do algoritmo. O que reforça a ideia de que a comunicação bidirecional do algoritmo tende a forçar o VNS_BLS a intensificar a busca em torno de uma solução considerada boa, não permitindo que haja uma diversificação das soluções.

4.4 Buscas Locais mais eficientes para a melhora da solução

Como explicado anteriormente, o VNS_BLS utiliza 36 buscas locais diferentes que executam simultaneamente. Sendo assim pode haver buscas locais que não contribuem com a melhora da solução, resultando em consumo desnecessário de recursos. Esta seção relata os resultados obtidos em relação ao uso das buscas locais para a otimização da solução. Todas as versões do VNS_BLS são analisadas separadamente com o objetivo de verificar quais buscas locais são mais efetivas para cada versão. As figuras apresentam a média de uso das buscas locais das 5 execuções em escala logarítmica.

A Figura - 4.9 apresenta a relação de uso das buscas locais do VNS_BLS de todas as versões.

Das 36 buscas locais, nenhuma das versões utilizou todas para otimização da solução. Nota-se que 24 buscas são utilizadas em todas as versões. Portanto o VNS_BLS desperdiça tempo de processamento com buscas que não auxiliam na melhora da solução. As buscas

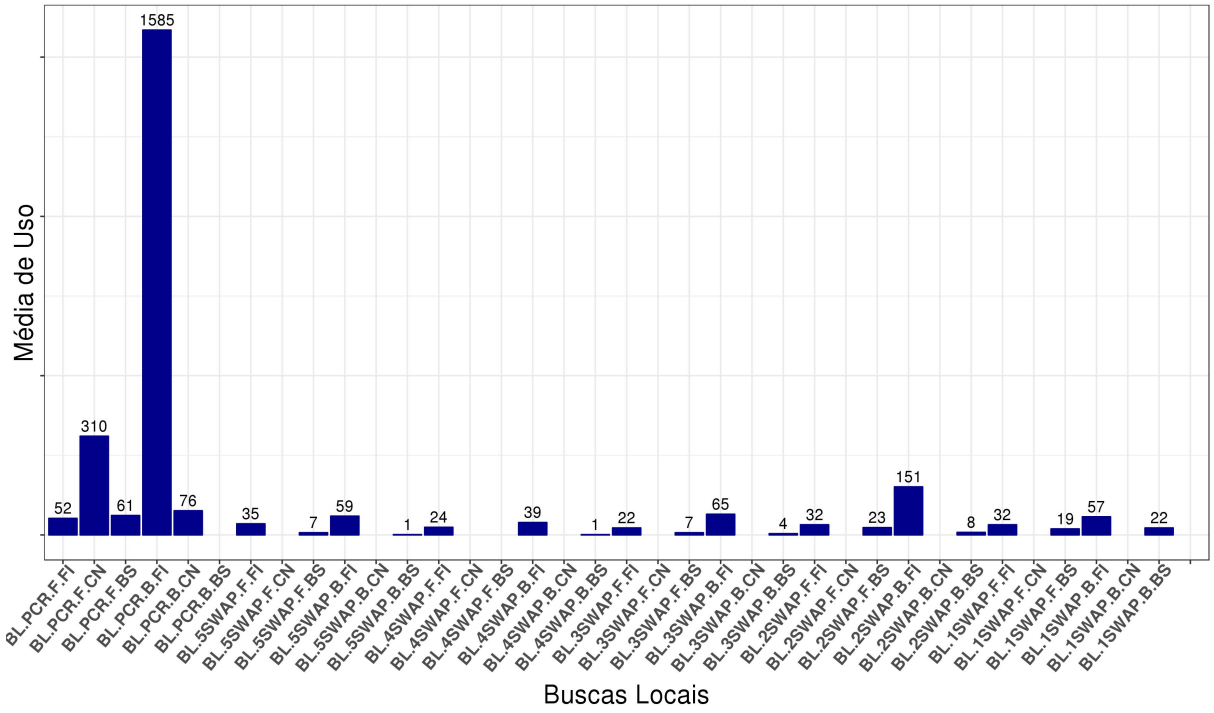


Figura 4.9: Média de utilização das buscas locais em todas as versões.

locais que utilizam os operadores mais simples, os quais fazem poucas modificações na solução, foram as que mais contribuíram para melhora da solução. As cinco buscas mais utilizadas em todas as versões são:

- BL.PCR.B.FI
- BL.PCR.F.CN
- BL.2SWAP.B.FI
- BL.PCR.B.CN
- BL.3SWAP.B.FI

Em contrapartida há doze buscas que não são utilizadas por nenhuma das versões caracterizando, assim, desperdício de recursos em comum em todas as versões, são elas:

- BL.PCR.B.BS
- BL.4SWAP.F.BS
- BL.1SWAP.F.CN

- BL.2SWAP.F.CN
- BL.3SWAP.F.CN
- BL.4SWAP.F.CN
- BL.5SWAP.F.CN
- BL.1SWAP.B.CN
- BL.2SWAP.B.CN
- BL.3SWAP.B.CN
- BL.4SWAP.B.CN
- BL.5SWAP.B.CN

A Figura - 4.10, Figura - 4.11, Figura - 4.12, Figura - 4.13 apresentam a taxa de utilização das buscas locais por versão do VNS_BLS.

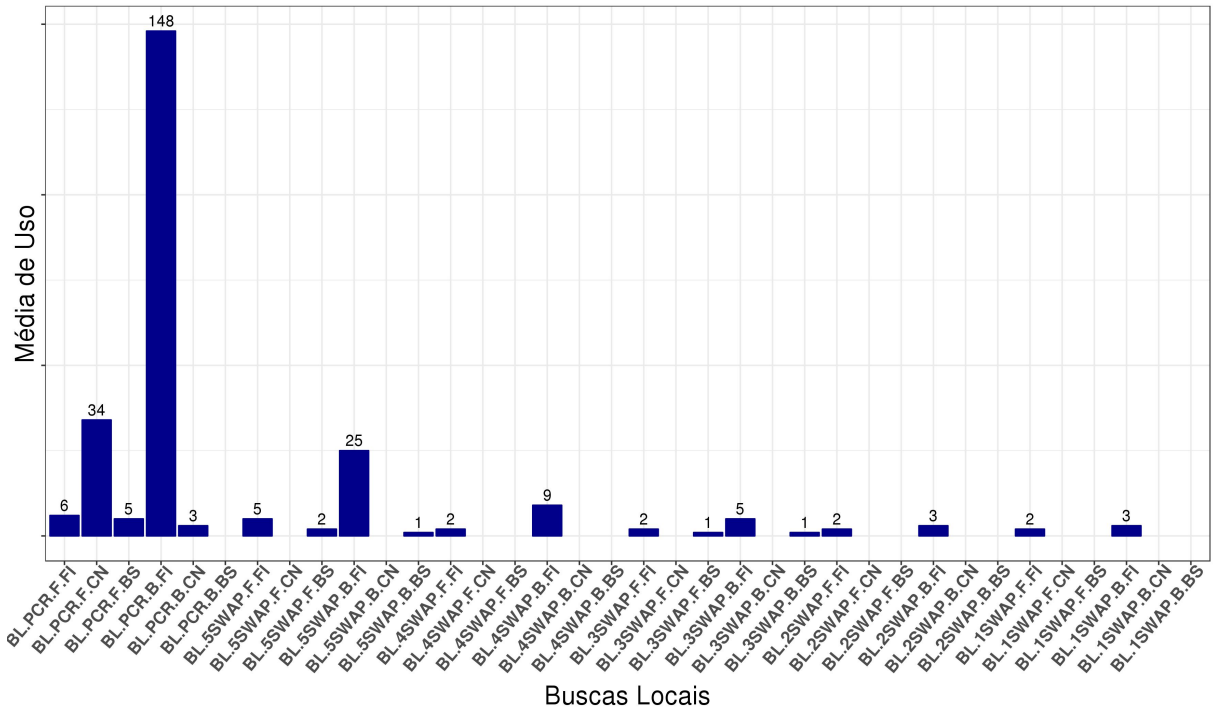


Figura 4.10: Média de utilização das buscas locais da versão VNS_BLS_SC_CO.

Há buscas locais que são utilizadas por todas as versões, como BL_PCR_B_FI. No entanto, algumas buscas se mostram úteis apenas para algumas versões como a BL.4SWAP.B.BS,

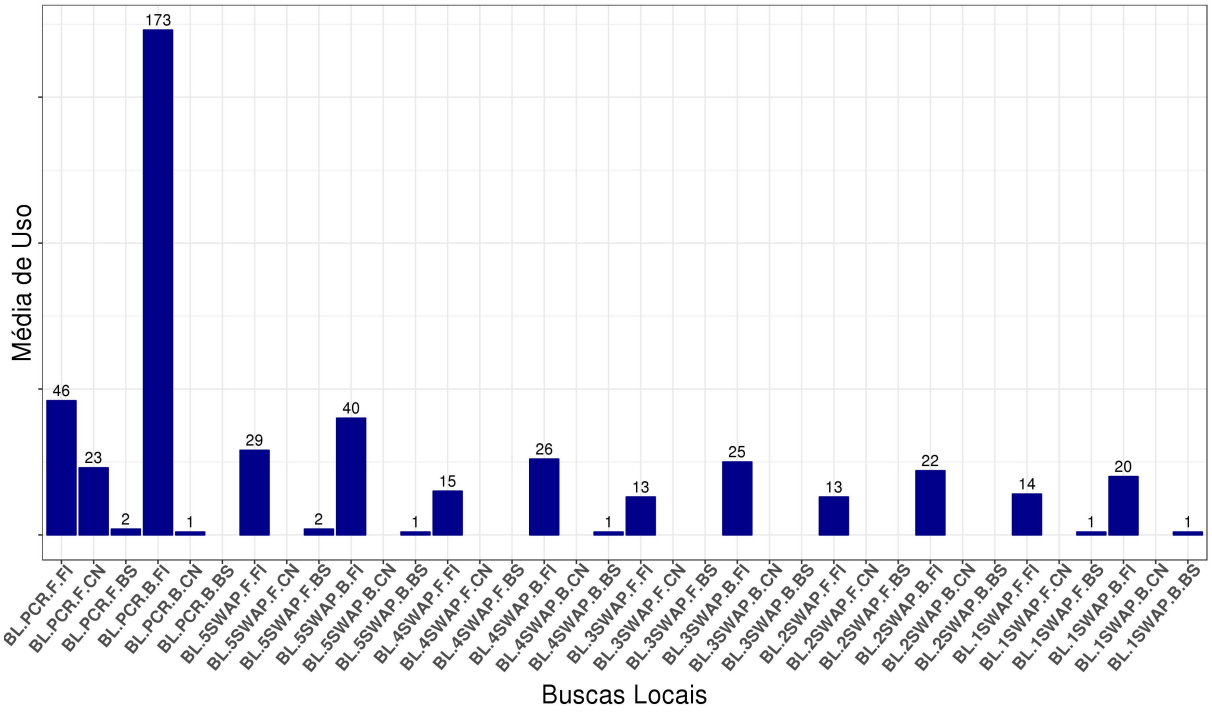


Figura 4.11: Média de utilização das buscas locais da versão VNS_BLS_CC_CO.

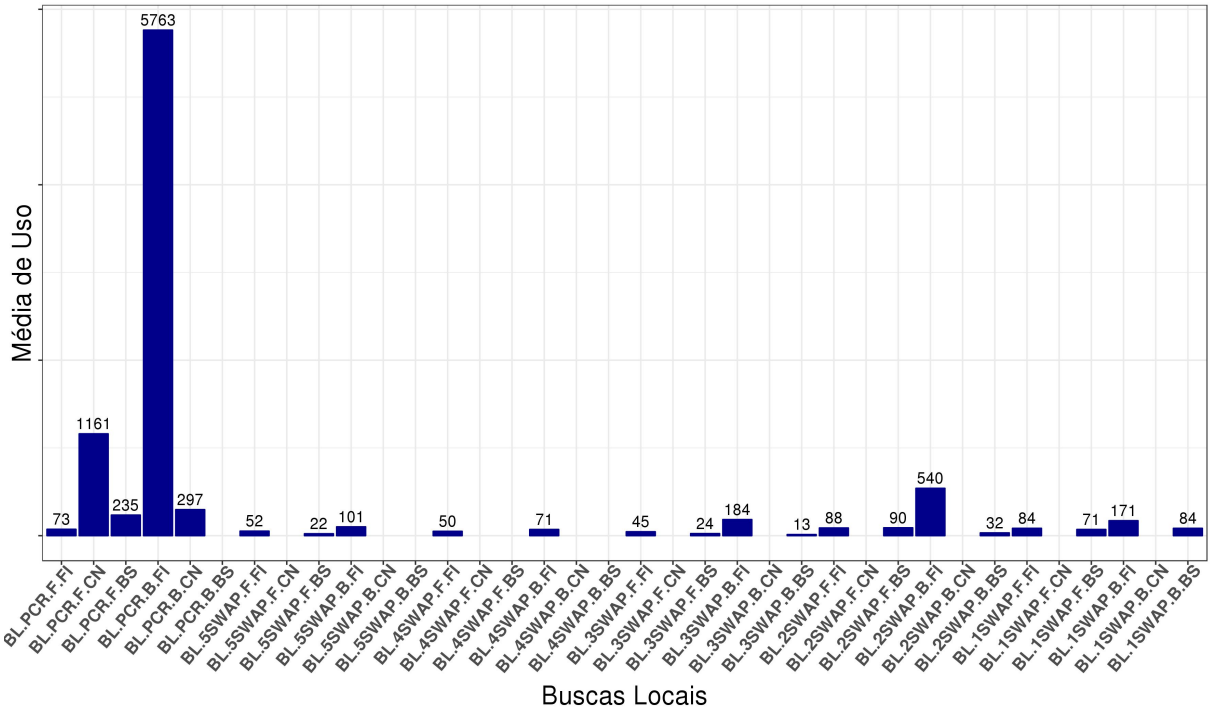


Figura 4.12: Média de utilização das buscas locais da versão VNS_BLS_SC_SO.

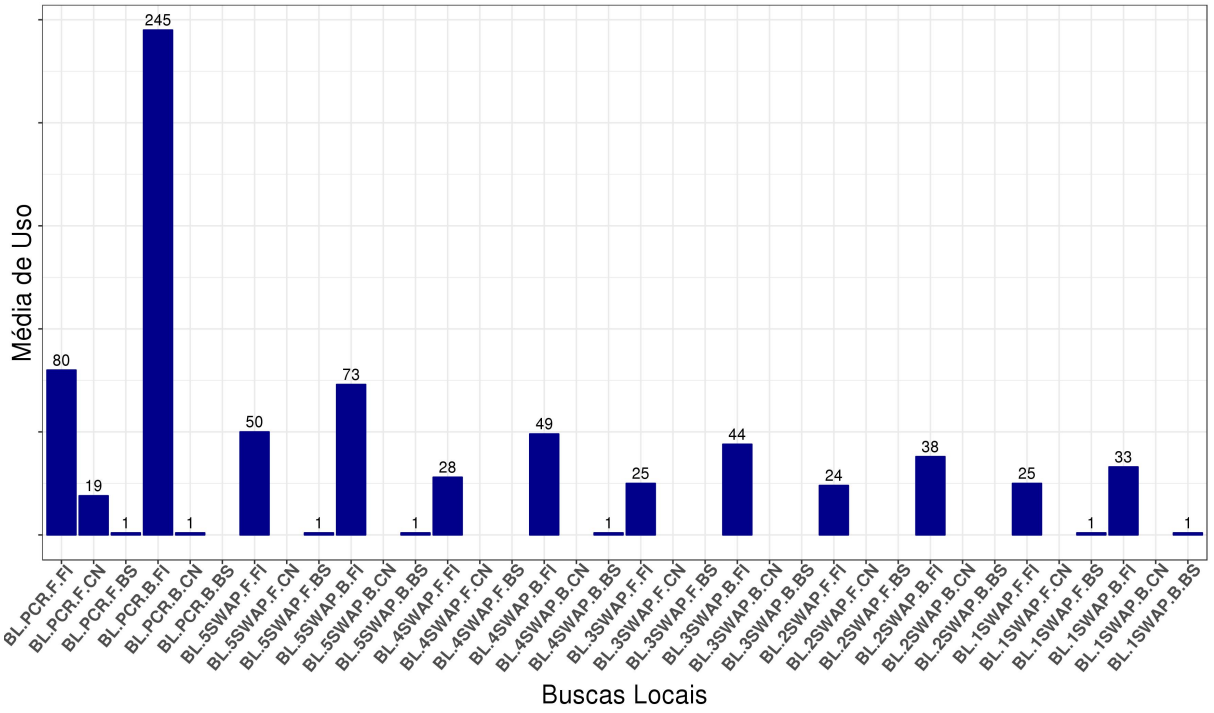


Figura 4.13: Média de utilização das buscas locais da versão VNS_BLS_CC_SO.

que teve obteve melhora de solução na versão VNS_BLS_CC_CO. Isso mostra que as versões acabam por explorar regiões diferentes do espaço de soluções. A buscas locais que tiveram baixa média de uso não podem ser desconsideradas, pois podem ser responsáveis por gerar soluções que podem ser melhoradas por outras buscas e atingir soluções finais boas.

Na versão VNS_BLS_SC_SO, como ilustra a Figura - 4.12, o uso das buscas é maior que das outras versões. Tendo como exemplo a BL_PCR_B_FI, que é utilizada em torno de 200 vezes em cada versão, na versão VNS_BLS_SC_SO ela é de 5763 utilizações. O aumento considerável da utilização se deve a característica da versão de permitir que cada busca local continue executando, pelo menos uma rodada de sua busca caso as outras ainda não tenham acabado. A característica de comunicação unidirecional com a memória compartilhada, colabora com este significativo aumento pois cada busca intensifica sua procura no ponto que está. A versão VNS_BLS_CC_SO, Figura - 4.13, não possui taxa de utilização alta das buscas, isso se dá pois apesar de possuir a característica de não ociosidade, ela permite a comunicação entre elas, assim há uma maior diversidade nas soluções, fazendo com que não haja tanta utilização de buscas locais em um mesmo ponto do espaço.

4.5 Comparação entre as versões do VNS_BLS

Devido as características de comunicação com a memória compartilhada e a possibilidade de execução enquanto outras buscas locais não finalizaram, foi possível implementar quatro versões do VNS_BLS. Estas versões foram discutidas na Seção 3.3. Dessa forma, esta seção tem como objetivo fazer um comparativo das versões para poder concluir qual possui melhor desempenho quanto a melhora da solução e consumo de processamento para obtenção da solução final. As figuras detalham a distribuição das soluções obtidas por cada versão. Os gráficos são plotados baseados nas soluções finais atingidas em cada uma das 5 execuções feitas de cada versão do VNS_BLS. Os resultados são mostrados no gráfico violino este gráfico é similar ao boxplot porém mostra também a distribuição de uma variável e transmite a densidade da distribuição (Hintze e Nelson, 1998).

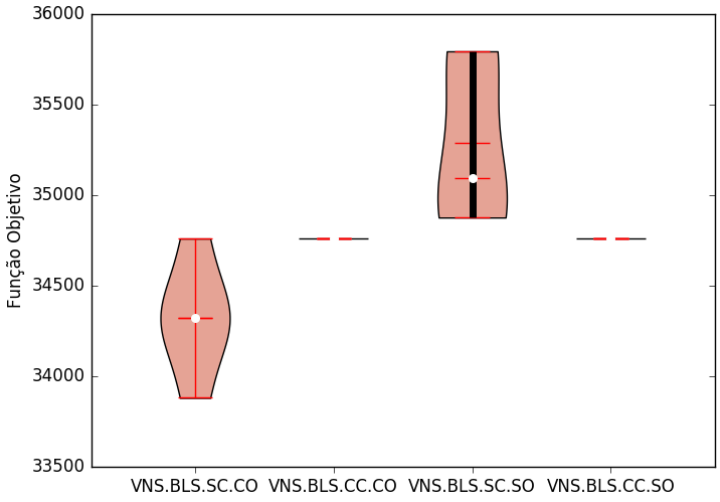
A Figura - 4.14 mostra a distribuição de valores das versões tanto de função objetivo como quantidade de jornadas para Inst412.

Todas as versões atingiram uma solução final com valor de função objetivo abaixo do 36.000 e quantidade de jornadas em torno de 80. A versão VNS_BLS_SC_SO, apesar de alcançar uma solução final em torno de 35.000 de função objetivo possui mais variações que as outras versões, variando de 35.800 até 35.000. A versão vns_BLS_SC_CO também obteve resultado variando de 34.800 a 33.700. Esta variação indica que as versões obtiveram mais soluções que vagaram pelo espaço de solução. Já para as outras variações, todas as execuções obtiveram a solução final com mesmo valor de função objetivo e quantidade de jornadas.

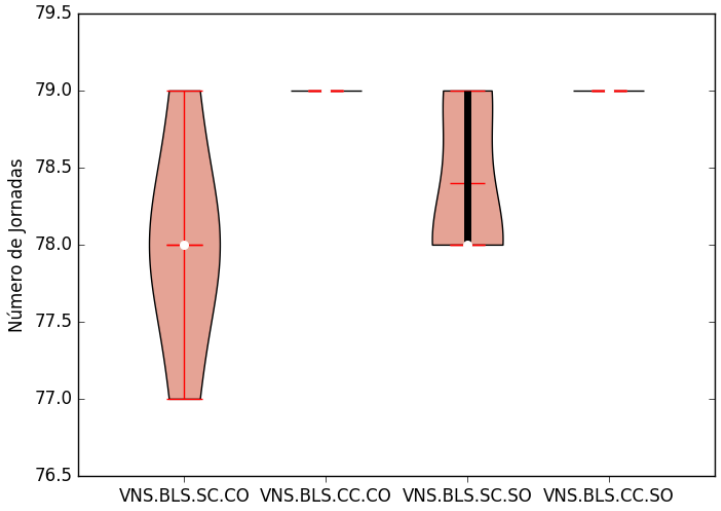
Quando consideramos a instância Inst2313, Figura - 4.15, a versão VNS_BLS_SC_CO é a que atinge menores valores de função objetivo. O VNS_BLS_SC_CO varia entre 160 até 170 mil de valor de função objetivo e 370 a 390 jornadas. Já a versão VNS_BLS_SC_SO foi a versão que obteve o desempenho menos promissor para esta instância. As versões VNS_BLS_CC_SO e VNS_BLS_CC_SO não obtiveram variação em suas soluções. Já a versão VNS_BLS_SC_SO varia seus resultados entre 180 a 175 mil de função objetivo, se mostrando inferior em relação a qualidade de solução das três versões.

Para a instância Inst3478, Figura - 4.16, a versão VNS_BLS_SC_SO apresentou os melhores resultados, apesar de mostrar grande variação nos valores das soluções ao longo das execuções.

Para todas as versões a quantidade de jornadas acompanha o valor de função objetivo, portanto quando menor o valor da função menor a quantidade de jornadas. As versões (Seção 3.3) podem ser classificadas com relação a quais obtiveram melhores valores de função objetivo da seguinte forma:



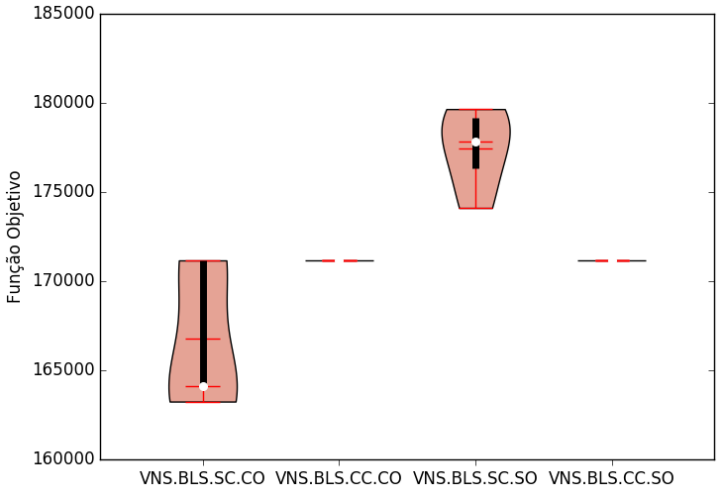
(a) Inst412 - Função Objetivo



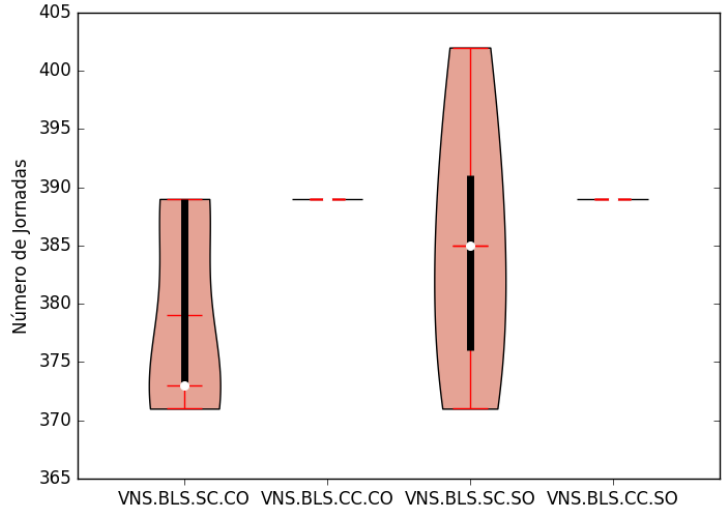
(b) Inst412 - Jornadas

Figura 4.14: Distribuição dos valores de função objetivo e quantidade de jornadas obtidos pelas soluções de todas as versões do VNS_BLS para a instância Inst412.

- Inst412 - VNS_BLS_SC_SO
- Inst2313 - VNS_BLS_SC_CO
- Inst3478 - VNS_BLS_SC_SO



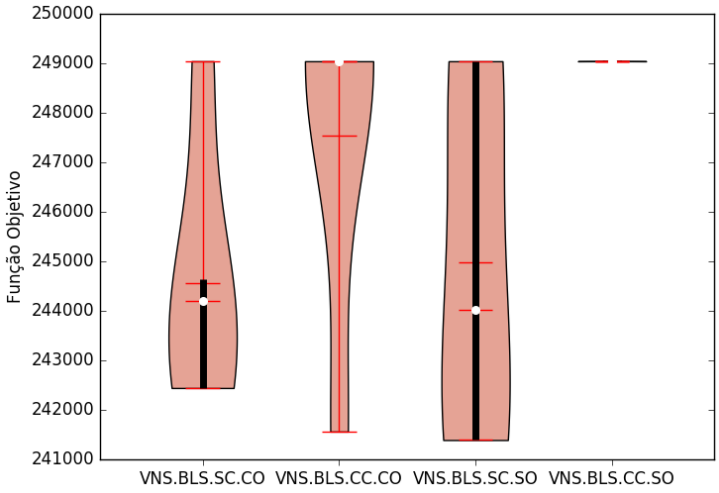
(a) Inst2313 - Função Objetivo



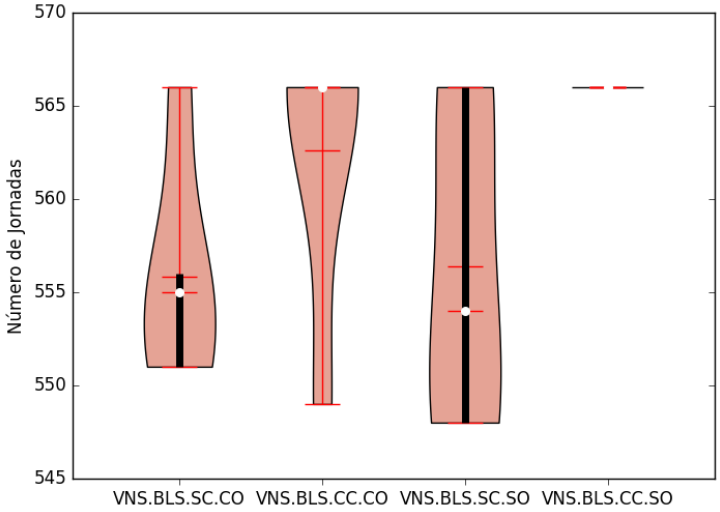
(b) Inst2313 - Jornadas

Figura 4.15: Distribuição dos valores de função objetivo e quantidade de jornadas obtidos pelas soluções de todas as versões do VNS_BLS para a instância Inst2323.

A versão VNS_BLS_SC_CO obteve as melhores soluções para a maioria das instâncias de teste. A seguir, será analisado os resultados obtidos pelas versões com relação a tempo de processamento gasto por cada uma. As figuras mostram a distribuição de tempo de processamento de cada versão do VNS_BLS para cada instância de teste. A Figura - 4.17 mostra a relação de tempo em segundos gasto pelas versões.



(a) Inst3478 - Função Objetivo



(b) Inst3478 - Jornadas

Figura 4.16: Distribuição dos valores de função objetivo e quantidade de jornadas obtidos pelas soluções de todas as versões do VNS_BLS para a instância Inst3478.

Nota-se que a versão VNS_BLS_SC_SO consome mais tempo de processamento, pois varia até 1.500 segundos para obter uma solução. As versões VNS_BLS_CC_CO VNS_BLS_CC_SO mantiveram um tempo de execução constante sem variações. Já a versão VNS_BLS_SC_SO variou entre 600 a 400 segundos.

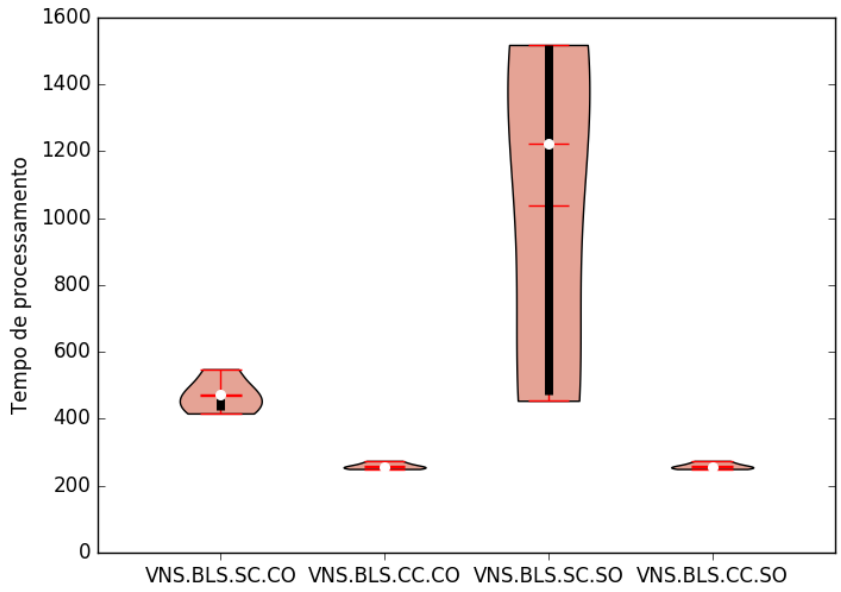


Figura 4.17: Distribuição do tempo de processamento gasto pelas versões do VNS_BLS para a instância Inst412.

A versão VNS_BLS_SC_CO, que obteve melhor desempenho quanto ao valor da função objetivo, Figura - 4.15, consome mais tempo de processamento, como mostra a Figura - 4.18. As versões VNS_BLS_CC_CO e VNS_BLS_CC_SO obtiveram o menor tempo de como observado na Figura - 4.15 mantiveram suas soluções constantes. Enquanto a versão VNS_BLS_SC_SO obteve variação no tempo de processamento assim como nos valores de função objetivo.

A versão VNS_BLS_SC_SO teve variação de tempo de processamento entre 200.000 a 380.000 segundos, como mostra a Figura - 4.19. Porém foi a versão que obteve melhores resultados de função objetivo. As versões VNS_BLS_SC_CO e VNS_BLS_CC_CO, variaram na mesma faixa de tempo, em torno de 250.000 a 130.000 segundos de processamento. A versão VNS_BLS_CC_SO, teve tempo de processamento inferior às outras versões, em torno de 30.000 segundos no entanto foi a versão que obteve os piores resultados com relação a função objetivo.

O tempo de processamento esta diretamente associado com a qualidade da solução, pois quanto maior o tempo gasto melhor as soluções obtidas. Classificando as versões com relação ao tempo de processamento, temos:

- Inst412 - VNS_BLS_CC_CO e VNS_BLS_CC_SO
- Inst2313 - VNS_BLS_CC_CO e VNS_BLS_CC_SO
- Inst3478 - VNS_BLS_CC_SO

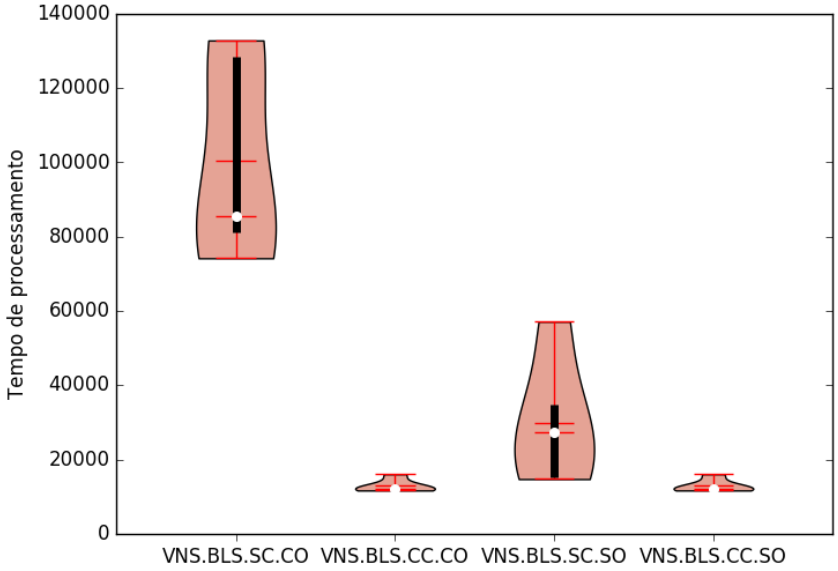


Figura 4.18: Distribuição do tempo de processamento gasto pelas versões do VNS_BLS para a instância Inst2313.

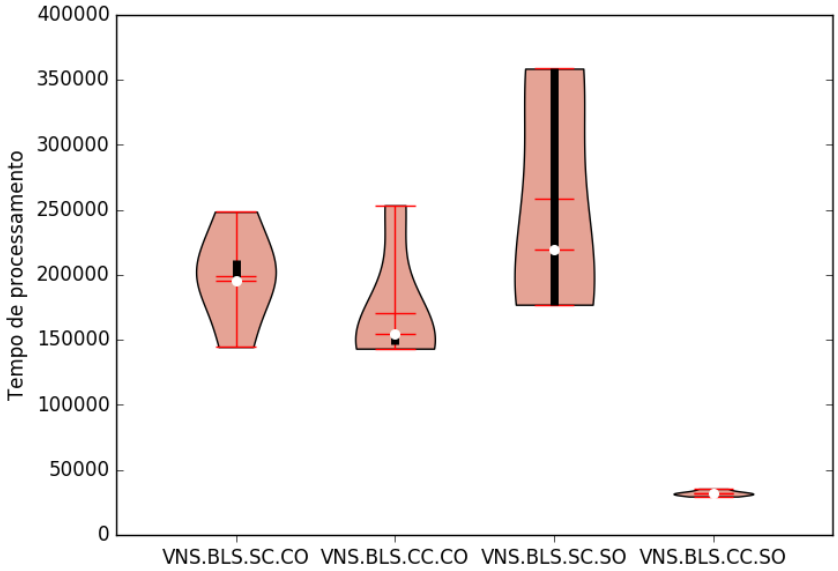


Figura 4.19: Distribuição do tempo de processamento gasto pelas versões do VNS_BLS para a instância Inst3478.

Nota-se que as versões que obtém os resultados de função objetivo mais altos são as que tiveram menor tempo de processamento.

4.5.1 Síntese dos resultados

A Tabela 4.5.1 apresenta os resultados de função objetivo, número de jornadas e tempo de execução obtidos pelo VNS_BLS. Para cada um é apresentado o melhor (min), pior (max) e média (med) dos valores alcançados.

		VNS_BLS_SC_CO			VNS_BLS_SC_SO			VNS_BLS_CC_SO			VNS_BLS_CC_CO		
Instância		C(x)	$\|J\|$	Tempo (seg)	C(x)	$\|J\|$	Tempo (seg)	C(x)	$\|J\|$	Tempo (seg)	C(x)	$\|J\|$	Tempo (seg)
Inst412	min	33.880	77	548	34.876	78	1.518	34.760	79	274	34.760	79	274
	med	38.453	79	469	35.287	78	1.037	34.760	79	259	34.760	79	259
	max	34.760	78	474	35.794	79	1.224	34.760	79	264	34.760	79	264
Inst2313	min	163.240	371	128.539	174.102	371	57.134	171.160	389	12.221	171.160	389	12.221
	med	166.760	379	100.488	177.417	385	29.844	171.160	389	12.957	171.160	389	12.957
	max	171.160	389	74.211	179.650	391	14.755	171.160	389	11.939	171.160	389	11.939
Inst3478	min	242.440	551	195.171	241.389	548	177.072	249.040	566	31.435	241.560	549	253.439
	med	244.552	555	199.057	244.974	556	258.123	249.041	566	32.082	247.544	562	170.746
	max	249.040	566	144.542	249.040	566	358.613	249.045	566	31.660	249.040	566	146.920

Tabela 4.3: Resultados obtidos pelas versões do VNS_BLS.

Os resultados indicam que existe uma relação entre a função objetivo e a quantidade de jornadas. Para as versões VNS_BLS_SC_CO e VNS_BLS_SC_SO a proporção de melhoria entre a melhor e pior solução tanto de valor da função objetivo como quantidade de jornadas, fica em torno de 3%. Já as versões VNS_BLS_CC_SO e VNS_BLS_CC_CO tem GAP pequeno entre a melhor e pior solução. Para as instâncias Inst412 e Inst2323 se mantém em zero o GAP. A instância Inst3478 a versão VNS_BLS_CC_SO tem GAP da melhor para a pior solução de 0.002% enquanto a VNS_BLS_CC_CO de 3%. Sendo assim conclui-se que as versões VNS_BLS_SC_CO e VNS_BLS_SC_SO tendem a otimizar mais a solução final.

Portanto conclui-se que um valor baixo de função objetivo está associado com menor número de jornadas, ou seja valor baixo de função objetivo tende a diminuir o número de jornadas. Considerando que escalas com menos jornadas, indicam menos motoristas designados e menos possíveis quebras de restrições, este é um resultado coerente.

Vale ressaltar que algumas versões obtém tempo de processamento da pior solução maior que tempo de processamento da melhor solução. A tendência em algoritmos heurísticos é que melhores soluções sejam encontradas em um tempo maior. A versão VNS_BLS_SC_SO por exemplo, para Inst3478 obteve GAP de 102% de tempo de processamento que a melhor solução. Já a versão VNS_BLS_CC_SO, para Inst3478, obteve um tempo em torno de 0.71% a mais. As outras versões, para todas as instâncias mantém o comportamento esperado, onde o tempo consumido para obter as melhores soluções é maior que os tempos consumidos para as soluções com função objetivo alto.

Contudo, na maioria dos casos as versões do VNS_BLS foram capazes de obter a melhor solução em tempo até 55% menor do que o obtido pela execução que encontrou a pior solução. Isto demonstra que a estratégia de executar buscas locais em simultâneo é promissora para reduzir o tempo de resposta e ainda fornecer boas soluções.

4.6 Comparação com outras abordagens

Esta seção apresenta uma comparação entre as duas versões que apresentaram melhor desempenho quanto ao valor de função objetivo da solução final do VNS_BLS e duas abordagens encontradas na literatura. A Tabela - 4.4 apresenta uma comparação do VNS_BLS_SC_CO, VNS_BLS_SC_SO e as abordagens de Dos Santos *et al.* (2016) e Sakayma *et al.* (2014), ambas sequenciais. A Tabela - 4.4 apresenta os valores de função objetivo, tempo de processamento, relatada em minutos, e as jornadas das soluções obtidas por VNS_BLS e as abordagens de Dos Santos *et al.* (2016) e Sakayma *et al.* (2014).

Tabela 4.4: Comparação função objetivo e número de jornadas.

	VNS_BLS_SC_CO			VNS_BLS_SC_SO			(Dos Santos *et al.*, 2016)			(Sakayma *et al.*, 2014)		
Instância	F(x)	Tempo	$\|J\|$	F(x)	Tempo	$\|J\|$	F(x)	Tempo	$\|J\|$	F(x)	Tempo	$\|J\|$
Inst3478	244.552	3.317	555	244.974	4.302	556	218.076	21.635	496	243.984	7.305	554
Inst2313	166.760	1.674	379	177.417	497	385	149.655	19.610	341	168.117	18.743	378
Inst412	38.453	7	79	35.287	17	78	31.788	369	79	35.502	8	80

Os valores de função objetivo e jornadas das soluções obtidas pelas versões VNS_BLS_SC_CO e VNS_BLS_SC_SO são maiores, que as soluções atingidas por Dos Santos *et al.* (2016). Em comparação com os resultados de Sakayma *et al.* (2014) as versões do VNS_BLS se mostraram bastante próximo.

A Tabela - 4.5 apresenta o GAP entre as versões VNS_BLS_SC_CO e VNS_BLS_SC_SO com as abordagens de Dos Santos *et al.* (2016) e Sakayma *et al.* (2014).

Tabela 4.5: GAP de função objetivo e tempo de processamento entre VNS_BLS e duas abordagens da literatura.

Instância	VNS_BLS_SC_CO				VNS_BLS_SC_SO			
	(Dos Santos *et al.*, 2016)		(Sakayma *et al.*, 2014)		(Dos Santos *et al.*, 2016)		(Sakayma *et al.*, 2014)	
	F(x)	Tempo	F(x)	Tempo	F(x)	Tempo	F(x)	Tempo
Inst3478	-10,8%	552%	-0,23%	120%	-10,9%	402%	-0,40%	69%
Inst2313	-10,2%	1071%	0,81%	1019%	-15,6%	3845%	-5,2%	3671%
Inst412	-17%	5171%	-0.0007%	14%%	-0.0009%	2070%	-0.00006%	52%

Em todas as comparações é possível observar que as versões do VNS_BLS se mostraram maior em relação ao valor de função objetivo. No entanto, o GAP em alguns casos, é

menor que 1%. Apenas a comparação da versão VNS_BLS_SC_CO com o trabalho de Dos Santos *et al.* (2016) mostrou GAP maior que 10%.

Em comparação com a abordagem de Sakayma *et al.* (2014) o algoritmo VNS_BLS_SC_SO se mostrou GAP pior quanto a tempo de processamento. Sendo assim, considerando o GAP das soluções e o tempo de processamento, mesmo que a abordagem do VNS_BLS perca em relação a valor de função objetivo, ele se mostra competitivo devido a apresentar melhor desempenho de processamento.

4.7 Retirando buscas locais sem uso

Dada a análise de quais buscas locais são mais utilizadas na melhora da solução, foi executado um experimento para todas as versões do VNS_BLS retirando as buscas que se mostraram pouco úteis na Seção 4.4.

A Figura - 4.20 mostra a média de utilização das buscas locais que de fato otimizaram a solução. Em VNS_BLS_SC_CO foi utilizado apenas 18 buscas locais. Com a redução, nota-se que todas as buscas locais foram utilizadas em algum momento para melhorar a solução.

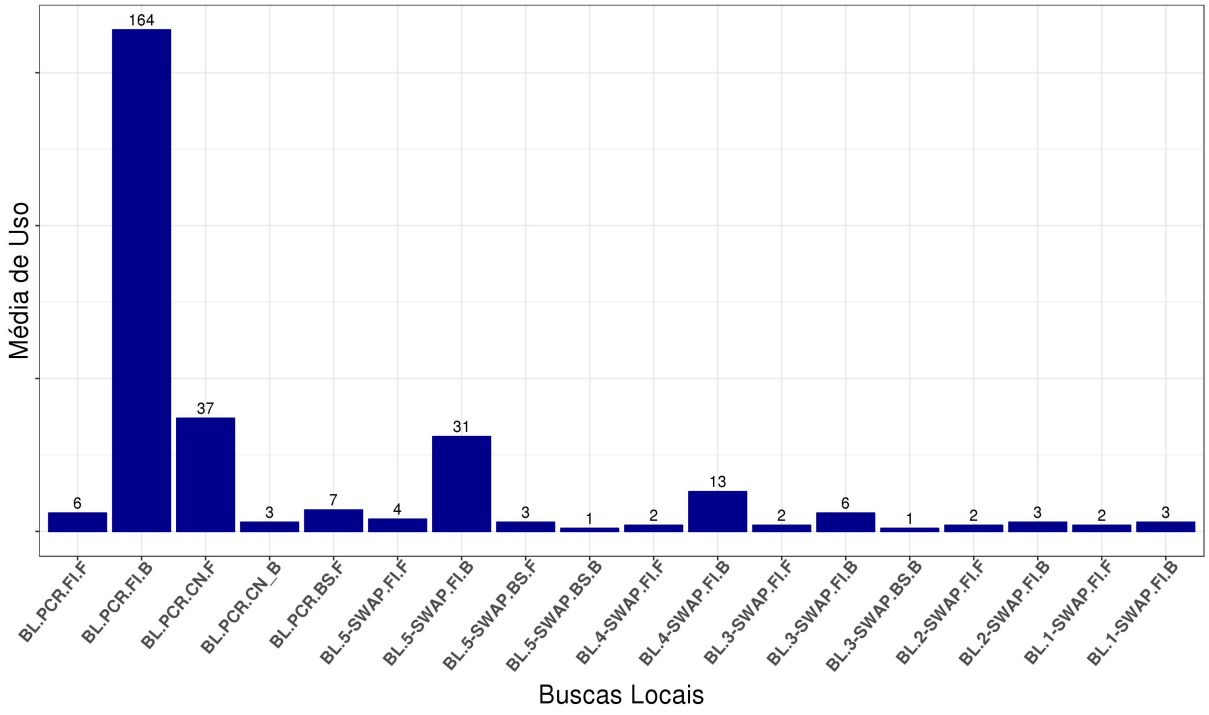


Figura 4.20: Taxa de utilização das buscas locais da versão VNS_BLS_SC_CO.

Na versão VNS_BLS_SC_SO, foram utilizadas 22 buscas locais, Figura - 4.21, destas observa-se que quatro não contribuíram para a melhora da solução. Como o algoritmo tem característica heurística, é possível que ele tenha explorado regiões do espaço onde estas buscas não se mostraram efetivas para melhora da solução. As versões VNS_BLS_CC_CO e VNS_BLS_CC_SO, Figura - 4.22 e Figura - 4.23 respectivamente, tiveram o mesmo comportamento.

A proporção de uso das buscas se mantém, porém os valores podem variar das versões com todas as buscas, esse comportamento se dá devido a característica heurística do algorítimo. Porém é importante notar que as buscas mais e menos usadas se mantem, diferenciando apenas nos valores.

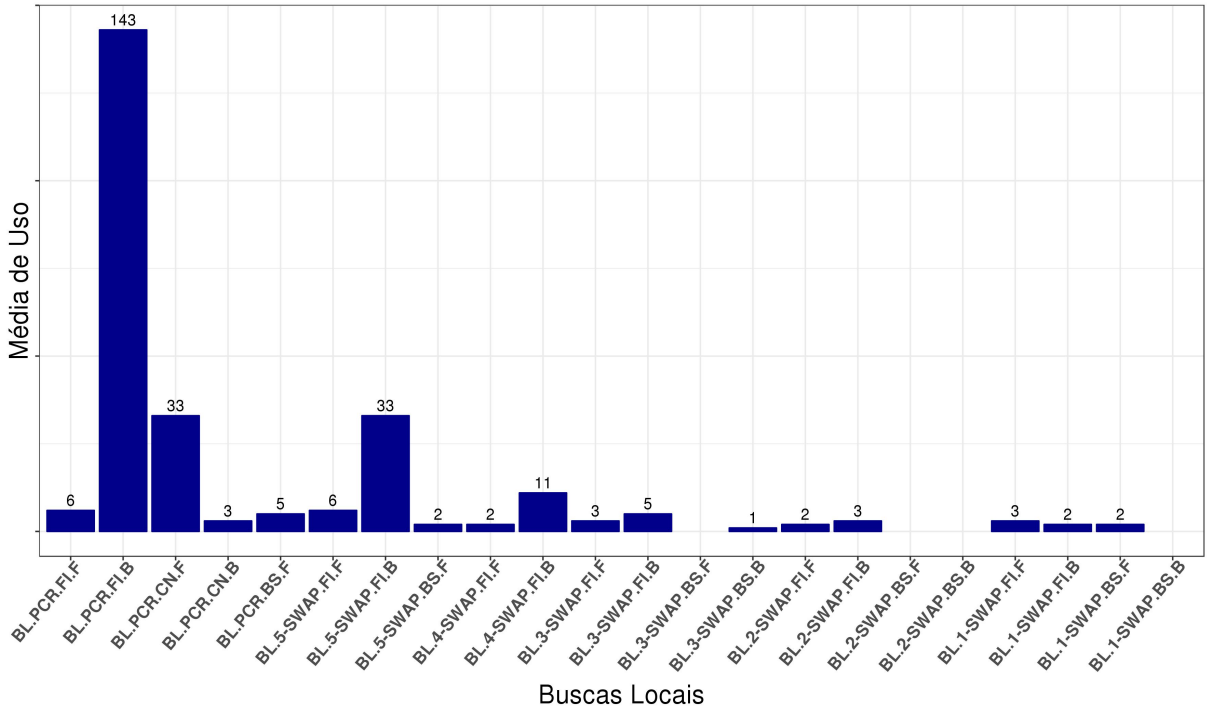


Figura 4.21: Taxa de utilização das buscas locais da versão VNS_BLS_SC_SO.

A versão VNS_BLS_SC_CO para Ints412, ficou com intervalo de solução entre 35.000 e 33.800, utilizando 18 buscas locais, Figura - 4.24(a). Comparando com os resultados da versão do algoritmo que executou as 36 buscas locais, Figura - 4.14, percebe-se que ambas ficaram com o mesmo intervalo de função objetivo para as soluções.

As versões VNS_BLS_CC_CO e VNS_BLS_CC_SO para Inst412, Figura - 4.24(a), verifica-se que a versão com as buscas locais retiradas, obteve resultados igual a versão que executa as 36 buscas locais. A Figura - 4.14 demonstra que VNS_BLS_CC_CO e VNS_BLS_CC_SO manteve os resultados em torno de 34.700 de função objetivo.

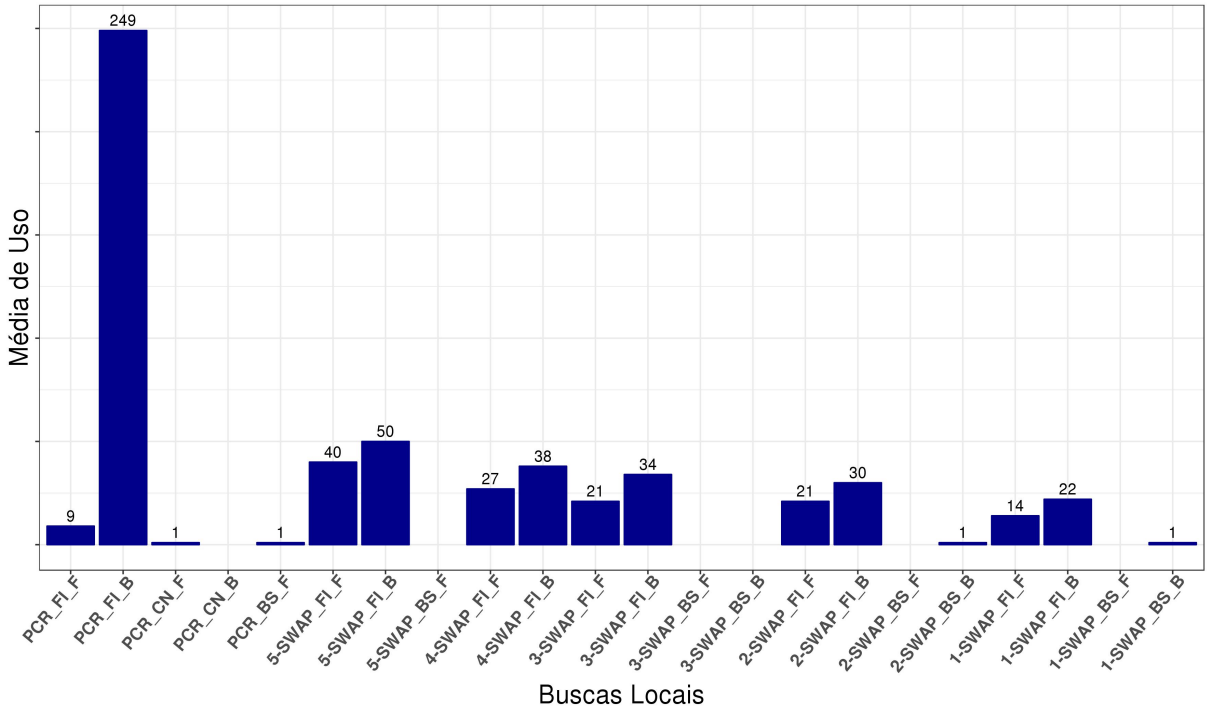


Figura 4.22: Taxa de utilização das buscas locais da versão VNS_BLS_CC_CO.

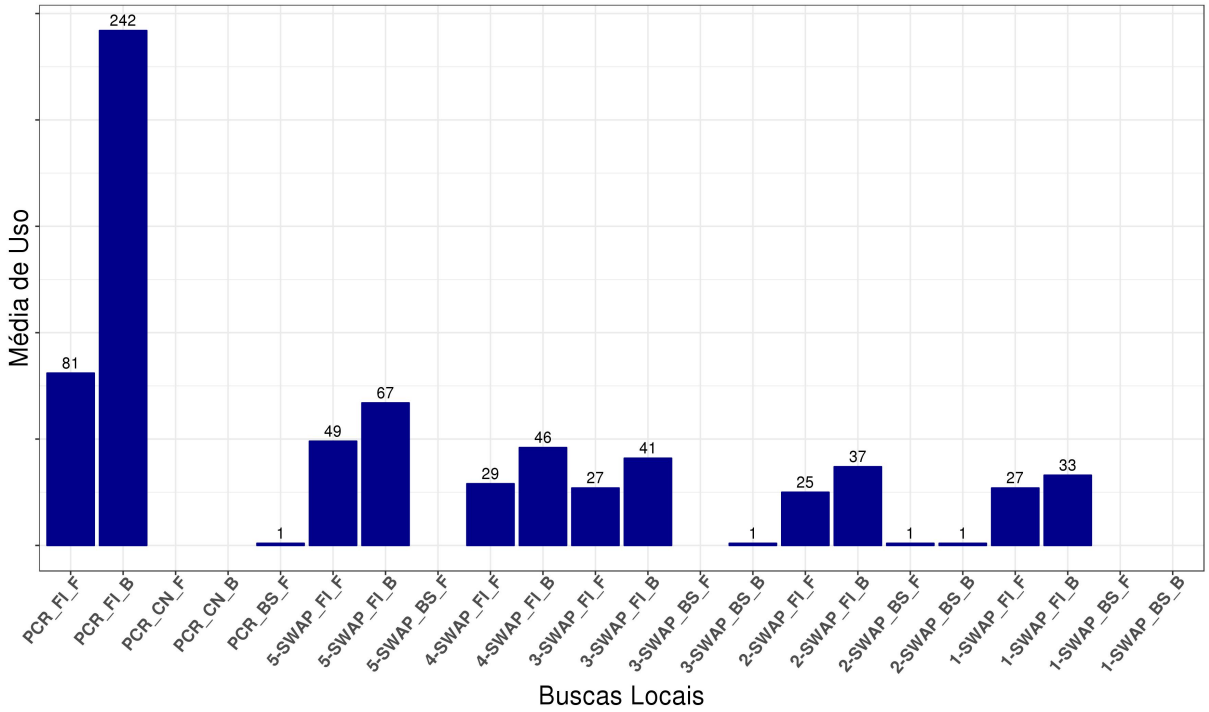
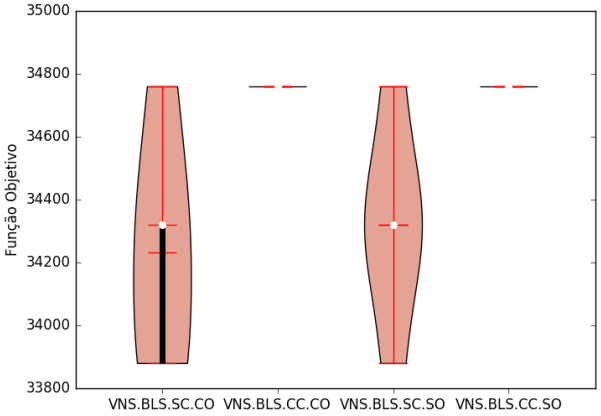
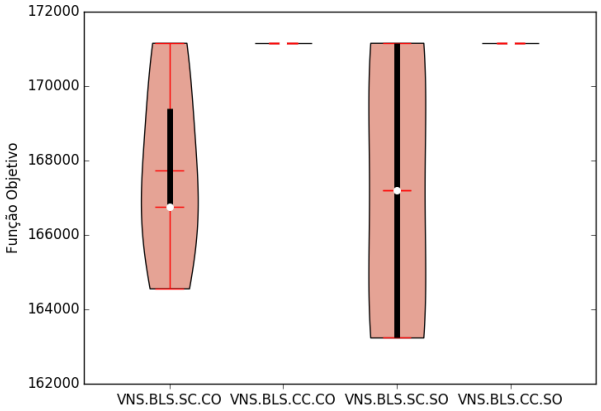


Figura 4.23: Taxa de utilização das buscas locais da versão VNS_BLS_CC_SO.

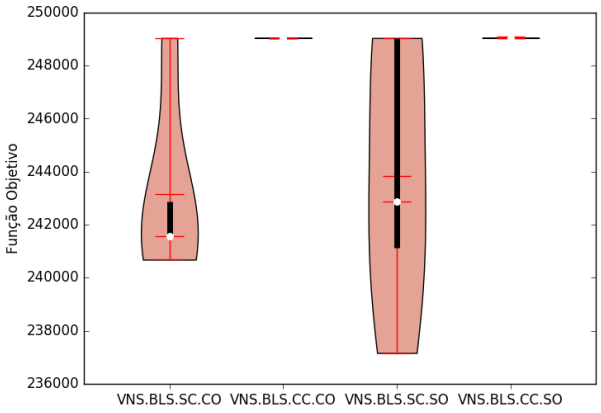
A distribuição das soluções obtidas para Inst2313 pelas versões do VNS_BLS é apresentada na Figura - 4.24(b). Em comparação com os resultados apresentados na Figura - 4.15, observa-se que a distribuição dos valores de função objetivo mantiveram-se iguais pra todas as versões.



(a) Inst412



(b) Inst2313

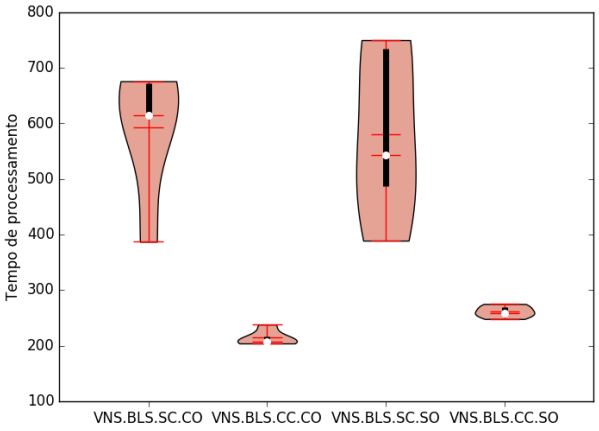


(c) Inst3478

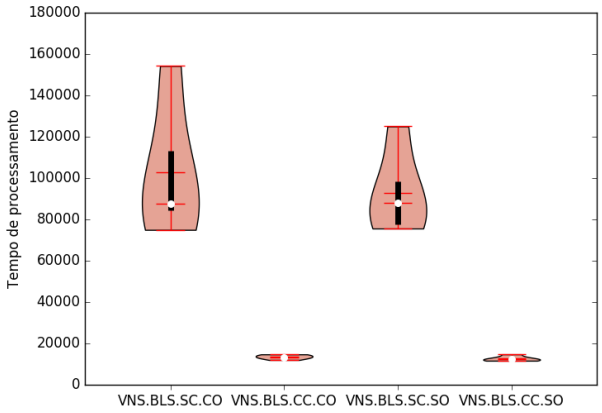
Figura 4.24: Distribuição de função objetivo.

Para Inst3478, a característica se manteve, ou seja, os resultados da versão com as buscas locais retiradas foi capaz de obter o mesmo intervalo de valor de função objetivo que a versão com 36 buscas. A Figura - 4.24(c) apresenta os resultados da versão com as buscas retiradas. Quando comparado com Figura - 4.16, observa-se as versões VNS_BLS_SC_CO, VNS_BLS_CC_SO e VNS_BLS_CC_SO tiverem o mesmo resultado da versão com todas as buscas. A versão VNS_BLS_CC_CO atinge função objetivo de 249.000 enquanto a solução da versão com as 36 buscas locais varia de 249.000 a 241.000.

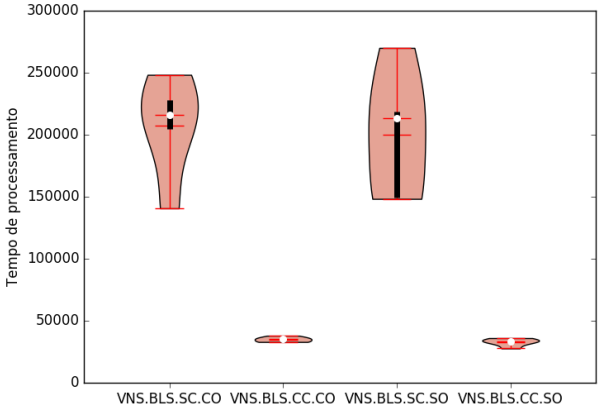
Em geral as versões com os buscas locais retiradas gastaram o mesmo tempo de processamento que as versões com todas as buscas locais, como pode ser visto Figura - 4.25 e Figura - 4.17, Figura - 4.18, Figura - 4.19. Com uma pequena variação para mais em alguns casos. De acordo com as análise é possível concluir que para todas as versões do VNS_BLS, retirar as buscas locais que não demonstram ser efetivas mantém a qualidade da solução e em algumas situações diminui o tempo de processamento.



(a) Inst412



(b) Inst2313



(c) Inst3478

Figura 4.25: Tempo de processamento gasto pelas versões do VNS_BLS com buscas locais desligadas.

5

Conclusão

A elaboração de escalas de motoristas é um problema recorrente a empresas de transporte que precisam montar escalas de acordo com restrições e custos operacionais. Quando consideramos especificamente empresas de transporte público, muitas vezes estas lidam com restrições como: horas trabalhadas, tempo de parada para descanso, horas extras entre outras restrições possíveis. Este tipo de problema é conhecido como Problema de Escalonamento de Motoristas e foi definido na literatura como um NP-Difícil.

Devido a estes fatores e as dimensões que as instâncias do problema podem atingir, é comum recorrer a utilização de meta-heurísticas para resolução do PEM. Há ainda a possibilidade de utilizar técnicas para execução simultânea de trechos do algoritmo para resolução do PEM, estas técnicas são pouco exploradas. Sendo assim, este trabalho investigou a proposta de uma meta-heurística baseada em VNS, denominada VNS_BLS, que utiliza buscas locais em simultâneo para resolver o PEM.

Os resultados computacionais mostraram que a abordagem é válida, pois o VNS_BLS otimizou em até 30% as soluções iniciais das instâncias de testes utilizadas. Quando comparado com duas abordagens apresentadas na literatura, VNS_BLS perde em relação ao valor de função objetivo. Porém, possui tempo de processamento inferior as outras abordagens. Esta característica estimula seu uso, considerando que o GAP de tempo de processamento é consideravelmente alto.

Como as buscas locais são executadas de forma independente, foi possível identificar quais contribuem para a melhora da solução. Notou-se que das 36 buscas locais, 24 de fato melhoram a solução. Isto indica que VNS_BLS utiliza processamento desnecessário em buscas que não melhoram a solução.

Devido a algumas características do VNS_BLS foi possível fazer quatro versões do algoritmo, são elas VNS_BLS_SC_CO, VNS_BLS_CC_CO, VNS_BLS_SC_SO e VNS_BLS_CC_SO. Estas versões foram comparadas a fim de verificar qual possui melhor desempenho. Concluiu-se que as versões que possuem comunicação com a memória compartilhada, tanto para armazenamento como para recuperação da solução, obtém resultados menos significativos que as versões que não possuem esta característica de comunicação. Portanto, as versões VNS_BLS_SC_CO e VNS_BLS_SC_SO foram as que se mostraram mais promissoras quanto a qualidade da solução.

Quando comparado com outras abordagens presentes na literatura, o VNS_BLS se mostrou competitivo, visto que em relação ao tempo de processamento ele alcançou resultado até 5000% menor. Quando comparado com qualidade da solução, o VNS_BLS alcançou resultados no máximo 17% pior. Analisando este ganho de processamento comparado com a qualidade, conclui-se que o VNS_BLS é uma abordagem competitiva. Como trabalhos futuros são enumerados:

- Aumento da quantidade de instâncias de teste. Executar experimentos com outras instâncias do problema, para que haja mais insumos para analisar o comportamento do algoritmo.

- Comparação com mais abordagens da literatura, abordagens que utilizem outras meta-heurísticas para resolução do PEM.

- Desenvolver novas estratégias que proporcionem uma melhora na qualidade da solução final, explorar outras formas de paralelizar a meta-heurística.

- Desenvolver uma estratégia que dinamicamente desative as buscas locais que não contribuem na melhora da solução. Implementação de uma versão que consiga analisar quais buscas não estão contribuindo com a melhora da solução e dinamicamente desligue a busca.

- Implementação de diferentes tipos de pertubação da solução, outras abordagens de *shake* para que seja possível explorar mais regiões do espaço de busca.

- Implementação de outras funções objetivos, que avaliem outros custos referentes ao PEM, por exemplo: gasto de combustível ou rotas da cidade.

- Ampliar o número de execuções de experimento com o propósito de melhorar a acurácia estatística dos resultados computacionais.

REFERÊNCIAS

ALBA, E. *Parallel metaheuristics: a new class of algorithms*, v. 47. John Wiley & Sons, 2005.

AZENCOTT, R. *Simulated annealing parallelization techniques*. John Wiley & Sons, 1992.

BADEAU, P.; GUERTIN, F.; GENDREAU, M.; POTVIN, J.-Y.; TAILLARD, E. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, v. 5, n. 2, p. 109 – 122, parallel Computing in Transport Research, 1997.

BLAIS, J.-Y.; LAMONT, J.; ROUSSEAU, M. The hastus vehicle and manpower scheduling system at the société de transport de la communauté urbaine de montréal. *Interfaces*, v. 20, n. 1, p. 26–42, 1990.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, v. 35, n. 3, p. 268–308, 2003.

CALVI, R. *Um algoritmo para o problema de escalonamento de tripulação em empresas de ônibus*. Dissertação de Mestrado, Universidade 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`

CNI-IBOPE Avaliação do transporte público piora de 2011 a 2014. `http://www.ibope.com.br/pt-br/noticias/Documents/RSB%2027%20Mobilidade%20Urbana%20Setembro%202015.pdf`, accessed: 2017-06-18, 2014.

Constantino, A. A.; Landa-Silva, D.; de Melo, E. L.; de Mendonça, C. F. X.; Rizzato, D. B.; Romão, W. A heuristic algorithm based on multi-assignment procedures for nurse scheduling. *Annals of Operations Research*, v. 218, n. 1, p. 165–183, 2014.

Constantino, A. A.; de Mendonca Neto, C. F. X.; de Araujo, S. A.; Landa-Silva, D.; Calvi, R.; dos Santos, A. F. Solving a large real-world bus driver scheduling problem with a multi-assignment based heuristic algorithm. *jucs*, v. 23, n. 5, p. 479–504, 2017.

Crainic, T. G.; Crisan, G. C.; Gendreau, M.; Lahrichi, N.; Rei, W. A concurrent evolutionary approach for rich combinatorial optimization. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, New York, NY, USA: ACM, 2009a, p. 2017–2022.

Crainic, T. G.; Crisan, G. C.; Gendreau, M.; Lahrichi, N.; Rei, W. Multi-thread integrative cooperative optimization for rich combinatorial problems. In: *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009b, p. 1–8.

Crainic, T. G.; Gendreau, M.; Hansen, P.; Mladenović, N. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, v. 10, n. 3, p. 293–314, 2004.

Crainic, T. G.; Toulouse, M. Parallel meta-heuristics. In: *Handbook of metaheuristics*, Springer, p. 497–541, 2010.

Davidovic, T.; Crainic, T. G. Parallel local search to schedule communicating tasks on identical processors. *Parallel Computing*, v. 48, p. 1 – 14, 2015.

Dias, T. G.; de Sousa, J. P.; Cunha, J.; *et al.* Genetic algorithms for the bus driver scheduling problem: a case study. *Journal of the Operational Research Society*, v. 53, n. 3, p. 324–335, 2002.

Djenic, A.; Radojicic, N.; Maric, M.; Mladenovic, M. Parallel vns for bus terminal location problem. *Applied Soft Computing*, v. 42, p. 448 – 458, 2016.

Dos Santos, Flausino, A.; Constantino, A.; Romão, W. Algoritmos baseados na meta-heurística VNS aplicados ao problema de escalonamento de motoristas de ônibus. *Simpósio Brasileiro de Pesquisa Operacional*, p. 1388–1399, 2016.

Dos Santos, A. F. *Algoritmos baseados na meta-heurística vns aplicados ao problema de escalonamento de motoristas de ônibus.* Dissertação de Mestrado, Universidade Estadual de Maringá, 2016.

Eskandarpour, M.; Zegordi, S. H.; Nikbakhsh, E. A parallel variable neighborhood search for the multi-objective sustainable post-sales network design problem. *International Journal of Production Economics*, v. 145, n. 1, p. 117 – 131, 2013.

Fores, S. *Column generation approaches to bus driver scheduling.* Tese de Doutoramento, University of Leeds, 1996.

Garca-Lopez, F.; Melian-Batista, B.; Moreno-Perez, J. A.; Moreno-Vega, J. M. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics*, v. 8, n. 3, p. 375–388, 2002.

Gonçalves, T. L. *Meta-heurísticas para o problema de programação de tripulações.* Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, 2010.

Hansen, P.; Mladenović, N.; Moreno Pérez, J. A. Variable neighbourhood search: methods and applications. *4OR*, v. 6, n. 4, p. 319–360, 2008.

Hillier, F. S.; Lieberman, G. J. *Introdução a pesquisa operacional.* McGraw Hill Brasil, 2013.

Hintze, J. L.; Nelson, R. D. Violin plots: A box plot-density trace synergism. *The American Statistician*, v. 52, n. 2, p. 181–184, 1998.

Knausz, M. *Parallel variable neighbourhood search for the car sequencing problem.* Fakultat fur Informatik der Technischen Universitat Wien., 2008.

Li, S. *Hyper-heuristic cooperation based approach for bus driver scheduling.* Tese de Doutoramento, Université de Technologie de Belfort-Montbeliard, 2013.

Marinho, E. H.; Ochi, L. S.; Drummond, L. M.; Souza, M. J. F.; Silva, G. P. Busca tabu aplicada ao problema de programação de tripulações de ônibus urbano. *Simpósio Brasileiro de Pesquisa Operacional, XXXVI*, p. 1471–1482, 2004.

de Melo, E. L.; Constantino, A. A.; Rizzato, D. B.; Romão, W. Um algoritmo heurístico de duas fases para escalonamento de enfermeiros com balanceamento de atendimento às preferências. *Simpósio Brasileiro de Pesquisa Operacional*, v. 30, 2010.

Mladenovic, N. A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. In: *papers presented at Optimization Days*, 1995.

Osman, I. H.; Laporte, G. Metaheuristics: A bibliography. *Annals of Operations Research*, v. 63, n. 5, p. 511–623, 1996.

Pentico, D. W. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, v. 176, n. 2, p. 774 – 793, 2007.

Portugal, R.; Lourenço, H. R.; Paixão, J. P. Driver scheduling problem modelling. *Public Transport*, v. 1, n. 2, p. 103–120, 2009.

Rizzato, D. B.; Melo, E. L.; Constantino, A. A. Automação e otimização do processo de escalonamento de enfermeiros. *XXX Encontro Nacional de Engenharia de Produção*, 2010.

Sakayma, R. Z. *Avaliação da meta-heurística vns para um problema de planejamento operacional de transporte público.* Dissertação de Mestrado, Universidade Estadual de Maringá, 2014.

Sakayma, R. Z.; Constantino, A. A.; Romão, W. Meta-heuristica vns aplicada a um problema de planejamento operacional de transporte publico. In: *XLVI Simpósio Brasileiro de Pesquisa Operacional*, 2014, p. 1632–1643.

Sanchez-Oro, J.; Sevaux, M.; Rossi, A.; Marti, R.; Duarte, A. Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. *Electronic Notes in Discrete Mathematics*, v. 47, p. 85 – 92, 2015.

Sevkli, M.; Aydin, M. E. Parallel variable neighbourhood search algorithms for job shop scheduling problems. *Journal of Management Mathematics*, v. 18, n. 2, p. 117–133, 2007.

Silva, G. P.; da Cunha, C. B. Uso da técnica de busca em vizinhança de grande porte para a programação da escala de motoristas de ônibus urbano. *Transportes*, v. 18, n. 2, 2010.

Silva, G. P.; Souza, M. J. F.; ALVES, J. Simulated annealing aplicadoa programação da tripulação no sistema de transporte público. *Encontro Nacional de Engenharia de Produção*, 2002.

Silva, G. P.; Souza, M. J. F.; Reis, J. v. A. d. Um método exato para otimizar a escala de motoristas e cobradores do sistema de transporte público. *Congresso de Pesquisa e Ensino em Transporte*, 2004.

Yunes, T. H.; Moura, A. V.; De Souza, C. C. Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, v. 39, n. 2, p. 273–288, 2005.