LUÍSA BRANDÃO CAVALCANTI

ROTEIRIZAÇÃO DE REBOCADORES PARA ATENDIMENTO DE PLATAFORMAS
OFFSHORE, COM OPERAÇÕES SINCRONIZADAS E JANELAS DE TEMPO

São Paulo
2021

LUÍSA BRANDÃO CAVALCANTI

ROTEIRIZAÇÃO DE REBOCADORES PARA ATENDIMENTO DE PLATAFORMAS
OFFSHORE, COM OPERAÇÕES SINCRONIZADAS E JANELAS DE TEMPO

Versão Corrigida

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção de
título de Doutora em Ciências.

São Paulo
2021

TERMO DE JULGAMENTO

Nome: Luísa Brandão Cavalcanti

Título: Roteirização de rebocadores para atendimento de plataformas *offshore*, com operações sincronizadas e janelas de tempo

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção de título de Doutora em Ciências.

Aprovada em 23 de fevereiro de 2021.

Banca Examinadora

Prof. Dr. André Bergsten Mendes

Instituição: EP-USP

Julgamento: Aprovada

Prof. Dr. Miguel Cezar Santoro

Instituição: EP-USP

Julgamento: Aprovada

Prof. Dr. Leonardo Junqueira

Instituição: EP-USP

Julgamento: Aprovada

Prof^a Dr^a: Laura Silvia Bahiense da Silva Leite

Instituição: UFRJ (Externo)

Julgamento: Aprovada

Prof. Dr. Paulo Ribas

Instituição: Externo

Julgamento: Aprovada

LUÍSA BRANDÃO CAVALCANTI

ROTEIRIZAÇÃO DE REBOCADORES PARA ATENDIMENTO DE PLATAFORMAS
OFFSHORE, COM OPERAÇÕES SINCRONIZADAS E JANELAS DE TEMPO

Versão Corrigida

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção de
título de Doutora em Ciências.

Área de Concentração: Engenharia Naval e
Oceânica.

Orientador: Prof. Dr. André Bergsten
Mendes

São Paulo
2021

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única da autora e com a anuência do seu orientador.

São Paulo, 23 de abril de 2021.

Assinatura da autora: Luísa Cavalcanti

Assinatura do orientador: Al. N. Zs

Catálogo-na-publicação

Cavalcanti, Luísa Brandão

ROTEIRIZAÇÃO DE REBOCADORES PARA ATENDIMENTO DE PLATAFORMAS *OFFSHORE*, COM OPERAÇÕES SINCRONIZADAS E JANELAS DE TEMPO / L. B. Cavalcanti – versão corr. -- São Paulo, 2021.
178 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo.
Departamento de Engenharia Naval e Oceânica.

1.Roteirização 2.Rebocadores 3.Programação Matemática 4.Heurística
5.Estruturas Offshore I.Universidade de São Paulo. Escola Politécnica.
Departamento de Engenharia Naval e Oceânica II.t.

Aos meus pais

AGRADECIMENTOS

Ao meu orientador André Bergsten, primeiramente por ter me transmitido serenidade nos momentos mais difíceis, mas também pelas ideias e contribuições que iluminaram o caminho percorrido para chegar a esta tese.

Aos professores da minha banca de qualificação, Marco Antônio Brinati e Miguel César Santoro, cujas críticas foram fundamentais para o encaminhamento deste trabalho.

Aos meus pais, Angela e Magnus, por serem essa grande fortaleza da minha vida, onde sempre encontro acolhimento e apoio, além de reconhecimento pelos desafios que decidi encarar.

Ao meu companheiro de vida Pedro, pelo apoio especialmente na reta final desta pesquisa.

À minha prima Bel e ao seu marido Xixo, pela honra de terem me tornado madrinha da Teresa, mesmo eu estando tão ausente, pela necessidade de me dedicar a esta pesquisa. Amo muito vocês três.

Às minhas amigas, que tanto me escutaram falar deste doutorado e que foram meu principal colo nos momentos difíceis.

*Si me caí, es porque estaba caminando. Y
caminar vale la pena, aunque te caigas.*
(Eduardo Galeano)

RESUMO

A exploração de petróleo em alto mar depende de recursos produtivos que são movimentados e alocados às plataformas por rebocadores especiais, denominados *Anchor Handling Tug Supply* (AHTS), muitas vezes exigindo a atuação simultânea de mais de uma embarcação. Se por um lado o afretamento AHTS é caro, por outro o não-atendimento de um serviço requerido pode ocasionar interrupção nas atividades de uma plataforma, gerando prejuízo econômico. Assim, torna-se necessária a gestão eficiente da frota de rebocadores, com planejamento das atividades previstas para curto a médio prazo. Neste trabalho, a metodologia da Pesquisa Operacional (PO) é empregada na proposição de modelo matemático de otimização, capaz de designar as embarcações que devem realizar cada serviço demandado e criar rotas, definindo não só a sequência de tarefas executadas por cada rebocador como a data de início das atividades, com o objetivo de minimizar o custo total do programa. O problema abordado se diferencia de problemas clássicos de roteirização, por haver múltiplos modos de atendimento de cada tarefa, restrições de sincronização de operações e janelas de tempo. Sua formulação generaliza problema sabidamente NP-difícil, portanto os métodos de solução propostos baseiam-se em meta-heurísticas conhecidas, cujos desempenhos são testados em um conjunto de instâncias aqui propostas. Dentre as contribuições desta tese à PO, destacam-se: apresentação de modelo matemático para um novo problema de roteirização, cuja estratégia de modelagem se provou mais eficiente que formulações tradicionais; e desenvolvimento de novo movimento de inserção de tarefa em posições intermediárias de um conjunto de rotas, essencial à aplicação de meta-heurísticas para solução de problemas de roteirização com operações sincronizadas. Apesar deste tipo de movimento ser usual em problemas de roteirização, esta é a primeira aplicação ao problema com operações sincronizadas, devido à alta complexidade gerada pela interdependência entre as rotas.

Palavras-chave: Roteirização. Sequenciamento de tarefas. Sincronização de recursos. Logística offshore. Petróleo.

ABSTRACT

Offshore oil extraction depends on productive resources that must be moved and allocated to the platforms by special tug vessels, known as Anchor Handling Tug Supply (AHTS), often requiring the simultaneous operation of more than one vessel. If, on the one hand, AHTS chartering is expensive, on the other hand, the non-attendance of a required service can cause interruption in the activities of a platform, generating prohibitive economic loss. Thus, it is necessary to efficiently manage the tugboat fleet, by scheduling activities planned for short to medium terms. In this work, the Operational Research (OR) methodology is used to propose a mathematical optimization model, capable of designating the vessels that must perform each service required and creating routes, by defining not only the sequence of tasks performed by each tug, but also the start date of each activity, in order to minimize the total cost of the program. The problem addressed differs from classic routing problems with time windows, in that there are multiple modes to perform each task and some of these tasks require multiple, synchronized resources. As the formulated model generalizes a problem known to be NP-hard, the solution methods proposed in this work are based on meta-heuristics and their performance is tested on a set of instances, which are also proposed here. Among the contributions of this thesis to the OR, the following stand out: presentation of a mathematical model for a new routing problem, whose modeling strategy proved to be more efficient than traditional formulations; and development of a new movement that inserts a given task at intermediate positions of a set of routes, which is essential to the application of meta-heuristics to solve synchronized routing problems. Although this type of movement is usual in routing problems, this is the first application to a problem with synchronized operations, due to the high complexity generated by the interdependence between the routes.

Keywords: Routing. Scheduling. Synchronization. Offshore Logistics. Oil and Gas.

LISTA DE FIGURAS

FIGURA 1 – ESQUEMA DE ROTAS PARCIAIS DE TRÊS EMBARCAÇÕES, ATENDENDO SETE TAREFAS.....	100
FIGURA 2 – ILUSTRAÇÃO DO IMPACTO CAUSADO EM TODAS AS ROTAS DEVIDO INSERÇÃO DE UMA TAREFA EM APENAS UMA DELAS, CONSIDERANDO O PROGRAMA ORIGINAL ILUSTRADO PELA FIGURA 1	100
FIGURA 3 – SEQUÊNCIA DOS PROCESSOS USADOS NA INSERÇÃO DA TAREFA, EXPLICITANDO OS MOMENTOS EM QUE AS ESTRUTURAS AUXILIARES SÃO CONSTRUÍDAS E USADAS (QUADROS DESTACADOS COM UMA ESTRELA).....	102
FIGURA 4 – EXEMPLO DE VISUALIZAÇÃO GRÁFICA DE UMA SOLUÇÃO PARCIAL	104
FIGURA 5 – REPRESENTAÇÃO DO CÁLCULO DE FOLGA DA ÚLTIMA TAREFA EXECUTADA NO PROGRAMA.	104
FIGURA 6 – REPRESENTAÇÃO DO CÁLCULO DE FOLGA DA PENÚLTIMA TAREFA DE UMA ROTA PARCIAL.....	105
FIGURA 7 – REPRESENTAÇÃO DA SOLUÇÃO PARCIAL APÓS INSERÇÃO DE UMA TAREFA NA SOLUÇÃO ANTERIOR.....	106
FIGURA 8 – GRÁFICO COM RESULTADOS DA ANÁLISE DE SENSIBILIDADE DO NÚMERO DE CLASSES DAS INSTÂNCIAS DE 10 TAREFAS	135
FIGURA 9 – GRÁFICO COM RESULTADOS DA ANÁLISE DE SENSIBILIDADE DO PERCENTUAL DE TAREFAS COM RESTRIÇÕES DE SINCRONIZAÇÃO NAS INSTÂNCIAS DE 10 TAREFAS.	136
FIGURA 10 – GRÁFICO COM RESULTADOS DA ANÁLISE DE SENSIBILIDADE DA DEMANDA MÉDIA DE REBOCADORES POR TAREFA, NAS INSTÂNCIAS DE 10 TAREFAS	138
FIGURA 11 – ILUSTRAÇÃO DE SOLUÇÃO OBTIDA PELA HEURÍSTICA CONSTRUTIVA PARA A INSTÂNCIA 6, NO FORMATO DE DIAGRAMAS DE GANTT.....	156

LISTA DE PSEUDOCÓDIGOS

PSEUDOCÓDIGO 1 – ALGORITMO DA HC IMPLEMENTADA.....	96
PSEUDOCÓDIGO 2 – CÁLCULO DAS FOLGAS (<i>SLACK TIMES</i>) DAS TAREFAS JÁ ROTEIRIZADAS.	106
PSEUDOCÓDIGO 3 – ALGORITMO DE CÁLCULO DAS POSIÇÕES EM QUE UMA TAREFA PODE SER INSERIDA, CONSIDERANDO A SOLUÇÃO PARCIAL DADA POR UM CONJUNTO DE ROTAS, E ARMAZENAMENTO NA MATRIZ $A_{jobm} \times n$	109
PSEUDOCÓDIGO 4 – ALGORITMO QUE LISTA SUCESSORES DIRETOS E INDIRETOS DE UMA TAREFA.....	111
PSEUDOCÓDIGO 5 – ALGORITMO DA FUNÇÃO QUE VERIFICA SE UMA TAREFA PODE SER INSERIDA SIMULTANEAMENTE EM DETERMINADAS POSIÇÕES DE UM CONJUNTO DE ROTAS.....	112
PSEUDOCÓDIGO 6 – ALGORITMO QUE INSERE TAREFA NAS ROTAS PARCIAIS DA SOLUÇÃO E CALCULA OS PARÂMETROS	113
PSEUDOCÓDIGO 7 - HEURÍSTICA CONSTRUTIVA COM MÚLTIPLOS RECOMEÇOS (HCMR).....	118
PSEUDOCÓDIGO 8 - ALGORITMO DA BUSCA LOCAL COM MÚLTIPLOS RECOMEÇOS (BLMR).....	119
PSEUDOCÓDIGO 9 – ALGORITMO DA GRASP	120
PSEUDOCÓDIGO 10 – ALGORITMO RVTMR IMPLEMENTADO	122
PSEUDOCÓDIGO 11 – ALGORITMO DESTRÓI E REPARA, USADO NO RVMTR	123
PSEUDOCÓDIGO 12–ALGORITMO DA HEURÍSTICA CONSTRUTIVA DESENVOLVIDA NAS ETAPAS INICIAIS DA PESQUISA, POSTERIORMENTE DESCARTADA	172
PSEUDOCÓDIGO 13 – ALGORITMO DA PRIMEIRA FASE DA HEURÍSTICA CONSTRUTIVA, ADITIVA	173
PSEUDOCÓDIGO 14 – ALGORITMO QUE ADICIONA UMA TAREFA NO FINAL DAS ROTAS PARCIAIS DE UMA SOLUÇÃO E CALCULA O <i>EARLIEST STARTING TIME (ES)</i> E A DATA DE CONCLUSÃO, $t_{Conclusao}$	174
PSEUDOCÓDIGO 15 –ALGORITMO DA SEGUNDA FASE DA HEURÍSTICA CONSTRUTIVA, QUE USA INSERÇÃO.....	175

PSEUDOCÓDIGO 16 – ALGORITMO DA FUNÇÃO QUE VERIFICA SE UMA TAREFA PODE SER INSERIDA SIMULTANEAMENTE EM DETERMINADAS POSIÇÕES DE UM CONJUNTO DE ROTAS	176
PSEUDOCÓDIGO 17–ALGORITMO DA HEURÍSTICA CONSTRUTIVA PROPOSTA APÓS A QUALIFICAÇÃO, APRESENTADA NO L SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL (CAVALCANTI; MENDES, 2018).....	178

LISTA DE TABELAS

TABELA 1 – PARÂMETROS-CHAVES DAS INSTÂNCIAS DA LITERATURA E DAS PROPOSTAS NESTE TRABALHO. 80	
TABELA 2 – REGRAS DE ORDENAÇÃO USADAS PELA HEURÍSTICA CONSTRUTIVA	115
TABELA 3 – PARÂMETROS FIXADOS PARA AS 36 INSTÂNCIAS.....	126
TABELA 4 – PARÂMETROS DESCRITIVOS DAS INSTÂNCIAS USADAS NOS EXPERIMENTOS COMPUTACIONAIS DESTE TRABALHO, DESTACANDO ALTERAÇÕES EM RELAÇÃO AO TRABALHO DE MENDES (2007): NÚMERO DE TAREFAS, PERCENTUAL DE TAREFAS SINCRONIZADAS, FROTA TOTAL E POR CLASSE, DURAÇÃO MÉDIA DAS TAREFAS, QUANTIDADE MÉDIA DE REBOCADORES POR TAREFA E OCUPAÇÃO	127
TABELA 5 – PARÂMETROS-CHAVES DAS INSTÂNCIAS DA LITERATURA E DAS PROPOSTAS NESTE TRABALHO.	128
TABELA 6 – PARÂMETROS DESCRITIVOS DAS INSTÂNCIAS USADAS NOS EXPERIMENTOS DE ANÁLISE DE SENSIBILIDADE DO NÚMERO DE CLASSES. OS VALORES ALTERADOS ESTÃO EM NEGRITO.	130
TABELA 7 – PARÂMETROS DESCRITIVOS DAS INSTÂNCIAS USADAS NOS EXPERIMENTOS DE ANÁLISE DE SENSIBILIDADE DO PERCENTUAL DE CLIENTES COM RESTRIÇÃO DE SINCRONIZAÇÃO (OU SEJA, COM QUANTIDADE DE REBOCADORES DEMANDADOS SUPERIOR A UM). VALORES EM NEGRITO SINALIZAM ALTERAÇÃO EM RELAÇÃO À INSTÂNCIA ORIGINAL.	131
TABELA 8 – PARÂMETROS DESCRITIVOS DAS INSTÂNCIAS USADAS NOS EXPERIMENTOS DE ANÁLISE DE SENSIBILIDADE DA QUANTIDADE MÉDIA DE REBOCADORES DEMANDADOS POR TAREFA. NOVAMENTE, VALORES EM NEGRITO SINALIZAM ALTERAÇÃO EM RELAÇÃO À INSTÂNCIA ORIGINAL.....	133
TABELA 9 – RESULTADOS DOS EXPERIMENTOS DE ANÁLISE DE SENSIBILIDADE DO NÚMERO DE CLASSES. OS VALORES ALTERADOS ESTÃO EM NEGRITO.	135
TABELA 10 – RESULTADOS DOS EXPERIMENTOS DE ANÁLISE DE SENSIBILIDADE DO PERCENTUAL DE CLIENTES COM RESTRIÇÃO DE SINCRONIZAÇÃO. VALORES EM NEGRITO SINALIZAM ALTERAÇÃO EM RELAÇÃO À INSTÂNCIA ORIGINAL E LINHAS EM TONALIDADE MAIS CLARA INDICAM INSTÂNCIA INVIÁVEL.	137
TABELA 11 – RESULTADOS DA ANÁLISE DE SENSIBILIDADE DA QUANTIDADE MÉDIA DE REBOCADORES DEMANDADOS POR TAREFA. VALORES EM NEGRITO SINALIZAM ALTERAÇÃO EM RELAÇÃO À INSTÂNCIA ORIGINAL E INSTÂNCIAS INVIÁVEIS APARECEM SINALIZADAS.....	139

TABELA 12 – CARACTERÍSTICAS DOS 3 MODELOS MATEMÁTICOS TESTADOS, PARA COMPARAÇÃO: QUANTIDADE DE VARIÁVEIS (TOTAL E INTEIRAS) E DE RESTRIÇÕES (TODAS AS RESTRIÇÕES SÃO LINEARES).	141
TABELA 13 – RESULTADOS DA AVALIAÇÃO DOS MODELOS MATEMÁTICOS: TEMPO DE EXECUÇÃO E GAP FINAL OBTIDOS PELO GUROBI PARA CADA MODELAGEM E INSTÂNCIA TESTADAS.....	142
TABELA 14 – RESUMO DOS RESULTADOS OBTIDOS COM O GUROBI, MOSTRANDO O TEMPO PARA OBTENÇÃO DE UMA SOLUÇÃO VIÁVEL (TERCEIRA COLUNA) E DA MELHOR SOLUÇÃO (ÚLTIMA COLUNA), LIMITADO A DUAS HORAS.	144
TABELA 15 – MELHORES UPPER E LOWER BOUNDS OBTIDOS PELO GUROBI PARA CADA INSTÂNCIA, DESTACANDO EM NEGRITO OS IDS DAS INSTÂNCIAS CUJA SOLUÇÃO ÓTIMA FOI ENCONTRADA.	146
TABELA 16 – PARÂMETROS DA HCMR, ADOTADO APÓS TESTES DE CALIBRAÇÃO.	147
TABELA 17 – RESULTADOS DA HEURÍSTICA CONSTRUTIVA E COMPARAÇÃO COM DESEMPENHO DO GUROBI, NO EXPERIMENTO USADO PARA ENCONTRAR APENAS UMA SOLUÇÃO FACTÍVEL.....	148
TABELA 18 – PARÂMETROS DA HCMR, ADOTADO APÓS TESTES DE CALIBRAÇÃO.	149
TABELA 19 – RESULTADOS DA HEURÍSTICA CONSTRUTIVA COM MÚLTIPLOS RECOMEÇOS (HCMR) NAS INSTÂNCIAS PROPOSTAS PARA O PRVJTSIN-HET.	150
TABELA 20 – PARÂMETROS DA BLMR, ADOTADO APÓS TESTES DE CALIBRAÇÃO.	151
TABELA 21 – RESULTADOS DA BUSCA LOCAL COM MÚLTIPLOS RECOMEÇOS (BLMR) NAS INSTÂNCIAS PROPOSTAS PARA O PRVJTSIN-HET.....	151
TABELA 22 – PARÂMETROS DE CALIBRAÇÃO DA GRASP.	152
TABELA 23 – RESULTADOS DA HEURÍSTICA GRASP NAS INSTÂNCIAS PROPOSTAS PARA O PRVJTSIN-HET.....	152
TABELA 24 – PARÂMETROS DA RVTMR, ADOTADO APÓS TESTES DE CALIBRAÇÃO.	153
TABELA 25 – RESULTADOS DA HEURÍSTICA DE REMOÇÃO DE VETOR DE TAREFAS COM MÚLTIPLOS RECOMEÇOS (RVTMR) NAS INSTÂNCIAS PROPOSTAS PARA O PRVJTSIN-HET.	154
TABELA 26 – MELHORES RESULTADOS OBTIDOS PARA 5 EXECUÇÕES DOS QUATRO MÉTODOS PROPOSTOS E COMPARAÇÃO COM O GUROBI	158

SUMÁRIO

1	INTRODUÇÃO	15
1.1	DO PROBLEMA ABORDADO.....	15
1.2	DO OBJETIVO	16
1.3	DO MÉTODO	16
1.4	ESTRUTURA DESTE TEXTO	20
2	REVISÃO DA LITERATURA	22
2.1	DA LOGÍSTICA <i>OFFSHORE</i>	22
2.2	DA PROGRAMAÇÃO DE TAREFAS (<i>SCHEDULING</i>)	24
2.2.1	Programação em máquinas paralelas com recursos de processamento	26
2.2.2	Programação de tarefas com alocação de equipes	29
2.2.3	Inserção de tarefa em programa pronto	33
2.3	DA ROTEIRIZAÇÃO COM SINCRONIZAÇÃO DE RECURSOS	41
2.3.1	Definições e tipos de sincronização.....	41
2.3.2	Problemas de roteirização com coleta e entrega.....	47
2.3.3	Problemas de roteirização de veículos com restrições de sincronização.....	50
2.4	DOS MODELOS MATEMÁTICOS PARA PROBLEMAS DE ROTEIRIZAÇÃO COM RESTRIÇÕES DE SINCRONIZAÇÃO OU DE PRECEDÊNCIA E JANELAS DE TEMPO.....	53
2.4.1	Formulações com nós fictícios para representar visitas sincronizadas a um mesmo local ou execução simultânea da mesma tarefa	53
2.4.2	Formulação sem nós adicionais para tarefas sincronizadas.....	58
2.4.3	Modelagem compacta para o problema de coleta e entrega	62
2.5	DOS MÉTODOS DE SOLUÇÃO	64
2.5.1	Busca em Vizinhança de Grande Porte (LNS) e variações aplicadas à roteirização de veículos	69
2.5.2	Busca Local Iterativa (ILS) aplicada à roteirização de veículos	74
2.5.3	<i>Greedy Randomized Adaptive Search Procedure</i> (GRASP) aplicada à roteirização de veículos	75
2.5.4	Métodos do tipo Destroi-e-Repara aplicados à roteirização de veículos.....	75
2.6	CONCLUSÕES DA REVISÃO DA LITERATURA	78
3	DESENVOLVIMENTO DO MODELO MATEMÁTICO DE OTIMIZAÇÃO	82
3.1	DEFINIÇÕES GERAIS	83
3.2	FORMULAÇÃO CLÁSSICA, COM INDEXAÇÃO POR REBOCADOR	85
3.2.1	Versão reduzida do problema, com pré-processamento dos dados da instância ..	89
3.3	FORMULAÇÃO COMPACTA, COM INDEXAÇÃO POR CLASSE.....	89
4	PROPOSIÇÃO DE HEURÍSTICA CONSTRUTIVA	93
4.1	NOMENCLATURA E DEFINIÇÕES	93
4.2	A HEURÍSTICA CONSTRUTIVA	95
4.2.1	Funcionamento geral da heurística	95
4.2.2	Complexidade do movimento de inserção de tarefa.....	99
4.2.3	Visão geral do uso de estruturas auxiliares para o movimento de inserção	101
4.2.4	Cálculo e uso de estruturas auxiliares: folgas das tarefas.....	103
4.2.5	Cálculo e uso de estruturas auxiliares: matriz $A(k)$, com posições viáveis de inserção de uma tarefa k	107
4.2.6	Cálculo e uso de estruturas auxiliares: sucessores das tarefas.....	110
4.2.7	Apresentação do algoritmo de inserção de tarefa em rotas paralelas	112

4.3	REGRAS PARA ORDENAÇÃO DE TAREFAS E ESCOLHA DO MODO.	114
5	PROPOSIÇÃO DE MÉTODOS HEURÍSTICOS PARA SOLUÇÃO DO PROBLEMA ABORDADO	118
5.1	MÚLTIPLOS RECOMEÇOS DA HEURÍSTICA CONSTRUTIVA	118
5.2	BUSCA LOCAL COM MÚLTIPLOS RECOMEÇOS	119
5.3	GRASP PARA O PRVJT-SIN-HET	120
5.4	REMOÇÃO DE VETOR DE TAREFAS COM MÚLTIPLOS RECOMEÇOS	121
6	EXPERIMENTOS COMPUTACIONAIS	125
6.1	PROPOSIÇÃO DE INSTÂNCIAS	125
6.1.1	Apresentação das instâncias do PRVJTSin-Het	126
6.1.2	Comparação com instâncias da literatura	127
6.1.3	Análise de sensibilidade dos parâmetros que diferenciam as instâncias propostas de outras instâncias da literatura.....	129
6.2	TESTES COM PACOTE OTIMIZADOR DO MERCADO	140
6.2.1	Avaliação dos modelos matemáticos propostos	140
6.2.2	Obtenção de limitantes inferiores e superiores para as instâncias, usando o otimizador Gurobi	143
6.2.3	Parâmetros para avaliação dos métodos de solução propostos.....	145
6.3	RESULTADOS DOS MÉTODOS DE SOLUÇÃO PROPOSTOS.....	147
6.3.1	Obtenção de solução viável usando heurística construtiva	147
6.3.2	Heurística construtiva com múltiplos recomeços.....	149
6.3.3	Busca Local com múltiplos recomeços	151
6.3.4	GRASP	152
6.3.5	RVTMR	153
6.4	ANÁLISE DOS RESULTADOS	155
7	CONCLUSÕES.....	160
	REFERÊNCIAS	164
	APÊNDICE A - Versões anteriores da Heurística Construtiva	171

1 INTRODUÇÃO

1.1 DO PROBLEMA ABORDADO

Na indústria da exploração de petróleo, alguns recursos produtivos são empregados tanto na manutenção de poços em operação como na perfuração de novas áreas. Para operações em alto mar (normalmente chamadas pelo termo equivalente em inglês, *offshore*), as empresas petrolíferas podem utilizar sondas semissubmersíveis que, por serem estruturas móveis, podem ser deslocadas de tempos em tempos para prospectar novas regiões (MENDES, 2007).

Para fazer o deslocamento destas unidades marítimas e a ancoragem no novo local de operação, são empregados rebocadores denominados *Anchor Handling Tug Supply* (AHTS), usualmente afretados pela empresa por períodos de médio-longo prazo (SHYSHOU; GRIBKOVSKAIA; BARCELÓ, 2010). Estas embarcações também são responsáveis pelo reboque e ancoragem das plataformas de produção, que sucede as etapas de perfuração e completação dos poços. Por conta disto, a frota de AHTS contratada deve ser suficiente para executar os diferentes tipos de tarefa em uma região geográfica delimitada, englobando diversos poços de exploração de petróleo; e, uma vez contratada, o planejamento das atividades dos rebocadores da frota (que normalmente é heterogênea) passa a ser crucial para garantir o equilíbrio econômico das operações em questão.

Na logística, o planejamento é comumente classificado em três níveis, a saber: estratégico, que abrange um horizonte de tempo de anos ou mesmo décadas, e engloba o dimensionamento de recursos e de infraestrutura necessários para atender uma demanda projetada, cujo nível de incerteza é alto; tático, que pode cobrir meses da operação prevista, a um grau de incerteza menor, e normalmente envolve a alocação dos recursos já disponíveis, mas pode também incluir a previsão de redução ou aumento dos recursos; operacional, que lida com uma demanda conhecida para os próximos dias ou poucas semanas, dependendo da dinâmica do negócio tratado, e programa o uso dos recursos disponíveis para atendê-la (BALLOU, 2009; CHOPRA; MEINDL, 2013). Neste sentido, quando o planejamento das operações realizadas pela frota AHTS é tratado em nível estratégico, a empresa de exploração de petróleo deve dimensionar a frota necessária para atender as operações previstas e definir a infraestrutura de apoio para estas embarcações, por exemplo. Já no nível tático, o planejamento de uso da frota já contratada pode levar em conta a previsão dos serviços

a realizar no próximo mês, por exemplo, adotando intervalos de dias ideais para sua realização. Ou seja, além de definir quais tarefas serão executadas por cada um dos rebocadores disponíveis, deve-se definir a ordem de atendimento das tarefas e as datas em que cada tarefa será executada, caso os roteiros criados sejam cumpridos. Desta forma, é possível prever os custos relacionados ao programa criado e, se necessário, priorizar o atendimento das tarefas mais urgentes.

Nesta pesquisa, é abordado o planejamento tático do uso de uma frota heterogênea AHTS já contratada, supondo que a quantidade de rebocadores é suficiente para atender a demanda. As especificidades do problema em questão, apresentadas nos capítulos de revisão da literatura e de modelagem matemática, tornam sua resolução tão complexa que a obtenção de um programa que minimize os custos envolvidos é inviável na prática, mesmo que a quantidade de atividades demandadas seja pequena.

1.2 DO OBJETIVO

Conforme explicado na seção anterior, esta tese aborda um problema real da logística *offshore*, de gestão da frota de rebocadores para realização de atividades como reboque e ancoragem de sondas de perfuração e unidades de produção de petróleo. Mais especificamente, o objetivo desta pesquisa é propor um algoritmo capaz de gerar o planejamento das atividades de uma frota AHTS de forma eficiente, o que inclui designar as embarcações responsáveis pela execução de cada tarefa demandada, determinar a sequência de execução das tarefas alocadas a cada uma das embarcações e calcular o instante de início de cada tarefa, minimizando os custos. Assume-se, portanto, que a frota foi previamente dimensionada e é suficiente para atender a todas as tarefas, e que a demanda de atividades é conhecida, assim como são conhecidos todos os custos envolvidos.

O objetivo desta pesquisa pode ser resumido ao atendimento da seguinte questão:

- Como alocar os rebocadores de maneira eficiente, atendendo todos os serviços demandados pelas plataformas *offshore* em certo horizonte de planejamento?

1.3 DO MÉTODO

Para responder à questão enunciada, propõe-se o uso da Pesquisa Operacional (PO), uma abordagem científica para auxílio na tomada de decisão em sistemas complexos (ALTAY; GREEN, 2006), que aplica métodos matemáticos e a alta capacidade de cálculo dos computadores modernos a problemas difíceis enfrentados por gestores

(BRADLEY; HAX; MAGNANTI, 1977). Mais especificamente, esta pesquisa enquadra-se no ramo de PO denominado programação matemática (do inglês *mathematical programming*), que busca a alocação ótima de recursos para atender atividades que competem entre si, sujeita a um conjunto de restrições impostas pela natureza do problema estudado. Assim, no âmbito da PO, a questão fundamental deste trabalho pode ser respondida por um modelo matemático de otimização, cuja solução seja um programa de atendimento de todas as atividades demandadas, que obedeça às restrições operacionais do problema real. Uma solução ótima de tal modelo minimiza os custos envolvidos.

Os métodos de solução para os modelos matemáticos citados constituem outra ramificação importante da PO, associada à programação matemática, na qual são desenvolvidos e testados algoritmos capazes de resolver modelos de otimização, chegando ou não em solução ótima. Neste contexto, um algoritmo pode ser definido como um conjunto estruturado de instruções genéricas, usualmente envolvendo operadores lógicos, capaz de gerar uma solução para o modelo. Caso o algoritmo comprovadamente chegue à solução ótima, diz-se que o método em questão é otimizante; caso contrário, o método é denominado heurístico e o próprio algoritmo também pode ser chamado de heurística de solução. Tendo em mente tais definições, os modelos matemáticos que não possuem algoritmo capaz de resolvê-los em tempo polinomial são denominados NP-difíceis (SCHOLL; KLEIN; JÜRGENS, 1997), para os quais métodos heurísticos são especialmente úteis.

Na literatura, há duas abordagens para avaliar o desempenho de métodos de solução heurísticos: avaliar o pior caso possível teórico; ou fazer estudos experimentais em séries de problemas teste, comparando os resultados médios em relação a *Lower Bounds* ou soluções ótimas, nos casos em que é possível encontrá-las (EDIS; OGUZ; OZKARAHAN, 2013). O **Lower Bound (LB)** representa um valor da função objetivo que limita inferiormente a solução ótima, em oposição ao **Upper Bound (UB)**, que é um limitante superior. Para um problema de minimização, como o tratado nesta pesquisa, o UB representa o valor da melhor solução que se conhece para o problema; enquanto o LB é um valor tal que se sabe que não existe solução capaz de gerar valor inferior a ele. Técnicas de obtenção de LB incluem, por exemplo, a resolução de uma versão relaxada do problema original, por exemplo, com a transformação de variáveis discretas em contínuas (ARENALES *et al.*, 2007; KHANAFER; CLAUTIAUX; TALBI, 2010).

Neste trabalho são apresentados dois modelos matemáticos de otimização que representam o problema de gestão da frota AHTS, de tal forma que a solução de tais modelos possa ser aplicada ao caso real. No caso estudado, uma solução de qualquer um dos modelos é um programa de atendimento das tarefas demandadas, que designa um rebocador ou conjunto de rebocadores que atenderá cada tarefa e especifica o horário (ou data) de início de cada uma delas. Isto implica traçar uma ordem de atendimento das tarefas para cada embarcação e, já que as tarefas estão associadas a localizações geográficas, dadas pelo posicionamento das plataformas, a sequência das tarefas compõe a rota que será percorrida pela embarcação. Assim, para cada rebocador haverá um conjunto de tarefas que serão atendidas segundo um roteiro contido na solução, levando em conta o tempo necessário para deslocamento da embarcação e outras restrições operacionais enumeradas no capítulo 3.

Entre os modelos apresentados nesta tese, um é baseado nos modelos já apresentados por Mendes (2007), e outro foi desenvolvido ao longo desta pesquisa, baseando-se em uma nova estratégia de modelagem publicada para problemas de Coleta e Entrega, definido no capítulo de revisão da literatura. Portanto, a contribuição desta pesquisa para a área de PO não se restringe ao método de solução desenvolvido, mas inclui a apresentação de uma nova formulação para o problema de roteirização estudado. Como tal problema é NP-difícil (MENDES, 2007) e as instâncias propostas, também baseadas em trabalho anterior de Mendes (2007), são complexas, propõe-se o uso de heurísticas na sua solução. Uma vez que as instâncias foram modificadas para aumentar a semelhança com o problema real, não há resultados prévios com os quais comparar o desempenho dos métodos propostos. Para contornar isto, o software otimizador Gurobi é empregado na obtenção de LBs e UBs para as instâncias propostas.

Todos os métodos desenvolvidos utilizam um movimento para a inserção de tarefa nas rotas dos rebocadores, cuja inovação está associada a uma característica importante do problema, que é o fato da maioria das tarefas demandarem mais de um rebocador para sua execução, por motivos técnicos ou de segurança. Tal movimento é o primeiro a ser apresentado para este subtipo de problema de roteirização, que possui restrições de sincronia de operações. A necessidade da atuação simultânea de dois recursos gera interdependência entre as rotas da solução, tornando qualquer alteração nestas rotas muito mais complexa do que nos demais problemas de roteirização. Por conta desta característica, métodos de solução já utilizados em outros problemas de roteirização

não são aplicáveis ao problema desta pesquisa, pois gerariam uma quantidade proibitiva de cálculos a cada iteração (LIU; TAO; XIE, 2019).

Talvez por conta desta lacuna de métodos de solução adequados, os problemas de roteirização com sincronização de recursos vêm ganhando atenção nos últimos anos (AIT HADDADENE; LABADIE; PRODHON, 2016). Pouco após o início da pesquisa que resultou nesta tese, Ait Haddadene, Labadie e Prodhon (2016) propuseram uma heurística híbrida para resolver um problema que também possui interdependência entre rotas, porém em menor escala (há poucas tarefas que exigem mais de um recurso e no máximo são requeridos dois recursos simultâneos). Para modificar uma solução viável de maneira eficiente, os autores fazem testes de viabilidade antes de aplicar movimentos comumente adotados em heurísticas voltadas a problemas de roteirização. A abordagem aplicada no presente trabalho é similar, mas o movimento desenvolvido faz a inserção simultânea da tarefa em mais de uma rota, diferentemente dos movimentos usados naquele trabalho. Além disto, os problemas tratados nas duas pesquisas são diferentes.

Mais recentemente, Liu Tao e Xie (2019) apresentaram um método de solução para um problema equivalente ao tratado por Ait Haddadene, Labadie e Prodhon (2016). A heurística apresentada obtém um desempenho impressionante nas instâncias daquele problema, que possui janelas de tempo e restrição de sincronização, mas tem frota homogênea e não possui restrições de incompatibilidade. Além disto, nas instâncias discutidas na literatura, são exigidos até dois recursos por tarefa e o percentual de tarefas com restrição de sincronização é muito inferior ao do problema que motiva a presente pesquisa. Esta diferença explica por que os algoritmos propostos por outros autores podem negligenciar a importância de fazer a inserção em um conjunto de rotas, uma vez que o grau de interdependência entre elas é muito inferior ao das instâncias desta tese.

De acordo com a revisão da literatura realizada, desde sua introdução por Mendes (2007) não houve outra publicação tratando o problema aqui estudado. Os métodos propostos pelo autor usam a estrutura do modelo matemático, primeiramente relaxando a integralidade das variáveis binárias para obtenção de uma solução inicial viável; e depois, na fase de melhoria das soluções, inserindo restrições que redefinem o espaço percorrido em busca local, realizada por um software de otimização. Contudo, os resultados mostram que o tempo computacional destas abordagens é elevado, tornando até mesmo a obtenção de uma solução viável impraticável, dependendo da

instância. Para contornar esta questão, o autor apresenta uma heurística construtiva, capaz de obter soluções viáveis em todos os cenários testados. Finalmente, ao conjugar o uso de tal heurística com um método otimizante, Mendes (2007) chega nos melhores resultados de sua pesquisa, que ainda assim exigem alto tempo computacional.

Levando tais resultados em conta, na presente pesquisa propõe-se o uso de uma heurística construtiva conjugada a uma heurística de melhoria, que não se baseie na estrutura matemática do problema, se assemelhando às heurísticas tradicionalmente empregadas na resolução de problemas de roteirização. No processo de melhoria, são utilizados movimentos similares aos métodos tradicionais, como a remoção e reinserção de uma tarefa em rotas diferentes ou a troca de posição da tarefa numa mesma rota, adaptando-os para lidar eficientemente com a questão da interdependência entre rotas. O método foi inspirado na heurística híbrida proposta por Ait Haddadene, Labadie e Prodhon (2016) e, à exemplo daqueles autores, também usa testes de viabilidade para agilizar a construção de solução viável.

Até a etapa de qualificação, um movimento de inserção foi desenvolvido e embutido na heurística construtiva. Nas etapas seguintes, tal movimento foi associado a um movimento inverso, de remoção de tarefas das rotas, tanto para melhorar o desempenho da heurística construtiva como para construir a vizinhança de uma solução viável, permitindo a realização de uma busca local nesta região do espaço de soluções. Para guiar o processo de busca, foram testadas meta-heurísticas capazes de fugir de regiões de mínimo local de baixa qualidade, explorando melhor o espaço de soluções, com objetivo de viabilizar uma ferramenta de planejamento eficiente, confiável e ágil para a gestão da frota AHTS em escala real.

1.4 ESTRUTURA DESTE TEXTO

Após este capítulo introdutório, o segundo capítulo dessa tese contém a revisão da literatura, que apoia a definição do problema-objeto desta pesquisa e do seu escopo, além de levantar possíveis métodos de solução para tal problema. No capítulo 3, são apresentados os modelos matemáticos de otimização desenvolvidos para o problema de roteirização de rebocadores. Já no quarto e quinto capítulos são propostos, respectivamente, métodos de obtenção de solução viável (heurística construtiva) e de obtenção de solução de melhor qualidade. Os experimentos computacionais realizados com os métodos de solução propostos são descritos no capítulo 6. Finalmente, as

conclusões desta tese são expostas no último capítulo, explicitando as limitações da pesquisa e sugerindo os caminhos para futuros desenvolvimentos.

2 REVISÃO DA LITERATURA

Na literatura da PO, o problema-objeto desta pesquisa pode ser encaixado nas classes do Problema de Roteirização de Veículos (VRP, do inglês *Vehicle Routing Problem*) e do Problema de Programação de Tarefas (MSP, do inglês *Machine Scheduling Problem*), especialmente versões do MSP com tempos de configuração dependentes da sequência (*sequence-dependent set-up times*). Em ambos os casos, se adiciona ao problema clássico um conjunto de restrições denominado de sincronização de operações, conforme definição de Drexl (2012).

Levando em conta as similaridades do problema com as classes de problemas mencionadas, a revisão da literatura foi dividida em algumas frentes, de forma a cobrir tanto a definição do problema prático que motiva esta pesquisa, como problemas similares e seus métodos de solução, situando o problema em uma lacuna do conhecimento existente. Neste capítulo são citados os trabalhos mais relevantes de cada uma das frentes de pesquisa adotadas.

A primeira frente de pesquisa é referente à aplicação que motiva esta pesquisa, para definir o problema-objeto e averiguar outras versões ou outros tratamentos já dados ao tema na literatura. A segunda frente refere-se aos problemas de MSP, focando em problemas em que a configuração de máquinas é em paralelo, nos quais é preciso atribuir cada uma das tarefas a uma máquina e determinar, para cada máquina, a melhor ordem de execução das tarefas para atender determinado objetivo. Como as máquinas estão em paralelo, duas atividades podem se sobrepor no cronograma desde que não estejam sendo executadas pela mesma máquina, assim como no problema de roteirização de uma frota de veículos. A última frente de pesquisa executada abrange os problemas de roteirização de veículos, especialmente versões com restrições de sincronização (DREXL, 2012), que têm grande aderência ao problema estudado.

2.1 DA LOGÍSTICA OFFSHORE

A exploração de petróleo em alto mar é apoiada por diversos tipos de embarcações, como apontado por Seixas et al. (2016): há, por exemplo, embarcações que responsáveis por determinar o melhor ponto para perfuração do solo, ou para lançar em solo marítimo as tubulações necessárias para operação da plataforma, conectando tais tubulações à infraestrutura instalada; por sua vez, plataformas de produção são transladadas, ancoradas e conectadas à infraestrutura submersa graças ao apoio de embarcações específicas.

Neste contexto, os rebocadores AHTS são responsáveis pelo transporte e ancoragem de unidades de perfuração, essenciais à extração do petróleo. Os equipamentos de perfuração são compartilhados entre as plataformas, de forma que uma vez que o trabalho é concluído em um certo local, a unidade deve ser removida e transportada até outra plataforma, por mais de um rebocador (SHYSHOU; GRIBKOVSKAIA; BARCELÓ, 2010).

Apesar das embarcações do tipo AHTS serem caras, o custo de deixar uma unidade de perfuração esperando até que haja um número suficiente de rebocadores disponíveis para sua realocação é muito mais alto (MENDES, 2007). Contudo, o cronograma atrelado aos equipamentos costuma ser feito individualmente, causando sobreposição das atividades demandadas por unidades diferentes, o que dificulta o planejamento das atividades da frota AHTS. Tal fato motivou Shyshou, Gribkovskais e Barceló (2010) a desenvolverem um modelo de simulação de eventos discretos para resolver o problema de dimensionamento da frota destes rebocadores. O método escolhido permite incorporar facilmente a natureza estocástica do problema, já que as operações dos navios dependem das condições climáticas, que geram incerteza no planejamento de suas operações. Como o problema tratado por aqueles autores é tático-estratégico, abrangendo um horizonte de planejamento de um ano, torna-se crucial levar em conta tais incertezas.

Os trabalhos de Shyshou, Gribkovskais e Barceló (2010) e Mendes (2007) são os únicos encontrados que lidam especificamente com os rebocadores AHTS. Mas outros aspectos da logística *offshore* que também lidam com a roteirização de embarcações aparecem com maior frequência e podem ajudar no desenvolvimento desta pesquisa. Por exemplo, Aas, Halskau e Wallace (2009) apresentam os diversos tipos de *supply vessels* e tratam o problema de composição da frota destes navios, tanto em termos de quantidade como das diferentes características que os navios que compõem a frota devem apresentar. Este tipo de embarcação, segundo Seixas et al. (2016), foi especialmente desenvolvido para transportar suprimentos até as plataformas *offshore* e coletar equipamentos e itens utilizados, para descarte ou reciclagem nos pontos de apoio localizados na costa. Em seu trabalho, Aas, Halskau e Wallace (2009) expõem para a comunidade científica o desafio de dimensionar a frota de *supply vessels*, levantando os aspectos importantes do uso deste tipo de embarcação, pois a composição da frota é essencial na garantia de que as unidades de produção e extração de petróleo não tenham suas atividades interrompidas. Porém, ao mesmo

tempo, os custos envolvidos na cadeia de exploração do petróleo, que compreende o fluxo de mercadorias e serviços para apoio e abastecimento destas unidades, devem ser minimizados tanto quanto possível, constituindo uma compensação financeira crucial para a indústria de óleo e gás. Os autores fazem uma boa caracterização do problema, se embasando em um caso real, e sugerem possíveis caminhos para pesquisas futuras sobre a composição da frota de navios de suprimento de qualquer petroleira.

Outros autores tratam problemas de roteirização inseridos no contexto de transporte marítimo, mas fora da logística de exploração de petróleo. Por exemplo, Fagerholt (2001) analisa o problema de roteirização de navios para atender a demanda de movimentação de carga entre diversos portos, com a possibilidade de contratação de outro armador para realizar a entrega de parte da demanda. Este problema é enxergado como um problema de coleta e entrega com frota heterogênea e janelas de tempo que podem ser violadas até um certo limite. A abordagem de solução empregada particiona a solução, gerando primeiramente uma quantidade considerável de roteiros possíveis (regras heurísticas reduzem o número de candidatos) e depois resolvendo o problema de otimização (com heurística). O modelo é testado com dados reais, gerando resultados que mostram que é possível reduzir o custo total (incluindo custo de inconveniência de não obedecer a algumas janelas de tempo) se as janelas não forem tratadas como restrições rígidas. Em três dos quatro casos testados, a solução ótima é obtida, sendo que na opção não atendida o horizonte de tempo do planejamento é mais longo do que seria na vida real. A limitação da abordagem diz respeito à geração de roteiros candidatos, que pode ser muito dispendiosa em problemas com poucas restrições, o que não é o caso da roteirização tratada no artigo.

2.2 DA PROGRAMAÇÃO DE TAREFAS (*SCHEDULING*)

Problemas de programação de tarefas (ou *job scheduling*, na terminologia em inglês) são aqueles em que é preciso definir a ordem em que um conjunto finito de tarefas será executado, assim como os instantes de início (e consequentemente de fim) de processamento de cada tarefa (BISKUP; FELDMANN, 2001; BRUCKER, 1998; BRUCKER; KNUST, 2012; PINEDO, 2016). Este processamento pode ser feito utilizando um único recurso ou um conjunto homogêneo ou heterogêneo de recursos (usualmente chamados de máquinas), trabalhando em série e/ou em paralelo. As tarefas podem ter prazo de entrega (*due date*), ou seja, um instante de tempo máximo

para sua finalização, que pode ser comum a todas as tarefas ou específico para cada uma delas; e podem ter uma data de liberação (*release date*), de forma que o início de cada tarefa não possa acontecer antes de determinado instante de tempo (BRUCKER, 1998; PINEDO, 2016). Estas restrições podem ser rígidas (*hard constraints*), caso seja impossível desrespeitá-las (como a data de liberação, por exemplo); ou flexíveis (*soft constraints*), quando é possível por exemplo atrasar a tarefa desde que se arque com um custo por este atraso, normalmente dado em função do tamanho do atraso e denominado peso de penalidade (*penalty weight*). Por conta disto, restrições rígidas são aquelas que, se obedecidas por uma solução, a tornam viável; enquanto as flexíveis refletem a qualidade da solução, que é tão maior quanto maior for o grau de atendimento destas restrições (RAHIMIAN; AKARTUNALI; LEVINE, 2016). Além disso, é comum que problemas de *scheduling* tenham restrições associadas à ordem de execução das tarefas, exigindo que uma tarefa seja executada antes de outra, o que é denominado restrições de precedência (*precedence constraints*) (BRUCKER, 1998; PINEDO, 2016). Finalmente, alguns problemas de *scheduling* especificam um intervalo de tempo necessário para preparo da máquina antes da execução de determinada tarefa, chamado *tempo de set-up*, que pode depender da máquina utilizada (*machine-dependent set-up time*) ou da ordem de processamento (*sequence-dependent set-up time*), ou seja, da tarefa executada imediatamente antes da tarefa em questão pela mesma máquina (BRUCKER; KNUST, 2012).

O tomador da decisão que fará o planejamento da execução das atividades pode levar em conta objetivos como minimizar o tempo total para produção de todos os itens, ou maximizar o número de produtos terminados antes de seu prazo de entrega. Por exemplo, numa indústria com demanda para produção de diversos produtos, é possível que os itens possam ser produzidos em qualquer uma das máquinas disponíveis (produção em paralelo), sendo que máquinas diferentes gastam tempos diferentes para produzir o mesmo produto (máquinas heterogêneas). Deve-se determinar, então, em qual máquina cada produto será executado e a ordem em que produtos alocados a uma mesma máquina será executada. Assim, a solução de problemas de *scheduling* é composta pela ordem em que as tarefas devem ser processadas em cada máquina, denominada sequência, e pelo instante de início de cada tarefa da sequência, que junto com a sequência compõe um programa (neste texto, usado equivalentemente ao termo em inglês *schedule*) (BISKUP; FELDMANN, 2001). Cabe destacar que, dependendo

das características do problema, é possível que haja tempo ocioso entre duas atividades processadas consecutivamente numa mesma máquina.

2.2.1 Programação em máquinas paralelas com recursos de processamento

Modelos de Programação em Máquinas Paralelas (PMS, do inglês *Parallel Machine Scheduling*), em que as máquinas operam em paralelo, tradicionalmente consideram um único tipo de recurso no sequenciamento de tarefas, como mostram as revisões de literatura de Cheng e Sin (1990), Edis, Oguz e Ozkarahan (2013) e Mokotoff (2001). Contudo, alguns problemas reais da indústria da manufatura ou voltados à computação envolvem mais de um recurso no processamento de produtos ou na execução de tarefas (EDIS; OGUZ; OZKARAHAN, 2013).

Por isso, na literatura do PMS há uma classe de problemas com recursos de processamento adicionais. Recursos de processamento atuam simultaneamente a uma máquina no processamento de uma tarefa, em oposição aos recursos de *input* ou *output*, cuja atuação é antes ou depois do processamento da tarefa por uma máquina, respectivamente (EDIS; OGUZ; OZKARAHAN, 2013). Estes recursos podem ser renováveis, o que significa que após a execução de uma tarefa qualquer, a fração de sua capacidade que foi alocada ao processamento da tarefa torna-se disponível novamente para o processamento de outras tarefas; ou não renováveis, se, uma vez alocada, esta capacidade é consumida e não pode ser usada para processar outras atividades (SLOWINSKI, 1980). Além disto, recursos podem ser discretos ou contínuos, segundo o tipo de variável associada à quantidade (ou fração da capacidade) do recurso alocada a cada tarefa (SLOWINSKI, 1980). Por fim, no caso dos recursos renováveis, sua alocação pode ser estática ou dinâmica: no primeiro caso, um recurso adicional (ou fração da capacidade total do recurso) é associado a uma única máquina ao longo de todo o horizonte de planejamento, podendo ser usado no processamento de qualquer tarefa processada na máquina em questão; já na formulação dinâmica, se o recurso é associado a uma máquina para o processamento de uma tarefa, antes e após seu processamento o mesmo recurso pode estar associado ao processamento de outra tarefa em outra máquina.

Dentro desta classe de problemas, é de especial interesse à presente pesquisa o problema de Programação de Tarefas em Máquinas Paralelas com Restrição de Recursos (RCPMSP, do inglês *Resource Constrained Parallel Machine Scheduling Problem*), definido da seguinte maneira: é dado um conjunto de n tarefas a ser

processado em um conjunto de m máquinas paralelas não relacionadas, com apoio de um conjunto u de recursos adicionais. Há uma quantidade finita de unidades disponíveis de cada recurso pertencente a u , denotada pelo vetor $b = \{b_1, b_2, \dots, b_u\}$. O tempo de processamento p_{ij} é especificado para cada par tarefa-máquina e, durante seu processamento, cada tarefa pode exigir o uso de recursos adicionais a quantidades especificadas por um vetor $R(i) = \{R_1(i), R_2(i), \dots, R_u(i)\}$. Assim, a resolução do RCPMSP é difícil porque exige não só uma alocação eficiente das tarefas, mas a determinação de agrupamentos eficientes para o processamento simultâneo das tarefas a cada instante de tempo, garantindo que a capacidade de cada recurso é respeitada ao longo de toda a programação (BŁAZEWCZ *et al.*, 1986; EDIS; OGUZ; OZKARAHAN, 2013).

Há ainda a classe de problemas denominada Programação de Máquinas Paralelas com Recursos Flexíveis (PMFRS, do inglês *Parallel Machine Flexible Resource Scheduling*), introduzida por Daniels e Mazzola (1993) e Daniels, Hoopes e Mazzola (1996). Neste caso, um conjunto de n tarefas, N , é processado pelo conjunto de m máquinas, sendo dada a alocação de cada tarefa a uma única máquina (na literatura denominada como máquina dedicada) (DANIELS; HOOPES; MAZZOLA, 1996; DANIELS; MAZZOLA, 1994; EDIS; OGUZ; OZKARAHAN, 2013). Assim como no RCPMSP, o processamento de cada tarefa em uma máquina requer o uso simultâneo de um recurso adicional, cuja capacidade é limitada e renovável, mas nesse caso o recurso adicional é único e o tempo de processamento da tarefa na máquina depende linearmente da quantidade de unidades de recurso adicional alocada para sua execução. Para modelar matematicamente esta particularidade do PMFRS, são definidos modos de processamento de cada tarefa $i \in N$, de tal forma que cada modo corresponde a uma determinada quantidade do recurso adicional alocada a sua execução. Assim, é possível associar uma tarefa i a um conjunto de modos de processamento K_i , e o tempo de processamento da tarefa ao modo $k \in K_i$, com k igual ao número de unidades do recurso adicional, é denotado por $\hat{p}_{ik}$. Segue que uma generalização possível para o PMFRS é dada pelo problema sem alocação prévia de tarefas a máquinas, denominado PMFRS Não-Específico (UPMFRS, de *Unspecified PMFRS*). No UPMFRS, o tempo de processamento depende da máquina escolhida e da quantidade do recurso adicional utilizada (ou seja, do modo empregado), e é denotado por $\hat{p}_{ijk}$.

A definição de modos também aparece na literatura de *project scheduling*, uma classe de problemas de programação em que se deseja traçar um cronograma de execução de diversos projetos, sendo que cada projeto contém uma série de tarefas próprias, a executar numa ordem específica. Mais especificamente, no problema de Programação de Múltiplos Projetos com Recursos Restritos e Múltiplos Modos (MMRCMPSP, do inglês *Multi-Mode Resource Constrained Multi-Project Scheduling Problem*), definido da seguinte maneira: é dado um conjunto de **projetos** (equivalente a *tarefas* ou *jobs* na nomenclatura usual de *scheduling*), sendo cada projeto associado a um conjunto de atividades (ou operações) que devem ser executadas em determinada ordem, sem interrupção e sobreposição temporal; a um instante de liberação; e a um prazo de entrega (ARTIGUES; ROUBELLAT, 2000). Assim, a primeira atividade de uma tarefa só pode ser iniciada a partir da data de *release* do projeto e é desejável que a última atividade termine no máximo no prazo do projeto, pois após este instante começa a contar o atraso na entrega do projeto, atrelado a uma penalização. Cada atividade de cada projeto pode ser executada de diferentes modos, sendo que cada modo usa uma determinada quantidade de cada um dos recursos disponíveis e tem uma certa duração. Finalmente, a função objetivo é de minimização do maior atraso, que pode ser reduzido à minimização do *makespan*.

Apesar de serem frequentes na literatura, os autores que tratam o problema com tempo de processamento dependente da quantidade de recurso (adicional) alocada normalmente adotam simplificações para diminuir sua complexidade. Poucos autores tratam a versão dinâmica do problema, ou seja, aquela em que o recurso adicional pode ser designado para atuar simultaneamente à operação de diferentes máquinas ao longo do programa (EDIS; OGUZ; OZKARAHAN, 2013). Além disso, são poucos os trabalhos que não tratam a alocação tarefa-máquina como dada e que, na mesma modelagem, consideram o tempo dependente da quantidade de recurso adicional utilizada.

A revisão da literatura de Edis, Oguz e Ozkarahan (2013) mostra a escassez dos problemas RCPMSP, PMFRS e UPMFRS na literatura e apresenta algumas possibilidades de extensão dos modelos matemáticos para o PMFRS e UPMFRS dinâmicos. Os autores destacam que poucos modelos usam função objetivo diferente do *makespan* e que nenhuma das publicações estudadas apresenta mais de um critério na função objetivo. Entre os problemas menos estudados e considerados mais promissores pelos autores, são citados os RCPMSPs com um tipo de recurso adicional e máquinas não relacionadas ou uniformes, tempos de processamento arbitrários e

funções objetivos relacionadas ao prazo de entrega das tarefas; além, claro, das versões dinâmicas dos PMFRS e UPMFRS.

2.2.2 Programação de tarefas com alocação de equipes

A literatura de problemas de geração de cronograma para equipes, considerando tarefas a realizar, é vasta e não será abordada neste trabalho. Contudo, a formulação proposta por Dohn, Kolind e Clausen (2009) possui muitas semelhanças com o problema de roteirização de rebocadores e, por conta disto, será explanada a seguir.

Dohn, Kolind e Clausen (2009) abordam o Problema de Alocação de Força-tarefa com Janelas de Tempo, Restrições de formação de Equipe e número limitado de times (m-MAPTWTC, sigla do inglês *Manpower Allocation Problem with Time Windows, job-Teaming Constraints and limited number of m teams*). O objetivo do problema é definir as equipes que executarão cada tarefa demandada, respeitando janelas de tempo das tarefas e as jornadas de trabalho dos indivíduos, e maximizando o número total de tarefas atendidas. Os autores citam aplicações no setor de tratamento médico domiciliar, hospitais, prestadores de serviços técnicos e gestão de equipe de solo para aeroportos, que é o problema que motivou sua pesquisa.

As equipes, neste caso, são responsáveis por todas as atividades entre a chegada e partida de uma aeronave no aeroporto, como desembarque, transporte e entrega das bagagens nas esteiras rolantes e limpeza da aeronave. Assim, um time heterogêneo precisa ser composto e alocado para cada chegada de aeronave, garantindo que o número mínimo de integrantes habilitados para executar cada tarefa é respeitado, assim como a janela de tempo disponibilizada pelo aeroporto e as jornadas de trabalho dos funcionários. Um aspecto importante deste problema é que a solução deve equalizar a carga de trabalho dos times, para não gerar atritos ou insatisfação dos funcionários envolvidos.

O modelo matemático proposto é de Programação Inteira, decomposto pelo método de Dantzig-Wolfe em um problema *master* e um subproblema *pricing*, resolvido por geração de colunas e *branch-and-price*. O processo de *branching* garante a integralidade das variáveis e a sincronização das tarefas de cooperação. As instâncias testadas usam dados reais de aeroportos europeus.

Há algumas diferenças importantes entre m-MAPTWTC e o problema abordado nesta pesquisa. São elas: cada tarefa demanda um certo número de times, que os autores modelam como se cada time demandado executasse uma subtarefa da tarefa em

questão, mas não é necessário executar todas as subtarefas de uma atividade; além disto, não é necessário atender todas as tarefas; os times são tratados como um recurso fixo e não um combinação de indivíduos; e os intervalos de tempo mínimo entre tarefas não dependem do recurso alocado, como é comum em problemas que não envolvam o deslocamento de recursos entre pontos geográficos.

No modelo matemático proposto por Dohn, Kolind e Clausen (2009), assume-se que são conhecidos os seguintes conjuntos e parâmetros:

- Conjunto K de trabalhadores (recursos humanos) disponíveis, sendo os indivíduos categorizados em m tipos de função, $T_1, T_2 \dots T_m$.
- Um conjunto J de n tarefas $i \in J = \{1, 2 \dots n\}$, sendo cada tarefa descrita por uma tupla $\{[v_{i1}, v_{i2} \dots v_{im}], p_i, [s_i, e_i]\}$, onde o primeiro vetor contém o número demandado de trabalhadores de cada tipo, a duração do serviço e a janela de tempo para início da tarefa.
- Os trabalhadores iniciam e terminam sua jornada de trabalho em um centro de serviços, denotado por $j = 0$ no início das rotas e por $j = n + 1$ no final.
- Sempre que uma tarefa $j \in J$ puder ser executada logo após outra tarefa $i \in J: i \neq j$ pelo mesmo recurso, é conhecido o valor de tempo de deslocamento entre as duas tarefas d_{ij} . Além disso, o tempo de deslocamento entre o centro de serviços e cada uma das tarefas é denotado por d_{0j} . O conjunto de todos os pares (i, j) assim definidos é denotado por A .

São variáveis de decisão:

- x_{ijk} : binária que indica se o trabalhador k executa a tarefa j imediatamente após executar a tarefa i (ou após sair do centro de serviços)
- w_i : data/hora de início da tarefa i

A função objetivo é minimizar uma soma ponderada do número de recursos utilizados e da distância total viajada:

$$\min z = \left(\alpha \cdot \sum_j \sum_k x_{0jk} + \beta \cdot \sum_k \sum_{(i,j) \in A} d_{ij} x_{ijk} \right) \quad \text{Eq. 2-1}$$

Sujeito a:

- Atender a demanda de cada tipo de trabalhador, para todas as tarefas:

$$\sum_{j \in \Delta^-(i)} \sum_{l=K_{k-1}+1}^{K_k} x_{jil} = v_{ik} \forall i \in J, k = 1, 2 \dots m \quad \text{Eq. 2-2}$$

- Exatamente um arco ligando o centro de serviços a uma tarefa (ou nó final das rotas) deverá ser selecionado para cada recurso (funcionário) disponível:

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1 \forall k \in K \quad \text{Eq. 2-3}$$

- As rotas devem ser contínuas, por isso o número de arcos de entrada em um nó selecionados deverá ser igual à quantidade selecionada de arcos de saída deste nó, para cada funcionário:

$$\sum_{i \in \Delta^-(j)} x_{ijk} - \sum_{i \in \Delta^+(j)} x_{jik} = 0 \forall i \in J, k \in K \quad \sum_{j \in \Delta^+(0)} x_{0jk} = 1 \forall k \in K \quad \text{Eq. 2-4}$$

- Para cada rota, é preciso selecionar um arco ligando a última tarefa executada por cada funcionário ao nó final das rotas, que representa o próprio centro de serviços:

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1 \forall k \in K \quad \text{Eq. 2-5}$$

- Não pode haver sobreposição temporal na execução de tarefas executadas por um único funcionário (esta restrição também garante que não haja subciclos nas rotas e que o início de uma atividade depende da chegada de todos os funcionários ao nó em questão):

$$w_j \geq w_i + p_i + d_{ij} + M(x_{ijk} - 1) \forall k \in K, (i, j) \in A \quad \text{Eq. 2-6}$$

- A data de início da tarefa precisa respeitar a janela de tempo do cliente:

$$s_i \leq w_i \leq e_i \forall i \in J \quad \text{Eq. 2-7}$$

- Por fim, é declarado o domínio das variáveis:

$$x_{ijk} \in \{0, 1\} \forall i, j \in J, k \in K \\ w_i \in \mathbb{R}^+ \forall i \in J \quad \text{Eq. 2-8}$$

Em trabalho mais recente, Rahimian, Akartunali e Levine (2016) tratam do problema de definição de turnos de enfermeiros em hospital, conhecido na literatura como *Nurse Rostering* e definido da seguinte maneira: dado um horizonte de planejamento pré-definido e uma equipe finita de enfermeiros disponíveis, deve-se definir os dias e turnos

de trabalho que cada profissional deverá cumprir, tal que sejam obedecidas restrições operacionais, contratuais e as preferências de cada profissional.

Segundo os autores, meta-heurísticas não são tão eficientes quando a estrutura do problema é muito complexa, tornando desafiador encontrar uma solução de qualidade, ou até mesmo uma solução viável, em tempo de execução razoável. Por outro lado, abordagens exatas como *Integer Programming* e *Constraint Programming* são ótimas em resolver problemas complexos, mas incapazes de resolver instâncias de média ou larga escala (RAHIMIAN; AKARTUNALI; LEVINE, 2016). Para contornar este impasse, os autores recorrem aos métodos híbridos, que combinam técnicas exatas e meta-heurísticas. Normalmente, o método exato escolhido é usado para obter uma solução que satisfaça parte das restrições do problema, para na sequência ser aprimorada pela meta-heurística. Para resolver o problema proposto, os autores desenvolvem um algoritmo híbrido que utiliza uma meta-heurística e um método exato, mais especificamente o VNS e a Programação Inteira (*Integer Programming*), aproveitando os pontos fortes de ambas as abordagens. Embora a combinação entre métodos exatos e heurísticos, chamada de *matheuristic*, não seja novidade, os autores argumentam que a forma como combinam os métodos o é, pois o IP é encapsulado no VNS, servindo para diversificar e intensificar a busca local, como mecanismo de perturbação e, depois da finalização do VNS, ele é novamente utilizado para melhorar a solução até que certo limite de tempo seja atendido. Na fase encapsulada, o IP é usado dentro de uma abordagem do tipo destrói-e-recria (*ruin-and-recreate*), em que partes da solução que geram altas penalidades são removidas e uma nova solução é recriada pelo IP.

Antes da execução da *matheuristic* proposta, uma heurística gulosa é usada para gerar uma solução inicial. O algoritmo proposto é iterativo e, por tentativa e erro, tenta gerar programas viáveis para cada um dos profissionais da equipe individualmente, selecionando aleatoriamente os dias em que o profissional trabalhará e, depois, os turnos, respeitando padrões de trabalho viáveis. Caso a carga total de trabalho resultante ou o número de finais de semana trabalhados ultrapassem limites pré-estabelecidos nas restrições do modelo, a solução é descartada e o processo é reiniciado. Segundo os autores, são necessários 100 ciclos deste processo até gerar um cronograma de trabalho viável para um profissional.

2.2.3 Inserção de tarefa em programa pronto

Problemas de inserção são uma classe especial dos problemas de programação de tarefas em que parte das atividades a programar já faz parte de um cronograma pronto e as demais atividades precisam ser encaixadas em tal programa, com intuito de minimizar o instante de término da última atividade a ser processada (ou seja, o *makespan* da solução) (GRÖFLIN; KLINKERT, 2006). Tais problemas, segundo Groflin e Klinkert (2006), podem ser usados tanto para aplicações diretas, inspirados em sistemas produtivos dinâmicos, como em métodos heurísticos que usem buscas locais. Neste último caso, é preciso que o método de busca também seja capaz de encontrar uma ou mais atividades que possam ser retiradas de sua posição original num programa, para então resolver o problema de inserção.

No *job shop*, em que cada operação de uma tarefa deve ser executada em uma máquina específica, segundo uma ordem dada, e uma tarefa só passa por uma máquina uma única vez, dois tipos de inserção podem ser utilizadas (GRÖFLIN; KLINKERT, 2006):

- de atividade (*activity*, operação), em que apenas uma operação precisa ser inserida no programa original;
- de tarefa (*job*), em que um conjunto de atividades pertencentes a um mesmo *job*, portanto contendo suas próprias restrições de precedência, deve ser inserido no programa.

Assim, a complexidade do problema depende do tipo de inserção e das características do programa original e do objeto a inserir em tal programa. Groflin e Klinkert (2006) dividem esta complexidade em dois aspectos: (1) do problema de viabilidade, ou seja, da determinação de haver solução viável para o problema; e (2) do problema de inserção, de encontrar uma solução viável que minimize o *makespan* obtido. A seguir são detalhadas técnicas de inserção já aplicadas a problemas de programação de tarefas e de projetos (*project scheduling*).

Werner e Winkler (1995) apresentam uma heurística de duas fases para resolver o problema clássico do *job shop* (JS). A primeira fase da heurística, construtiva, inicia-se com a adição da tarefa de maior tempo total (soma de tempos de processamento de todas as suas operações) às máquinas, respeitando a ordem em que suas operações devem ser executadas. Em seguida, a operação de maior tempo dentre as operações das tarefas remanescentes é adicionada ao programa, testando todas as possibilidades

de sua inserção e escolhendo aquelas de menor custo marginal (custo marginal é igual ao *makespan* do maior caminho que passe por esta tarefa - que não necessariamente é o caminho crítico). Ressalta-se que ao invés de simplesmente adotar a melhor alternativa (de menor custo marginal), os autores pegam um conjunto das k melhores soluções parciais e as exploram a cada iteração, cada uma tornando-se mãe de novas soluções parciais, e sempre acatando as k melhores soluções parciais filhas dentre todas as soluções filhas resultantes (ou seja, mesmo que elas não tenham vindo da modificação de uma mesma solução mãe). Este processo é repetido até que todas as operações tenham sido adicionadas ao programa, resultando numa solução factível. Então a segunda fase da heurística é aplicada, tomando como base a melhor solução inicial obtida pela construtiva, LR_0 . Primeiramente, seleciona-se uma tarefa que tenha ao menos uma operação em um bloco do caminho crítico da solução inicial, LR_0 (bloco é um conjunto de operações processadas no caminho crítico sem intervalo de parada entre elas), e seleciona-se sua operação de maior tempo de processamento, verificando seu tipo, que determina o tipo de movimento que será aplicado (as operações são classificadas segundo o tipo de movimento que pode ser feito com ela, pra direita ou pra esquerda da programação original). Deleta-se tal tarefa do programa (todas suas operações) e atualiza-se a solução parcial remanescente. Então, conforme o tipo de movimento que será feito, uma a uma as operações da tarefa são reinseridas na programação, seguindo a ordem em que devem ser processadas à direita ou à esquerda de sua posição original. Para cada operação reinserida, são testadas consecutivamente as posições em que sua inserção é possível, até chegar a um critério de parada destes testes, que tem a intenção de indicar o afastamento de uma boa região de inserção da tarefa. Porém, apenas as k melhores programações resultantes são levadas em conta. No método proposto, ou todas as operações são reinseridas na posição original ou à direita dela, ou são todas reinseridas na posição original ou à esquerda dela, resultando uma nova solução completa e viável (ou melhor, num conjunto de novas soluções completas e viáveis). Finalmente, a vizinhança é composta pelos melhores programas resultantes da modificação para direita ou para esquerda de todas as operações da tarefa em questão que estejam em blocos na solução mãe desta vizinhança. Então, os melhores vizinhos (soluções completas viáveis) são selecionados para, por sua vez, se tornarem soluções mães de novas vizinhanças. Esta estratégia de busca baseia-se na estratégia de *path search*, que, similarmente ao *Simulated Annealing* e à Busca Tabu (*Tabu Search*), tenta escapar de regiões de ótimos locais.

O problema de inserção de uma tarefa inicialmente não prevista em um programa já pronto, com a intenção de minimizar a soma de todos os atrasos, também é tratado por Duron, Proth e Wardi (2005). Neste caso, o recurso é único, as tarefas já programadas possuem prazo *soft*, mas a nova tarefa possui prazo *hard*, sendo que nenhuma tarefa possui data de liberação. Não são permitidas interrupções ou sobreposições temporais nos processamentos das tarefas. Além disso, a nova tarefa possui duas sub tarefas (ou operações, atividades) que devem ser executadas em sequência, com um intervalo fixo de tempo entre o fim da primeira sub tarefa e o início da segunda. Os autores testam 4 formas de solução para o problema, sendo que uma delas trata a tarefa como um bloco, ou seja, como uma tarefa única de duração igual à soma das durações das sub tarefas e do tempo de ociosidade entre elas, portanto ignorando a possibilidade de executar outra atividade no intervalo de ociosidade entre as sub tarefas. Como para a aplicação prevista é necessário que o tempo computacional, necessário para inserir a nova tarefa no programa já pronto, seja baixo, a solução ótima serve apenas como *Lower Bound* para avaliar a qualidade das demais soluções.

Como esperado, os resultados mostram que é importante aproveitar o tempo ocioso entre sub tarefas, pois o desempenho da heurística que trata a tarefa como um bloco único foram sempre os piores, em todas as instâncias testadas (DURON; PROTH; WARDI, 2005). Entretanto, os resultados devem ser avaliados com cautela, já que apenas um programa gerado randomicamente foi usado nos testes, com 11 diferentes tipos de tarefas para inserção (ou seja, cada algoritmo foi executado 11 vezes, sempre numa mesma programação). Além disso, é curioso que o programa inicial tenha tempos de ociosidade entre tarefas uma vez que não há restrição de disponibilidade (*release date*) das tarefas e que a função objetivo seja de minimização de atrasos, e não de desvio em relação ao prazo, pois isso significa que pode existir programa melhor do que o inicial e que, ao inserir a nova tarefa neste programa, os algoritmos podem estar aproveitando desta “folga” para melhorar a solução inicial. Contudo, é possível que tais instâncias repliquem fielmente a aplicação que os autores tinham em mente.

No trabalho de Gröflin e Klinkert (2006) é apresentada uma formulação genérica do problema de inserção, aplicável a uma variedade de problemas de *scheduling*, inclusive: (1) **Job Shop com Múltiplos Processadores de Tarefas** (MPTJS, de *Multi-Processor-Task Job Shop* ou **Multi-Resource Job Shop**), em que uma ou mais operações devem ser realizadas por mais de um recurso simultaneamente, mantendo a restrição de que cada recurso é capaz de executar apenas uma tarefa por vez, sem

interrupções; (2) **Generalized Blocking Job Shop**, em que não há pulmão entre as máquinas disponíveis, por isso, ao terminar o processamento de uma atividade, a máquina só fica livre quando a atividade recém concluída estiver ocupando a próxima máquina ou ela não necessitar passar por mais nenhuma operação (ou seja, se estiver sido finalizada); e (3) **Generalized No-Wait Job Shop**, que não permite tempo ocioso entre a execução de operações consecutivas de uma tarefa, o que indiretamente significa que também não há pulmão entre máquinas. Os autores apresentam um método para determinar se existe solução viável para uma inserção desejada e para encontrar *Upper* e *Lower Bounds* para a solução ótima.

Em trabalho subsequente (GRÖFLIN; KLINKERT; DINH, 2008), os autores abordam a generalização do JS em que o processamento de cada operação de cada tarefa utiliza um subconjunto de máquinas de maneira simultânea. Mantêm-se as restrições de não interrupção ou sobreposição de tarefas na programação individual de cada máquina. No texto, os autores descrevem o problema de inserção de uma tarefa no MPTJS e apresentam uma descrição poliédrica (*polyhedral description*) de todas as soluções viáveis, resumida a seguir. Também é apresentado um método de cálculo de LB para o problema em questão e para o caso especial do JS (o JS é caso especial do MPTJS em que, para qualquer operação i de um job j , O_{ij} , o subconjunto de máquinas associadas ao seu processamento, M_{ij} , tem módulo 1).

Os autores estendem a formulação de programação disjuntiva do JS para o MPTJS. Nesta formulação, o problema é descrito por um grafo disjuntivo em que:

- os nós são as operações pertencentes a todas as tarefas, mais dois nós fictícios (ou *dummies*) que representam operações de duração nula a serem processadas antes e depois de todas as demais;
- os arcos são divididos em subconjuntos. O primeiro, A , contém todos os arcos direcionais que representam as restrições de precedência intrínsecas a cada tarefa, conectando pares de operações de um mesmo *job* que devem ser processadas consecutivamente, assim como os arcos que conectam os nós fictícios de início e fim a todas as operações; o segundo contém os arcos que conectam pares de operações pertencentes a tarefas diferentes e que utilizam pelo menos uma máquina em comum, e por isso não podem se sobrepor no programa-solução. Desta forma, se as atividades r e s são de tarefas diferentes e utilizam uma máquina em comum, então o grafo do problema deve incluir um

par de arcos disjuntivos e complementares da seguinte forma: um iniciado em r apontando para s e outro iniciado em s apontando para r . Para facilitar a representação deste par de arcos disjuntivos, é usada uma aresta (arco não direcional) conectando um nó r a um nó s , representada pelo subconjunto E (de aresta em inglês, *edge*). Por fim, o conjunto contendo todas as arestas, ou família de E , é referenciado como $\mathcal{E}$

- o comprimento de cada arco de E e de A é igual à duração da tarefa que os iniciam.

Usando tal representação, uma seleção de arcos de E é **completa** se contém pelo menos um arco de cada aresta do grafo e é **acíclica** se não contém subciclo. Uma solução viável para o problema é acíclica e completa, portanto, contém exatamente um arco de cada aresta de $\mathcal{E}$.

Uma formulação matemática de otimização é apresentada para o problema, minimizando o *makespan*, que na representação adotada é igual ao início de processamento da tarefa *dummy* final. O modelo inclui um conjunto de restrições disjuntivas para garantir a não sobreposição de operações que utilizam máquina em comum mas não pertencem ao mesmo *job* (já que para operações do mesmo *job* a própria ordenação intrínseca ao *job* garante a não sobreposição de atividades). A formulação matemática apresentada é, então, equivalente à seguinte formulação: dentre todas as seleções viáveis no grafo descrito anteriormente, encontre aquela que minimiza o maior caminho entre os nós fictícios inicial e final.

Por sua vez, o problema de inserção de tarefa é: dado um *job* J e uma seleção T viável para um determinado grafo disjuntivo, que representa todos os jobs já programados, encontre a seleção S de arcos incidentes a J tal que a união de S e T forme uma seleção viável em G (grafo do problema) e que minimiza o *makespan*. Os arcos incidentes em J são aqueles que possuem arcos complementares, ou seja, formam arestas ligando operações de *job* J a operações do grafo original.

Entretanto, ressalta-se que o problema de encontrar inserção viável pode deixar de ser trivial quando precisa ser resolvido como parte de uma heurística de busca local, de forma que alguns arcos são pré-selecionados (ou arbitrariamente excluídos da solução) para que seja gerada solução diferente da original, ou para que a nova solução não se distancie muito da solução que a originou (GRÖFLIN; KLINKERT; DINH, 2008). Assim como no problema tratado no presente trabalho, no problema genérico definido pelos

autores o tempo de processamento das operações não depende da máquina utilizada. Contudo, naquele caso não há mais de um conjunto de máquinas que pode ser adotado para execução de cada operação. Além disso, os tempos de *set-up* não são dependentes da sequência de processamento, o que seria necessário para adaptar o modelo de *scheduling* para um modelo de roteirização.

Por sua vez, Artigues, Michelon e Reusser (2003) abordam dois tipos de problema derivados do Problema de Programação de Projetos com Recursos Restritos (RCPSP, do inglês *Resource-Constrained Project Scheduling Problem*). No RCPSP, um conjunto de atividades precisa ser processado utilizando um conjunto de recursos renováveis, de forma a minimizar o *makespan* da solução (programação) final. Cada recurso do conjunto possui uma capacidade finita e é renovável, o que significa que as unidades de recursos alocadas a uma certa atividade voltam a ficar disponíveis após o fim do processamento da mesma, para uso de outra atividade. Desta forma, pode-se interpretar que cada recurso é um tipo de máquina e que dispõe-se de um conjunto de máquinas de cada tipo operando em paralelo, ou seja, cada unidade de um recurso k representa uma máquina do conjunto de R_k máquinas do tipo k . Restrições de precedência entre duas atividades i e j determinam que a atividade j só pode ser iniciada após a conclusão de i , e cada atividade mobiliza r_{ik} unidades de cada recurso k , bloqueando-as durante um intervalo de tempo igual a p_i . O primeiro tipo de RCPSP tratado é o estático: deseja-se determinar o instante de início de um conjunto de atividades, respeitando restrições de disponibilidade de recursos e de precedência entre atividades. O segundo é a versão dinâmica do RCPSP, em que é necessário revisar uma programação em curso devido ao surgimento de uma perturbação no programa, que neste caso significa o surgimento de uma atividade inesperada (ARTIGUES; MICHELON; REUSSER, 2003).

Os autores utilizam o conceito de posições de inserção dominantes, já empregado na literatura para variantes do JS (ARTIGUES; MICHELON; REUSSER, 2003), para melhorar o algoritmo proposto anteriormente, com objetivo de inserir uma nova atividade em uma solução existente. O número de posições dominantes pode ser reduzido a $O(n^2)$, sendo n o número de atividades, e os autores apresentam um algoritmo com tempo de complexidade $O(n^2m)$ em que m é o número de recursos renováveis. Um algoritmo de *rescheduling* é proposto para a versão dinâmica do problema, comparado a uma heurística baseada em regra de prioridade. Para a versão estática do RCPSP, os autores apresentam duas heurísticas contendo o algoritmo de

inserção encapsulado: uma construtiva simples e uma busca tabu, que a cada iteração reinsere uma ou mais atividades, modificando a solução corrente.

O modelo matemático de otimização é definido e a modelagem apresentada utiliza um grafo de fluxo de recursos entre as atividades, em uma rede denominada pelos autores de *activity-on-node (AON)-flow network*. Para gerar um fluxo viável para o RCPSP estático, é utilizada uma extensão do Esquema de Geração Paralela de Programa (SGS, do inglês *parallel Schedule Generation Scheme*), que constrói um programa incrementando passo-a-passo de programas parciais, avançando no tempo a cada iteração graças à adição de uma atividade aos programas parciais. A seleção da atividade a adicionar no instante corrente, t_g , é feita somente dentre atividades selecionáveis, ou seja, atividades que utilizam somente os recursos disponíveis em t_g e cujas atividades precedentes foram finalizadas até o instante t_g . Após a adição de uma atividade, calcula-se o menor tempo de conclusão de todas as atividades em progresso, e o algoritmo avança para t_{g+1} .

O algoritmo de inserção INS, por sua vez, usa como *input* uma solução viável e uma atividade que deve ser inserida nesta solução. Usando as capacidades dos arcos e os **instantes mínimos e máximos para início da atividade** de cada nó (respectivamente *earliest starting times* e *latest starting times* dos nós, o INS é capaz de achar a posição ótima para inserção dessa atividade, considerando que as precedências já estabelecidas pela solução de input devem ser mantidas. Para tal, o algoritmo explora um conjunto de cortes dominantes de capacidade máxima. Os autores aplicam o algoritmo de inserção para o RCPSP dinâmico, comparando seus resultados com a aplicação da heurística construtiva SGS, atribuindo o valor do *earliest starting time* ao instante de início de cada atividade. Além disso, o algoritmo de inserção é misturado com o SGS para criar um método de solução para o problema estático, ou seja, de programação de todas as atividades partindo de um programa vazio. Por fim, o algoritmo de inserção é embutido em uma busca tabu, na geração da vizinhança das soluções exploradas a cada iteração da meta-heurística.

Os autores concluem que a heurística construtiva combinada ao algoritmo de inserção é mais robusta que o método guloso sozinho e que a inclusão deste algoritmo em uma meta-heurística é promissora, mas poderia ter resultados melhores se a solução inicial fosse melhor, o que é deixado para pesquisas futuras (ARTIGUES; MICHELON; REUSSER, 2003).

No trabalho de Artigues e Roubellat (2000) também é proposto um algoritmo de inserção para resolver problemas de reprogramação de tarefas (*rescheduling*) devido ao surgimento de uma nova tarefa a ser executada. Neste caso, o problema tratado é da MMRCMPSP (Programação de Múltiplos Projetos com Recursos Restritos e Múltiplos Modos, já definido na seção 2.2.1). Como os próprios autores afirmam, o algoritmo também pode ser usado em buscas locais, para formar vizinhanças de soluções, desde que seja combinado com outro algoritmo que remova a tarefa de um programa já pronto.

Os autores usam uma representação da solução em grafo, que inclui tanto a ordem de execução das atividades como o fluxo de recursos utilizados para sua execução, e apresentam um algoritmo que trabalha diretamente nessa representação para inserir uma nova atividade no programa já montado. Para reduzir a complexidade do problema, Artigues e Roubellat (2000) usam um sub grafo da solução original para fazer a inserção, excluindo da representação aqueles nós e arcos que não são afetados com a inserção da nova atividade. É interessante observar que, embora a inserção desta atividade possa provocar grandes mudanças no programa original, já que os recursos são limitados e que há diversas maneiras de executar as atividades, neste problema a posição de inserção da atividade em relação às demais atividades é dada, já que a nova atividade faz parte de um projeto já programado e está vinculada a restrições de precedência e de sucessão. Assim, sua inserção pode provocar alterações no sub grafo que representa os fluxos de recursos, mas não no sub grafo que representa a ordem de execução das atividades dentro de cada projeto (embora possa haver alteração na ordem de execução ou finalização dos projetos como um todo, devido justamente a alterações no fluxo de recursos). Apesar do método poder ser executado em diversos problemas derivados do *job shop*, além do próprio, não são contemplados os problemas em que a atividade inserida não está associada a um projeto já programado e/ou não possui restrições de precedência e sucessão, podendo ser encaixada em qualquer posição (relativamente à ordem de execução das atividades). Além disso, os próprios autores desencorajam o uso do método para problemas em que os tempos de *set-up* entre atividades variam conforme a sequência de execução programada, como no caso do problema-objeto desta pesquisa.

2.3 DA ROTEIRIZAÇÃO COM SINCRONIZAÇÃO DE RECURSOS

Na primeira parte desta seção, em 2.3.1, são apresentadas as definições e a taxonomia relativa a problemas de otimização com restrições de sincronização, conforme encontradas na literatura. Em 2.3.2, são citados trabalhos relevantes de problemas de roteirização para coleta e entrega, evidenciando as similaridades com o problema tratado nesta pesquisa. Por fim, em 2.3.3, são apresentados os trabalhos que focaram em problemas de roteirização com restrições de sincronização e janelas de tempo. Os modelos matemáticos dos trabalhos citados estão apresentados na seção 2.4, separados nas subseções de acordo com a estratégia de modelagem adotada.

2.3.1 Definições e tipos de sincronização

Segundo Prins (2004), o Problema de Roteirização de Veículos (VRP, do inglês *Vehicle Routing Problem*) é definido em uma rede completa direcionada, $G = (V, E)$, na qual V representa o conjunto de nós (ou vértices, as duas nomenclaturas são equivalentes) enquanto E , o conjunto de arcos (ou arestas, nomenclatura mais utilizada quando o arco é bidirecional) que conecta todos os pares de nós. O índice i é usado para se referir aos nós, com $i = \{0, 1, \dots, n\}$, tal que o nó $i = 0$ representa um depósito com m veículos de capacidade W e oferta de carga; e os demais nós, $i > 0$, representam clientes com demanda não negativa $q_i < W$, tal que a demanda total (soma das demandas de todos os nós) é inferior ou igual à oferta. Por sua vez, os arcos (i, j) representam o trajeto entre dois pontos da rede, $i \in V$ e $j \in V$ com $i \neq j$, portanto ligando clientes ou depósito a cliente, e estão associados a um custo de deslocamento não negativo c_{ij} (com $c_{ij} = c_{ji}$), que na prática usualmente representa a distância ou o tempo de deslocamento entre i e j (PRINS, 2004).

O VRP consiste em determinar um conjunto de m viagens, iniciadas e terminadas no depósito $i = 0$, tal que a demanda de todos os clientes é atendida e cada cliente é visitado uma única vez, respeitando-se a capacidade dos veículos (PRINS, 2004). Nas diversas formulações do VRP, m pode ser um dado de entrada do problema, que restringe o número máximo de viagens, ou uma variável de decisão. O VRP é comprovadamente NP-Difícil (PRINS, 2004), o que motivou o desenvolvimento de diversas heurísticas para solucionar instâncias de dimensão compatível a problemas reais. Heurísticas construtivas, que partem de uma solução vazia e a cada iteração adicionam um cliente ainda não visitado a uma rota, até que todos os clientes tenham

sido adicionados, são em geral mais rápidas e simples que meta-heurísticas, que normalmente partem de uma solução viável (obtida por uma heurística construtiva) e, usando um processo estruturado, aplicam modificações nesta solução, guardando a melhor solução obtida ao longo do processo, até atingir um critério de parada. Prins⁽²⁰⁰⁴⁾, por exemplo, apresenta uma heurística híbrida baseada no Algoritmo Genético (GA, do inglês *Genetic Algorithm*) para uma versão do VRP denominada VRP com Distância restrita (DVRP, do inglês *Distance-constrained VRP*), em que o custo total a minimizar é composto por duas parcelas: o custo de deslocamento, como no VRP; e o custo de atendimento de cada cliente, d_i . Ainda, no DVRP o custo de cada viagem não pode ultrapassar um limite (não negativo) L (PRINS, 2004).

Muitos problemas derivam desta definição clássica do VRP, conforme restrições ou objetivos diversos são adicionados ao modelo matemático. Conforme a taxonomia proposta por (DESROCHERS *et al.*, 1999), é dentre os problemas de roteirização com múltiplos recursos que surgem restrições de sincronização, como as encontradas no presente trabalho. Exemplos de aplicações que contam com este tipo de sincronização incluem o problema de planejamento de colheita no contexto da produção agrícola (KARLSSON; RÖNNQVIST; BERGSTRÖM, 2004), de roteirização de tratores e reboques (DREXL, 2012; FRANCIS; ZHANG; SMILOWITZ, 2007), de planejamento de atendimento médico residencial (AIT HADDADENE; LABADIE; PRODHON, 2016; BERTELS; FAHLE, 2006; EN-NAHLI *et al.*, 2016; EVEBORN; FLISBERG; RÖNNQVIST, 2006; LIU, Ran; TAO; XIE, 2019; REDJEM; MARCON, 2016) e de geração de cronograma para equipes de prestação de serviços, como entrega e posterior instalação de móveis e eletrodomésticos (HOJABRI *et al.*, 2018), e de gestão de equipes técnicas em portos (LI; LIM; RODRIGUES, 2005), hospitais (RAHIMIAN; AKARTUNALI; LEVINE, 2016) e aeroportos (DOHN; KOLIND; CLAUSEN, 2009).

Problemas desta classe são denominados de **VRP com Múltiplas Restrições de Sincronização** (VRPMS, do inglês *Vehicle Routing Problem with Multiple Synchronization constraints*), no qual mais de um veículo pode ou deve ser utilizado para realizar uma atividade (DREXL, 2012). Essa característica tem uma consequência importante para os métodos de solução do problema: pelo fato de uma mesma atividade aparecer no roteiro de mais de um veículo (ou seja, o nó está presente em mais de uma viagem), uma alteração feita em uma rota pode afetar outras rotas e, inclusive, tornar a solução inviável (DREXL, 2012). Na literatura, esta característica é denominada de **interdependência entre rotas**.

Existem vários tipos de sincronização aplicáveis ao VRPMS. Drexl ⁽²⁰¹²⁾ faz uma revisão da literatura de VRPMSs, comentando detalhes da modelagem e métodos de solução adotados nas diversas publicações. Utilizando a variante do VRP como base, o autor classifica os diversos tipos de sincronização possíveis e aborda suas causas. Para cada tipo de sincronização, uma lista de trabalhos é apresentada e classificada quanto aplicação, objetos a sincronizar, abordagem de solução e tipo de variável adotada no modelo MIP. A classificação sugerida por Drexl (2012), com base em sua revisão da literatura, e adotada no presente trabalho é:

- Sincronização de tarefa: sendo **tarefa** uma atividade obrigatória que precisa ser contemplada na solução do problema, usando ou não unidades de algum recurso limitado, e **veículo** um objeto móvel autônomo ou não, cuja capacidade pode ser usada para execução da tarefa, define-se sincronização de tarefa como a restrição de que cada tarefa precisa ser executada exatamente uma vez por um ou mais veículos. Assim, qualquer solução viável precisa designar a tarefa a um ou mais veículos.
- Sincronização de operação: uma **operação** é algo que pode ou deve ser realizado por um veículo em um nó específico, para facilitar ou mesmo possibilitar a execução de uma tarefa. A sincronização de operação refere-se à simultaneidade temporal das operações de dois ou mais veículos, no mesmo local ou em locais diferentes. Por exemplo, em problemas de roteirização de veículos que permitem transbordo da carga sem armazéns intermediários, ou seja, que exigem a presença simultânea de dois veículos no mesmo ponto, há sincronização de operações. Este tipo de sincronização pode gerar o que se chama de **janela de tempo dinâmica**, que faz com que o início de uma operação dependa da execução de outra operação – no exemplo do transbordo de carga, o veículo que chega primeiro ao ponto em que acontecerá a transferência da carga só pode iniciar a operação depois que o(s) outro(s) veículo(s) também chega ao local. Além disso, a sincronização de operação pode ser puramente espacial (dois veículos precisam visitar um mesmo nó, mas não é preciso levar em conta os instantes em que isso acontece), com precedência (a operação desempenhada por um dos veículos precisa acontecer depois de outra operação desempenhada por outro veículo, com um intervalo de tempo mínimo ou máximo entre tais operações), ou exata (o intervalo de

tempo entre o início de duas operações executadas por veículos diferentes é fixo, podendo ser igual a zero caso as operações precisem ser simultâneas).

- Sincronização de movimento: este tipo de sincronia está presente em problemas que consideram veículos não-autônomos em sua formulação, pois este tipo de veículo só pode se mover quando acompanhado por um veículo autônomo. Assim, a sincronização de movimento significa que o veículo não autônomo só pode percorrer um arco da rede se, no mesmo intervalo de tempo, outro veículo autônomo compatível também percorrer o mesmo arco. Um veículo não autônomo pode ser uma caçamba (veículo com rodas não-motorizado que depende de veículo trator para se locomover) ou, quando os motoristas (autônomo ou não autônomo, a depender das restrições) são considerados na formulação do problema, os próprios veículos motores (por exemplo os ônibus de uma empresa de transporte urbano) são considerados não autônomos.
- Sincronização da carga: em cada nó, a diferença final entre a quantidade de carga descarregada por veículos e a quantidade carregada deve ser exatamente igual à demanda do nó, que em algumas classes de VRPs pode ter valor nulo ou negativo (nó com oferta de carga).
- Sincronização de recurso: quando este tipo de sincronização se aplica, a qualquer instante do horizonte de planejamento, a utilização total ou o consumo de um determinado recurso por cada um dos veículos é menor ou igual a um limite fixo. Por exemplo, no problema do transbordo da carga sem depósito intermediário é possível que em uma transferência a carga transferida para um veículo passivo (que só recebe carga) só passa ser proveniente de um único outro veículo ativo (que pode receber e ofertar carga), sendo a quantidade de veículos ativos escassa.

Assim como em problemas de *job scheduling*, também há variações do VRP com restrições temporais. Porém, ao invés de data de liberação e prazo, estas restrições normalmente aparecem no formato de **janelas de tempo** (*time windows*), que podem ser definidas da seguinte maneira: dado um intervalo $[a, b]$, com $a \leq b$, a tarefa associada a tal intervalo deve ser iniciada em algum instante de tempo t tal que $a \leq t \leq b$. Ao incluir este tipo de restrição ao VRP, obtém-se o Problema de Roteirização

de Veículos com Janelas de Tempo (VRPTW, do inglês *Vehicle Routing Problem with Time Windows*).

Ao incluir as janelas de tempo, restrições de sincronização e de precedência Bredström e Rönnqvist (2008) introduziram o Problema de Roteirização de Veículos com Janelas de Tempo e restrições de Sincronização e Precedência (PRVJT-SP ou VRPTW-SP, do inglês *Vehicle Routing Problem with Time Windows, Synchronization and Precedence constraints*, também denominado *Vehicle Routing Problem With Time Windows And Synchronization Constraints*, neste caso abreviado como VRPTWSyn) (AFIFI; DANG; MOUKRIM, 2013; AIT HADDADENE; LABADIE; PRODHON, 2016; EN-NAHLI *et al.*, 2016; LIU, Ran; TAO; XIE, 2019). Os autores partem da formulação clássica do problema de roteirização de veículos e a complementam com restrições de sincronização, apresentando um modelo matemático genérico, no sentido de que não se atém a uma aplicação específica. Tal modelo é NP-Hard porque engloba o VRPTW (que se sabe ser NP-Hard) como caso especial (AIT HADDADENE; LABADIE; PRODHON, 2016). Na formulação proposta, que usa uma representação em grafo, para garantir a visita de mais de um veículo nos casos em que isto é necessário, o nó que representa o pedido em questão foi duplicado e uma restrição de acoplamento, que força visitas “irmãs” a começarem ao mesmo tempo, foi adicionada ao modelo. O atendimento das restrições de precedência é resolvido na própria representação do problema, que não inclui arcos saindo de uma atividade a para outra atividade b , caso b tenha de ser concluída antes de a . Ainda, se houver necessidade de espaçar as visitas entre as atividades b e a de uma certa quantidade de tempo, então um tempo adicional ao deslocamento é atrelado ao arco que liga as duas atividades.

Bredström e Rönnqvist (2008) utilizam uma heurística baseada no método *local branching* para resolver o problema, usando uma abordagem de duas fases: utilizando variáveis auxiliares, primeiro é feita a designação de veículo para atender cada cliente e depois um método otimizador é usado para criar a sequência de atendimento e determinar as datas de início das tarefas. Os próprios autores desencorajam a utilização do método para resolver problemas em escala real e ressaltam a importância de investigar a aplicabilidade de métodos heurísticos para sua solução. Ainda, nas instâncias propostas pelos autores e utilizadas pelos demais trabalhos que tratam o VRPTW-SP, apenas cerca de 10% das tarefas exigem a atuação simultânea de mais de um veículo.

Ait Haddadene, Labadie e Prodhon (2016), Afifi, Dang e Moukrim (2013) e En-Nahli et al. (2016) aplicam o VRPTW-SP ao problema de programação de atividades de *Home Health Care*, em que pacientes recebem serviços médicos em suas residências. Neste contexto, os veículos representam os cuidadores profissionais e os pontos de demanda são serviços associados a clientes. Então, além de associar um cuidador a cada uma das tarefas, levando em conta as habilidades específicas dos profissionais e seu horário de trabalho, que é modelado como um recurso finito, deseja-se determinar a ordem em que as visitas serão realizadas e o momento em que cada serviço será iniciado.

Afifi, Dang e Moukrim (2013) consideram apenas a minimização dos custos (ou tempo) de locomoção e aplicam a meta-heurística *Simulated Annealing* (SA) na sua solução. Nas instâncias testadas, o método obtém melhores resultados do que os reportados na literatura até então, mas não há comparação com um valor de LB, exceto nos casos em que a solução ótima é conhecida.

En-Nahli et al. (2016) também consideram apenas a minimização do tempo de locomoção e propõem uma heurística híbrida, que combina as meta-heurísticas *Variable Neighborhood Descent* (VND) e *Iterated Local Search* (ILS), partindo de uma solução inicial obtida com uso de uma heurística construtiva baseada no movimento de inserção de tarefa nas rotas em construção. Apesar de obter bons resultados nas instâncias menores de Bredström e Rönnqvist (2008), os autores concluem que o método precisa ser aprimorado antes de ser aplicável a problemas de escala real, ou mesmo nas instâncias maiores propostas por Bredström e Rönnqvist (2008).

Ait Haddadene, Labadie e Prodhon (2016) propõem um modelo de Programação Inteira Mista, ou MIP - *Mixed Integer (Linear) Programming*, cujo objetivo é minimizar o custo total de atendimento a todos os pontos (medido pelos custos de locomoção entre clientes) e maximizar a satisfação dos clientes, levando em conta eventuais preferências em ser atendidos por determinado cuidador. Para sua solução, os autores apresentam uma meta-heurística híbrida, que combina *Greedy Randomized Adaptive Search Procedure* (GRASP) com *Iterated Local Search* (ILS): assim como a GRASP, cada iteração parte de uma solução gerada por procedimento guloso randomizado e em seguida aplica melhorias nessa solução; no algoritmo proposto, estas melhorias são feitas pela aplicação da ILS, que primeiro perturba a solução inicial (gerada pela GRASP) e depois faz uma busca local aplicando uma sequência de estruturas de vizinhança (os autores testaram duas estratégias na busca local, uma randomizada e outra baseada na VND). Ainda, a busca local executada a cada iteração computa

parâmetros que permitem determinar se o movimento em questão tem potencial para gerar filhos viáveis, graças a testes de viabilidade que consideram tanto as janelas de tempo como as restrições de sincronização. Os testes realizados mostram que soluções geradas podem não ter boa qualidade em ambos os aspectos da função objetivo.

2.3.2 Problemas de roteirização com coleta e entrega

Entre os problemas de roteirização de veículos, há um subconjunto de problemas que trata da roteirização com coleta e entrega de carga ou de passageiros, ou *Pick-up and Delivery Vehicle Routing Problems* (PDVRP), na terminologia em inglês. Neste caso, para atender a demanda, o veículo deverá se dirigir ao local de início de cada viagem solicitada; coletar a carga ou o passageiro, consumindo assim parte de sua capacidade de transporte; e seguir para o ponto de destino, onde a carga ou passageiro será entregue, liberando a capacidade de transporte que havia sido bloqueada. Entre a coleta e entrega de um produto (ou embarque e desembarque de uma pessoa), o veículo pode visitar outros pontos de coleta (ou embarque), desde que sua capacidade de transporte seja respeitada. Aplicações deste tipo de problema incluem o planejamento do serviço público de transporte de passageiros com dificuldade de locomoção, como o Atende+, prestado pela cidade de São Paulo, ou a roteirização de motoristas para realização de transporte de pequenas encomendas, hoje muito comum em grandes centros urbanos.

Do ponto de vista da modelagem matemática, no PDVRP a coleta e entrega de produtos pode ser modelada incluindo nós com demanda positiva e negativa, de forma que pontos de coleta possuem valores positivos de consumo de capacidade e os de entrega, negativos; e com acréscimo de restrições para exigir que o mesmo veículo atenda os nós que se referem à mesma viagem (ou seja, ao par origem-destino da mesma solicitação de transporte) e que a visita ao nó de origem preceda a visita ao nó de destino. Usando a terminologia de programação de tarefas, o PDVRP é um VRP com restrições de precedência entre serviços e a exigência que tais serviços precisem ser atendidos pelo mesmo recurso – no contexto de *scheduling*, é como se uma mesma máquina precisasse processar duas tarefas que tenham restrição de precedência entre si. Esta relação entre o PDVRP, os problemas de programação de tarefas e os VRPs com sincronização de recursos é abordada por Bredström e Rönnqvist (2008), que

também fornecem exemplos de aplicações reais que combinam características destas três classes de problemas.

A literatura também trata de casos do PDVRP em que há restrições temporais atreladas à demanda, seja pela exigência de cumprir uma janela de tempo para a coleta ou entrega, seja restringindo o tempo entre as duas operações. Para aplicações de transporte de passageiros, por exemplo, além de haver janelas para embarque e desembarque, é desejável que o usuário do serviço não passe um tempo exageradamente grande dentro do veículo, a fim de garantir um nível de serviço adequado. No modelo matemático, isto é garantido por uma restrição que limita a diferença temporal entre as visitas de dois nós associados ao local de coleta e entrega de uma mesma viagem. No caso do *Pick-up and Delivery Vehicle Routing Problem with Time Windows* (PDVRPTW), como aqui se denomina o PDVRP com acréscimo de janelas de tempo para o atendimento dos pedidos, esta diferença será menor ou igual a uma constante não negativa. Ressalta-se que bastaria inferir que tal diferença fosse igual a zero e retirar a restrição que exige que um mesmo veículo atenda a origem e destino da mesma viagem, para que tal restrição passe a ser de sincronização de operações, como no caso do problema desta pesquisa. Há outras restrições, como de consumo de capacidade dos veículos, que não são aplicáveis à roteirização de rebocadores, e que distinguem as duas famílias de problemas. Exemplos de modelos matemáticos são encontrados no trabalhos de Bredström e Rönnqvist (2008).

Rousseau, Gendreau e Pesant ⁽²⁰¹³⁾ abordam uma variante do PDVRPTW, que denominam *Synchronized Dynamic Vehicle Dispatching Problem* (SDVDP), ou Problema Dinâmico de Despacho de Veículos traduzindo literalmente, em que veículos devem ser roteirizados para atender um conjunto de pedidos de coleta de carga em clientes espalhados geograficamente. Os pedidos possuem uma janela de tempo rígida e alguns deles devem ser executados por dois tipos de veículo: um regular e um especial. Veículos regulares representam aqueles que carregam as mercadorias coletadas, por isso possuem capacidade limitada; enquanto os especiais representam os que levam equipes de suporte para realizar serviços antes ou depois da coleta. A janela de tempo que limita a chegada do veículo de serviço para atender um cliente especial (cliente que requer veículo regular e de serviço) é definida relativamente à chegada do veículo de coleta, diferentemente do problema abordado nesta pesquisa. Além disso, no máximo um veículo de cada tipo é demandado. O termo "dinâmico" faz referência ao fato de que nem todos os pedidos de coleta de carga são conhecidos no

início do horizonte de planejamento, ou seja, no momento da primeira otimização. Ao contrário, após o início da execução dos roteiros, surgem novos pedidos para coleta de carga, e o modelo de otimização deve ser capaz de tentar acomodar tais pedidos, inserindo as respectivas visitas nas rotas já existentes, sem alterar a ordem das demais tarefas nem a designação dos veículos que as atenderão. Como a frota disponível é limitada e as janelas de tempo são rígidas, a modelagem permite que os pedidos de coleta de carga sejam negados, caso não seja possível encaixá-los nas rotas.

É proposto um conjunto de instâncias para compor o benchmark do problema, derivadas de instâncias da literatura. Nas instâncias propostas, o percentual de clientes especiais (que exigem sincronização de operações) é 10%. Uma das análises conduzidas pelos autores comparou os resultados da heurística adotando percentuais de 5, 25 e 50% de clientes especiais para as instâncias propostas, contudo o número de veículos especiais disponíveis também foi alterado nesses casos, pois foi definido em função (direta) deste percentual.

Os autores utilizam uma heurística híbrida para a solução do problema, combinando Programação por Restrições com duas abordagens de busca local: uma que adota a primeira possibilidade de inserção da tarefa que gere solução viável (*first insertion*) e outra que verifica a melhor possibilidade de inserção da tarefa encontrada dentro de um limite de tempo (*best insertion*). O método é dinâmico, devido à própria natureza do problema. No início da execução e toda vez que surge um pedido novo, um controlador do processo de inserção iterativa aciona o operador de inserção. Este operador pega a última solução do modelo matemático, ou seja, que contempla os clientes já alocados em rotas; constrói o modelo de inserção do novo pedido, usando CP; e resolve o modelo. Caso a inserção possa ser feita, a solução é passada para o controlador de inserções e, caso não tenha chegado um novo pedido de inserção de cliente, o controlador envia a solução corrente para o método de busca local. A busca é executada até que um critério de parada seja atingido ou um novo pedido de inserção chegue ao controlador.

A busca local utiliza operadores tradicionais de problemas de roteirização, a saber: *2-Opt*, *Or-Opt*, *Cross*, *Exchange* e *Relocate*. Duas meta-heurísticas são testadas para guiar a busca, uma estratégia simples de "*descent*" e a *Guided Local Search* (GLS), que penaliza certos movimentos que tendem a revisitar áreas da região já exploradas, de maneira similar ao que é feito pela Busca Tabu. Para a simples *descent strategy*, não fica claro quando nem como é feita a escolha do movimento, se é a toda iteração

ou só no início da busca, se é aleatório ou segue uma ordem específica. Os resultados mostraram que a GLS com a estratégia de "*best insertion*" produziram melhores resultados, de maneira geral. Um ponto interessante do método de solução proposto é que o operador de inserção e o método de busca local não precisam ser alterados para comportar as restrições de sincronização e as janelas de tempo dinâmicas (definidas em função da data de chegada do veículo capacitado), pois esta parte da heurística é feita pelo modelo de Programação por Restrições.

Em suma, a literatura de PDVRPTW foi consultada com intuito de verificar as semelhanças com o problema aqui tratado e, com base nelas, avaliar a aplicabilidade de abordagens e estratégias de modelagem para a roteirização de rebocadores. É interessante observar que algumas publicações do VRP com restrições de sincronização parecem se basear no PDVRPTW, ao criar nós fictícios para representar a visita de mais de um recurso ao mesmo nó, como exemplificado na seção 2.5.

Dentre os artigos consultados, destaca-se o trabalho de Furtado, Munari e Morabito (2017), que apresenta uma formulação mais compacta para o PDVRPTW, em que o índice do veículo não aparece na indexação da variável de decisão de fluxo, mas sim no índice do nó inicial da rota, que é guardado por uma variável auxiliar. Os autores documentaram uma diminuição significativa no tamanho do problemas modelado, e por isto o trabalho serviu de base para um dos modelos matemáticos desenvolvidos nesta tese.

2.3.3 Problemas de roteirização de veículos com restrições de sincronização

Ait Haddadene, Labadie e Prodhon (2016) apresentam uma nova variante do PRV aplicável ao contexto de prestação de cuidados médicos a domicílio, ou *Home Health Care* em inglês, em que são consideradas simultaneamente restrições de Janela de Tempo (JT), de Simultaneidade (veículos que precisam estar no mesmo ponto de demanda ao mesmo tempo) e de Precedência entre as visitas programadas (P), que aqui será denominada abreviadamente por PRVJT-SP.

Nesta aplicação, os veículos representam os cuidadores (enfermeiros e afins) e os pontos de demanda são serviços associados a clientes. Assim, além de associar um cuidador a cada uma das tarefas, levando em conta as habilidades específicas dos cuidadores e seu horário de trabalho, que é um recurso finito, deseja-se determinar a ordem em que as visitas serão realizadas e o momento em que cada serviço é iniciado.

O modelo MIP desenvolvido tem como objetivo minimizar o custo do atendimento a todos os pontos, medido pelos custos de locomoção entre clientes, e maximizar a satisfação dos pacientes, levando em conta suas preferências em ser atendidos por determinado recurso. Tal modelo é NP-Hard porque engloba o PRVJT, que se sabe ser NP-Hard, como caso especial. Para sua solução, os autores desenvolvem e testam uma meta-heurística híbrida, que combina a GRASP com a ILS. Os testes realizados mostram que soluções geradas podem não ter boa qualidade em ambos os aspectos da função objetivo. Os autores argumentam que esta estratégia de modelagem chega a um modelo significativamente menor do que a tradicional, que usa nós especiais (também chamados de fictícios ou virtuais) para modelar a sincronização de serviços. Hojabri et al. (2018) apresentam o PRV com múltiplas restrições de sincronização, ou *Vehicle Routing Problem with Multiple Synchronization Constraints* (VRPMS), em que alguns clientes precisam ser visitados por dois veículos, com uma janela de tempo para a realização das tarefas. A motivação para sua modelagem veio de um problema real de roteirização de veículos para grandes lojas que, além de vender e entregar artigos para seus clientes, também vendem o serviço de instalação de alguns destes produtos. Como parte das entregas envolve apenas a descarga da mercadoria comprada, há dois tipos de veículos considerados na modelagem do problema: os que carregam os produtos e realizam as entregas, e os que transportam as equipes de instalação. Para cada entrega existe uma janela de atendimento acordada com o cliente, enquanto a janela de instalação claramente está associada ao momento da entrega do produto, podendo iniciar-se um pouco antes ou logo depois da entrega, a depender do tipo de produto.

O problema é modelado usando um grafo, definido pelo conjunto de vértices que representam os clientes e arcos que representam o deslocamento entre pares de clientes. Os autores dividem os clientes entre aqueles que compraram serviço de instalação, os chamados clientes especiais, e os que não compraram, ou clientes regulares. Enquanto os regulares são representados por um único vértice, associado a uma demanda (em unidades que consomem a capacidade do veículo), uma janela de tempo e uma duração (descarga e entrega do produto), os clientes especiais possuem um vértice regular (idêntico ao dos demais clientes) e um vértice especial, que deve ser visitado por um veículo de serviço dentro de uma janela de tempo vinculada à visita do veículo de entrega, e associado a um tempo de serviço. Os veículos de entrega

possuem capacidade limitada enquanto os de serviço têm capacidade ignorada no modelo.

Dentre as diferenças entre o problema abordado por Hojabri *et al.* (2018) e o problema estudado nesta pesquisa, destacam-se:

- apesar de haver dois tipos de veículo, um nunca pode substituir o outro;
- o veículo de entrega é liberado assim que sua tarefa é concluída, independente da equipe do veículo de serviço já ter finalizado a instação, ou mesmo chegado ao local;
- cada cliente especial exige exatamente um veículo de cada tipo;
- não há o conceito de atraso neste problema, portanto a função objetivo considera apenas a distância total percorrida pelos veículos designados para atender a demanda.

Parragh e Doerner (2018) abordam dois problemas similares, o Problema de Roteirização de Veículos com Janela de Tempo e Sincronização de pares (PRVJTS) e o de Roteirização e Programação de Técnicos de Serviço com Sincronização por Pares (tradução literal de *Service Technician Routing and Scheduling Problem with pairwise Synchronization*), dando exemplos de diversas aplicações reais para cada um, como atendimento médico domiciliar, programação de cronograma de atividades de uma equipe de funcionários, designação de equipe de bordo para atender os vôos de uma empresa aérea, entrega e bombeamento de mistura de concreto para obras, snow plowing, definição de cronograma e roteirização de navios de carga, entre outros.

Os modelos MIP dos dois problemas se diferem por um conjunto de restrições e alguns parâmetros, por isso a estratégia de modelagem e solução é a mesma: os autores propõem a criação de um vértice adicional para cada tarefa que demande dois recursos sincronizados, de forma que cada nó do grafo seja visitado por no máximo um veículo da frota. Como a própria nomenclatura do problema indica, no máximo 2 recursos são demandados por tarefa e uma tarefa pode não ser executada, apesar de haver penalização na função objetivo por cada nó não visitado. Os autores usam a ALNS para resolver os dois modelos, apresentam um algoritmo similar ao proposto por Kovacs *et al.* (2012), que combina a metaheurística de Busca Local Adaptativa em Grande Vizinhaça (abreviada como ALNS, do inglês *Adaptative Large Neighborhood Search*) com Programação por Restrições (CP, *Constraint Programming*).

Liu, Tao e Xie (2019) tratam do PRVJT-S em que um conjunto de recursos (ou frota de veículos), disponível inicialmente em um depósito central, deve atender serviços demandados por um conjunto de clientes espalhados geograficamente, sendo que cada cliente só pode receber o veículo dentro de uma janela de tempo pré-especificada e alguns serviços demandam mais de um veículo simultaneamente. Os autores reforçam que tal problema é uma extensão do PRVJT, que é sabidamente NP-hard, o que o torna um bom candidato para uso de heurísticas. Contudo, eles também argumentam que as restrições de sincronização do PRVJT-S tornam a aplicação de heurísticas baseadas em busca em vizinhança extremamente ineficiente, já que toda vez que um cliente é inserido, possivelmente é necessário recalcular as datas de início de todas as atividades já programadas. Os autores propõem um método de solução baseado na *Adaptative Large Neighbourhood Search* (ALNS) que obtém um desempenho impressionante nas instâncias testadas. Os autores ressaltam que criar uma heurística para esta variante de PRVJT-S não é um processo fácil.

2.4 DOS MODELOS MATEMÁTICOS PARA PROBLEMAS DE ROTEIRIZAÇÃO COM RESTRIÇÕES DE SINCRONIZAÇÃO OU DE PRECEDÊNCIA E JANELAS DE TEMPO

Durante o processo de revisão da literatura, foram encontrados diversos trabalhos tratando problemas semelhantes ao desta pesquisa. Embora nenhum dos modelos propostos possa ser aplicado diretamente à roteirização de rebocadores AHTS, por conta das especificidades de tal aplicação, as estratégias de modelagem da literatura são úteis à formulação de seu modelo matemático.

A seguir são apresentados os modelos que embasaram as formulações proposta no capítulo 3. Neste texto, foi mantida a nomenclatura de conjuntos, parâmetros e variáveis adotadas pelos autores, para minimizar erros de digitação. Além disso, os termos veículos e recursos foram adotados mesmo para problemas que tratam de recursos humanos, como equipes de enfermeiros e médicos.

2.4.1 Formulações com nós fictícios para representar visitas sincronizadas a um mesmo local ou execução simultânea da mesma tarefa

Parragh e Doerner (2018) apresentam dois modelos similares para os dois problemas estudados em seu artigo, o PRVJT-S com restrições de precedência entre tarefas (PRVJT-SP) e o de roteirização de técnicos para prestação serviço a domicílio, com

sincronização por pares (STRSP). Ambos os modelos são representados em grafo direcional $G = (V, A)$, utilizando os seguintes parâmetros:

- Ao conjunto de n tarefas demandadas, é acrescido um conjunto de S' tarefas fictícias, que contém um nó adicional para cada tarefa com restrição de sincronização (no máximo dois recursos são demandados pelo mesmo cliente). Os pares de tarefas sincronizadas compõem o conjunto S .
- $V' = \{1, 2, \dots, n\} \cup S'$ é o conjunto de vértices que representam a execução de serviços nos locais onde são demandados, sendo cada $i \in V'$ associado a uma janela de tempo $[e_i, l_i]$, uma duração de serviço s_i e uma multa o_i caso o serviço não seja atendido pela solução. Dependendo do problema em questão (PRVJT-SP ou STRSP), cada vértice $i \in V'$ possui uma demanda d_i ou um vetor $Q_i = \{q_{i1}, q_{i2}, \dots, q_{i|Q|}\}$ que especifica o nível de qualificação necessário em cada habilidade $f \in Q$, q_{if} .
- K é o conjunto de m veículos, de capacidade C , disponíveis para execução das tarefas. Para cada recurso $k \in K = \{1, 2, \dots, m\}$ há um vetor que define suas habilidades, composto por parâmetros p_f^k , cujo valor equivale ao nível de qualificação do veículo k na habilidade f .
- Para cada veículo k é criado um nó que representa o início de sua rota e outro que representa o fim, sempre se referindo à mesma localidade geográfica (o depósito de veículos). Assim, são criados os conjuntos de nós de origem $D_o = \{n + 1, n + 2, \dots, n + m\}$ e de destino $D_d = \{n + m + 1, n + m + 2, \dots, n + 2m\}$, e a união de ambos $D = D_o \cup D_d$.
- O conjunto completo de vértices do grafo denotado por V e definido como $V = V' \cup D$.
- $A = \{(i, j) | i \in V \setminus D_d, j \in V \setminus D_o, i \neq j\}$ é o conjunto de arcos do grafo, sendo cada arco (i, j) associado a um tempo t_{ij} e a um custo c_{ij} .

As variáveis de decisão são:

- x_{ij}^k é uma binária que indica se o nó $(i, j) \in A$ faz parte da rota k
- B_i é uma variável contínua que indica o início da atividade $i \in V'$

A função objetivo é minimizar o custo total das viagens e o número total de clientes não atendidos, pela minimização da multa associada aos nós negligenciados pela solução:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} x_{ij}^k c_{ij} + \sum_{i \in V'} o_i \left(1 - \sum_{k \in K} \sum_{j | (i,j) \in A} x_{ij}^k \right) \quad \text{Eq. 2-9}$$

Sujeito a:

- Apenas um veículo k pode atender um nó que represente a realização de uma tarefa:

$$\sum_k \sum_{j | (i,j) \in A} x_{ij}^k \leq 1 \forall i \in V' \quad \text{Eq. 2-10}$$

- A rota de cada veículo deverá ser contínua, então se um veículo chega a um cliente, ele precisa sair do mesmo. Similarmente, ele só pode sair do nó se tiver chegado nele, vindo de outro nó da rede (que pode ser outro cliente ou pode ser nó de origem da rota):

$$\sum_{i | (i,j) \in A} x_{ij}^k - \sum_{i | (i,j) \in A} x_{ji}^k = 0 \forall j \in V', k \in K \quad \text{Eq. 2-11}$$

- Todas as rotas precisam ser iniciadas e finalizadas no depósito de recursos. Assim, exatamente um arco saindo do nó de origem e um arco entrando no nó de destino deverá ser selecionado – e pode ser um único arco:

$$\sum_{j | (n+k,j) \in A} x_{n+k,j}^k = 1 \forall k \in K$$

$$\sum_{i | (i,n+m+k) \in A} x_{i,n+m+k}^k = 1 \forall k \in K$$

- As capacidades dos recursos precisam ser respeitadas ou, para o problema de roteirização de prestadores de serviços técnicos, o nível de experiência em cada habilidade requerida precisa ser atendido:

$$\sum_i d_i \sum_j x_{ij}^k \leq C \forall k \in K$$

$$q_{if} \sum_{j | (i,j) \in A} x_{ij}^k \leq p_f^k \forall i \in V', k \in K, f \in Q$$

Os autores comentam que a restrição anterior poderia ter sido modelada diferentemente, substituindo os parâmetros relativos às habilidades requeridas pelo cliente e possuídas pelos recursos pela definição de compatibilidade entre cada recurso k e cada serviço i . Neste caso, haveria uma restrição garantindo que sempre que o recurso k for incompatível com o serviço i , a variável x_{ij}^k valeria zero.

- A data de início de uma atividade precisa levar em conta o tempo despendido por todas as atividades atendidas pelos veículos a ela alocados, incluindo os tempos de deslocamento:

$$B_j \geq (B_i + s_i + t_{ij})x_{ij}^k \forall (i, j) \in A, k \in K$$

- A data de início de cada atividade tem que estar dentro de sua janela de atendimento:

$$e_i \leq B_i \leq l_i \forall i \in V$$

- Uma tarefa que demande mais de um recurso só pode ser iniciada quando ambos estiverem disponíveis, portanto os nós que referem-se à mesma atividade precisam ter a mesma data de início:

$$B_i = B_j \forall (i, j) \in S$$

- Domínio das variáveis:

$$x_{ij}^k \in \{0, 1\}$$

$$B_i \in \mathbb{R}$$

Parece haver um problema na formulação dos autores ao não exigir que se um dos nós de uma tarefa com restrição de sincronização é atendido, o outro também o seja. Supondo que apenas um dos nós pertença a uma rota, apesar da restrição $B_i = B_j$ forçar que um valor seja atribuído à data de início do outro nó, a restrição $B_j \geq (B_i + s_i + t_{ij})x_{ij}^k$ permite que o valor de x_{ij}^k seja zero para todos os veículos $k \in K$. Ademais, a restrição $B_j \geq (B_i + s_i + t_{ij})x_{ij}^k$ pode ser substituída por $B_j \geq B_i + s_i + t_{ij} + (x_{ij}^k - 1)M$ (sendo M uma constante de valor grande suficiente) para tornar o modelo um MILP, mas o problema citado persistiria. Para contornar essa questão, pode-se incluir a restrição:

$$\sum_k \sum_{i \in V} x_{ij}^k = \sum_k \sum_{i \in V} x_{ih}^k \quad \forall (j, h) \in S$$

Outra formulação que utiliza a mesma estratégia de modelagem, isto é, com acréscimo de nós fictícios para representar visitas sincronizadas de mais de um recurso, é a proposta por de Liu, Tao e Xie (2019), também definida no grafo $G = (V, A)$, usando os seguintes conjuntos e parâmetros:

- K é o conjunto de veículos, sendo que os veículos possuem um turno máximo H para excutar todas as atividades que lhes forem atribuídas. Se qualquer veículo $k \in K$ for utilizado na solução, há um custo fixo ξ_k associado.

- N é o conjunto dos nós que representam todos os clientes, composto pela união dos seguintes subconjuntos: $N_1 = \{1, 2, \dots, n_1\}$ e $N_2 = \{n_1 + 1, n_1 + 2, \dots, n_1 + n_2\}$ conjuntos de nós representando clientes que necessitam (N_2) ou não necessitam (N_1) sincronização de operações.
- Nesta formulação, cada vértice do grafo só poderá ser visita uma vez, ou seja, por apenas um veículo. Então, para representar a visita de outros veículos a um mesmo cliente, são criados tantos nós fictícios quando necessários para cada nó $i \in N_2$. Por exemplo, se o cliente $i = n_1 + 1$ exige a operação sincronizada de 3 veículos, são gerados dois nós fictícios i' e i'' , com $t_{ii'} = t_{ii''} = t_{i'i''} = t_{i'i} = t_{i''i} = t_{i''i'} = \infty$ e janelas de tempo e durações idênticas aos parâmetros do nó original. O conjunto de nós fictícios associados a os clientes com restrição de sincronização é denotado por N'_2 . Para facilitar a notação, também são criados os conjuntos $N' = N \cup N'_2$, $V' = V \cup N'_2$ e P^{sync} , este último contendo todos os pares $(i, j) | i \in N', j \in N', i \neq j$ de vértices que se referem ao mesmo cliente.
- $V = \{0, n_1 + n_2 + 1\} \cup N$ é o conjunto de vertices do grafo, formado pela união de N com dois nós fictícios, que representam o início e fim de todas as rotas, definidos a fim de evitar subciclos na solução ou para de fato representar o depósito inicial e final dos recursos. Cada cliente $i \in N$ é associado a um tempo de serviço τ_i e a uma janela de tempo $[a_i, b_i]$ rígida, dentro da qual o serviço precisa ser iniciado. Para os nós fictícios de V , define-se que $[a_0, b_0] = [a_{n_1+n_2+1}, b_{n_1+n_2+1}]$ tal que $b_0 - a_0 = H$; e $\tau_0 = \tau_{n_1+n_2+1} = 0$.
- A é o conjunto de arcos ligando os pares de vértices do grafo, sendo cada arco associado a um tempo (custo) de deslocamento t_{ij} , que vale 0 para o arco ligando os nós fictícios (t_{0, n_1+n_2+1}) e infinito para t_{ii} .

Variáveis de decisão:

- x_{ijk} é uma binária que indica se o veículo k atende o cliente i e, em seguida, viaja até o cliente j para também atendê-lo.
- u_i é a data de início do serviço executado no nó $i \in V$

A função objetivo é minimizar o custo total de transporte, dado pela soma dos custos fixos e variáveis de cada veículo da frota:

$$\sum_{i \in V' \setminus \{n_1+n_2+1\}} \sum_{j \in V' \setminus \{0\}} \sum_{k \in K} x_{ijk} t_{ij} + \sum_{j \in N'} \sum_{k \in K} \xi_k x_{0j}$$

Sujeito a:

- A todo nó que represente visita a cliente, deve haver exatamente uma visita atribuída: apenas um veículo o visita e apenas uma vez

$$\sum_{k \in K} \sum_{j \in V'} x_{ijk} = 1 \forall i \in N'$$

- As rotas devem ser contínuas, sem formação de subciclo, o que é garantido pelo balanço nos nós que representam clientes e pelas restrições de início e fim nos depósitos inicial e final, respectivamente

$$\begin{aligned} \sum_{i \in V'} x_{ijk} &= \sum_{i \in V'} x_{jik} \quad \forall j \in N', k \in K \\ \sum_{j \in V'} x_{0jk} &= 1 \text{ e } \sum_{i \in V'} x_{i, n_1+n_2+1, k} = 1 \quad \forall k \in K \end{aligned}$$

- O instante de início de um serviço no cliente pertencente a determinada rota é maior ou igual ao instante de início do serviço imediatamente anterior

$$u_j \geq u_i + \tau_i + t_{ij} + M(x_{ijk} - 1) \quad \forall i, j \in V' | (i, j) \in A, k \in K$$

- A janela de tempo de cada serviço deve ser obedecida

$$a_i \leq u_i \leq b_i \quad \forall i \in V'$$

- O início de um serviço que possua restrição de sincronização só pode se dar após a chegada de todos os recursos (veículos) necessários

$$u_i = u_j \quad \forall (i, j) \in P^{sync}$$

- Domínio da variável binária

$$x_{ijk} \in \{0, 1\} \quad \forall i, j \in V', k \in K$$

2.4.2 Formulação sem nós adicionais para tarefas sincronizadas

Ait Haddadene, Labadie e Prodhon (2016) apresentam um modelo de Programação Inteira Linear Mista (ou MILP na sigla em inglês, de *Mixed Integer Linear Program*), definido no grafo não direcional $G = (V, E)$, para o VRPTWSyn. A formulação assume que são conhecidos os seguintes parâmetros e conjuntos:

- S : conjunto de s tipos de serviços demandados pelos clientes;
- K : conjunto finito de veículos (que nesta aplicação representam os cuidadores) disponíveis para atender as demandas dos clientes, indexados por k , sendo que cada veículo só pode operar dentro de sua janela de tempo $[\alpha_k, \beta_k]$. **Cada veículo desempenha apenas um tipo de serviço**, o que é modelado com o auxílio de um conjunto de s parâmetros $o_{ks'}$, sendo $o_{ks'} = 1$ caso s' seja o

serviço compatível com o veículo em questão. Ainda, os autores denotam como $K_{s'}$, o subconjunto de veículos capazes de executar o serviço s' .

- N : conjunto de n nós que representam os clientes, indexados por i . Cada cliente demanda um conjunto de serviços $S_i = \{m_{i1}, m_{i2} \dots m_{is}\}$, com $m_{is'} = 1$ para os serviços demandados e pelo cliente i e $m_{is'} = 0$ caso o serviço em questão não seja necessário. Todas as atividades precisam ser iniciadas dentro da janela de tempo do cliente $[a_i, b_i]$. Para cada serviço demandado por cada cliente, há uma duração associada D_{is} . Além disso, para cada par de serviços u e r demandados por um cliente i , é necessário respeitar um intervalo entre as duas atividades, denotado por $gap_{iur} = gap_{iru}$. Caso as atividades precisem ser sincronizadas, tem-se $gap_{iur} = 0$.
- $L = \{d, f\}$ conjunto com os dois nós de início e fim das rotas – cada veículo k só pode sair do nó d a partir da data α_k e precisa chegar em f até β_k ;
- $V = N \cup L$ conjunto com todos os nós definidos acima;
- Para cada par de cliente i e veículo k , é definido um parâmetro que representa o quanto o cliente gostaria de ser atendido por aquele veículo (cuidador/enfermeiro), $Pref_{ik} \in R$, sendo que quanto menor seu valor, maior a preferência;
- E : conjunto de arcos ligando todos os pares de nós $(i, j) | i \in N, j \in N, i \neq j$, sendo cada arco associado a um custo c_{ij} e um tempo de deslocamento T_{ij} ;

Variáveis de decisão:

- x_{ijk} : binária que indica se o arco (i, j) é percorrido pelo veículo k , ou seja, se o veículo é designado para ambos os clientes e atende j logo após finalizar o atendimento de i .
- t_{ik} : instante de início da execução de um serviço no cliente i pelo veículo k

Como função objetivo, os autores adotam a minimização do custo variável de transporte e da penalidade total por alocar recursos preteridos pelos clientes para atendê-los:

$$\min \sum_{i \in V \setminus \{f\}} \sum_{j \in V \setminus \{d\}} \sum_{k \in K} c_{ij} x_{ijk} + \sum_{i \in N} \sum_{j \in V \setminus \{d\}} \sum_{k \in K} Pref_{ik} x_{ijk} \quad \text{Eq. 2-12}$$

Tal que:

- Para cada veículo, deverá ser selecionado exatamente um arco ligando o nó inicial a qualquer outro nó de V e um arco ligando qualquer nó de V ao nó final:

$$\sum_{j \in N} x_{djk} = 1 \quad \forall k \in K \quad \text{Eq. 2-13}$$

$$\sum_{i \in N} x_{ifk} = 1 \quad \forall k \in K \quad \text{Eq. 2-14}$$

Nota-se que os autores optaram por forçar que o veículo seja utilizado. Caso não seja este o caso, basta definir que as somatórias são feitas em $j \in V \setminus \{d\}$ e $i \in V \setminus \{f\}$

- Para garantir a continuidade das rotas, além do primeiro conjunto de restrições define-se o equilíbrio nos nós (número de arcos que entra é igual ao que sai):

$$\sum_{i \in V \setminus \{f\}} x_{ijk} = \sum_{i \in V \setminus \{d\}} x_{jik} \in N \quad \forall j \in N, k \in K \quad \text{Eq. 2-15}$$

- É preciso selecionar um veículo compatível com cada um dos serviços demandados por cada cliente:

$$\sum_{j \in V \setminus \{d\}} \sum_{k \in K_{s'}} x_{ijk} = m_{is'} \quad \forall i \in N, s' \in S \quad \text{Eq. 2-16}$$

- Não pode haver sobreposição temporal na execução de tarefas atendidas pelo mesmo veículo. Assim, antes de iniciar o atendimento de um cliente, é necessário finalizar o atendimento do anterior e se deslocar até o cliente em questão:

$$t_{jk} \geq t_{ik} + (T_{ij} + D_{is'})x_{ijk} - b_i(1 - x_{ijk}) \\ \forall i, j \in V, s' \in S: m_{is'} + m_{js'} = 2, k \in K_s, \quad \text{Eq. 2-17}$$

Aqui os autores aproveitam para só definir a restrição para o conjunto de variáveis que fizer sentido, ignorando tarefas que são incompatíveis com cada veículo. É interessante a forma como os autores evitam a definição do tradicional *big M*, uma constante grande o suficiente para relaxar a restrição quando $x_{ijk} = 0$, a substituindo pelo limitador da janela de tempo da tarefa i , uma vez que este valor será necessariamente maior ou igual a t_{ik} , tornando o lado direito da inequação nulo ou negativo.

- O atendimento de cada cliente está restrito à sua janela de tempo:

$$a_i \sum_{j \in N} x_{ijk} \leq t_{ik} \leq b_i \sum_{j \in N} x_{ijk} \quad \forall i, j \in V, s' \in S: m_{is'} + m_{js'} = 2, k \in K_s, \quad \text{Eq. 2-18}$$

Os autores utilizam o somatório de x_{ijk} em j para deixar livre o instante de início de clientes não atendidos.

- A janela de operação (ou expediente) dos veículos deve ser respeitada:

$$\alpha_k \leq t_{dk} \leq \beta_k \quad \forall k \in K \quad \text{Eq. 2-19}$$

$$\alpha_k \leq t_{fk} \leq \beta_k \quad \forall k \in K \quad \text{Eq. 2-20}$$

- O intervalo entre o início de dois serviços demandados pelo mesmo cliente precisa ser atendido:

$$\sum_{k \in K_r} t_{ik} - \sum_{k \in K_{s'}} t_{ik} \leq \text{gap}_{is'r} \quad \forall i \in N, s', r \in S: m_{ir} + m_{is'} = 2 \text{ \& } r \neq s' \quad \text{Eq. 2-21}$$

$$\sum_{k \in K_r} t_{ik} - \sum_{k \in K_{s'}} t_{ik} \geq -\text{gap}_{is'r} \quad \forall i \in N, s', r \in S: m_{ir} + m_{is'} = 2 \text{ \& } r \neq s' \quad \text{Eq. 2-22}$$

$$\sum_{k \in K_r} t_{ik} - \sum_{k \in K_{s'}} t_{ik} \geq \text{gap}_{is'r} - Mz_i \quad \text{Eq. 2-23}$$

$$\forall i \in N, s', r \in S: m_{ir} + m_{is'} = 2 \text{ \& } r \neq s'$$

$$\sum_{k \in K_r} t_{ik} - \sum_{k \in K_{s'}} t_{ik} \leq -\text{gap}_{is'r} + M(1 - z_i) \quad \text{Eq. 2-24}$$

$$\forall i \in N, s', r \in S: m_{ir} + m_{is'} = 2, r \neq s'$$

Os autores definem z_i como binária que vale 1 caso $\sum_{k \in K_r} t_{ik} \geq \sum_{k \in K_{s'}} t_{ik}$ e a constante $M = \max_{k \in K} \beta_k + \max_{i \in N, s', r \in S: s' \neq r} \text{gap}_{isr} + 1$. As restrições acima linearizam a expressão $|\sum_{k \in K_r} t_{ik} - \sum_{k \in K_{s'}} t_{ik}| = \text{gap}_{isr}$

- Domínio das variáveis de decisão:

$$x_{ijk} \in \{0,1\} \quad \forall i, j \in V, k \in K \quad \text{Eq. 2-25}$$

$$z_i \in \{0,1\} \quad \forall i \in N \quad \text{Eq. 2-26}$$

$$t_{ik} \in \mathcal{R}_*^+ \quad \forall i \in V, k \in K \quad \text{Eq. 2-27}$$

Ao adotar variáveis atreladas aos arcos e também ao recurso, ou seja, indexadas por nó de origem, destino e recurso utilizado, tudo se passa como se houvesse um grafo para cada veículo da frota, sendo que neste problema os grafos são idênticos entre si, já que o tempo e o custo de locomoção são independentes do recurso alocado. A relação entre estes grafos é estabelecida somente pela variável que marca o instante de início das atividades. Os autores argumentam que esta estratégia de modelagem é

significativamente menor do que usar nós fictícios para modelar a sincronização de serviços, como nos modelos apresentados na próxima seção.

2.4.3 Modelagem compacta para o problema de coleta e entrega

Furtado, Munari e Morabito (2017) definem o PDVRPTW no grafo $G = (N, A)$, formado pelo conjunto de nós N e de arcos A e propõem um Modelo de Programação Inteira Linear Mista (MILP) que é uma evolução da modelagem proposta por Lu e Dessouky (2006), conforme explicitado a seguir.

Conjunto de parâmetros e variáveis utilizados:

- N é o conjunto de nós da rede, que inclui o conjunto de nós de coleta $P = \{1, 2 \dots n\}$; o conjunto dos de entrega $D = \{n + 1, n + 2 \dots 2n\}$; o conjunto de nós de início e fim das rotas $N_0 = \{0, 2n + 1\}$. No modelo de Lu e Dessouky (2006), este último conjunto é composto por nós fictícios que representam os nós de origem de cada recurso, o que o torna mais adequado a problemas de frota heterogênea. Neste caso, define-se o conjunto $N'_0 = \{2n + 1, 2n + 2 \dots 2n + m + 1\}$ e a rota de cada veículo $k \in K = \{1, 2 \dots m\}$ é iniciada no nó de índice $i = 2n + k$ e finalizada no nó indexado como $j = 2n + k + 1$.
- Cada nó $i \in N$ da rede possui uma janela de tempo para início do atendimento $[e_i, l_i]$ e um parâmetro associado à carga demandada ou ofertada no nó, q_i , assumindo um valor negativo nos nós de entrega, positivo nos de coleta e nulo em nós de início de rota.
- A é o conjunto de arcos da rede, representando o deslocamento entre dois locais com demanda de serviço ou entre o depósito inicial e os pontos de coleta de carga. Cada arco $(i, j) \in A$ é associado a um custo e a um tempo de deslocamento, c_{ij} e t_{ij} .

Variáveis de decisão:

- x_{ij} : binária que indica se o arco (i, j) faz parte da solução, isto é, é trafegada por um dos veículos da rota
- B_i : variável contínua não-negativa que indica a data de início do serviço no nó i
- Q_i : variável contínua não negativa que indica a carga do veículo após a visita em um nó i

- v_i : variável contínua que assume o valor do índice do primeiro nó da rota que atende o nó i (seu valor será sempre inteiro, mas não há necessidade de modelar a variável como inteira).

A função objetivo é minimizar o custo devido às viagens realizadas:

$$\min z = \sum_i \sum_j c_{ij} x_{ij} \quad \text{Eq. 2-28}$$

Sujeito a:

- Um e apenas um arco entrando e saindo de cada nó de demanda (de entrega ou coleta de produto) deve ser selecionado:

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in P \cup D \quad \text{Eq. 2-29}$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in P \cup D \quad \text{Eq. 2-30}$$

- Não pode haver sobreposição temporal na execução de duas tarefas alocadas para o mesmo veículo. Assim, a data de início do serviço em determinado nó deverá respeitar todo o trajeto executado pelo veículo, o que também garante a não existência de subciclos na solução:

$$B_j \geq B_i + t_{ij} - M(1 - x_{ij}) \quad \forall i, j \in N \quad \text{Eq. 2-31}$$

- A carga no veículo, após visitar um nó, leva em conta o tipo de serviço executado no nó (coleta ou entrega) e todos os serviços anteriores:

$$Q_j \geq Q_i + q_j - M(1 - x_{ij}) \quad \text{Eq. 2-32}$$

- A data de início da atividade está contida em sua janela de tempo:

$$e_i \leq B_i \leq l_i \quad \forall i \in N \quad \text{Eq. 2-33}$$

- A capacidade dos veículos precisa ser respeitada:

$$\max\{0, q_i\} \leq Q_i \leq \min\{Cap, Cap + q\} \quad \forall i \in N \quad \text{Eq. 2-34}$$

- O mesmo veículo que realiza a entrega de uma carga deve realizar a coleta, previamente:

$$v_{n+i} = v_i \quad \forall i \in P \quad \text{Eq. 2-35}$$

$$B_{n+i} \geq B_i + t_{i,n+i} \quad \forall i \in P \quad \text{Eq. 2-36}$$

Os autores colocaram o deslocamento na inequação, embora seja supérflua, já que a restrição que define as datas de início de tarefas executadas pelo menos veículo já inclui este tempo.

- A variável que indica o veículo que atende um nó i deverá assumir o valor do índice do primeiro nó da rota ligada a i . Para isso, o arco saindo de um nó de origem de rota é usado para linkar as duas variáveis de decisão x_{ij} e v_j . Na modelagem dos autores, é criado um nó inicial $i = 0$ onde todas as rotas são iniciadas e um nó $2n + 1$ onde todas as rotas são finalizadas, sem a necessidade de criar nós artificiais para representar os veículos, uma vez que neste problema a frota é heterogênea e não há parâmetros específicos de cada recurso (como data de liberação ou jornada de trabalho, por exemplo).

$$v_j \geq j \cdot x_{0j} \forall j \in P \cup D \quad \text{Eq. 2-37}$$

$$v_j \leq i \cdot x_{0j} - n(x_{0j} - 1) \forall j \in P \cup D \quad \text{Eq. 2-38}$$

- O mesmo link é feito usando os demais arcos da rota:

$$v_j \geq v_i + n(x_{ij} - 1) \forall i, j \in P \cup D \quad \text{Eq. 2-39}$$

$$v_j \leq v_i + n(1 - x_{ij}) \forall i, j \in P \cup D \quad \text{Eq. 2-40}$$

- Finalmente, o domínio das variáveis de decisão é dado:

$$x_{ij} = \{0,1\} \forall (i,j) \in A \quad \text{Eq. 2-41}$$

$$B_i, Q_i, v_i \in \mathbb{R}_*^+$$

Interessante notar que, como não pode haver subciclo (graças às restrições de datas de início das atividades executadas no nós) e todo nó de serviço precisa ter um arco entrando e saindo dele, necessariamente haverá a seleção de um nó de origem da rota. Isto faz com que as seguintes restrições tradicionais de escolha do arco inicial tornem-se supérfluas:

$$\sum_{j \in N \setminus \{2n+1\}} x_{ij} = 1 \forall i \in N_0 \setminus \{2n+m+1\} \quad \text{Eq. 2-42}$$

$$\sum_{i \in N \setminus \{2n+m+1\}} x_{ij} = 1 \forall j \in N_0 \setminus \{2n+1\} \quad \text{Eq. 2-43}$$

2.5 DOS MÉTODOS DE SOLUÇÃO

Problemas de otimização combinatória possuem como característica um espaço de solução vasto, o que dificulta a aplicação de métodos de solução otimizantes, ou seja,

que comprovadamente são capazes de encontrar solução ótima, para instâncias reais. Enquanto alguns destes problemas possuem algoritmos gulosos (*greedy*) capazes de chegar em soluções ótimas ou sub ótimas, a maioria deles precisa ser resolvida com uma combinação de método guloso para obter rapidamente uma solução viável, ainda que de baixa qualidade, seguida por um processo de busca pelo espaço de soluções, guiada por um conjunto de regras e procedimentos (PINEDO, 2016).

A Busca Local (BL) ou Busca em Vizinhança é um processo de busca estruturado, que examina uma região restrita do espaço de soluções, a fim de encontrar a melhor solução (ou as melhores) contidas neste espaço, conhecida como solução de mínimo local (JOHNSON; PAPADIMITRIOU; YANNAKAKIS, 1985). Para tanto parte-se de uma solução inicial (ou um grupo de soluções iniciais) e nela(s) são feitas pequenas alterações, chamadas de movimentos, para gerar outras soluções possíveis ao problema. Este conjunto de soluções, formadas a partir de aplicações do movimento na(s) solução(ões) original(is), é chamado de vizinhança da solução. A cada iteração, a solução que serve de base para gerar a vizinhança é chamada de solução corrente. Construída a vizinhança, um critério de seleção é aplicado na escolha da próxima solução corrente e, caso haja melhoria em relação à solução inicial, ela se torna a nova solução incumbente, que é a melhor solução encontrada ao longo da busca. A definição da solução corrente é normalmente feita usando um dos critérios a seguir: ou a primeira solução vizinha gerada que apresentar melhoria em relação à inicial é selecionada (estratégia conhecida como *first descent*), ou os movimentos são aplicados exaustivamente, até gerar todas as soluções vizinhas à inicial, e só então a melhor delas é selecionada (*steepest descent* ou *best descent*). Após a seleção da solução corrente, com atualização ou não da incumbente, os movimentos são repetidos para construir a vizinhança desta nova solução (ou do novo grupo de soluções). Tal processo é repetido até que um critério de parada seja atingido. Em algumas heurísticas de Busca Local, isto acontece quando nenhuma das soluções vizinhas à solução corrente apresente melhoria na função objetivo (ou na função usada para comparar soluções alternativas); enquanto outras são finalizadas após determinado tempo ou número máximo de iterações. Há também aplicações de Busca Local que só aceitam soluções vizinhas que melhorem a solução corrente, ou até mesmo a solução incumbente do problema; ou que aceitam soluções piores a uma certa probabilidade. Caso a regra adotada para atualização da solução corrente não seja satisfeita, o método pode ser

finalizado, ou outra vizinhança da solução corrente pode ser gerada – com a aplicação de outro movimento.

Heurísticas que funcionam da maneira descrita anteriormente, ou seja, por repetição do ciclo definição de solução corrente - alteração na solução - atualização de solução incumbente, são classificadas por Schrimpf et al. ⁽²⁰⁰⁰⁾ como algoritmos de melhoria iterativa. Segundo a definição dos autores, algoritmos de melhoria iterativa começam com solução inicial e prosseguem realizando pequenas modificações na solução corrente como intuito de encontrar novas soluções, melhores que a original; cada solução encontrada é avaliada, seja por seu valor na função objetivo ou por uma função aproximada, e uma regra de decisão de aceitação da solução é aplicada (SCHRIMPF et al., 2000). Os autores enumeram as seguintes regras para aceitação da solução corrente:

- Caminho aleatório (*random walk*): toda nova solução é aceita como solução corrente (pressupõe que não há geração de vizinhança);
- Aceitação gulosa (*Greedy Acceptance*): toda solução vizinha que seja melhor que a solução corrente é aceita, mesmo que seja pior que a incumbente;
- Aceitação com aleatoriedade (como na *Simulated Annealing*): toda solução vizinha que seja melhor que a corrente, enquanto soluções piores são aceitas segundo uma probabilidade, que diminui ao longo da execução do algoritmo;
- Aceitação com limitante de piora (*Threshold Acceptinn*): soluções melhores ou cuja diferença em relação à solução corrente estejam dentro de um valor pré-estipulado são sempre aceitas como nova solução corrente;
- *The Great Deluge Algorithm*: apenas soluções acima de um certo nível de qualidade são aceitas.

Com a evolução da pesquisa usando a Busca Local para melhoria de soluções iniciais, constatou-se que estes métodos possuem uma tendência a ficar presos em regiões de mínimo local da solução. Além disso, a qualidade da solução inicial passa a ser determinante para a evolução da busca (LOURENÇO; MARTIN; STÜTZLE, 2003).

Para evitar que isto aconteça, foram criadas formas para causar perturbações na solução corrente ou incumbente, modificações maiores e em geral com componentes aleatórios mais relevantes do que as aplicadas nos movimentos de busca local, a fim de sair de uma região do espaço de soluções. Conforme este tipo de abordagem foi sendo testado, notou-se que as metodologias de melhor desempenho faziam uma

ponderação entre os processos de intensificação da busca, relativos à maior cobertura de um espaço de soluções reduzido, normalmente feito pela própria busca local; e de diversificação, que têm como cobrir uma região mais ampla do espaço de soluções, ainda que cada micro região não seja completamente vasculhada. As regras de aceitação de solução corrente com piora na função objetivo, citadas anteriormente, são estratégias para justamente diversificar a busca. Nesta tese, as abordagens de busca focadas na diversificação são chamadas de Busca Global (BG), para diferenciá-las da BL.

Neste contexto, surgiram métodos de BL e BG para a solução de problemas de otimização combinatória. Então, convencionou-se chamar de meta-heurística o “conjunto de procedimentos que guiam e modificam as operações de heurísticas subordinadas, para produzir melhores soluções a problemas de otimização de difícil solução”, segundo definição de Ribeiro e Hansen (RIBEIRO; HANSEN, 2002). Em outras palavras, meta-heurísticas são algoritmos que, partindo de uma solução inicial de um problema de otimização, fazem uma busca estruturada pelo espaço de soluções, usando estratégias para evitar que a busca se restrinja a uma região de mínimo local. Algumas das meta-heurísticas conhecidas são: *Simulated Annealing*, Busca Tabu, Algoritmos Genéticos, *Scatter Search*, *Greedy Randomized Adaptive Search* (GRASP), *Large Neighbourhood Search* (LNS) e suas variantes: Busca em Vizinhança de Larga Escala (VLNS, da terminologia em inglês *Very Large Neighbourhood Search*), Busca Adaptativa em Vizinhança de Larga Escala (ALNS, *Adaptative Large Neighbourhood Search*), Busca em Vizinhança Variável (VNS, *Variable neighborhood Search*), Busca Local Iterativa (ILS, de *Iterated Local Search*), Colônia de Formigas, entre outras. Uma diferenciação importante entre as meta-heurísticas diz respeito à sua chamada “memória”: enquanto alguns métodos modificam a solução corrente sem considerar o caminho percorrido pelo processo de busca, como o *Simulated Annealing*; outros consideram as configurações já examinadas em soluções passadas, ao utilizar informações relevantes sobre essas soluções na formação de novas vizinhanças, como Busca Tabu (GLOVER; LAGUNA, 1998), *Scatter Search* e *Path-relinking* (GLOVER; LAGUNA; MARTI, 2000).

De acordo com a literatura de PRV e de sequenciamento de tarefas, para problemas cuja complexidade torna desafiadora até mesmo a obtenção de uma única solução viáveis, as meta-heurísticas não são tão eficientes (RAHIMIAN; AKARTUNALI; LEVINE, 2016). Por outro lado, abordagens exatas como a Programação por Restrições

(*Constraint Programming*) são ótimas em resolver problemas complexos, mas incapazes de resolver instâncias de média ou larga escala. Para contornar este impasse, nos últimos anos vêm surgindo na literatura métodos híbridos, denominados de pseudo-exatos ou mateheurísticas (tradução livre do termo em inglês *matheuristics*, jogo de palavras que substitui o termo *meta* por *math*, em alusão à *mathematics*), resultantes da combinação de técnicas exatas com meta-heurísticas. Normalmente, o método exato escolhido é usado para obter uma solução que satisfaça parte das restrições do problema, para na sequência ser aprimorada pela meta-heurística.

Uma das razões para a dificuldade em encontrar soluções para o problema de roteirização com sincronização de operações é a interdependência de rotas. Segundo Rousseau, Gendreau e Pesant (2013), o elevado número de interconexões entre rotas significa que, para inserir um novo cliente, pode ser necessário recalcular as datas de visita de todos os clientes já inseridos nas rotas. Embora os autores tenham adotado a Programação por Restrições, uma alternativa para lidar com o alto custo computacional na modificação de soluções é não averiguar vizinhanças inteiras.

Neste sentido, abordagens do tipo Destruir-e-Reparar (tradução de *ruin-and-recreate*) propõem um procedimento de busca que remove uma parte da solução corrente e a reconstrói usando uma regra simples. O procedimento usado na reconstrução pode variar a cada iteração, à exemplo do que é feito por métodos de busca que utilizam mais de uma estrutura de vizinhança, como a Busca em Vizinhança Variável. Segundo Rahimian, Akartunali e Levine (2016), “uma estratégia de destruir-e-recrutar parte da melhor solução obtida, então destrói esta solução parcialmente e a recria a partir da solução remanescente, com objetivo de obter uma solução diferente da original e de melhor qualidade”.

Para embasar o desenvolvimento de métodos para a solução do problema abordado nesta tese, a literatura pertinente foi consultada. Dada a grande quantidade de meta-heurísticas existentes e sua constante evolução, a proposta aqui não é oferecer uma visão abrangente sobre todos os métodos, tampouco apresentar uma lista exaustiva dos algoritmos existentes. Alternativamente, as próximas subseções apresentam de maneira sucinta alguns dos métodos já aplicados a problemas relacionados ao abordado nesta tese, que serviram de base para o desenvolvimento deste trabalho.

Os autores tratam do VRPTWSyn, em que um conjunto de recursos (ou frota de veículos), disponível em um depósito inicial, deve atender serviços demandados por um conjunto de clientes espalhados geograficamente, sendo que cada cliente só pode

receber o veículo dentro de uma janela de tempo pré-especificada e alguns serviços demandam mais de um veículo simultaneamente. Tal problema é uma extensão do VRPTW, que é sabidamente NP-hard, o que o torna um bom candidato para uso de heurísticas. Contudo, os autores argumentam que as restrições de sincronização do VRPTWSyn tornam a aplicação de heurísticas baseadas em busca em vizinhança extremamente ineficiente, já que toda vez que um cliente é inserido, possivelmente é necessário recalcular as datas de início de todas as atividades já programadas.

2.5.1 Busca em Vizinhança de Grande Porte (LNS) e variações aplicadas à roteirização de veículos

Shaw (1998) introduz uma heurística de busca local, a *Large Neighbourhood Search*, e a testa em duas variantes do PRV: Roteirização de Veículos Capacitados (PRVC) e PRVJT, com janela de tempo. Em ambos os casos, há apenas um depósito inicial, a frota de veículos é infinita, homogênea e capacitada.

A LNS funciona como uma busca local de um único movimento, de remoção e reinserção de um subconjunto de visitas a clientes na solução, com o diferencial de usar inteligência na escolha destes clientes e de simplificar os cálculos na inserção, pelo uso de propagação de restrições, para evitar movimentos que gerem soluções inviáveis ou que tenham custo computacional (tempo) muito alto (SHAW, 1998). Isto porque, segundo o autor, há dois fatores que afetam o desempenho da busca local: como o conjunto de nós a remover é escolhido e o processo de reinserção.

A escolha dos clientes a remover baseia-se num atributo que tem como objetivo refletir o grau de relação (tradução livre do termo em inglês, *relatedness*) entre os nós da rede. A cada iteração do LNS, sorteia-se um conjunto de visitas a remover da solução, com alto grau de relação entre si. Para diversificar a busca, é usado um parâmetro que controla o quanto o grau de relação pesa na escolha dos nós a remover, de forma que em algumas iterações os nós são escolhidos de forma aleatória. A quantidade de nós a remover aumenta quando é atingido um determinado número de iterações sem melhoria na função objetivo, até um valor máximo (30 clientes). Para os problemas testados pelo autor, dois aspectos são considerados para avaliar a relação entre dois clientes: sua posição geográfica, isto é, quanto mais próximos, mais relacionados são os clientes; e o fato de estarem sendo atendidos pelo mesmo veículo. Contudo, o autor esclarece que para outras aplicações, aspectos específicos do problema devem ser considerados, modificando a função de grau de relação entre dois nós da rede.

O procedimento de reinserção usa o algoritmo otimizante *branch-and-bound*, combinado com heurísticas para diminuir a complexidade do movimento e limitar a busca a uma parte da árvore de *branch and bound*, reduzindo o tempo computacional necessário – mais especificamente, é empregada a técnica da Busca com Discrepância Limitada (*Limited Discrepancy Search*) para não explorar todos os galhos da árvore de *branch-and-bound*, limitando-se àqueles com menor grau de discrepância. Assim, após a destruição parcial da solução, cada cliente removido tem seu custo de reinserção calculado para todas as posições legais de inserção. Três critérios determinam se a posição da inserção é ilegal para dado cliente: (i) a capacidade do veículo associado àquela rota não é suficiente para acomodar o cliente que se deseja inserir; (ii) os prazos da tarefa a inserir ou de alguma das tarefas postergadas devido à inserção são rompidos caso haja inserção; e (iii) o custo da solução parcial mais o acréscimo pela inserção da tarefa é superior à melhor solução encontrada até o momento. Após calcular os custos incrementais da reinserção, os clientes não-visitados são ordenados decrescentemente de acordo com o menor custo incremental para sua própria inserção. Seguindo esta ordem, os clientes são reinseridos um-a-um de volta à solução, sempre na posição (viável) que gere o menor incremento de custo. Após cada reinserção, todos os clientes ainda não inseridos são reavaliados quanto ao custo de reinserção nas posições legais remanescentes e o conjunto é reordenado. Ainda, caso a qualquer momento um cliente tenha apenas uma posição legal de reinserção, ele é imediatamente reinserido na solução, independentemente de sua posição no conjunto pré-ordenado. Tal método é apelidado de heurística de inserção por ponto mais longe (*farthest insertion*), já que primeiro são reencaixados os clientes que gerem maior custo. O LNS é testado por Shaw (1998) em benchmarks dos problemas abordados, com diferentes configurações (valores para seus parâmetros de calibração da heurística). O método foi capaz de melhorar as soluções conhecidas à época para algumas das instâncias da literatura, e teve desempenho médio similar aos melhores métodos conhecidos até então, com a vantagem de ser mais simples de implementar, segundo o autor. Uma diferença importante com relação ao problema de roteirização abordado pelo autor e o tratado neste trabalho é que a frota daquele é ilimitada. Isto permite que a solução inicial adotada possua um cliente por veículo e rota, como feito pelo autor, o que não é viável para a roteirização de rebocadores.

Já Rousseau, Gendreau e Pesant (2013) propõem heurística híbrida para a solução de um problema de roteirização dinâmica, com restrições de sincronização. O método

proposto combina Programação por Restrição (CP) com duas abordagens de Busca Local: uma que adota a primeira possibilidade de inserção da tarefa que gere solução viável (*first insertion*) e outra que verifica a melhor possibilidade de inserção da tarefa encontrada dentro de um limite de tempo (*best insertion*). O método é dinâmico, devido à própria natureza do problema abordado por aqueles autores: no início da execução e toda vez que surge um pedido novo, um controlador do processo de inserção iterativa aciona o operador de inserção. Este operador pega a última solução do modelo matemático, ou seja, que contempla os clientes já alocados em rotas; constrói o modelo de inserção do novo pedido, usando CP; e resolve o modelo. Caso a inserção possa ser feita, a solução é passada para o controlador de inserções e, caso não tenha chegado um novo pedido de inserção de cliente, o controlador envia a solução corrente para o método de busca local. A busca é executada até que um critério de parada seja atingido ou até que um novo pedido de inserção chegue ao controlador.

Nessa aplicação, a busca local utiliza operadores tradicionais de problemas de roteirização, a saber: *2-Opt*, *Or-Opt*, *Cross*, *Exchange* e *Relocate*. Duas meta-heurísticas foram testadas para guiar a busca: a abordagem tradicional da BL e a Busca Local Guiada, ou *Guided Local Search* (GLS), que penaliza movimentos que tendam a revisitar áreas da região já exploradas, de maneira similar ao que é feito pela Busca Tabu. Assim, foram testadas quatro variantes de busca local: BL com *first insertion*, BL com *best insertion*, GLS com *first insertion* e GLS com *best insertion*. Os resultados mostraram que a GLS com aceitação de vizinho por *best insertion* produziram melhores resultados, de maneira geral. Um ponto interessante do método de solução proposto é que o operador de inserção e o método de busca local não precisam ser alterados para comportar as restrições de sincronização e as janelas de tempo dinâmicas (definidas em função da data de chegada do veículo capacitado), pois esta parte da heurística é feita pelo modelo de CP.

Por sua vez, Hojabri et al. (2018) apresentam um algoritmo que combina o ALNS com CP (Constraint Programming), da seguinte maneira: a cada iteração, um certo número de clientes da solução corrente é removido da solução corrente, usando um operador de remoção escolhido à uma probabilidade que varia conforme os resultados obtidos ao longo da busca; em seguida, o problema de reinserção dos clientes removidos é modelado usando CP, e um pacote de otimização (ILOG CP Optimizer) resolve o subproblema. O subproblema de CP visa simplesmente satisfazer a restrição de que o novo valor da função objetivo deve ser menor do que o da solução corrente (ou pouco

pior, caso seja sorteada a possibilidade de aceitar uma solução pior), dentro de um intervalo de tempo que também varia ao longo das iterações do ALNS. Ou seja, é possível que o CP não consiga encontrar uma nova solução e, caso isto ocorra antes que o critério de parada seja atingido, o processo é simplesmente reiniciado partindo da mesma solução corrente (novamente é sorteado um operador de remoção e são removidos alguns clientes, criando um novo subproblema de CP a resolver). Para obter a solução inicial do problema, o algoritmo proposto por Solomon ⁽¹⁹⁸⁷⁾ é utilizado primeiro para os vértices regulares, e então para os especiais, uma vez que as janelas de tempo forem determinadas para estes clientes.

O método desenvolvido por Rahimian, Akartunali e Levine, 2016) é aplicado a um problema de programação de tarefas para equipe de enfermeiros, que possui similaridades ao problema de roteirização com sincronização de recursos. Os autores apresentam um algoritmo híbrido que utiliza uma meta-heurística e um método exato, mais especificamente o VNS e a Programação Inteira (*Integer Programming*), aproveitando os pontos fortes de ambas as abordagens. Os autores argumentam que a forma como os métodos foram combinados na heurística híbrida criada é inovador: a IP é encapsulado no VNS, servindo para diversificar e intensificar a busca local, como mecanismo de perturbação; e depois da finalização do VNS, a IP é utilizada para melhorar a solução até que certo limite de tempo seja atendido. Na fase encapsulada, a IP é usada dentro de uma abordagem do tipo destrói-e-recria (vide seção 2.5.4), em que partes da solução que geram altas penalidades são removidas e uma nova solução é recriada. Antes da execução da metaheurística proposta, uma heurística gulosa é usada para gerar uma solução inicial. O algoritmo proposto é iterativo e, por tentativa e erro, tenta gerar programas viáveis para cada um dos profissionais da equipe individualmente, selecionando aleatoriamente os dias em que o profissional trabalhará e, depois, os turnos, mas respeitando padrões de trabalho viáveis. Caso a carga total de trabalho resultante ou o número de finais de semana trabalhados ultrapassem limites pré-estabelecidos nas restrições do modelo, a solução é descartada e o processo é reiniciado. Segundo os autores, são necessários 100 ciclos deste processo até gerar um cronograma de trabalho viável para um profissional.

A atual heurística estado-da-arte para o PRVJTSin é a apresentada por Liu, Tao e Xie (2019). O método, baseado na ALNS, parte de uma solução inicial e, a cada iteração, um movimento de remoção de clientes é sorteado, assim como a quantidade de clientes a remover e o movimento de reinserção. Cinco movimentos de remoção e dois de

inserção são considerados no sorteio, sendo que a probabilidade de escolha de cada movimento varia conforme o desempenho dos mesmos ao longo da execução da heurística. Ao remover e reinserir os clientes, apenas uma solução vizinha é construída, assim como na abordagem de Destruir-e-Reparar. A solução criada só é aceita como solução corrente caso satisfaça a um critério definido pelos autores.

Baseado no método *Simulated Annealing*, o critério de aceite da solução vizinha varia ao longo da execução da heurística e leva em conta não só a função objetivo estabelecida no modelo matemático (custo ou tempo total de transporte), como também as violações das janelas de tempo das tarefas e dos veículos (nesta aplicação, a janela dos veículos corresponde à jornada de trabalho dos funcionários). Para isso, é proposta uma função objetivo “aumentada”, que atribui pesos aos elementos de violação da viabilidade da solução, sendo que tais pesos variam ao longo da execução da heurística. O processo é finalizado quando o número de iterações atinge um limite, ou seja, o critério de parada é exclusivamente por número de iterações. Assim, ao longo da execução, soluções inviáveis são aceitas a uma probabilidade que diminui conforme o número de soluções viáveis encontradas aumenta, graças aos pesos mencionados anteriormente. Aparentemente, a viabilidade de uma solução só é checada quando a solução incumbente é atualizada, pois é neste momento os parâmetros associados à função de aceite das soluções correntes são atualizados.

A obtenção de solução inicial é feita por aplicação de uma heurística construtiva simples: em cada passo, uma tarefa não roteirizada é escolhida aleatoriamente e encaixada na posição que resulte no menor incremento na função objetivo aumentada. Desta forma, caso seja possível gerar uma solução viável, pressupõe-se que ela será preferida em relação a uma alternativa inviável, pois esta última terá maior valor na função objetivo aumentada.

Os autores ressaltam a dificuldade de modificar a solução deste tipo de problema, por conta da grande quantidade de interconexões entre rotas, e detalham os artifícios usados para aumentar a eficiência computacional do método, que envolvem não avaliar todas as janelas de tempo a cada iteração e alguns truques de programação. Como apontado pelos autores, a implementação de heurísticas para os problemas da classe PRVJTSin não é trivial.

Para calibrar seu algoritmo, Liu et al. (2013) modificam algumas das instâncias originalmente propostas por Solomon (1987) e por Gehring e Homberger (1999), a fim de obter instâncias pequenas, médias e grandes compatíveis com o PRVJTSin. Foram

selecionadas 6 instâncias de Gehring e Homberger com 200 clientes e 12 instâncias de Solomon com 100 clientes, mas metade destas foi reduzida, sorteando apenas 30 clientes para compor o problema. Para cada uma das instâncias menores do VRPTW, foram criadas 3 instâncias do VRPTWSyn, com 10, 30 e 50% de clientes com restrição de sincronização. Para as médias e grandes, também foram criadas 3 instâncias novas por instância do VRPTW, usando 5, 10 e 20% dos clientes. A forma de atribuição da demanda para cada cliente com restrição de sincronização não foi explicitada no artigo, porém solicitamos as instâncias para os autores e verificamos que o número máximo de veículos por cliente foi 2.

Como cada instância original (i.e. do VRPTW) corresponde a 3 problemas diferentes, conforme a lógica adotada para o posicionamento geográfico dos clientes, os autores usaram as três variantes de cada instância. Assim, foram criadas 18 instâncias de cada tamanho (30, 100 ou 200 clientes).

Segundo os autores, o principal parâmetro a calibrar é o número de clientes removidos, q , pois ele controla a acurácia e a velocidade do método. A cada iteração da heurística, q é selecionado aleatoriamente dentro de um intervalo, portanto a calibração foi feita via intervalo: 5 a 15% dos clientes; 15 a 30%; 5 a 30%; 15 a 50% e 30 a 50%. Ao fim dos testes, foi selecionado o intervalo de 15 a 30% do total de clientes da instância, que conseguiu bons resultados sem consumir muito tempo.

Para atestar o desempenho do método, os autores usaram as instâncias de Bredstrom e Ronnqvist (2008), originalmente propostas para o VRPTW com restrição de precedência e sincronização, inspirado no problema de scheduling de equipe para cuidados médicos domiciliares. Nesse conjunto de instâncias, há problemas com 20, 50 e 80 clientes, dos quais em média 10% exige a presença simultâneas de dois recursos - sim, apenas dois. Os clientes são distribuídos uniformemente em torno do depósito de origem e destino das rotas, e três funções objetivo são testadas para cada instância, para possibilitar a comparação direta com o desempenho de métodos de solução publicados previamente.

2.5.2 Busca Local Iterativa (ILS) aplicada à roteirização de veículos

A ILS, introduzida por Lourenço, Martin e Stützle (2003), funciona da seguinte forma: começando por uma boa solução viável, obtida pela aplicação combinada de uma heurística gulosa e de método de Busca Local, a cada iteração a solução corrente é perturbada, com a intenção de explorar outras regiões do espaço de solução, e a BL é

refeita. Caso a solução obtida seja melhor que a incumbente, esta última é atualizada. O processo de perturbação seguida de busca local é repetido por um número pré-determinado de iterações.

Ait Haddadene, Labadie, e Prodhon (2016) apreentam uma meta-heurística híbrida, que combina a GRASP com a ILS: assim como a GRASP, cada iteração parte de uma solução gerada por procedimento guloso randomizado e em seguida aplica melhorias nessa solução; no algoritmo proposto, estas melhorias são feitas pela aplicação da ILS, que primeiro perturba a solução inicial (gerada pela GRASP) e depois faz uma busca local aplicando uma sequência de estruturas de vizinhança (os autores testaram duas estratégias na busca local, uma randomizada e outra baseada na VND). Ainda, a busca local executada a cada iteração computa parâmetros que permitem determinar se o movimento em questão tem potencial para gerar filhos viáveis, graças a testes de viabilidade que consideram tanto as janelas de tempo como as restrições de sincronização.

2.5.3 Greedy Randomized Adaptative Search Procedure (GRASP) aplicada à roteirização de veículos

Proposta por Feo e Resende (1995), a meta-heurística GRASP usa uma população de solução iniciais como ponto de partida para a execução de buscas locais paralelas. Então, a melhor solução encontrada é adotada como solução final.

Como indicado pelos próprios autores, as regras e processos adotados na Busca Local podem variar. Por exemplo, pode ser adotada uma lista restrita de vizinhos candidatos à solução corrente, ao invés de examinar a vizinhança toda; ou usar a memória de soluções passadas para guiar a busca, por exemplo por *path-relinking* (FESTA; RESENDE, 2011; MARTÍ; RESENDE; RIBEIRO, 2013; RESENDE; RIBEIRO, 2014).

A meta-heurística é aplicada para resolver PRV por Kontoravdis e Bard (1995). Além da aplicação de Ait Haddadene, Labadie e Prodhon (2016), já citada neste trabalho, a combinação do GRASP com ILS para resolver PRV também é feita por Prins (2009).

2.5.4 Métodos do tipo Destrói-e-Repara aplicados à roteirização de veículos

Segundo Christiaens e Vanden Berghe (2020), a estratégia de destruir-e-reparar (*Ruin-and-Recreate*, **R&R**) foi proposta originalmente por Dees e Smith (1981), mas só recebeu esta nomenclatura no artigo de Schrimpf *et al.* (2000). Esta abordagem explora a região próxima a uma solução corrente, similarmente ao que é feito pelos métodos de busca em vizinhança, porém apenas um vizinho da solução corrente é avaliado.

Para gerar o vizinho, primeiro destrói-se uma parte considerável da solução corrente, para então reconstruí-la. Assim, a R&R costuma aparecer combinada a outras meta-heurísticas, encapsulada na fase de BL. Por exemplo, (PISINGER; ROPKE, 2007) propõem uma estrutura padrão para a ALNS usando diversos movimentos para as fases de destruição e recriação da solução, e aplicam o método a alguns problemas de roteirização, entre eles uma variante de coleta e entrega com janelas de tempo.

Schrimpf *et al.* (2000) argumentam que a R&R generaliza as regras de aceitação com aleatoriedade e com limitante de piora. No R&R, partes extensas de uma solução corrente para o problema é destruída e, em seguida, reconstruída de maneira diferente, com objetivo de melhorar a solução. Para aqueles autores, o poder desta abordagem está no fato de que ao destruir uma parte suficientemente grande da solução, aumenta-se a flexibilidade para reconstruir uma solução melhor. Três problemas são resolvidos por R&R no trabalho citado: PRVJT, Problema de Otimização em Rede e Problema do Caixeiro Viajante.

O PRVJT abordado, de particular interesse para esta tese, tem como função objetivo a minimização do número de veículos empregado e, secundariamente, da distância total viajada. A heurística desenvolvida utiliza dois modos de destruição, denominados radial e aleatório; e um método de reconstrução (reparo) simples: escolhe-se aleatoriamente um dos nós do conjunto formado pelos nós removidos no processo de destruição, e o adiciona na melhor posição possível (isto é, que gere o menor incremento de custo na função objetivo), respeitando as janelas de tempo. Caso não seja possível adicionar o nó em nenhuma das rotas existentes, é adicionado um novo veículo à solução. Assim, os nós são readicionados um-a-um na melhor posição possível, de maneira gulosa (sem levar em conta o impacto desta decisão no encaixe das demais tarefas removidas e ainda não reinseridas nas rotas).

A cada iteração da heurística, um dos modos de destruição é escolhido aleatoriamente, à mesma probabilidade, e em seguida é executado o algoritmo de R&R na solução corrente. A nova solução passa por uma decisão de aceitação, substituindo ou não a solução corrente. Para a destruição radial, o número de nós removidos é menor ou igual a 30% da quantidade de nós contemplados pela solução; enquanto para a destruição aleatória adotou-se no máximo 50% dos nós. Depois da reconstrução completa da solução, usando critério de melhor inserção para cada tarefa, se a nova solução for pior que a corrente, ela só é aceita caso a distância entre as duas for inferior a um limitante. O valor deste limitante, inicialmente igual à metade do desvio padrão da função objetivo

obtida por aplicação de uma regra de aceitação de caminhada aleatória, diminui exponencialmente a cada iteração da heurística. Em contrapartida, se a nova solução for melhor que a corrente, esta última é substituída pela nova solução e comparada à solução incumbente. Após o processo decisório de troca da solução corrente, ela passa novamente pelo mesmo processo de R&R (seleção de método de destruição e dos parâmetros envolvidos, destruição e reconstrução da solução, avaliação e redefinição da solução corrente), até que este ciclo tenha sido repetido 40.000 vezes.

Uma das vantagens do método, enfatizada pelos autores, é a possibilidade de paralelizar os cálculos, ganhando muita eficiência. Os passos de reconstrução da solução, responsáveis por 90% do tempo computacional total da heurística, podem ser paralelizados por veículo/rota, já que tais cálculos são independentes rota-a-rota. Contudo, para VRPs com restrição de sincronização isto não é verdade.

Christiaens e Vanden Berghe (2020) argumentam que seu método de solução é facilmente adaptável às diversas variações do VRP, além de ser simples de implementar e capaz de obter boas soluções rapidamente. A meta-heurística proposta usa uma lógica simples para os passos de R&R de uma heurística de melhoria, utilizando um único método de destruição e um único método de reconstrução da solução, diferentemente do que publicações prévias propõem. Segundo eles, a R&R é especialmente relevante para problemas como o abordado nesta tese, em que explorar a vizinhança completa ou um subconjunto dos vizinhos de uma solução requer um tempo computacional muito alto. Para testar a eficácia do método, o algoritmo é encapsulado em um algoritmo de *Simulated Annealing* e a heurística resultante é testada em benchmarks da literatura para 10 variações do PRV, partindo do foco principal deste trabalho: o PRV de frota homogênea, capacitada e ilimitada.

O objetivo por trás de um movimento de destruição proposto é criar dois tipos de folga na solução parcial: de capacidade, ou seja, criar capacidade ociosa nos veículos em uso; e espacial, pelo aumento da área de alcance dos veículos impactados pelo movimento. Tendo estes objetivos em mente, o movimento de destruição remove pedaços das rotas que sejam (geograficamente) próximos entre si. Cada um destes pedaços é uma sequência de clientes visitados consecutivamente pelo mesmo veículo, chamada de vetor (tradução livre de *string*), cuja cardinalidade (número de clientes inclusos no vetor) é crucial para a eficiência do método. Apesar do método de destruição ser único, há duas formas de definir os vetores a remover, adotadas à mesma probabilidade: uma remove uma sequência contínua de nós, a outra remove

uma sequência de nós com uma interrupção intermediária, preservando um subconjunto de nós dentro dessa sequência.

Após a destruição parcial da solução, é empregado um método guloso para reconstruir as rotas, apelidado de Inserção Gulosa com Piscadelas (tradução literal de *Greedy Insertion with Blinks*). Após ordenar o conjunto de nós/clientes removidos, a cada iteração é selecionado o primeiro cliente da lista e são examinadas todas as rotas que possuam folga suficiente para acomodá-lo. Para cada uma destas rotas, calcula-se o custo incremental da inserção do cliente em cada posição da rota e, ao fim do processo, o cliente é inserido naquela de menor custo. Contudo, o algoritmo avalia apenas uma parte das posições de inserção, sorteando se avaliará ou não cada uma delas de acordo com uma probabilidade, denominada de *blink rate*. Esta taxa de piscadelas é adotada porque os testes computacionais executados pelos autores os levaram a concluir que inserir cada cliente sempre na melhor posição possível não leva a melhores soluções. Caso não seja possível inserir o cliente em nenhuma das rotas existentes, adiciona-se um veículo à solução, alocado para atender o cliente em questão.

2.6 CONCLUSÕES DA REVISÃO DA LITERATURA

Na literatura de PRV, as classes de problemas que mais se aproximam do problema tratado aqui são as PRVJT-Sin, que incluem restrições de sincronização de recursos e janelas de tempo. Contudo, em nenhuma das aplicações encontradas há a consideração de uma frota heterogênea, mesmo nos casos em que há preocupação em atender certos clientes usando veículos específicos, por questões de preferência dos clientes. Isso porque, nas poucas publicações que lidam com o PRVJT-Sin, assume-se que qualquer veículo pode atender qualquer cliente. Não obstante, neste trabalho a frota considerada é formada por duas ou mais classes de rebocadores, de potências diferentes, sendo que uma tarefa que pode ser atendida por embarcação de determinada classe, também pode ser atendida por embarcação de classe superior (se houver). Desta forma, a demanda de uma tarefa não está associada a um único conjunto de recursos, podendo ser atendida por vários conjuntos diferentes, mas não por qualquer combinação de rebocadores. Além disto, no problema tratado aqui não há restrições de precedência entre as tarefas.

Se por um lado a existência de múltiplos modos de atendimento de cada tarefa diferencia o problema estudado do PRVJT-Sin, ela o aproxima de problemas de

Programação de Máquinas Paralelas com Recursos Flexíveis, também definido neste capítulo. Porém, no PMFRS o tempo de execução da atividade é proporcional à quantidade de recursos extra empregados e existe uma máquina que necessariamente é usada na execução de cada tarefa, portanto apenas os recursos adicionais são considerados flexíveis. Ressalta-se que, ainda assim, as abordagens já publicadas para resolução do PMFRS raramente tratam simultaneamente a designação da máquina que processará cada tarefa e outras especificidades do problema que aumentam sua complexidade, como por exemplo a variabilidade do tempo de execução da tarefa em função da quantidade de recurso adicional alocado.

Para criar um movimento de inserção de tarefa nas rotas dos rebocadores, usado na heurística construtiva, também foram consultados trabalhos de *scheduling* que abordam o problema de inserção de tarefa em programa pronto, tanto nas áreas de programação de tarefas como de projetos. Naquele contexto, o problema de inserção é modelado por si só, inspirado no problema real do surgimento de uma nova tarefa não prevista anteriormente, ou seja, quando o cronograma em curso foi gerado. Ressalta-se que, apesar dos modelos de inserção já publicados não lidarem com todas as características que diferenciam o problema abordado aqui dos demais, a literatura mostra que a inserção de uma tarefa em cronograma é um problema tão complexo que mesmo a obtenção de uma solução viável exige o uso de métodos heurísticos (GRÖFLIN; KLINKERT; DINH, 2008).

O problema abordado nesta pesquisa é NP-difícil, uma vez que generaliza o Problema de Roteirização de Veículos com Janelas de Tempo, comprovadamente *NP-difícil* (AIT HADDADENE; LABADIE; PRODHON, 2016). Mais especificamente, no caso especial em que cada operação demandada exige um único rebocador específico, o problema tratado nesta pesquisa torna-se equivalente ao PRVJT.

Em problemas NP-difíceis, a obtenção de soluções ótimas mesmo para pequenas instâncias pode ser inviável, exigindo alto tempo computacional. Nestes casos, torna-se necessário o desenvolvimento de métodos de resolução por aproximação, que sejam capazes de chegar em boas soluções em tempo reduzido. Neste sentido, meta-heurísticas vêm sendo empregadas com sucesso para resolver este tipo de problema, de natureza combinatória (DELORME; IORI; MARTELLO, 2016).

As meta-heurísticas aplicadas na solução de problemas similares ao tratado aqui utilizam algoritmos gulosos para gerar suas soluções iniciais. Em alguns casos, é permitido romper janelas das tarefas, por isso também é proposto um procedimento de

reparo da solução, para torná-la viável. Como apontado por Ait Haddadene, Labadie e Prodhon (2016), a roteirização de veículos com operações sincronizadas vem recebendo mais atenção nos últimos anos, mas ainda carece de método de solução eficiente. No método proposto por aqueles autores, os movimentos utilizados para modificar uma solução viável lidam com uma posição de uma rota e não com a tarefa, ou seja, se a tarefa que ocupa a posição em questão também aparecer em outra rota, o movimento tentará mexer a tarefa apenas na rota selecionada e, caso isso não seja viável devido à interdependência entre rotas, o movimento é simplesmente descartado. Esta abordagem produz bons resultados para o problema tratado pelos autores, provavelmente porque a quantidade de tarefas sincronizadas é pequena em relação à demanda total, representando cerca de 10% das tarefas. Assim, a chance de o movimento ser descartado é muito menor do que seria no problema tratado nesta pesquisa, em que quase todas as tarefas exigem mais de uma embarcação. Por isso, no presente trabalho propõe-se o uso de um movimento que altera soluções viáveis pela remoção e reinserção de tarefas nas rotas das embarcações disponíveis, independentemente do número de embarcações envolvido.

Dos trabalhos que abordam os problemas mais similares com o desta pesquisa, apresentados na seção 2.3, nenhum deles possui mais de uma classe ou tipo de veículo, como é o caso do problema de roteirização de rebocadores AHTS. Além disto, o número de tarefas que exigem a atuação simultânea de mais de um recurso é relativamente pequeno, e no máximo são requeridos 2 veículos por tarefa. A Tabela 1 apresenta os valores médios do percentual de clientes com restrição de sincronização ($sincr./n$ [%]), da quantidade média ($\overline{Q}_i$) e máxima ($\max Q_i$) de recursos por tarefa, além da quantidade de clientes.

Tabela 1 – Parâmetros-chaves das instâncias da literatura e das propostas neste trabalho.

Parâmetro	n	$sincr./n$ [%]	Classes da frota (c)	$\overline{Q}_i$	$\max Q_i$
Parragh e Doerner (2018)	20 a 80	10%	1	<2	2
Hojabri <i>et al.</i> , (2018)	25 a 200	5, 25 e 50%	1	<2	2
Ait Haddadene, Labadie e Prodhon (2016)	20 a 80	10%	1	<2	2
Rousseau, Gendreau e Pesant, (2013)	100 a 1.000	10%	1	<2	2
Liu, Tao e Xie (2019)	30 a 200	10, 30 e 50%: 30 clientes 5 a 20%: 100 e 200 clientes	1	<2	2

As instâncias propostas neste trabalho (vide seção 6.1) possuem 80 a 90% de tarefas com restrição de sincronização. Ainda, até 4 rebocadores podem ser exigidos simultaneamente. Isto significa que neste trabalho a interdependência entre rotas é maior, o que desafia os métodos de soluções propostos por outros autores e resumidos na seção 2.5.

Finalmente, conclui-se que o problema abordado nesta tese se difere dos encontrados na literatura por unir as seguintes particularidades:

- É um problema de roteirização com frota heterogênea, em que os veículos são subdivididos em classes, sendo que dentro da mesma classe os veículos são equivalentes (mesma velocidade de navegação) e intercambiáveis entre si;
- Há restrições de compatibilidade, informando se a classe é compatível com a tarefa e, caso seja, é possível substituir qualquer veículo desta classe por outro de uma classe superior;
- Possui restrições de sincronização, que exigem que mais de um rebocador esteja presente para a execução da tarefa;
- A maioria das tarefas exige mais de um veículo (80 a 90% das tarefas requer 2 ou mais rebocadores) e até quatro rebocadores podem ser necessários.

Sendo assim, é preciso propor um modelo matemático que atenda estas características do problema, tendo como base os modelos já publicados para trabalhos semelhantes (vide seção 2.4); e desenvolver método de solução para ele.

3 DESENVOLVIMENTO DO MODELO MATEMÁTICO DE OTIMIZAÇÃO

Nesta pesquisa, é abordado um problema real da logística *offshore* da indústria de exploração de petróleo: o planejamento do uso da frota de rebocadores, um tipo específico de embarcação conhecido como AHTS. Há dois fatores que aumentam a complexidade do problema: primeiro, as operações demandadas podem exigir a alocação simultânea de mais de um rebocador, por questões de segurança ou pela complexidade da operação; segundo, a frota é heterogênea e dividida em classes, sendo que rebocadores de uma classe superior sempre podem substituir rebocadores da classe imediatamente inferior. Além disto, todas as tarefas possuem uma data de início ideal, segundo o cronograma das atividades da plataforma, e seu atraso prejudica atividades subsequentes, à exemplo do que ocorre em alguns problemas de *scheduling*.

Para se referir ao fato de que há várias formas de atender as tarefas, devido ao fato de rebocadores de mesma classe serem equivalentes entre si e poderem substituir rebocadores de classes inferiores, convencionou-se dizer que há múltiplos modos de atender as tarefas. Destaca-se que a existência de **múltiplos modos**¹ de execução significa não só que existe mais de um conjunto de embarcações capaz de realizar uma mesma tarefa, como que tais conjuntos podem ter composições diferentes em termos da quantidade de rebocadores de cada tipo. Neste texto, denomina-se **configuração** a descrição de um conjunto de rebocadores que usa as suas identificações, ou seja, que enumera os IDs das embarcações designadas para atender uma tarefa; em oposição ao termo **modo**, que descreve o subconjunto de rebocadores apenas pela quantidade de embarcações de cada classe (ou seja, cada tipo). Ou seja, duas configurações de atendimento de uma tarefa podem ser do mesmo modo, caso a quantidade de embarcações de cada classe seja igual nas duas configurações, mas ao menos uma das embarcações de uma não esteja presente na outra.

Para serviços demandados por plataformas de produção, o atraso da chegada do AHTS pode causar uma interrupção no serviço, gerando um prejuízo proporcional ao tempo de parada. Por sua vez, os atrasos nas operações das unidades de perfuração são penalizados pelos custos de oportunidade, dados pelas taxas diárias de afretamento das sondas. Por outro lado, dependendo da tarefa, é possível adiantar sua execução

¹ Ao longo do texto, estão destacados em negrito os termos utilizados pela autora para se referir a aspectos importantes do problema, sempre da primeira vez em que são mencionados. Assim, o leitor poderá facilmente encontrar o seu significado, caso precise voltar a este capítulo futuramente.

em alguns dias sem prejuízo para a operação. Assim, nas instâncias deste problema, cada tarefa possui uma **data prevista** (ou data ideal) para sua execução e uma janela de dias em que é viável realizá-la. Dentro desta janela, cada dia após a data ideal é considerado um dia de **atraso** no início da execução da tarefa, sobre o qual incorre os custos de interrupção ou de oportunidade, a depender do tipo da atividade.

Mas os custos relacionados à gestão da frota não se devem apenas aos atrasos, como também ao consumo de combustível e outros custos associados ao uso da embarcação. Sendo assim, um cronograma de realização das tarefas pode ser avaliado quanto aos **custos operacionais** e de **penalidade** (por atraso na realização das tarefas). Em outras palavras, neste caso a função objetivo possui múltiplos critérios, sendo um deles a penalidade por atraso de cada tarefa.

Neste capítulo, são apresentadas possíveis formulações matemáticas para o problema de roteirização de frota heterogênea de rebocadores, para execução de tarefas associadas a janelas de tempo, sincronização de operações, penalidade por atraso e múltiplos modos de execução. Assim como os modelos encontrados na literatura (vide seção 2.4), as formulações apresentadas nas próximas seções são definidas em um grafo direcional, composto por nós (ou vértices), que representam as tarefas a realizar, e por arcos, que representam a locomoção dos recursos (rebocadores AHTS) entre a realização das tarefas. Na formulação tradicional, o grafo é definido para cada rebocador, ou seja, a variável de decisão é indexada pelo arco e pelo identificador da embarcação. Na formulação alternativa, desenvolvida nesta pesquisa, o grafo é definido para cada classe de rebocador, resultando em uma redução considerável do tamanho do problema. Esta formulação inspirou-se em uma estratégia de modelagem publicada recentemente para o problema de coleta e entrega (vide seção 3.3), sendo adaptada para o problema de roteirização de rebocadores. Por se tratar do mesmo problema, as definições de conjuntos e nomenclaturas adotadas foram padronizadas e estão apresentadas a seguir, na seção 3.1.

3.1 DEFINIÇÕES GERAIS

Dado um conjunto J de n tarefas demandadas e um conjunto $\mathcal{M}$ de m rebocadores, necessários à realização das tarefas, deseja-se criar um programa de atendimento que contemple a execução de todas as tarefas com o menor orçamento possível, respeitando algumas restrições.

Neste problema, os rebocadores deverão iniciar suas rotas a partir de um local previamente conhecido (que pode tanto ser o local onde o rebocador realizou seu último serviço, como um cais onde serviços de manutenção são executados), e precisam se deslocar aos locais em que cada tarefa precisa ser realizada, já que tarefas representam serviços de atendimento a plataformas *offshore*. Algumas tarefas possuem local de início e fim diferentes e a maioria delas exige a atuação simultânea de mais de uma embarcação.

O início da execução de cada uma das tarefas $i \in \mathcal{J}$ está restrito a uma **janela de tempo** rígida, representada por $[a_i, b_i]$, tal que a_i é o instante (ou data) mais cedo em que a atividade pode ser iniciada, enquanto b_i é o instante a partir do qual a tarefa não pode mais ser iniciada. Há também uma data desejada para o início da tarefa, d_i , a partir da qual há uma penalidade financeira por cada dia de atraso, denotada por β_i . Desta forma, o custo pago pelo atraso de uma tarefa é proporcional ao tamanho deste atraso (diferença positiva entre início da execução da tarefa e b_i). Cada tarefa possui uma **duração** D_i , que inclui eventuais deslocamentos entre seus locais de início e fim, de forma que os rebocadores alocados para sua execução só serão liberados para as próximas atividades após sua conclusão. A **demand** é definida pelo vetor $\bar{Q}_i = \{q_i^1 \dots q_i^c\}$, que expressa o **modo básico** de atendimento da tarefa, sendo q_i^s a quantidade mínima de rebocadores classe s necessários para a execução de i e Q_i a quantidade total de rebocadores necessários, $Q_i = \sum_{s \in \mathcal{S}} q_i^s$.

Como as tarefas estão associadas a locais geográficos de início e fim, são conhecidas as distâncias que um rebocador qualquer precisará navegar, se for selecionado para a execução consecutiva de duas tarefas. Assim, são dadas as distâncias de navegação para realizar a tarefa j logo depois da i , $Dist_{ij}$, e para realizá-las na ordem inversa, $Dist_{ji}$. Ressalta-se que, como o local de início e fim de uma mesma tarefa podem ser diferentes, as distâncias $Dist_{ij}$ e $Dist_{ji}$ não são necessariamente iguais.

Cada rebocador $r \in \mathcal{M}$ pertence a uma classe s_r e possui uma **data de liberação**, R_r , antes da qual não podem iniciar nenhum percurso. O conjunto de embarcações é heterogêneo, subdividindo-se em c subconjuntos homogêneos. Cada subconjunto $\mathcal{M}^s | \mathcal{M}^s \subseteq \mathcal{M}$ é denominado **classe** e designado pelo índice $s = \{1 \dots c\}$, sendo que o conjunto de todas as classes é designado por $\mathcal{S}$. Ou seja, cada classe $s \in \mathcal{S} = \{1 \dots c\}$ possui m_s rebocadores equivalentes entre si, que navegam à mesma velocidade v^s , e possuem o mesmo custo por unidade de distância percorrida, CV^s . Quanto menor o

valor de s , menor a potência da embarcação, por isso, à exceção da última classe de embarcações $s = c$, uma embarcação da classe s sempre pode ser substituída por embarcação da classe $s' > s$. Desta forma, define-se também o subconjunto de navios compatíveis com a classe s , $\tilde{\mathcal{M}}^s \subseteq M$, que contém todos os navios das classes $s' \geq s$, $\forall s \in \{1, 2 \dots c\}$.

Neste texto, o problema assim descrito é denominado **Problema de Roteirização de Veículos com Janelas de Tempo, Sincronização de Recursos e frota Heterogênea (PRVJTSin-Het)**, usando termos já adotados para a classe de Problemas de Roteirização de Veículos, à qual o PRVJTSin-Het pertence.

3.2 FORMULAÇÃO CLÁSSICA, COM INDEXAÇÃO POR REBOCADOR

A representação do problema é feita em um grafo $G_{AHTS} = (V, \mathcal{A})$, da seguinte forma:

- Cada tarefa i é representada por um único nó, portanto o conjunto de todos os nós também é representado por $\mathcal{J}$.
- O início e fim das rotas de cada embarcação $r \in \mathcal{M}$ é representado por um par de nós equivalentes: um que tem apenas arcos saindo de si, O_r^+ , e um que apenas recebe arcos, O_r^- . O nó O_r^+ representa o local onde o rebocador r fica disponível a partir de sua data de liberação. Já o nó O_r^- representa o local para onde este rebocador deve se deslocar ao final do percurso. Nas instâncias estudadas neste trabalho, estes nós finais são fictícios, ou seja, não estão atrelados a uma localização geográfica e servem apenas para representar o final de uma rota, garantindo a continuidade das mesmas (para satisfazer às restrições impostas no modelo matemático). O conjunto de todos os nós de origem das embarcações é representado por $\mathcal{O}^+ = \{O_1^+, O_2^+ \dots O_m^+\}$, enquanto os que representam o destino final é representado por $\mathcal{O}^- = \{O_1^-, O_2^- \dots O_m^-\}$.
- O conjunto de todos os nós do grafo G_{AHTS} é definido como $V := \mathcal{O}^+ \cup \mathcal{J} \cup \mathcal{O}^-$.
- Para representar a execução consecutiva das duas tarefas por um mesmo recurso, todos os pares de tarefas i, j (com $i \in \mathcal{J}, j \in \mathcal{J}$ e $i \neq j$) definem um arco iniciado em i e terminado em j (de forma que $(i, j) \neq (j, i)$). O conjunto de todos os arcos assim formados é representado por $\mathcal{A}$.

- Além disso, são definidos arcos iniciados em $\mathcal{O}^+$ e incidentes em $\mathcal{J}$, e arcos iniciados em cada nó de $\mathcal{J}$ e incidentes em $\mathcal{J}^-$. Os subconjuntos de tais arcos é representado por A^+ e A^- , respectivamente.
- São também definidos arcos fictícios (ou virtuais) para representar rotas nulas. Tais arcos conectam todo par de nós $\mathcal{O}_r^+$ e $\mathcal{O}_r^-$, para todo $r \in \mathcal{M}$, sendo seu conjunto representado por $A^* = \{(\mathcal{O}_1^+, \mathcal{O}_1^-) \dots (\mathcal{O}_r^+, \mathcal{O}_r^-) \dots (\mathcal{O}_m^+, \mathcal{O}_m^-)\}$.
- O conjunto de arcos do grafo é denotado por $\mathcal{A} := A^+ \cup A \cup A^- \cup A^*$.

Levando em conta o grafo assim definido, definem-se os seguintes parâmetros:

- O tempo necessário para cada rebocador r percorrer o arco (i, j) , T_{ij}^r , obtido pela divisão da distância $Dist_{ij}$ pela velocidade da embarcação r , ou seja, v^{sr} .
- O custo para o rebocador r percorrer o arco (i, j) , CV_{ij}^r , igual ao tempo de percurso T_{ij}^r multiplicado pelo custo variável da classe do rebocador, CV^{sr} .
- O horizonte de planejamento é representado por H .

Uma solução para o problema é uma seleção de arcos em G_{AHTS} que atribui valores às seguintes variáveis:

- x_{ij}^r , binária que vale 1 se o rebocador $r \in \mathcal{M}$ percorrer o arco $(i, j) \in A$ em sua rota; e 0 caso contrário;
- $\hat{s}_i^r$, instante de chegada do navio $r \in \mathcal{M}$ ao nó $i \in \mathcal{J}$, considerando tarefas atribuídas à embarcação r e realizadas antes de j ;
- s_i , instante de início da tarefa $i \in \mathcal{J}$;
- l_i , atraso da tarefa $i \in \mathcal{J}$, medido em relação a b_i .

A função objetivo z é composta de duas parcelas: a dos custos de deslocamento (proporcionais à duração total da rota percorrida); e a dos custos referentes à penalidade por atraso na execução das tarefas. Assim, tem-se:

$$\min z(S) = \sum_{i \in \mathcal{J}} \beta_i l_i + \sum_{i \in V \setminus \mathcal{O}^-} \sum_{j | (i,j) \in A} \sum_{r \in \mathcal{M}} CV_{ij}^r x_{ij}^r \quad \text{Eq. 3-1}$$

Na expressão (1), a função $z(S)$ define o custo total de uma solução S , viável. A primeira parcela da função $z(S)$ corresponde ao custo total de penalização por atraso no início das tarefas; enquanto a segunda parcela corresponde ao custo de deslocamento. Cabe

dizer que dependendo do modelo de custos para uso da frota, esta função objetivo pode ser alterada. Por exemplo, é possível que se deseje diferenciar o tempo que as embarcações passam navegando, ociosas (aguardando outros rebocadores chegarem ao local de início de uma tarefa) ou executando uma tarefa. Isto exigiria a criação de variáveis auxiliares para computar tais custos, mas pode ser feito sem alterações significativas no modelo.

Uma solução ótima S^* possui o menor valor possível $z^* = z(S^*)$. Qualquer solução viável deve, além de atribuir valores às variáveis de decisão definidas anteriormente, atender às seguintes restrições:

$$\sum_{j \in \mathcal{J} \cup \mathcal{O}_r^-} x_{o_r^+ j}^r = 1 \quad \forall r \in \mathcal{M} \quad \text{Eq. 3-2}$$

$$\sum_{j \in \mathcal{J} \cup \mathcal{O}_r^+} x_{j o_r^-}^r = 1 \quad \forall r \in \mathcal{M} \quad \text{Eq. 3-3}$$

$$\sum_{r \in \mathcal{M}^s} \sum_{i \in \{V \setminus \mathcal{O}^+ \mid (i,j) \in A\}} x_{ij}^r \geq q_j^s \quad \forall j \in \mathcal{J}, \forall s \in \mathcal{S} \quad \text{Eq. 3-4}$$

$$\sum_{r \in \mathcal{M}} \sum_{i \in \{V \setminus \mathcal{O}^+ \mid (i,j) \in A\}} x_{ij}^r \geq Q_j \quad \forall j \in \mathcal{J} \quad \text{Eq. 3-5}$$

$$\sum_{i \in \{V \mid (i,j) \in A^+ \cup A\}} x_{ij}^r = \sum_{i \in \{V \mid (j,i) \in A \cup A^+\}} x_{ji}^r \quad \forall j \in \mathcal{J}, \forall r \in \mathcal{M} \quad \text{Eq. 3-6}$$

$$\hat{s}_i^r \geq R^r + T_{o_r^+ i}^r - (1 - x_{o_r^+ i}^r) \times H \quad \forall i \in \mathcal{J}, \forall r \in \mathcal{M} \quad \text{Eq. 3-7}$$

$$\hat{s}_j^r \geq s_i + D_i + T_{ij}^r - (1 - x_{ij}^r)(b_i - D_i - T_{ij}^r) \quad \forall r \in \mathcal{M}, \forall (i,j) \in A \quad \text{Eq. 3-8}$$

$$s_i \geq \hat{s}_i^r \quad \forall i \in \mathcal{J}, \forall r \in \mathcal{M} \quad \text{Eq. 3-9}$$

$$s_i \geq a_i \quad \forall i \in \mathcal{J} \quad \text{Eq. 3-10}$$

$$s_i \leq b_i \quad \forall i \in \mathcal{J} \quad \text{Eq. 3-11}$$

$$l_i \geq s_i - b_i \quad \forall i \in \mathcal{J} \quad \text{Eq. 3-12}$$

$$\begin{aligned} x_{ij}^r &\in \{0,1\} \quad \forall (i,j) \in \mathcal{A}, r \in \mathcal{M} \\ s_i, l_i &\in \mathbb{R}^+ \quad \forall i \in \mathcal{J} \\ \hat{s}_i^r &\in \mathbb{R}^+ \quad \forall i \in \mathcal{J}, \forall r \in \mathcal{M} \end{aligned} \quad \text{Eq. 3-13}$$

O conjunto de restrições apresentado na equação Eq. 3-2 define que toda rota deve ser iniciada no seu depósito de origem, conectando-o a apenas um outro nó, mesmo que tal nó seja o depósito final, caso a embarcação não seja utilizada no programa. Similarmente, o conjunto Eq. 3-3 especifica que toda rota deve terminar no depósito final O_r^- , partindo de um (e apenas um) outro nó da rede, que pode ser o inicial se a embarcação não for utilizada. As inequações referenciadas como Eq. 3-4 fazem com que, para cada tarefa e cada classe, a quantidade de rebocadores de classe igual ou superior designados para atendê-la deve ser maior ou igual à quantidade demandada pela tarefa para aquela classe. Observa-se que tais restrições permitem que um mesmo rebocador alocado seja contado duas vezes para a mesma tarefa, uma vez na restrição correspondente à sua própria classe e outra na restrição da classe imediatamente inferior. Então, para garantir que o total de embarcações alocadas é suficiente para atender a demanda da tarefa, as inequações Eq. 3-5 garantem que o total de rebocadores alocados seja igual ou superior à quantidade demandada – como o problema minimiza custos, a solução naturalmente terá a quantidade de embarcações alocadas igual à demanda. Por sua vez, o conjunto de equações Eq. 3-6 garante o balanço de cada nó do conjunto $\mathcal{J}$, ou seja, dos nós que representam tarefas e não o início e fim das rotas, pela restrição de que a seleção de arcos deve ser tal que a quantidade de arcos incidentes na tarefa é igual à quantidade de arcos iniciados na tarefa, para cada rebocador do conjunto.

Para atribuir um valor de início da atividade, é preciso levar em conta a janela de tempo dinâmica em tarefas com sincronia de operações, já que seu início só pode ocorrer após a chegada de todas as embarcações necessárias. Assim, as restrições representadas em Eq. 3-7 definem o instante de chegada de cada embarcação à primeira tarefa de sua respectiva rota, atribuindo o valor da data de liberação da embarcação mais o tempo de deslocamento entre sua origem e o local de início da primeira atividade. Similarmente, as inequações Eq. 3-8 definem que, para qualquer tarefa j que não seja a primeira de uma rota r , o início de sua execução é limitado pelo instante de finalização da tarefa anterior a ela, i , mais o tempo de deslocamento entre i e j pelo rebocador r . Então, a consideração da janela dinâmica é garantida pelas inequações Eq. 3-9, que fazem com que o início de uma tarefa seja igual ou posterior ao instante de chegada de cada uma das embarcações a ela alocadas. Além disto, as restrições Eq. 3-7 e Eq. 3-8 vinculam as variáveis de decisão x_{ij}^r e $\hat{s}_j^r$, enquanto as

restrições Eq. 3-9 vinculam $\hat{s}_j^r$ e s_j . Para cumprir as janelas de tempo vinculadas às tarefas, Eq. 3-10 indica que o início de uma tarefa é no mínimo igual ao início de sua janela hard, a_i , enquanto Eq. 3-11 garante que o início seja anterior ou igual ao limitante superior da janela hard, b_i . Já o atraso da tarefa, utilizado no cálculo da penalidade, é calculado graças ao conjunto de inequações Eq. 3-12. O último conjunto de restrições define o domínio das variáveis de decisão.

3.2.1 Versão reduzida do problema, com pré-processamento dos dados da instância

Devido à existência de janelas de tempo para o atendimento das demandas nas plataformas, possivelmente há sobreposição temporal de janelas de dois ou mais pedidos diferentes, de forma que tais tarefas não possam ser executadas pelo mesmo rebocador. Aliás, mesmo que não haja sobreposição das janelas, como as tarefas possuem determinada duração e podem estar associadas a plataformas diferentes, é possível que duas tarefas não possam pertencer à mesma rota.

Esta característica faz surgir a possibilidade de, ao invés de definir arcos ligando todos os pares de pontos do grafo que representa o problema, sejam apenas criados os arcos que representem associação viável entre os pares de tarefa. Assim, é possível que exista o arco (i, j) mas não exista a representação do caminho inverso, (j, i) ; ou ainda, que não haja nenhum dos dois arcos.

Usando esta estratégia de redução do problema, não são criadas variáveis x_{ij}^r caso o arco (i, j) represente uma sequência de tarefas que não possa ser executada pela embarcação $r \in \mathcal{M}$. O grafo adotado para o problema é $G' = (V, \mathcal{A}')$, com $\mathcal{A}' := A^+ \cup A' \cup A^- \cup A^*$. O novo conjunto de arcos, $A' \subset A$, é formado apenas por pares de tarefas que possam ser executadas pelo mesmo rebocador naquela ordem (primeiro a tarefa i e depois a j , para o arco (i, j)), considerando a embarcação AHTS mais veloz da frota. Assim, a formulação matemática é idêntica à apresentada anteriormente, porém possui uma quantidade inferior de variáveis e de restrições.

3.3 FORMULAÇÃO COMPACTA, COM INDEXAÇÃO POR CLASSE

Inspirada nas formulações compacta propostas por Lu e Dessouky (2004) e Furtado, Munari e Morabito (2017) para o problema de roteirização com coleta e entrega, a estratégia de modelagem a seguir utiliza nós virtuais para simbolizar o início da rota de cada embarcação e variáveis x_{ij}^s , ou seja, indexadas por classe da embarcação, s .

Originalmente, o modelo foi construído sem pré-processamento dos dados da instância, com arcos ligando todos os pares de pontos, nos dois sentidos. Por sugestão de participantes de um congresso internacional, onde a primeira versão do modelo foi apresentado, uma versão reduzida do modelo foi criada, utilizando o pré-processamento de dados para remover arcos inviáveis (novamente, usando a velocidade das embarcações de classe mais alta, que são as mais velozes da frota). O modelo resultante também é definido em um grafo direcional, $G = (V', \mathcal{A}')$, formado pelos seguintes conjuntos e parâmetros:

- Assim como no modelo tradicional, cada tarefa i é representada por um único nó, portanto o conjunto de todos os nós também é representado por $\mathcal{J}$.
- Conjunto de m nós representando o início da rota de cada um dos rebocadores, indexados como $i \in \mathcal{O} = \{n + 1, n + 2 \dots n + m\}$. Cada nó $i \in \mathcal{O}$ possui uma janela de tempo $[a_i, b_i]$ associada a si, sendo seu início dado pela data de liberação do rebocador, ou seja $a_i = R_{i-n}$, e seu fim igual ao horizonte de planejamento do problema $b_i = H$. Os demais parâmetros listados para os nós de tarefas ($i \in \mathcal{J}$) são zerados, ou seja, $d_i = \beta_i = D_i = 0 \forall i \in \mathcal{O}$.
- Nó fictício para representar o final de todas as rotas, $i = n + m + 1$, com janela de tempo $[a_i, b_i] = [0, H]$.
- $V' = \mathcal{O} \cup \mathcal{J} \cup \{n + m + 1\}$ é o conjunto completo de nós (ou vértices) do grafo.
- O conjunto de arcos do grafo, $\mathcal{A}'$, é formado apenas por pares de nós $(i, j): i \in \mathcal{O} \cup \mathcal{J}, j \in \mathcal{J} \cup \{n + m + 1\}, i \neq j$, tais que seja factível executar a tarefa j logo após a tarefa i sem romper a janela de tempo de j , considerando a velocidade da embarcação mais rápida disponível, $\max\{v^s\}$. Como a distância entre as duas tarefas é conhecida e as velocidades e custos dos rebocadores são dados por classe, são calculados os seguintes parâmetros para cada arco: um vetor de distâncias associado, $\bar{T}_{ij} = \{T_{ij}^1 \dots T_{ij}^c\}$ sendo T_{ij}^s o tempo que um rebocador de classe s leva para navegar da plataforma (local) final da tarefa i até o local inicial da tarefa j ; e um vetor de custos, $\bar{CV}_{ij} = \{CV_{ij}^1 \dots CV_{ij}^c\}$.

As variáveis de decisão usadas no modelo são:

- x_{ij}^s : variável inteira, que indica o número de rebocadores de classe s a trafegar no arco i, j .

- y_{ij}^s : variável binária que indica se ao menos um rebocador de classe s percorre o arco (i, j) .
- t_i : variável contínua que indica a data de início da tarefa i .
- l_i : variável contínua que equivale ao atraso (a ser penalizado na função objetivo) no início da tarefa i .

A função objetivo é dada por:

$$\min z(S) = \sum_{j \in J} \beta_j l_j + \sum_{s \in S} \sum_{j \in J} \sum_{i \in J \cup \emptyset: (i,j) \in \mathcal{A}'} x_{ij}^s CV_{ij}^s \quad \text{Eq. 3-14}$$

Uma solução ótima S^* possui o menor valor possível $z^* = z(S^*)$. Qualquer solução viável deve, além de atribuir valores às variáveis de decisão definidas anteriormente, atender às seguintes restrições:

$$\sum_{j \in J \cup \{n+m+1\}} x_{ij}^{s_{i-n}} = 1 \quad \forall i \in \mathcal{O} \quad \text{Eq. 3-15}$$

$$\sum_{i \in J \cup \emptyset} x_{i,n+m+1}^s = m_s \quad \forall s \in S \quad \text{Eq. 3-16}$$

$$\sum_{i \in J \cup \emptyset} x_{ij}^s = \sum_{k \in J \cup \{n+m+1\}} x_{jk}^s \quad \forall s \in S, j \in J \quad \text{Eq. 3-17}$$

$$\sum_{s \in S: s \geq s'} \sum_{i \in J \cup \emptyset} x_{ij}^s \geq q_j^{s'} \quad \forall s' \in S, j \in J \quad \text{Eq. 3-18}$$

$$\sum_{s \in S} \sum_{i \in J \cup \emptyset} x_{ij}^s = Q_j \quad \forall j \in J \quad \text{Eq. 3-19}$$

$$y_{ij}^s \geq \frac{x_{ij}^s}{Q_j} \quad \forall s \in S, j \in J, i \in \mathcal{O} \cup J: (i,j) \in \mathcal{A}' \quad \text{Eq. 3-20}$$

$$\begin{aligned} t_j &\geq a_i - H + y_{ij}^{s_{i-n}} (T_{ij}^{s_{i-n}} + H) \quad \forall i \in \mathcal{O}, j \in J: (i,j) \in \mathcal{A}' \\ t_j &\geq t_i - b_i + y_{ij}^s (T_{ij}^s + b_i + D_i) \quad \forall s \in S, i, j \in J: (i,j) \in \mathcal{A}' \end{aligned} \quad \text{Eq. 3-21}$$

$$a_i \leq t_i \leq b_i \quad \forall i \in J \quad \text{Eq. 3-22}$$

$$l_i \geq t_i - d_i \quad \forall i \in J \quad \text{Eq. 3-23}$$

$$\begin{aligned}
 x_{ij}^s &\in \mathbb{N} \\
 y_{ij}^s &\in \{0,1\} \\
 t_i, l_i &\in R^+
 \end{aligned}
 \tag{Eq. 3-24}$$

Os primeiros três conjuntos de restrições (Eq. 3-15 a 3-17) garantem que as rotas sejam contínuas, começando nos nós de origem de cada rebocador e terminando no nó final: exatamente um arco saindo de cada nó de origem deverá ser selecionado, a quantidade de rebocadores (de cada classe) que chegam a um nó de tarefa precisa ser idêntica à quantidade que sai dele, e a frota completa de cada classe deverá se locomover ao nó final. As restrições expressas em Eq. 3-18 e 3-19 garantem que a demanda de cada tarefa seja atendida, considerando as classes dos rebocadores alocados para sua execução. A conexão entre as variáveis x_{ij}^s e y_{ij}^s é feita na Eq. 3-20. A data de início de cada atividade depende da chegada de todos os rebocadores alocados a si, o que é garantido pelas duas inequações identificadas como Eq. 3-21 (a primeira refere-se aos arcos iniciados em $i \in \mathcal{O}$, enquanto a segunda trata dos demais arcos). Estas inequações garantem que não há formação de subciclos nas rotas, já que a data de início de cada tarefa depende da finalização da tarefa imediatamente anterior a ela, em todas as rotas que a atendam. A janela de tempo de cada tarefa é imposta por Eq. 3-22 e o cálculo do atraso, vinculando as variáveis t_i e l_i , é feito em Eq. 3-23. Por fim, o domínio das variáveis de decisão é declarado em Eq. 3-24.

4 PROPOSIÇÃO DE HEURÍSTICA CONSTRUTIVA

Por causa do grau de complexidade do problema tratado, a geração de uma solução inicial viável exige uma heurística construtiva elaborada, assim como a proposta por Ait Haddadene, Labadie e Prodhon (2016). Conforme apresentado no capítulo de revisão da literatura, tais autores desenvolveram um método de solução para um problema de roteirização similar, com sincronia de operações e janelas de tempo, mas com restrições de precedência e sem múltiplos modos de atendimento de cada tarefa.

Este capítulo apresenta a estrutura do método heurístico utilizado para construir uma solução viável do problema de roteirização de rebocadores para atendimento de plataformas offshore.

Nos pseudocódigos e equações apresentadas neste capítulo, além dos parâmetros e variáveis definidas no capítulo anterior, são utilizados os termos e conceitos definidos a seguir, na seção 4.1. O histórico de desenvolvimento da heurística está documentado no APÊNDICE A -Versões anteriores da Heurística Construtiva.

4.1 NOMENCLATURA E DEFINIÇÕES

Antes da explicação do procedimento, é importante definir alguns termos usados no texto. Uma **rota** ou **roteiro** é composta(o) pela sequência de tarefas executadas por uma embarcação, partindo do nó (fictício ou não) que representa o início da rota e tendo como última parada o nó (fictício ou não) que representa o fim. Como o início e fim de cada tarefa da rota estão associados a uma localização no espaço (pois as embarcações devem se deslocar para os locais geográficos de início e fim das tarefas, que possuem janelas de tempo de atendimento), listar uma sequência de tarefas é equivalente a listar uma sequência de posições geográficas que o rebocador percorrerá. Por sua vez, o **programa** de atendimento das tarefas (ou **solução** do problema), é composto(a) pelo conjunto de rotas de todas as embarcações, que contempla todas as atividades previstas, assim como pelas datas de início de cada atividade programada.

Na heurística construtiva, a cada iteração incrementa-se as chamadas **rotas parciais**, que são rotas contendo algumas das atividades demandadas, mas que juntas não formam uma **solução**, por não incluírem todas as tarefas. Similarmente, o **programa** (ou **solução**) **parcial** é definido pelo conjunto de rotas parciais. Nos algoritmos, enquanto uma $rota_k$ é um conjunto ordenado das tarefas executadas pela embarcação

k , na ordem de execução planejada, o conjunto de todas as m rotas é designado simplesmente por *rotas*.

Cabe esclarecer que, neste problema, os termos **tarefa**, **atividade** e **job** são usados para se referir à mesma coisa: operação executada com apoio de rebocadores AHTS, nas plataformas de exploração e produção de petróleo. Tarefas que exigem atuação de mais de um rebocador são chamadas de **tarefas com sincronização**, enquanto refere-se às demais como **tarefas simples**. Ainda, como definido no capítulo anterior, a **configuração** de uma tarefa diz respeito ao conjunto de embarcações designadas para sua execução, enquanto o **modo (de atendimento)** diz respeito à quantidade de rebocadores de cada uma das classes que compõem a configuração. Por exemplo, supondo cenário em que há 5 rebocadores identificados com os números de 1 a 5, sendo os 3 primeiros de classe 1 e os demais de classe 2, uma configuração pode ser definida pelo conjunto de rebocadores $\{1,2,4\}$. O modo desta configuração é $(2,1)$, ou seja, há dois rebocadores da primeira classe e um da segunda. É importante pontuar que mesmo uma tarefa simples pode ter mais de um modo, caso a classe do rebocador demandado não seja a mais alta disponível.

Algumas variáveis e estruturas de dados auxiliares são utilizadas nos algoritmos apresentados neste capítulo e merecem destaque. O instante mais cedo em que uma atividade j pode ser iniciada, levando em conta a rota ou o conjunto de rotas em que ela aparece, é denominado **Earliest Starting time** e designado por ES_j . A soma deste instante mínimo de início com o tempo de processamento D_j resulta no instante mais cedo em que ela pode ser finalizada, denotado por $tConclusao_j$ nos algoritmos. A **folga** (ou **slack time**) de uma tarefa é definida como o máximo atraso possível de seu início, medido em relação ao ES_j da solução corrente (parcial ou não), que não viola as janelas de atendimento nem dela e nem das tarefas afetadas pela mudança na sua data de início. Sua representação nos algoritmos é $folga_j$.

A atividade que precede diretamente a atividade j na rota r é denotada por $prec_j^r$ e a que sucede é suc_j^r . Ademais, o conjunto que armazena os sucessores diretos da tarefa j , qualquer que seja a rota, é denotado por $sucessores(j)$, e o conjunto de sucessores de todas as tarefas é armazenado em um vetor de vetores denotado por $\overline{Sucessores} = \{Sucessores(1) \dots Sucessores(j) \dots Sucessores(n)\}$.

A heurística construtiva faz a construção das rotas paralelamente, já que algumas tarefas são executadas por mais de um rebocador. A cada iteração, tenta-se adicionar

às rotas parciais uma tarefa ainda não roteirizada, selecionando uma configuração compatível com os modos de atendimento aplicáveis à tarefa em questão. Para cada configuração examinada, todas as posições de inserção que não gerem subciclos são levadas em conta. Caso não seja possível fazer a inserção sem romper nenhuma janela de tempo das atividades já roteirizadas, o processo é reiniciado, ou seja, as rotas são todas esvaziadas e tenta-se novamente alocar as tarefas aos rebocadores, usando outra ordem para inserção das tarefas e/ou variando a escolha das configurações, como será explicado mais adiante. Isto é repetido até que uma quantidade pré-determinada de soluções viáveis seja encontrada.

A heurística construtiva é usada para produzir uma solução inicial completa e viável, que pode servir de ponto de partida para uma busca local ou para uma busca regida por meta-heurística, sempre a fim de melhorar a solução obtida. Ainda, pode-se empregar a técnica denominada de **Múltiplos Recomeços** (MR), em que a heurística construtiva é executada diversas vezes e adota-se ou a melhor solução dentre todas as geradas, ou um conjunto diversificado de soluções. Ou seja, dependendo da heurística de melhoria adotada, a aleatoriedade da heurística construtiva pode ser usada para compor uma população de soluções diversificada, que sirva de ponto de partida para o método.

4.2 A HEURÍSTICA CONSTRUTIVA

Nos algoritmos apresentados neste e nos próximos capítulos, os procedimentos que requerem cálculos mais elaborados são realizados por **funções** (ou **métodos**), que utilizam como dados de entrada informações contidas em seus **argumentos**, além de dados definidos na instância do problema. O argumento de uma função é apresentado entre os parênteses que sucedem sua chamada ou definição.

4.2.1 Funcionamento geral da heurística

O procedimento global da heurística construtiva é apresentado no Pseudocódigo 1. Logo após inicializar as estruturas de dados necessárias, conforme linhas 2 até 4 do pseudocódigo, o processo de construção da solução é iniciado. Este processo, descrito a seguir, é repetido até que o tamanho da população de soluções iniciais seja atingido (*tamPop*) ou que o número mínimo de execuções da heurística tenha sido atingido (*nExp*), desde que ao menos uma solução viável tenha sido encontrada. Ou seja,

mesmo que o contador tenha ultrapassado o número de experimentos, a heurística não é finalizada antes que ao menos uma solução viável tenha sido encontrada.

Após inicializar a estrutura que guardará a solução encontrada, s^0 , são escolhidas as regras de ordenação e de seleção do modo de execução das tarefas (RO e RM , respectivamente), em função da iteração em que a heurística se encontra. Então, é feita a ordenação das tarefas de J , usando a regra selecionada RO , e as tarefas ordenadas são salvas em J' .

Pseudocódigo 1 – Algoritmo da HC implementada

```

1  Algoritmo hConstrutivaGulosa(tamPop, tamListaRestritaJ, tamListaRestritaM, nExp):
2  Lê o conjunto de tarefas demandadas,  $J$ 
3  Cria  $S^0 := \emptyset$ , conjunto para armazenar a população de soluções iniciais
4  contador  $\leftarrow$  1
5  Enquanto  $|S^0| < 1$  OU contador  $\leq$  nExp faça:
6    Inicializa  $s^0$ , solução com rotas vazias
7     $RO \leftarrow$  escolheRegraOrdenacao(contador)
8     $RM \leftarrow$  escolheRegraEscolhaDoModo(contador)
9     $J' \leftarrow$  ordenaTarefas( $J, RO$ )
10   verificaJanelas  $\leftarrow$  OK
11   Enquanto  $|J'| > 1$  E verificaJanelas = OK faça:
12      $k \leftarrow$  escolheTarefa(contador, tamListaRestritaJ,  $J'$ )
13      $J' := J' \setminus \{k\}$ 
14      $s^0 \leftarrow$  fazInsercaoEatualizaParametros( $k, s^0, \text{contador}, \text{tamListaRestritaM}, RM$ )
15     verificaJanelas  $\leftarrow$  verificaJT( $s^0$ )
16   Se  $|J'| = 0$  então:
17      $S^0 \cup \{s^0\}$ 
18   Se  $|S^0| \geq \text{tamPop}$  então: contador  $\leftarrow$  nExp
19   contador  $\leftarrow$  contador + 1
20   Retorna  $S^0$ 
21 Fim

```

Feito isto, inicia-se o processo de construção da solução: uma das tarefas $k \in J'$ é removida do conjunto ordenado e selecionada para ser inserida nas rotas parciais da solução em construção, s^0 . A escolha da tarefa k é feita pelo método **escolheTarefa**, usando uma lista restrita das tarefas de J' , de tamanho *tamListaRestritaJ*. Ou seja, no caso em que *tamListaRestritaJ* = 1, a cada iteração da HC a primeira tarefa de J' é escolhida, sem que haja aleatoriedade nesta decisão. Já se a lista restrita for de tamanho 3, por exemplo, o método **escolheTarefa** sorteará uma das três primeiras tarefas do conjunto ordenado J' . Conhecida a tarefa a inserir e o estado atual da solução, o método **fazInsercaoEatualizaParametros** verifica as opções de inserção de k em s^0 que não gerem subciclo. Usando uma regra de escolha de modos (RM), discutida na seção 4.3), e o tamanho da lista restrita, *tamListaRestritaM*, o método insere k em s^0 , ou seja, escolhe uma configuração de atendimento da tarefa k (subconjunto de rebocadores que seja compatível com a demanda da tarefa) e a coloca em uma posição

das rotas destes rebocadores. Caso esta inserção cause rompimento de alguma(s) das janelas de tempo das tarefas já roteirizadas, a função `verificaJT` retorna uma mensagem negativa e o processo todo é terminado prematuramente. Senão, o algoritmo volta ao passo de escolha da próxima tarefa a inserir na solução e repete todas as ações descritas, até que ou todas as tarefas tenham sido contempladas pela programação gerada para a frota de rebocadores, ou alguma janela tenha sido rompida.

Sempre que uma solução viável é gerada, ela é incluída na população de soluções iniciais, S^0 , e assim que este conjunto possuir o número de soluções almejadas, $tamPop$, o processo é finalizado. Assim, há três critérios de parada para esta heurística: (i) por número de iterações, (ii) pela quantidade mínima de soluções encontradas e (iii) pela quantidade máxima, ou tamanho da população de soluções viáveis. O algoritmo sempre para quando atinge o número de soluções buscadas, mas pode ser terminado prematuramente caso haja ao menos uma solução na população e o número máximo de iterações seja atingido.

Para a avaliação das possibilidades de inserção de tarefa nas rotas parciais dos rebocadores, além de levar em conta as diversas configurações compatíveis com k , é necessário examinar todas as combinações entre posições de inserção da tarefa nas rotas que compõem a configuração sob exame. Por exemplo, se a tarefa identificada como $k = 3$ puder ser atendida pelo conjunto de rebocadores identificados como 10 e 11, e o rebocador 10 possuir 1 tarefa na sua rota, enquanto o 11º possui 2 tarefas, é preciso examinar as seguintes combinações de inserção:

- Primeira posição da rota do rebocador 10 (antes da tarefa já roteirizada) e primeira posição da rota do rebocador 11 (antes da primeira tarefa da rota parcial);
- Primeira posição da rota do rebocador 10 e segunda posição da rota do rebocador 11 (entre as duas tarefas atuais);
- Primeira posição da rota do rebocador 10 e última posição da rota do rebocador 11 (após a segunda tarefa já roteirizada);
- Segunda posição da rota do rebocador 10 (após a tarefa já roteirizada) e primeira posição da rota do rebocador 11 (antes da primeira tarefa da rota parcial);
- Segunda posição da rota do rebocador 10 e segunda posição da rota do rebocador 11 (entre as duas tarefas atuais);

- Segunda posição da rota do rebocador 10 e última posição da rota do rebocador 11 (após a segunda tarefa já roteirizada).

Contudo, é possível que nem todas estas combinações sejam viáveis. O exemplo mais simples de uma situação deste tipo é se houver uma tarefa comum às duas rotas, ou seja, se a tarefa já atendida pelo rebocador 10 também estiver presente na rota do 11º rebocador. Neste caso, é inviável inserir a tarefa k antes da tarefa comum em um rebocador e após a tarefa comum no outro rebocador. Por isto, o método **fazInsercaoEatualizaParametros** utiliza as estruturas de dados auxiliares apresentadas na seção anterior deste capítulo (4.1), cujo cálculo é apresentada na seção 4.2.3. Cabe ressaltar aqui que combinações que geram subciclos são ignoradas, e que o método ordena cada uma das alternativas em função do número de tarefas com janelas de tempo rompidas e, para as soluções equivalentes neste critério, em função do custo incremental na função objetivo. Ainda, a lista restrita é composta apenas pelas opções que geram a menor quantidade de janelas rompida possível. Ou seja, se houver ao menos uma possibilidade de inserir k sem romper janelas, a lista restrita para escolha das posições de inserção conterá apenas alternativas que não rompam janelas, mesmo que só haja uma única opção. O mesmo comentário feito sobre a influência do *tamListaJ* na aleatoriedade da heurística é válido para a escolha da posição de inserção, ou seja, caso a lista restrita tenha tamanho igual a $tamListaM = 1$, não há aleatoriedade na escolha das posições de inserção.

Ao longo de toda a heurística, a solução parcial é descrita por estruturas de dados complementares ao conjunto *rotas*, que guarda a sequência de tarefas programada para cada rebocador r , $rota[r]$; e ao vetor *ES*, que armazena o instante mais cedo em que cada tarefa i pode ser executada, $ES[i]$, levando em conta a programação contida em *rotas*. A mais importante delas é o vetor *folga*, que armazena a quantidade de tempo $folga[i]$ que cada tarefa pode ser postergada (em função de seu $ES[i]$) sem romper nem a sua janela, nem as janelas das tarefas impactadas pelo seu adiamento. Estas estruturas são necessárias para lidar com a interdependência entre as rotas sem que seja necessário recalcular a solução toda após cada inserção. Por esta razão, antes de apresentar o método **fazInsercaoEatualizaParametros**, é preciso explicar o uso dos métodos e estruturas auxiliares. Assim, o texto a seguir é dividido da seguinte forma: as particularidades do problema da interdependência são discutidas na subseção 4.2.2, com apoio de um exemplo ilustrativo; os algoritmos usados nas

estruturas de dados auxiliares são apresentados nas subseções 4.2.3 a 4.2.6; e o pseudocódigo do movimento de inserção é finalmente apresentado em 4.2.7.

4.2.2 Complexidade do movimento de inserção de tarefa

Para definir a configuração de rebocadores que atenderá a tarefa, são avaliadas todas as possibilidades de inserção da tarefa nas rotas. Cada rota de cada subconjunto compatível de rebocadores (configuração) pode ter mais de uma posição viável para a inserção da tarefa, sendo preciso considerar todas as combinações possíveis entre as posições viáveis das rotas de uma mesma configuração. Isto eleva a complexidade do método **fazInsercaoEatualizaParametros**, pois é possível que duas posições de inserção de rotas diferentes sejam incompatíveis entre si, caso a inserção da tarefa em tais posições gere um subciclo no programa. Por exemplo, supondo que a tarefa adicionada, k seja inserida na posição originalmente ocupada por uma tarefa j em uma rota, em qualquer outra rota do modo escolhido a tarefa j não pode aparecer antes da posição de inserção de k . Portanto, a inserção só pode ser feita se houver ao menos uma combinação de navios em que existe posição de inserção válida compatível entre todas as rotas.

Destaca-se que uma rota impactada por uma inserção não precisa ter tarefa em comum com nenhuma das rotas do modo selecionado. Para ilustrar tal fato, toma-se como exemplo o esquema de programa parcial apresentado na **Figura 1**. Nela, são representadas três rotas parciais, das embarcações 1 a 3, usando uma circunferência identificada com o índice j para representar cada tarefa do roteiro. Na figura, a mesma tarefa é representada mais de uma vez se for executada por mais de uma embarcação, como é o caso da tarefa 5, e nestes casos as duas ou mais circunferências que representam a atividade aparecem alinhadas no eixo x , para mostrar que seu instante de início e fim é o mesmo independente da rota. Neste exemplo, nota-se que as rotas 1 e 3 não possuem tarefa em comum.

Figura 1 – Esquema de rotas parciais de três embarcações, atendendo sete tarefas.

Então, supõe-se que uma tarefa 8, ainda não roteirizada, pode ser inserida na rota 1 na posição ocupada originalmente pela tarefa 3, como representado na **Figura 2**. Esta inserção pode atrasar o início da tarefa 3 e, como a segunda embarcação também executa tal tarefa, as tarefas subsequentes na rota 2, de índice 5 e 6, também podem sofrer atrasos. Se este for o caso, então o roteiro 3 também será impactado e, por conta disso, a tarefa 7 também poderá sofrer atraso, apesar de não haver nenhuma atividade em comum entre as rotas 1 e 3.

Figura 2 – Ilustração do impacto causado em todas as rotas devido inserção de uma tarefa em apenas uma delas, considerando o programa original ilustrado pela **Figura 1**

Apesar das embarcações 1 e 3 não possuírem tarefas em comum, a inserção na rota 1 pode afetar tarefas da rota 3.

Cabe destacar que, no exemplo enunciado, diz-se que a inserção da tarefa 8 *pode* atrasar (e não que *atrasa*) o início da tarefa 3, assim como para as tarefas subsequentes. Isso se deve à forma como o instante mínimo de início de cada atividade (ES_j) é calculado, podendo ser restrito por sua própria janela de tempo ou pelo instante de chegada das embarcações necessárias. Ou seja, no exemplo dado, o ES_3 é igual ao maior valor entre as seguintes variáveis: o instante de chegada da embarcação 1 no ponto de início da tarefa 3, $s_{3,1}$; o instante de chegada da segunda embarcação no mesmo local, $s_{3,2}$; e o início da janela da tarefa 3, a_3^H . Então, se no programa original o início da atividade 3 era igual a a_3^H e se, após a inserção da tarefa 8, o novo instante de chegada da primeira embarcação no local de início da tarefa 3 for inferior a a_3^H , então não haverá atraso. Alternativamente, se originalmente o início da atividade 3 era igual a $s_{3,2}$ e o novo instante de chegada da primeira embarcação no local de início da tarefa 3 ainda for inferior a $s_{3,2}$, então também não há atraso.

Para lidar com a complexidade do movimento de inserção, são utilizadas estruturas de dados auxiliares, como a folga das tarefas, já definida no início deste capítulo. Os cálculos relacionados a tais estruturas são apresentados a seguir, no item 4.2.3.

4.2.3 Visão geral do uso de estruturas auxiliares para o movimento de inserção

Conforme dito anteriormente, para cada configuração compatível com a tarefa que se deseja inserir na solução parcial, são examinadas as combinações entre as posições de inserção da tarefa nestas rotas e, para cada combinação, avalia-se a viabilidade de inserir a tarefa em tais posições. Isto gera uma alta quantidade de cálculos a cada vez antes que a tarefa seja inserida na solução, prejudicando a velocidade da heurística. Alguns destes cálculos são repetidos várias vezes neste processo avaliação, já que um rebocador pode fazer parte de diversas configurações compatíveis com a tarefa.

Para evitar a repetição de cálculos a cada avaliação e diminuir a quantidade de combinações a examinar, assim que a tarefa que se deseja inserir é selecionada, cada uma das rotas é examinada individualmente, com a enumeração de todas as posições em que a tarefa poderia ser inserida sem violação de nenhuma janela de atendimento. Em seguida, para cada configuração compatível com a tarefa, é gerada uma lista contendo somente as combinações entre as posições de inserção viáveis, segundo a análise individual das rotas. Finalmente, cada combinação é examinada, ou seja, a tarefa é inserida nas posições especificadas e, caso não haja formação de subciclos, o

algoritmo calcula as novas datas de início e folgas das tarefas impactadas pela inserção, assim como o acréscimo na função objetivo.

Figura 3 – Sequência dos processos usados na inserção da tarefa, explicitando os momentos em que as estruturas auxiliares são construídas e usadas (quadros destacados com uma estrela).

A sequência de processos que envolvem a inserção de uma tarefa na solução parcial, explicitando os momentos em que as estruturas auxiliares são construídas e usadas, é apresentada na **Figura 3**. Nela, processos que aparecem entre os símbolos “<<” e “>>” referem-se aos processos globais da heurística construtiva, descritos de forma genérica para generalizar o movimento de inserção. Por exemplo, o “<< Processo de seleção da Tarefa (...)>>” faz parte do loop que escolhe, uma-a-uma, a próxima tarefa a ser colocada na solução, que no caso da HC é escolhida aleatoriamente entre as primeiras posições de uma lista ordenada.

Na execução da HC, na primeira chamada deste mecanismo de inserção não há nenhuma tarefa roteirizada, portanto as rotas estão todas vazias, ainda não há ES ou folgas das tarefas, e o processo “Avaliação individual das rotas” é suprimido. Ao fazer a inserção, a folga desta primeira tarefa inserida é calculada, como representado na **Figura 3**. Contudo, como as folgas são utilizadas na avaliação das posições viáveis de inserção da próxima tarefa, seu cálculo é o primeiro a ser explicado, a seguir.

4.2.4 Cálculo e uso de estruturas auxiliares: folgas das tarefas

Para determinar a folga de uma tarefa, é preciso conhecer a data limite para seu início, ou seja, o instante mais tarde em que ela pode ser iniciada sem romper nenhuma janela de tempo. A diferença entre este limite e a data de início programada para a tarefa representa sua folga, ou seja:

$$folga_j = limite_j - ES_j \quad \text{Eq. 4-1}$$

O limite do início da tarefa j , por sua vez, é calculado em função de sua própria janela de tempo (mais especificamente, de b_j) e do limite da tarefa executada imediatamente depois de j em cada rota à qual j pertence, levando em conta eventuais ociosidades nestas rotas. Portanto, se j for a última a ser executada em cada uma das rotas às quais pertence, seu limite será igual ao limitante superior de sua janela, b_j ; mas, se a tarefa não for a última executada por um ou mais rebocadores, é necessário que seja(m) conhecido(s) o(s) limite(s) da(s) tarefa(s) sucessora(s). Sendo assim, o algoritmo de cálculo das folgas percorre as tarefas roteirizadas a partir da última tarefa concluída (no programa parcial), até a primeira atividade programada, de forma a percorrer as rotas paralelamente, do final para o começo do horizonte de planejamento. O procedimento pode ser descrito da seguinte maneira: após detectar a tarefa de maior data de conclusão, que necessariamente será a última tarefa das rotas às quais pertence, sua folga é calculada usando $limite_j = b_j$. Então, parte-se para o cálculo da folga da próxima tarefa i de maior $tConclusao$, que pode ou não ter como sucessora direta a tarefa já calculada. Caso i tenha j como sucessora em uma rota r , o cálculo do limite da nova tarefa é:

$$limite_i \leq \begin{cases} b_i \\ ES_j + folga_j - T_{ij}^r - D_i \end{cases} \quad \text{Eq. 4-2}$$

Lembrando que, pela equação anterior, a expressão $ES_j + folga_j$ é igual a $limite_j$. Ou seja, o instante mais tarde em que a tarefa i pode ser iniciada não pode ser superior ao

limitante de sua janela, assim como não pode causar um atraso na tarefa j , que a sucede na rota r . Generalizando esta expressão, para levar em conta toda tarefa que suceda i em qualquer das rotas que atenda a tarefa i , chega-se à formulação abaixo.

$$\limite_i \leq \begin{cases} ES_j + folga_j - T_{ij}^r - D_i & \forall j \in J', r \in rotas: (i,j) \in rota[r] \end{cases} \quad \text{Eq. 4-3}$$

A figura a seguir ilustra o cálculo realizado, usando o exemplo de uma solução parcial com dois rebocadores e 3 tarefas. Novamente, o índice das tarefas aparece nas circunferências numeradas, mas nesta representação visual o tempo que está ocupado para execução da tarefa pelo rebocador aparece como um retângulo de coloração sólida. Por exemplo, a tarefa 6 estará em execução entre os dias 19 e 33, pelos rebocadores 1 e 2, portanto $ES_6 = 19$ e $tConclusao_6 = 33$. O tempo necessário para locomoção entre as tarefas também é representado por um retângulo, que neste caso foi colorido com um padrão quadriculado.

Figura 4 – Exemplo de visualização gráfica de uma solução parcial

Neste caso, o limite da tarefa 6, que é a última a ser executada, é definido por sua própria janela, que neste exemplo vale $b_6 = 21$. A figura a seguir representa o cálculo da folga resultante, igual a 2 dias, o que significa que a tarefa 6 pode ter seu início postergado 2 dias sem que haja rompimento de sua própria janela de tempo e de qualquer outra janela (de outras atividades).

Figura 5 – Representação do cálculo de folga da última tarefa executada no programa.

Para a tarefa 10, que antecede a tarefa 6 na rota do rebocador 2, o cálculo do limite depende de sua própria janela, que vai até $b_{10} = 14$, e da folga de sua sucessora. Pela

figura a seguir, nota-se que a tarefa 10 não pode ser postergada até o limitante de sua janela, pois isto romperia a janela da tarefa 6.

Figura 6 – Representação do cálculo de folga da penúltima tarefa de uma rota parcial

A expressão de cálculo do limitante da tarefa 10 fica:

$$limite_{10} \leq \begin{cases} b_{10} = 14 \\ \underbrace{ES_6}_{19} + \underbrace{folga_6}_2 - \underbrace{T_{10,6}^2}_{0,2} - \underbrace{D_{10}}_{8,8} = 12 \end{cases} \quad \text{Eq. 4-4}$$

Portanto, tem-se que $limite_{10}=12$ e $folga_{10} = 2$. Após o cálculo da folga da tarefa 10, é feito o da tarefa 1, que neste caso aparece em ambas as rotas. Sabendo que a janela da primeira tarefa é $[1,6]$, o cálculo do $limite_1$ fica:

$$limite_1 \leq \begin{cases} b_1 = 3 \\ \underbrace{ES_6}_{19} + \underbrace{folga_6}_2 - \underbrace{T_{1,6}^1}_{0,2} - \underbrace{D_1}_{8,5} = 12,3 \\ \underbrace{ES_{10}}_{10} + \underbrace{folga_{10}}_2 - \underbrace{T_{1,10}^1}_{0,3} - \underbrace{D_1}_{8,5} = 3,2 \end{cases} \quad \text{Eq. 4-5}$$

Como $ES_1 = 1,2$ (e $limite_1 = 3$), conclui-se que $folga_1 = 1,8$. Pela análise gráfica da solução, é possível notar que não existe nenhuma ociosidade na rota do rebocador 2, portanto a folga de uma tarefa será exatamente igual à folga de sua sucessora, a não ser que sua própria janela restrinja a folga, como ocorre para a tarefa 1 do exemplo. Ainda, no exemplo ilustrado, as folgas existentes permitem que seja feito o encaixe de uma nova tarefa na solução, como ilustra a **Figura 7**.

Figura 7 – Representação da solução parcial após inserção de uma tarefa na solução anterior

O algoritmo que calcula as folgas das tarefas roteirizadas, usando o processo de cálculo exemplificado, é apresentado no Pseudocódigo 2.

Na segunda linha do Pseudocódigo 2, é iniciado o conjunto J'' das atividades cujas folgas precisam ser calculadas, ou seja, atividades já roteirizadas (J' , usando a mesma notação do pseudocódigo da heurística construtiva). Na linha 3, as tarefas contidas em J'' são ordenadas de acordo com sua data de conclusão, não-crescentemente. Então, a cada iteração do *loop* entre as linhas 4 e 15, calcula-se a folga da tarefa de maior instante de conclusão entre aquelas cuja folga ainda não foi calculada e remove-se tal tarefa de J'' , conforme linhas 14 e 15. Desta forma, garante-se que os cálculos sejam feitos percorrendo as rotas do programa no sentido inverso.

Pseudocódigo 2 – Cálculo das folgas (*slack times*) das tarefas já roteirizadas.

```

1  Algoritmo calculaFolgas( $ES, tConclusao, rotas, Sucessores$ ):
2    Inicializa conjuntos  $J'' := J'$ 
3    Ordena  $J''$  não-crescentemente em  $tConclusao$ 
4    Enquanto  $J'' \neq \emptyset$ :
5       $j \leftarrow$  primeira tarefa de  $J''$ 
6       $limite \leftarrow b_j$ 
7       $rotasAverificar := \emptyset$ 
8      Se  $sucessores[j] \neq \emptyset$ :
9         $rotasAverificar \leftarrow$  conjunto de índices  $r$  de toda  $rota_k \in rotas$  que contêm  $j$ 
10     Para cada rota  $rr$  em  $rotasAverificar$ :
11        $limiteDaRota_{kk} \leftarrow ES_{suc_j^{rr}} + folga_{suc_j^{rr}} - T_{j,suc_j^{rr}}^{rr} - D_j$ 
12       Se ( $limiteDaRota_{kk} < limite$ ):
13          $limite \leftarrow limiteDaRota_{kk}$ 
14        $folga_j \leftarrow limite - ES_j$ 
15        $J'' = J'' \setminus j$ 
16  Fim

```

Na primeira interação, é calculada a primeira tarefa de J'' , a qual refere-se como j_{max} para facilitar as explicações. Como explicado anteriormente, a $folga_{j_{max}}$ é igual ao limitante superior de sua janela de tempo, $b_{j_{max}}$, menos o $ES_{j_{max}}$ uma vez que nenhuma tarefa é impactada pelo seu atraso. Como a tarefa j_{max} não tem sucessor, as linhas de 8 a 13 do pseudocódigo são puladas (graças à condicional que não preenche o conjunto $rotasAverificar$), fazendo com que a variável $limite$ seja igual ao seu valor de inicialização, definido na linha 6 como igual a $b_{j_{max}}$.

Para as demais tarefas de J'' , é verificado se há tarefa que a sucede em alguma rota e, caso uma tarefa j seja sucedida por uma tarefa i numa rota r , então $folga_j$ é igual à diferença entre ES_j e o menor dos seguintes valores: o final da sua própria janela, b_j ; e o máximo instante de início de j que não inviabiliza a tarefa i , igual a $ES_i + folga_i - T_{ji}^r - D_j$. Ainda, graças ao *loop* entre as linhas 10 e 13, se a tarefa j aparecer em mais de uma rota e tiver mais de um sucessor direto, o cálculo da folga leva em conta, além da janela de tempo de j , o menor $(ES_i + folga_i - T_{ji}^r - D_j)$ para qualquer tarefa i que sucede j em qualquer rota k do programa.

Voltando ao exemplo da **Figura 1** a primeira folga calculada para o programa ilustrado é a da tarefa 7, limitada apenas por b_7 . Em seguida, calcula-se a folga da tarefa 6, também limitada apenas por b_6 . Então, parte-se para a folga da quinta tarefa, limitada por b_5 , por $ES_6 + folga_6 - T_{56}^2 - D_5$ e por $ES_7 + folga_7 - T_{57}^3 - D_5$. Já a $folga_3$, calculada na sequência, é igual ao menor valor entre $b_3 - ES_3$ e $(ES_5 + folga_5 - T_{35} - D_3) - ES_3$. Já a segunda e a quarta tarefa têm suas folgas limitadas pelas suas próprias janelas de tempo e pelo máximo instante de início viável para tarefa 3. Por último, o cálculo da $folga_1$ leva em conta a folga da tarefa 2 e o limitante superior de sua própria janela, b_1 . Assim, todas as rotas existentes são averiguadas do fim para o início, atualizando os tempos de folga da última atividade ainda não calculada. Consequentemente, o tempo de folga de uma tarefa leva em consideração todas as rotas em que ela aparece e, como eventuais tarefas subsequentes já foram calculadas em iterações anteriores, é garantido que as janelas de todas as tarefas executadas após a tarefa atual estão sendo respeitadas.

4.2.5 Cálculo e uso de estruturas auxiliares: matriz $A(k)$, com posições viáveis de inserção de uma tarefa k

Conhecidas as folgas de cada tarefa já roteirizada, é possível verificar se a inserção de uma nova tarefa k numa posição ocupada por uma tarefa j na rota r é (individualmente) viável. Considera-se a posição viável se a folga da tarefa j for suficiente para acomodar o desvio causado pela inclusão da tarefa k entre j e a tarefa que precede j na rota r original, $prec_j^r$. Por isto, é preciso primeiro calcular qual será a data de início de k ao ser inserida em tal posição, para depois calcular o novo instante de início de j . Nas expressões abaixo, o sinal “'” indica o novo valor que uma variável

assumiria caso a tarefa k fosse de fato inserida na posição avaliada. Por exemplo, o novo valor de ES_j é denotado por ES'_j .

O instante de chegada da embarcação r na tarefa inserida, denotado por $s'_{k,r}$, é calculado da seguinte forma:

$$s'_{k,r} = ES_{prec_j^r} + D_{prec_j^r} + T_{prec_j^k,k}^r \quad \text{Eq. 4-6}$$

Como a avaliação da viabilidade da posição é individual, tudo se passa como se a tarefa fosse inserida apenas na rota r . Sob esta hipótese, os instantes mínimos de início das tarefas k e j são calculados segundo as expressões:

$$ES'_k \geq \begin{cases} s'_{k,r} \\ a_k \end{cases} \quad \text{Eq. 4-7}$$

$$ES'_j = ES'_k + D_k + T_{k,j}^r \quad \text{Eq. 4-8}$$

Então, considera-se viável a posição de inserção se ES'_k estiver contido na janela de tempo da atividade k e se o novo instante de início de j , ES'_j , estiver dentro do intervalo definido pela sua folga, ou seja, se as expressões $ES'_k \leq b_k$ e $ES'_j \leq ES_j + folga_j$ forem verdadeiras. Voltando ao exemplo da Figura 2, a posição ocupada pela tarefa 3 na rota 1 é viável se a inserção de 8 nesta rota não violar as janelas de atendimento de todas as tarefas potencialmente afetadas, ou seja, das tarefas 3, 5, 6 e 7.

Fazendo este cálculo para todas as posições de inserção possíveis de todas as rotas, constrói-se a matriz $A(k)$. Cada elemento $a_{r,pos} \in A(k)$, localizado na sua r -ésima linha e pos -ésima coluna, vale 1 se (e somente se) a tarefa k pode ser inserida na posição pos da rota r ; e 0 caso não possa. Assim, ainda em relação ao exemplo da Figura 2, caso a única opção para inserção da tarefa 8 seja a ilustrada na figura e o valor de n for igual a 8, então a matriz $A(8)$ é:

$$A(8) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

O algoritmo utilizado para construção da matriz $A(k)$ é apresentado no Pseudocódigo 3. O procedimento é iniciado com a criação da matriz, de m linhas e tamanho variável de colunas (cada linha possui tantas colunas quanto houver posições de inserção), atribuindo valor 0 para todas as posições criadas; e com a criação de um vetor auxiliar,

ES' , que representa as datas de início que cada tarefa teria após a inserção da tarefa k em determinada posição da rota. O vetor atual de ES é usado na inicialização de ES' .

Pseudocódigo 3 – Algoritmo de cálculo das posições em que uma tarefa pode ser inserida, considerando a solução parcial dada por um conjunto de rotas, e armazenamento na matriz $A(job)_{m \times n}$.

```

1. Algoritmo calculaA(k,rotas,j''):
2.   Inicializa A com valor 0 em todas as posições; e  $ES'_j = ES_j$ 
3.   Para toda  $r \in rotas$ 
4.      $pos \leftarrow$  última posição da rota  $r$ 
5.      $j \leftarrow$  tarefa na posição  $pos$  da rota  $r$ 
6.     Enquanto ( $j \neq O_r^+$ ) faça:
7.        $a_{r,pos} \leftarrow 0$ 
8.        $ES'_k \leftarrow a_k$ 
9.        $prec_j^r \leftarrow$  tarefa na  $(pos - 1)$  da rota  $r$  (i.e. que antecede  $j$  na rota  $k$ )
10.      Se ( $prec_j^r = O_r^+$ )
11.         $s'_{k,r} \leftarrow r^r + T_{O_r^+,k}^r$ 
12.      Senão
13.         $s'_{k,r} \leftarrow ES_{prec_j^r} + D_{prec_j^r} + T_{prec_j^r,k}^r$ 
14.      Se ( $s'_{k,r} > ES'_k$ )
15.         $ES'_k \leftarrow s'_{k,r}$ 
16.      Se ( $j = O_r^-$ )
17.        Se ( $ES'_k \leq b_k$ )
18.           $a(k)_{r,pos} \leftarrow 1$ 
19.        Senão, se ( $folga_j \geq D_k$ )
20.           $s'_{j,r} \leftarrow ES'_k + D_k + t_{k,j}^r$ 
21.          Se ( $s'_{j,r} > ES'_j$ )
22.             $ES'_j \leftarrow s'_{j,r}$ 
23.          Se ( $ES'_k \leq b_k$  &  $ES'_j - ES_j \leq folga_j$ )
24.             $a(k)_{r,pos} \leftarrow 1$ 
25.           $pos \leftarrow pos - 1$ 
26.           $j \leftarrow$  tarefa na posição  $pos$  da rota  $r$ 
27.   Retorna A
28. Fim

```

Então, para cada rota r do conjunto de rebocadores, identifica-se a tarefa que ocupa a última posição da rota r e percorre-se a rota no seu sentido inverso, da última até a primeira tarefa executada na rota, no *loop* entre as linhas 6 e 26. Ao percorrer a rota posição-a-posição, calcula-se primeiramente o menor instante em que a tarefa k poderia ser iniciada pelo rebocador r caso fosse inserida na posição em questão, nas linhas 10 a 15. Após o cálculo dos novo *earliest starting times* da tarefa inserida (k), denotado por ES'_k , caso a tarefa j (que ocupava a posição onde k foi inserida) seja o nó final da rota, O_r^- , então basta verificar se a janela da tarefa inserida é respeitada, o que é feito nas linhas 16 a 18 do pseudocódigo. Por outro lado, se j representar uma tarefa de fato, então seu novo instante de início é calculado entre as linhas 19 e 22, desde que a folga de j seja ao menos igual à duração da tarefa k inserida, conforme condicional apresentada na linha 19, que evita cálculos desnecessários. Finalmente, caso a janela de k seja respeitada e o atraso da tarefa j (medido pela diferença entre

seu novo instante de início e o atual) seja igual ou inferior à sua folga, então considera-se a posição de inserção viável, conforme linhas 23 e 24.

Uma vez montada a matriz $A(k)$, seu uso é simples: para uma configuração de rebocadores que seja compatível com a tarefa k , consulta-se a matriz para verificar se há ao menos um elemento de cada uma das respectivas linhas que valha 1 e, se houver, são geradas combinações de posições de inserção, pegando uma posição (que valha 1) de cada rota do conjunto de rebocadores.

4.2.6 Cálculo e uso de estruturas auxiliares: sucessores das tarefas

Finalmente, como a avaliação inicial das posições não verifica o conjunto de rotas que compõem o modo, mas sim as rotas individualmente, é preciso verificar a viabilidade da inserção simultânea nas posições escolhidas, garantindo que as janelas sejam respeitadas e que a inserção de k no conjunto de posições de inserção não forme subciclo. Para conferir o atendimento das janelas, o procedimento realizado é similar ao que foi feito na primeira fase da heurística e nas equações Eq. 4-6 a Eq. 4-8: calcula-se o instante de chegada de cada embarcação da configuração no nó correspondente à tarefa inserida, partindo da tarefa que precede a posição de inserção em cada uma das rotas. Então, o ES'_k é igual ao maior dos instantes de chegada ou ao início da janela da tarefa, o que ocorrer depois. Novamente, caso o ES'_k seja posterior ao b_k^H , considera-se a inserção inviável.

Caso a janela de tempo seja respeitada, parte-se para a verificação de subciclo. Primeiramente, para cada uma das posições de inserção do conjunto avaliado, assume-se a hipótese de que a tarefa k foi inserida em tal posição e atualiza-se a estrutura de *Sucessores'* de acordo com tal hipótese: a tarefa k é colocada na lista de sucessores da tarefa $prec_j^r$, $sucessores'(prec_j^k)$; a tarefa j é adicionada à lista $sucessores'(k)$; e é verificado se a tarefa j deve ser eliminada de $sucessores'(prec_j^r)$. Após tal atualização, utiliza-se o algoritmo do Pseudocódigo 4 para listar os sucessores diretos e indiretos da tarefa inserida.

Ou seja, são listadas as tarefas que sucedem k nas rotas em que ela foi inserida; as tarefas que sucedem as tarefas que sucedem k em qualquer rota do conjunto de rebocadores; e assim por diante. Então, se na lista $J^{Suc}(k)$, que contém todos os sucessores (diretos e indiretos) de k , a própria tarefa k aparecer, é porque um subciclo foi formado.

Pseudocódigo 4 – Algoritmo que lista sucessores diretos e indiretos de uma tarefa

```

1. Algoritmo listaSucessoresDiretosEIndiretos(k, sucessores):
2.   Inicializa conjunto  $J^{Suc} = \{k\}$ 
3.    $j \leftarrow$  primeira tarefa de  $J^{Suc}$ 
4.   Faça{
5.      $J^{Suc} = J^{Suc} \cup \text{sucessores}(j)$ 
6.     Se  $j \neq$  última tarefa de  $J^{Suc}$ 
7.        $j \leftarrow$  próxima tarefa de  $J^{Suc}$ 
8.   }Enquanto  $j \neq$  última tarefa de  $J^{Suc}$ 
9.   Retorna  $J^{Suc}$ 
10. Fim

```

Por exemplo, para o caso ilustrado na **Figura 1**, os elementos $Sucessores(6)$ e $Sucessores(7)$ são conjuntos vazios, enquanto $Sucessores(1) = \{2\}$ e $Sucessores(5) = \{6,7\}$, por exemplo. Supõe-se que ao olhar isoladamente as possibilidades de inserção nas rotas, uma combinação possível é a posição 4 da primeira rota, $rota_{1,4}$, com a posição 3 da terceira embarcação, $rota_{3,3}$. Ao tentar realizar tal inserção, por casa da primeira rota serão atualizados os conjuntos $Sucessores'(2) = \{3,8\}$ e $Sucessores'(8) = \{3\}$; e então, por causa da terceira embarcação, faz-se $Sucessores'(5) = \{6,7,8\}$ e $Sucessores'(8) = \{3,7\}$. Assim, a estrutura temporária $Sucessores'$ fica com os seguintes elementos:

$$\begin{aligned}
 Sucessores'(1) &= \{2\} \\
 Sucessores'(2) &= \{3,8\} \\
 Sucessores'(3) &= \{5\} \\
 Sucessores'(4) &= \{3\} \\
 Sucessores'(5) &= \{6,7,8\} \\
 Sucessores'(6) &= \emptyset \\
 Sucessores'(7) &= \emptyset \\
 Sucessores'(8) &= \{3,7\}
 \end{aligned}$$

Ao percorrer a lista de sucessores diretos e indiretos da tarefa inserida, 8, as tarefas listadas seriam $\{3,7,5,6,7,8\}$. Como uma tarefa não pode ser sucedida por ela mesma, identifica-se a formação de um subciclo e a combinação de posições de inserção é considerada inviável. Por outro lado, se outra possível combinação de posições de inserção para o mesmo modo fosse $rota_{1,4}$ e $rota_{3,2}$, o vetor $Sucessores'$ dessa opção seria:

$$Sucessores'(1) = \{2\}$$

$$\begin{aligned}
Sucessores'(2) &= \{3,8\} \\
Sucessores'(3) &= \{5\} \\
Sucessores'(4) &= \{3\} \\
Sucessores'(5) &= \{6,7\} \\
Sucessores'(6) &= \emptyset \\
Sucessores'(7) &= \emptyset \\
Sucessores'(8) &= \{3,5\}
\end{aligned}$$

Neste caso, os sucessores diretos e indiretos da tarefa 8 seriam $\{3,5,6,7\}$, o que significa que não haveria formação de subciclo. Portanto, neste exemplo o movimento de inserção de 8 na quarta posição da primeira embarcação e na segunda da quinta é viável.

4.2.7 Apresentação do algoritmo de inserção de tarefa em rotas paralelas

A função `fazInsercaoEatualizaParametros( $k, s^0, contador, tamListaRestritaM, RM$ )` gera uma nova solução pela inserção da tarefa k na solução s^0 . Neste processo, é calculado o menor instante de início e de fim (ES_j e $tConclusao_j$) possíveis para a tarefa inserida e são atualizadas as datas de início e folgas das demais tarefas, além do vetor de sucessores e demais estruturas de dados utilizadas. O cálculo das datas de início é feito pela função `propagaAtrasos(tarefa inserida, conjunto de rotas)`, que calcula o novo instante de início de todas as tarefas afetadas pela inserção, para todas as rotas impactadas. O Pseudocódigo 5 apresenta o método que tenta fazer a inserção de uma tarefa k em determinadas posições de um modo.

Pseudocódigo 5 – Algoritmo da função que verifica se uma tarefa pode ser inserida simultaneamente em determinadas posições de um conjunto de rotas

```

1  Algoritmo tentaInsercao(listaCombinacoes[comb], k, Sucessores, rotas):
2       $M_{k,g} \leftarrow$  conjunto de rebocadores especificado em listaCombinacoes[comb]
3       $ES'_k \leftarrow a_k^H$ 
4      Para todo  $r \in M_{k,g}$  faça:
5          posInsr  $\leftarrow$  posição de inserção na rotar, especificada em listaCombinacoes[comb]
6           $prec_k^r \leftarrow$  tarefa na posição (posInsr-1) de r
7          Se  $prec_k^r = O_k^+$  então  $s_k^r = r^k + t_{prec_k^r, k}^r$ 
8          Senão  $s_k^r = tConclusao_{prec_k^r} + t_{prec_k^r, k}^r$ 
9          Se  $s_k^r > ES'_k$  então  $ES'_k \leftarrow s_k^r$ 
10         Se  $ES'_k \geq b_k^H$  então retorna inviável
11         Senão, faça:
12             Sucessores' = atualizaSucessores(Sucessores, listaCombinacoes[comb], k)
13              $J^{Suc} =$  listaSucessoresDiretosEIndiretos(k, Sucessores')
14             Para todo  $j \in J^{Suc}$  faça:
15                 Se  $j = k$  então: retorna inviável
16         Retorna viável
17  Fim

```

Após inicialização do ES'_k com o limitante superior de sua janela, na linha 3, é calculada a menor data de chegada que cada um dos rebocadores do modo conseguiria atender, partindo da tarefa na posição anterior à posição em que a tarefa k seria inserida, entre as linhas 4 e 9. Toda vez que uma data de chegada s'^r_k é calculada, ela é comparada com o atual valor de ES'_k e, se for maior, o ES'_k é atualizado (linha 9). Ao fim do loop que verifica todos os instantes de chegada das embarcações do modo, verifica-se o atendimento da janela de tempo da tarefa inserida e, caso ela tiver sido violada, a função retorna o valor “inviável” (linha 10). Caso contrário, nas linhas 12 e 13 a estrutura de *Sucessores'* é atualizada e todos os sucessores diretos e indiretos da tarefa recém inserida são guardados no conjunto J^{Suc} . Então, caso a tarefa inserida esteja entre as tarefas deste conjunto, considera-se a inserção inviável. Por fim, caso o método não tenha sido interrompido nas linhas 10 ou 15, a inserção é considerada viável.

O funcionamento do método que faz a inserção (Pseudocódigo 6) é similar ao que tenta inserção, com a diferença que no primeiro as condições de viabilidade não são checadas e o valor de $tConclusao'_k$ também é calculado, somando o valor de $ES'_k + D_k$.

Pseudocódigo 6 – Algoritmo que insere tarefa nas rotas parciais da solução e calcula os parâmetros

```

1  Algoritmo fazInserçãoECalculaParametros(job,gSelecionado,rotas,Sucessores,ES,tConclusao):
2       $M_{job,g} \leftarrow$  conjunto de rebocadores especificado em gSelecionado
3       $ES' \leftarrow ES$  e  $ES'_{job} \leftarrow a_{job}^H$ 
4      Para todo  $k \in M_{job,g}$  faça:
5          posInsr  $\leftarrow$  posição de inserção na  $rota_k$  especificada em gSelecionado
6          Insere job na posInsr da  $rota_k$ 
7           $prec'_{job} \leftarrow$  tarefa na posição (posInsr-1) da  $rota_k$ 
8          Se  $prec'_{job} = O_k^+$  então  $s'^k_{job} = r^k + t^k_{prec'_{job},job}$ 
9          Senão  $s'^k_{job} = tConclusao_{prec'_{job}} + t^k_{prec'_{job},job}$ 
10         Se  $s'^k_{job} > ES'_{job}$  então  $ES'_{job} \leftarrow s'^k_{job}$ 
11          $tConclusao'_{job} \leftarrow ES'_{job} + p_{job}$ 
12          $Sucessores' = atualizaSucessores(Sucessores, listaCombinacoes[comb], job)$ 
13         tarefasAchegar  $\leftarrow sucessores'(job)$ 
14         Para todo  $j \in tarefasAchegar$  faça:
15             rotasAchegar  $\leftarrow$  conjunto de rebocadores que executam j
16             Para todo  $k \in rotasAchegar$  faça:
17                 posJ  $\leftarrow$  posição da tarefa j na  $rota_k$ 
18                  $prec'_j \leftarrow$  tarefa na posição (posJ-1) da  $rota_k$ 
19                 Se  $prec'_j = O_k^+$  então  $s'^k_j = r^k + t^k_{prec'_j,j}$ 
20                 Senão  $s'^k_j = ES'_{prec'_j} + p_{prec'_j} + t^k_{prec'_j,j}$ 
21                 Se  $s'^k_j > ES'_j$  então
22                      $ES'_j \leftarrow s'^k_j$ 
23                      $tConclusao'_j \leftarrow ES'_j + p_j$ 
24                     tarefasAchegar  $\leftarrow tarefasAchegar \cup sucessores'(j)$ 
25          $Sucessores \leftarrow Sucessores'$ 
26          $ES \leftarrow ES'$ 
27          $tConclusao \leftarrow tConclusao'$ 
28  Fim

```

Além disto, todos os sucessores diretos de k , ou seja, aqueles contidos em $sucessores'(k)$ têm seu ES' recalculados, considerando o $tConclusao'_k$, o tempo de deslocamento o seu ES antes da inserção. Sempre que houver necessidade de mudar o valor de ES'_j de uma tarefa j , o atraso é propagado para os seus sucessores diretos e indiretos, conforme linha 24 do pseudocódigo abaixo, que gera mais execuções do *loop* entre as linhas 14 e 24. Só depois da propagação de todos os atrasos, as estruturas *Sucessores*, *ES*, *tConclusao* recebem os valores de *Sucessores'*, *ES'*, *tConclusao'*, finalizando o método que faz a inserção.

4.3 REGRAS PARA ORDENAÇÃO DE TAREFAS E ESCOLHA DO MODO

Como visto na apresentação do algoritmo da heurística construtiva, duas listas restritas são usadas no processo de construção da solução, para escolha da próxima tarefa a inserir nas rotas e para a escolha de onde será feita esta inserção (i.e. em qual conjunto de rebocadores e em quais posições de suas rotas). Ambas as listas são previamente ordenadas, portanto o critério de ordenação utilizado pode ser calibrado de acordo com o tipo de problema e com as características das instâncias que se deseja resolver. A definição destas regras é uma forma de colocar inteligência no processo de criação da solução e, por isso, várias regras foram testadas nesta pesquisa.

Testes feitos nas menores instâncias disponíveis, cuja solução ótima é facilmente obtida por um algoritmo exato, indicaram que as regras de ordenação influenciam os resultados médios obtidos pela HC. Para a lista restrita de opções de inserção, foi possível identificar uma regra que obtém melhor desempenho em qualidade e em tempo computacional, que é a ordenação pelo custo incremental na função objetivo. Outros critérios de ordenação usados foram: quantidade de tarefas com penalização por atraso; número médio de tarefas por rota; quantidade de rebocadores em uso (ou seja, atendendo pelo menos uma tarefa); quantidade de rebocadores de classe superior em uso.

Além do critério de ordenação, duas regras *RM* foram testadas para a seleção do modo a usar na execução de cada tarefa: verificação exaustiva de todos os modos (e para cada modo, de todas as configurações de rebocadores correspondentes); e verificação modo-a-modo apenas até obter ao menos uma possibilidade viável de inserção da tarefa. Neste segundo caso, primeiro testa-se o modo básico de atendimento da tarefa, que utiliza a menor quantidade possível de rebocadores de classe superior; de houver possibilidade de inserir a tarefa em alguma configuração de rebocadores pertencente

ao modo básico, os outros modos não são averiguados, caso contrário, é examinado o próximo modo, remanejando apenas uma unidade de rebocadores de classe 1 para a classe 2 (em relação ao modo básico). O processo é repetido apenas até encontrar um modo que tenha uma ou mais possibilidade de inserção da tarefa, sem necessariamente averiguar todas as possibilidades. Desta forma, prioriza-se a adoção de modos que utilizem menos rebocadores mais caros.

Os testes realizados mostraram que as duas formas de averiguação são complementares, pois enquanto a verificação exaustiva leva muito mais tempo, mas resolve instâncias mais difíceis (isto é, instâncias em que o tempo necessário para o Gurobi encontrar a solução ótima foi mais alto); a verificação incremental dos modos é capaz de resolver instâncias que a exaustiva não resolveu. Por isso, as duas lógicas foram incorporadas ao método final, como explicado ao fim desta seção.

Para a lista restrita de tarefas, as regras de ordenação testadas também obtiveram desempenhos complementares. As oito Regras de Ordenação (RO) implementadas e testadas estão apresentadas na Tabela 2.

Tabela 2 – Regras de ordenação usadas pela Heurística Construtiva

Identificador da Regra de Ordenação	Explicação de regra usada na ordenação de tarefas
RO 1.	Não-decrescentemente pelo limite inferior da janela de tempo a_j , com desempate (também não-decrescentemente) pelo limite superior, b_j
RO 2.	Não-crescentemente pela demanda total, Q_j , com desempate (também não-crescentemente) pela demanda de rebocador de classe superior, q_j^c
RO 3.	Não-decrescentemente pelo limite superior da janela de tempo b_j , com desempate não-crescentemente pela duração D_j
RO 4.	Não-decrescentemente pelo limite superior da janela de tempo b_j , com desempate não-crescentemente pela demanda de rebocador de classe superior, q_j^c
RO 5.	Não-decrescentemente pelo limite superior da janela de tempo b_j , com desempate não-crescentemente pela demanda total, Q_j
RO 6.	Não-crescentemente pelo valor obtido da multiplicação da duração D_j pela demanda total Q_j , com desempate não-decrescentemente por b_j
RO 7.	Não-crescentemente pelo grau de incompatibilidade da tarefa, com desempate não-decrescentemente pelo tamanho da janela ($b_j - a_j$)
RO 8.	Não-crescentemente pelo grau de incompatibilidade, com desempate não-crescentemente pela demanda total, Q_j

Todas as RO desenvolvidas partem do princípio que se deve iniciar a construção da solução pelas tarefas mais difíceis de encaixar nas rotas, como comumente é feito em algoritmos de sequenciamento de tarefas. A diferença entre as regras está na escolha

dos parâmetros que tornam a inserção de uma tarefa mais difícil que as demais. Na primeira regra, supõe-se que devem ser adicionadas às rotas aquelas tarefas que possam ser iniciadas o quanto antes, priorizando as de prazo mais curto quando houver empate no início da janela de tempo. Já na segunda, o número de rebocadores demandado pela tarefa é o critério principal para a ordenação, sendo a quantidade de recursos de classe superior usada como critério de desempate. Isto porque espera-se que conforme a solução vá crescendo (com mais tarefas roteirizadas), fique mais difícil de encaixar tarefas que exijam grande número de rebocadores, especialmente aquelas que exijam mais recursos de classe superior, já que estes rebocadores podem atender a demanda de sua classe e das classes inferiores.

As RO 3 até 5 priorizam tarefas de prazo mais curto e, caso duas tarefas tenham o mesmo limite superior da janela de tempo, os critérios para desempate são: duração das tarefas, já que a dificuldade de inserção de tarefas mais longas aumenta conforme a solução parcial vai evoluindo; quantidade de rebocadores de classe superior demandados, escolhendo antes as com maior demanda; quantidade total de rebocadores necessários, novamente em ordem decrescente.

A sexta RO baseia-se na suposição de que tanto a duração quanto a quantidade de recursos demandados devem ser levados em conta na ordenação das tarefas, por isso a multiplicação destes dois parâmetros é usada como critério de ordenação. O limite superior da janela também é usado nesta RO, mas somente em caso de empate.

As duas últimas regras de ordenação, RO 7 e RO 8, utilizam o conceito de **grau de incompatibilidade** das tarefas, adaptado do problema de *Bin Packing* com compatibilidade entre objetos (SANTOS *et al.*, 2019). Para este contexto, um par de tarefas (i, j) é considerado incompatível entre si caso elas não possam ser executadas pelo mesmo rebocador sem que haja rompimento de suas janelas de tempo, considerando suas durações D_i e D_j e o menor tempo de deslocamento possível ($\min\{Dist_{ij}, Dist_{ji}\} / \max v^s$). O grau de incompatibilidade de uma tarefa i é igual ao número de tarefas com as quais a mesma não pode ser associada, isto é, pertencer à mesma rota.

Finalmente, definiu-se que a HC usaria uma combinação das regras e parâmetros calibráveis, da seguinte maneira:

- Nas primeiras oito iterações da HC, as listas restritas para escolha da tarefa ($tamListaRestritaJ$) e da posição de inserção ($tamListaRestritaM$) são iguais a

1, ou seja, não há aleatoriedade na construção da solução. A regra de ordenação de tarefas é adotada de acordo com o número da iteração (ou seja, começa-se pela RO1 e, na oitava iteração da HC, a RO8 é usada). Os modos são averiguados apenas até encontrar opções de inserção factíveis; mas, para cada modo que seja levado em conta, as configurações e posições de inserção de um modo são verificadas exaustivamente;

- Nas seguintes oito iterações ($contador \in [9,16]$), as listas restritas para escolha da tarefa ($tamListaRestritaJ$) e da posição de inserção ($tamListaRestritaM$) também são iguais a 1. A regra de ordenação de tarefas é adotada de acordo com o número da iteração menos 8 (ou seja, começa-se pela RO1 e, na oitava iteração da HC, a RO8 é usada). Antes de cada inserção, todos os modos são averiguados de maneira exaustiva, assim como as respectivas configurações e posições de inserção;
- A partir da 17ª iteração da HC, as listas restritas assumem valores definidos pelo usuário (vide capítulo 6). A RO adotada passa a ser fixa, RO5 e todos os modos são examinados para que seja escolhida uma posição de inserção para a tarefa. Novamente, as configurações de rebocadores e posições de inserção são examinadas de forma exaustiva para cada modo.

Esta versão final do método baseia-se nos experimentos realizados com as instâncias propostas neste trabalho. Os resultados obtidos ao longo do desenvolvimento do método mostram que para as versões sem aleatoriedade, não há uma regra de ordenação que se sobressaia às demais, assim como não é interessante adotar apenas uma regra para avaliação ou não de todos os modos. Contudo, para as versões aleatórias, uma das regras de ordenação teve melhor desempenho que as demais (RO número 5), e a consideração de todos os modos se mostrou importante para que todas as instâncias fossem resolvidas.

5 PROPOSIÇÃO DE MÉTODOS HEURÍSTICOS PARA SOLUÇÃO DO PROBLEMA ABORDADO

Neste capítulo, são apresentados os quatro métodos criados e testados para a solução do Problema de Roteirização de Veículos com Janelas de Tempo, Sincronização de recursos e frota Heterogênea, o PRVJTSin-Het:

- Múltiplos recomeços da heurística construtiva;
- Busca Local com múltiplos recomeços (número fixo de iterações e estratégia global);
- GRASP, adaptada para o PRVJTSin-Het (com Busca Local com parada por estagnação);
- Remoção de vetor de tarefas com múltiplos recomeços (Busca Local em região reduzida, usando método do tipo destruição-e-reconstrução)

Todos os métodos partem de uma ou mais soluções iniciais, portanto todas as aplicações foram combinadas com a heurística construtiva proposta no capítulo 4.

5.1 MÚLTIPLOS RECOMEÇOS DA HEURÍSTICA CONSTRUTIVA

Esta é abordagem mais simples implementada e consiste em executar a heurística construtiva um certo número de vezes e escolher a melhor solução gerada, conforme de observa no Pseudocódigo 7. Aqui, a única possibilidade de diversificação da solução é pelos aspectos aleatórios da construção da solução, ou seja, pela alteração na ordem em que as tarefas são adicionadas à solução, assim como pela escolha aleatória usando lista restrita das configurações e posições em que as tarefas são inseridas.

Pseudocódigo 7 - Heurística Construtiva com Múltiplos Recomeços (HCMR)

```

1  Algoritmo hHCMR(minIt,tamListaRestritaJ,tamListaRestritaM):
2    Cria s e s*, para armazenar solução corrente e incumbente
3    contador ← 1, solucoes ← 0
4    Enquanto contador ≤ minIt OU solucoes < 1 faça:
5      s ← hConstrutivaGulosa(1, tamListaRestritaJ, tamListaRestritaM, 1)
6      contador ← contador + 1
7      Se checaViabilidade(s) = Verdadeiro então:
8        Se z(s) < z(s*) OU solucoes < 1 então: s* ← s
9        solucoes ← solucoes + 1
10     Retorna s*
11  Fim

```

O algoritmo funciona da seguinte forma: a cada iteração da HCMR, a heurística construtiva é executada usando uma semente diferente, pela chamada do método *hConstrutivaGulosa*, apresentado anteriormente no Pseudocódigo 1. Caso a solução gerada, armazenada em *s*, seja considerada viável pelo método *checaViabilidade* e tenha valor de função objetivo inferior à incumbente *s**, atualiza-se *s**. O algoritmo é

finalizado quando ao menos uma solução tiver sido encontrada e a heurística construtiva já tiver sido executada determinado número de vezes ($minIt$). Por isso, o contador *solucoes* é incrementado toda vez que s^* é atualizada, e seu valor é levado em conta no loop “Enquanto $contador \leq minIt$ OU $solucoes < 1$ faça”.

5.2 BUSCA LOCAL COM MÚLTIPLOS RECOMEÇOS

A Busca Local com Múltiplos Recomeços (BLMR) é uma tentativa simples de explorar uma região maior do espaço de soluções. O algoritmo aplicado, descrito a seguir e apresentado no Pseudocódigo 8, parte de uma solução gerada pela heurística construtiva, usando parâmetros fixos para limitar as listas restritas (de escolha de tarefa e de seus locais de inserção na solução parcial).

Pseudocódigo 8 - Algoritmo da Busca Local com Múltiplos Recomeços (BLMR)

```

1  Algoritmo hBLMR( $maxItLocais, maxItGlobais$ ):
2    Cria  $s$ ,  $s'$  e  $s^*$ , para armazenar solução vizinha, corrente e incumbente
3     $contadorG \leftarrow 1$ 
4     $s^* \leftarrow$  hConstrutivaGulosa(1,  $tamListaRestritaJ$ ,  $tamListaRestritaM$ , 1)
5    Enquanto  $contadorG \leq maxItGlobais$  faça:
6       $s' \leftarrow s^*$ 
7      Inicializa  $S' := \emptyset$  (memória de soluções aceitas) e  $S := \emptyset$  (vizinhança da solução)
8       $contadorL \leftarrow 1$ 
9      Enquanto  $contadorL \leq nmaxItLocais$  faça:
10        $S :=$  geraVizinhança( $s'$ )
11        $s' \leftarrow$  encontraMelhorSolução( $S, S'$ )
12       Se checaViabilidade( $s'$ ) = Verdadeiro E  $z(s') < z(s^*)$  então:  $s^* \leftarrow s'$ 
13        $contadorL \leftarrow contadorL + 1$ 
14      $contadorG \leftarrow contadorG + 1$ 
15     Retorna  $s^*$ 
16  Fim

```

A cada iteração da BLMR, é aplicado um movimento que modifica a solução corrente, gerando sua vizinhança. Ao fim da iteração, a melhor solução vizinha é adotada como nova solução corrente, mesmo que sua qualidade seja inferior à da solução anterior. Isso significa que foi adotada a estratégia de *steepest descent*, com exploração exaustiva da vizinhança e adoção do melhor vizinho (RIBEIRO; HANSEN, 2002) ao invés de *first descent*, em que a primeira vizinha que gere melhoria seria aceita como nova solução corrente, sem exploração da vizinhança inteira. Ao completar uma quantidade pré-determinada de iterações, o processo é reiniciado a partir da melhor solução encontrada, ou seja, da solução incumbente do problema. O número de vezes que a busca é reiniciada também é pré-determinado.

O movimento **geraVizinhança**(s'), aplicado na construção da vizinhança da solução corrente s' , é de remoção de uma tarefa escolhida aleatoriamente, e reinserção nas rotas de todas as outras configurações de rebocadores compatíveis com a tarefa,

inclusive de outros modos de atendimento. Para que a heurística caminhe no espaço de soluções, é necessário permitir que as janelas de tempo das atividades sejam violadas ao longo da busca. Por isso, a função `encontraMelhorSolução` (S, S') avalia não só o valor na função objetivo como o grau de inviabilidade da solução, escolhendo a solução de menor custo entre aquelas com o menor número de tarefas com janelas rompidas. Outra característica importante desta função é que ela não permite a adoção de uma vizinha que já tenha sido solução corrente na mesma iteração da busca global. Para isto, toda vez que uma solução é aceita como corrente, ela é acrescentada ao conjunto S' , que representa a memória da heurística. Tal conjunto é esvaziado toda vez que a estratégia global é reiniciada.

5.3 GRASP PARA O PRVJT-SIN-HET

O algoritmo GRASP desenvolvido para este problema, apresentado no Pseudocódigo 9, baseia-se no modelo generalizado apresentado por Resende e Ribeiro (2014)..

Pseudocódigo 9 – Algoritmo da GRASP

```

1  Algoritmo hGRASP(tamPop,tamListaRestritaJ,tamListaRestritaM,numRecomecosSemMelhora):
2    Cria  $s$ ,  $s'$  e  $s^*$ , para armazenar solução criada, corrente e incumbente
3    Cria  $S'$ , conjunto para armazenar a população de soluções iniciais
4     $S' := \text{hConstrutivaGulosa}(\textit{tamPop}, \textit{tamListaRestritaJ}, \textit{tamListaRestritaM})$ 
5     $s^* \leftarrow \text{encontraMelhorSolucao}(S')$ 
6    Para cada  $s' \in S'$  faça:
7      contador  $\leftarrow 1$ 
8      Inicializa  $S' := \emptyset$  para armazenar a população de soluções correntes aceitas
9      Enquanto contador  $\leq \textit{numRecomecosSemMelhora}$  faça:
10        $S := \text{geraVizinhanca}(s')$ 
11        $s' \leftarrow \text{encontraMelhorSolucao}(S, S')$ 
12       Se  $\text{checaViabilidade}(s') = \text{Verdadeiro}$  E  $z(s') < z(s^*)$  então:
13         contador  $\leftarrow 0$ 
14          $s^* \leftarrow s'$ 
15       contador  $\leftarrow \textit{contador} + 1$ 
16  Retorna  $s^*$ 
17  Fim

```

Primeiramente, a heurística construtiva é usada para gerar uma população heterogênea de soluções viáveis, ou seja, de custo total diferente entre si. Então, executa-se um algoritmo de Busca Local partindo de cada solução desta população. Assim como na BLMR, a solução corrente (calculada pelo mesmo método, `encontraMelhorSolução`, conforme linha 11 do Pseudocódigo 9) pode ter janelas de tempo rompidas, então sua viabilidade é checada antes de comparar ao seu custo com a solução incumbente, para definir se esta última será atualizada (linhas 12 e 14). Contudo, o algoritmo adotado na busca é diferente, uma vez que o contador de iterações do processo de construção de vizinhança e adoção da melhor solução vizinha

é zerado toda vez que a solução aceita é melhor que a incumbente (linha 13 do Pseudocódigo 9). Ou seja, o número de iterações não é fixo como na BLMR e a busca só é finalizada quando estiver estagnada, segundo critério de estagnação pré-determinado (*numRecomecosSemMelhoria*).

Este método, além de simples de ser implementado, possui a vantagem de poder ser paralelizado, uma vez que as execuções de busca iniciada nas diferentes soluções iniciais não possuem relação entre si. Assim, se houver capacidade computacional para executar 4 algoritmos de busca local simultaneamente, por exemplo, será possível finalizar o GRASP com uma população inicial de 4 soluções no tempo que seria necessário para executar apenas uma das buscas locais.

5.4 REMOÇÃO DE VETOR DE TAREFAS COM MÚLTIPLOS RECOMEÇOS

Como apontado pela literatura de PRVJTSin, a interdependência entre as rotas de uma solução do problema, gerada pela necessidade de atuação simultânea de mais de um rebocador por tarefa, torna os movimentos de modificação de soluções e construção de vizinhança muito caros computacionalmente. Nestes casos, convém adotar a abordagem de destruir-e-reparar (R&R, de *Ruin-and-Recreate*), que além de não examinar toda uma vizinhança, exerce modificações mais abrangentes na solução corrente, o que aumenta as chances de chegar a uma solução viável no processo de reparo.

De fato, ao longo dos testes realizados com os algoritmos de busca local, constatou-se que o movimento de remoção e reinserção de tarefas é demorado e gera poucas soluções viáveis, exigindo um número alto de iterações para melhorar a solução inicial. Então, baseando-se no algoritmo de *Slack Induction by String Removal* (SISR, vide seção 2.5.4 desta tese), foi desenvolvido o método **Remoção de Vetor de Tarefas com Múltiplos Recomeços** (RVTMR), apresentado no Pseudocódigo 10.

Assim como na BLMR, o RVTMS é reiniciado algumas vezes, partindo da solução incumbente, até atingir o limite pré-determinado. Partindo de uma solução incumbente (que na primeira iteração é a solução gerada pela heurística construtiva), é feita uma modificação e reconstrução desta solução, chegando a uma nova solução corrente. Caso a solução corrente seja melhor que a incumbente, atualiza-se esta última e o contador de iterações sem melhoria é reiniciado. Então, a solução corrente é modificada e reconstruída, com o resultado novamente comparado à solução incumbente. Isto é

repetido até alcançar um certo número de iterações sem melhoria, também pré-determinado.

Pseudocódigo 10 – Algoritmo RVTMR implementado

```

1  Algoritmo hRVTMR(tamListaRestritaJ, tamListaRestritaM, maxItGlobais, numRecomecosSemMelhora,
   maxString, numConfigPorModo):
2    Cria s, s' e s*, para armazenar soluções parcial, corrente e incumbente
3    s* ← hConstrutivaGulosa(1, tamListaRestritaJ, tamListaRestritaM)
4    contadorG ← 1
5    Enquanto contadorG ≤ maxItGlobais faça:
6      Inicializa S := ∅ e J := ∅, memória para a busca local
7      s ← s*
8      contadorL ← 1
9      Enquanto contadorL ≤ numRecomecosSemMelhora faça:
10       arefaSemente ← sorteiaSementeEatualizaMemoria(n, J)
11       tamString ← sorteiaTamanho(2, maxString)
12       s ← destrói&repara(s, S, arefaSemente, tamString, numConfigPorModo)
13       S := S ∪ s
14       Se z(s) < z(s*) então:
15         s* ← s
16         contadorL ← 1
17       Senão
18         contadorL ← contadorL + 1
19       contadorG ← contadorG + 1
20     Retorna s*
21 Fim

```

Nota-se que, diferentemente de meta-heurísticas como ILS e VNS, a solução corrente é substituída em todas as iterações da RVTMS. Isto porque, nos testes preliminares do método, a versão do algoritmo que só altera a solução corrente caso a vizinha gerada seja melhor teve desempenho pior do que esta apresentada.

Na RVTMS, o movimento de destruição remove um subconjunto de tarefas da solução corrente e em seguida as reposiciona na solução. A composição deste subconjunto é aleatória e determinada por duas variáveis: uma tarefa semente e o número de tarefas removidas, sorteado de 2 até γ , sendo γ uma fração pré-estabelecida do total de tarefas n , tal que $\gamma \in \mathbb{Z}$, $2 \leq \gamma \leq n$. O movimento remove a tarefa semente e mais $\gamma - 1$ tarefas mais próximas a ela, classificadas segundo uma função denominada *adjacência*(i, j), definida abaixo:

$$adjacência(i, j) = \frac{1}{(|a_i - a_j| + |b_i - b_j| + |D_i - D_j| + Dist_{ij} + Dist_{ji})} \quad \forall i, j \in J, i \neq j \quad \text{Eq. 5-1}$$

E parâmetro tem como intuito refletir a relação entre duas tarefas demandadas, pela diferença nas suas janelas de tempo, duração e pela distância geográfica entre elas. Quanto menor for a soma de tais diferenças, maior a proximidade entre as tarefas.

Na implementação do algoritmo, foi definido um vetor $adj(i)$ para cada tarefa $i \in \mathcal{J}$, contendo os índices das tarefas $j \in \mathcal{J}: j \neq i$ ordenados decrescentemente em função do valor de $adjacência(i, j)$.

O algoritmo usado para destruir e reparar a solução corrente é apresentado Pseudocódigo 11. Primeiramente, as primeiras $tamString - 1$ tarefas do conjunto adjacente à semente são removidas da solução original, s , assim como a própria tarefa semente. Por exemplo, supondo que a tarefa 5, com $adj(5) = \{9, 3, 7, 6, 8, 4, 1, 10, 2\}$, seja escolhida como semente de uma sequência de remoção com tamanho 3, o movimento de remoção de vetor tira as tarefas 5, 9 e 3 da solução corrente, e armazena a solução parcial em s' .

Pseudocódigo 11 – Algoritmo destrói e repara, usado no RVMTR

```

1  Algoritmo destrói&repara( $s, \mathcal{S}, tarefaSemente, tamString, numConfigPorModo$ ):
2    Inicializa  $\mathcal{J}' := \emptyset$  para armazenar tarefas removidas
3     $s' \leftarrow s \setminus \{tarefaSemente\}$ 
4    Para cada  $i \in [1, tamString - 1]$  faça:
5       $s' \leftarrow s' \setminus \{adj(tarefaSemente)[i]\}$ 
6       $\mathcal{J}' \cup adj(tarefaSemente)[i]$ 
7    ordenaTarefas( $regraOrdenacao, \mathcal{J}'$ )
8    Enquanto  $|\mathcal{J}'| > 0$  faça:
9       $j \leftarrow \text{retornaPrimeiraTarefa}(\mathcal{J}')$ 
10      $\mathcal{J}' := \mathcal{J}' \setminus \{j\}$ 
11     Inicializa  $\mathcal{S}'' := \emptyset$  para armazenar possibilidade de reinserção de  $j$ 
12     Para cada  $modo \in \text{ModosCompatíveis}(j)$ 
13       Para cada  $configuracao \in [1, numConfigPorModo]$  faça:
14          $\mathcal{M}^j \leftarrow \text{sorteiaConfiguracaoCompativel}(modo)$ 
15          $s'' \leftarrow \text{insereTarefaNasMelhoresPosicoesDaConfiguracao}(s', j, \mathcal{M}^j)$ 
16          $\mathcal{S}'' \cup s''$ 
17      $s' \leftarrow \text{escolheMelhorSolucaoParcial}(\mathcal{S}'')$ 
18     Retorna  $s'$ 
19  Fim

```

Em seguida, o conjunto de tarefas removidas é ordenado não-decrescentemente de acordo com seu grau de compatibilidade, de forma que a primeira tarefa do conjunto seja a menos compatível com as demais, portanto mais difícil de encaixar na solução parcial sem romper janelas de atendimento. Uma-a-uma, as tarefas deste conjunto ordenado são reinseridas na solução, de maneira similar à da heurística construtiva. Todos os modos compatíveis com a tarefa são examinados, contudo, ao invés de averiguar todos os subconjuntos de rebocadores associados a cada modo, apenas algumas destas configurações são examinadas. A quantidade de configurações averiguadas é determinada pelo parâmetro $numConfigPorModo$ e a escolha destas configurações é feita aleatoriamente pela função `sorteiaConfiguracaoCompativel`.

A ideia de não olhar todas as opções de inserção da tarefa é baseada no *blinking rate* do SISR (CHRISTIAENS; VANDEN BERGHE, 2020), cuja proposta é não só diminuir o tempo computacional, mas diversificar as soluções examinadas, por não reconstruir sempre da mesma forma a solução previamente destruída. Enquanto aqueles autores não consideram algumas posições de inserção do cliente na rota de um veículo, escolhidas aleatoriamente, aqui as configurações compatíveis com os modos é que não são consideradas à exaustão.

Para cada configuração sorteada, são avaliadas todas as possibilidades que não geram subciclo e, dentre elas, é escolhido o conjunto de posições de inserção (ou a posição, caso a tarefa demande apenas um rebocador) que resulte no menor acréscimo no custo total. Então, a tarefa é reinserida na solução s' em tais posições, e o processo é recommçado para a próxima tarefa. Uma vez que todas as tarefas removidas tenham sido reinseridas na solução, o algoritmo retorna a nova solução completa, s' .

6 EXPERIMENTOS COMPUTACIONAIS

Este capítulo inicia-se com a apresentação de instâncias para o **PRVJTSin-Het**, na seção 6.1. Então, na seção 6.2.1 os modelos matemáticos propostos para o problema (vide capítulo 3) são testados nas instâncias menores, com intuito de identificar a melhor estratégia de modelagem para adoção nos testes seguintes.

Uma vez que a obtenção de soluções ótimas para instâncias do **PRVJTSin-Het** de tamanho médio a grande exige muito tempo de processamento, as soluções geradas pelos métodos propostos aqui nem sempre podem ser avaliadas por comparação com o valor das soluções ótimas. Alternativamente, a avaliação das soluções é feita por comparação com os limitantes inferior, **Lower Bound (LB)**, e superior, **Upper Bound (UB)**, de cada instância, tratativa adotada na literatura para problemas NP-difíceis como o **PRVJTSin-Het**. Para sua obtenção, foi usado um pacote de otimização do mercado, Gurobi Optimizer versão 9.0.1, conforme documentado na seção 6.2.2.

Obtidos os limitantes que viabilizam a avaliação de soluções do **PRVJTSin-Het**, parte-se para as execuções dos métodos de solução, cujos resultados estão apresentados em 6.3 e analisados em 6.4. Como mais de um computador foi utilizado nos experimentos, as características de cada máquina são apresentadas junto aos respectivos resultados, ao longo do texto deste capítulo.

6.1 PROPOSIÇÃO DE INSTÂNCIAS

As 36 instâncias propostas para este trabalho referem-se a um horizonte de planejamento de um mês e subdividem-se em três conjuntos de mesmo tamanho (ou seja, há 12 instâncias de cada tipo): (i) pequenas, de 10 tarefas; (ii) médias, de 20; e (iii) grandes, de 30. Tais instâncias se baseiam naquelas propostas por Mendes (2007), com modificações sugeridas pelo próprio autor, da seguinte maneira: rebocadores de uma mesma classe passaram a ter mesma velocidade média de navegação e mesmo custo variável; e o tamanho da frota foi reduzido nas instâncias de 20 tarefas.

Nesta seção, além de descrever os conjuntos de dados propostos, as instâncias do **PRVJTSin-Het** são comparadas com instâncias de problemas similares, que já estão difundidas na literatura. Desta forma, busca-se contextualizar o problema tratado nesta pesquisa, evidenciando as diferenças entre este e problemas já abordados por outros autores. Além disto, são feitas análises de sensibilidade dos parâmetros que diferenciam as instâncias propostas, mostrando o impacto de cada um na complexidade do problema estudado.

6.1.1 Apresentação das instâncias do PRVJTSin-Het

Os seguintes parâmetros são usados para descrever as instâncias:

- Número de tarefas demandadas, n ;
- Percentual de tarefas que demandam mais de um recurso, % *sincr.*;
- Número de embarcações de cada classe, m_s , e total, m ;
- Horizonte de planejamento, em dias, H ;
- Maior limitante superior de janela de atendimento de tarefa, $\max b_i$;
- Duração média das tarefas, $\bar{D}_i$, também em dias;
- Amplitude máxima da janela de atendimento das tarefas, definida pelo número máximo de dias entre a data de início desejada e os limitantes inferior e superior da janela, amp ;
- Número médio de embarcações demandas para atendimento de uma tarefa, $\bar{Q}_i$;
- O grau de ocupação nominal das embarcações, que representa a relação entre o tempo total das atividades demandadas e o tempo total disponível, $ocupação = (n \times \bar{D}_i \times \bar{Q}_i) / (m \times \max b_i)$;
- Número de classes de embarcações, c ;
- Duração mínima e máxima das tarefas, D_{min} e D_{max} ;
- Valor máximo e mínimo das penalidades diárias por atraso, β_{max} e β_{min} ;
- Probabilidade do local de início e fim da tarefa ser o mesmo, $prob_{i=j}$;
- Faixa de valores das coordenadas x e y associadas aos locais geográficos que fazem parte da instância $coord_{max}$ e $coord_{min}$

As tabelas a seguir apresentam os valores de tais parâmetros. Primeiramente, na Tabela 3 são mostrados os parâmetros fixos, ou seja, que possuem o mesmo valor para as 36 instâncias testadas. Já a Tabela 4 lista, para cada instância usada, os valores dos parâmetros que variam conforme a instância.

Tabela 3 – Parâmetros fixados para as 36 instâncias.

Parâmetro	Valor (unidade)
H	30 (dias)
c	2 (classes)
amp	5 a 7 (dias)
D_{max}	18 (dias)
D_{min}	2 (dias)
β_{max}	320.000 (\$/dia)
β_{min}	0 (\$/dia)
$prob_{i=j}$	80 (%)
$coord_{max}$	300 (milhas náuticas)
$coord_{min}$	0 (milha náutica)

Tabela 4 – Parâmetros descritivos das instâncias usadas nos experimentos computacionais deste trabalho, destacando alterações em relação ao trabalho de Mendes (2007): número de tarefas, percentual de tarefas sincronizadas, frota total e por classe, duração média das tarefas, quantidade média de rebocadores por tarefa e ocupação

ID da Instância	n	% <i>sincr.</i>	Frota total m		Frota por classe		$\overline{D}_i$	$\overline{Q}_i$	Ocupação
			Mendes (2007)	Este trabalho	m_1	m_2			
1	10	80%	12	12	8	4	9,9	2,4	83%
2	10	100%	12	12	8	4	9,9	2,6	79%
3	10	90%	13	13	9	4	10,1	2,8	91%
4	10	90%	13	13	8	5	11,9	2,4	71%
5	10	90%	12	12	8	4	12,1	2,6	105%
6	10	90%	16	16	11	5	12	2,8	75%
7	10	100%	11	11	8	3	10,1	2,4	82%
8	10	80%	12	12	8	4	9,9	2,6	77%
9	10	90%	13	13	9	4	10,1	2,8	68%
10	10	80%	12	12	8	4	12	2,4	89%
11	10	70%	14	14	10	4	12	2,6	68%
12	10	100%	15	15	10	5	12	2,8	83%
13	20	75%	22	19	14	5	9,9	2,4	100%
14	20	85%	23	20	12	8	9,9	2,6	89%
15	20	95%	27	23	15	8	9,9	2,8	89%
16	20	75%	27	27	20	7	11,95	2,4	73%
17	20	85%	29	25	18	7	11,95	2,6	104%
18	20	90%	31	25	18	7	12,05	2,8	100%
19	20	80%	22	17	11	6	10	2,4	94%
20	20	90%	27	16	11	5	9,5	2,6	91%
21	20	100%	27	24	17	7	9,1	2,8	73%
22	20	75%	25	19	13	6	12	2,4	126%
23	20	85%	29	21	16	5	12,05	2,6	99%
24	20	95%	31	23	15	8	12	2,8	91%
25	30	80%	20	20	14	6	9,9	2,4	85%
26	30	83%	21	21	14	7	10,1	2,6	91%
27	30	90%	23	23	15	8	10,1	2,8	82%
28	30	87%	24	24	15	9	12,0	2,4	97%
29	30	87%	25	25	16	9	12,0	2,6	83%
30	30	93%	26	26	17	9	12,1	2,8	89%
31	30	80%	21	21	14	7	10,0	2,4	79%
32	30	80%	22	22	16	6	10,0	2,6	89%
33	30	83%	24	24	16	8	10,0	2,8	75%
34	30	80%	24	24	16	8	12,1	2,4	84%
35	30	87%	28	28	19	9	12,0	2,6	73%
36	30	90%	29	29	19	10	12,0	2,8	87%

6.1.2 Comparação com instâncias da literatura

Deixando de lado as diferenças relativas à definição do problema em si, como a existência de modos para realização das tarefas ou a diversidade da frota disponível, a

principal característica que dificulta a solução das instâncias do PRVJTSin-Het é o alto percentual de tarefas com exigência de mais de um recurso e a quantidade de rebocadores demandadas por tarefa. Todas as publicações do PRVJT-S ou PRVJT-SP revisadas para este trabalho possuem valores inferiores desses dois parâmetros.

Para facilitar esta comparação, a Tabela 5 apresenta os valores médios do percentual de clientes com restrição de sincronização ($sincr./n$ [%]), da quantidade média ($\overline{Q_i}$) e máxima ($\max Q_i$) de recursos por tarefa, para as publicações anteriores e para esta pesquisa, além da quantidade de clientes.

Tabela 5 – Parâmetros-chaves das instâncias da literatura e das propostas neste trabalho.

Parâmetro	n	$sincr./n$ [%]	Classes da frota (c)	$\overline{Q_i}$	$\max Q_i$
Este trabalho	10 a 30	87%	2	2,6	4
Parragh e Doerner (2018)	20 a 80	10%	1	<2	2
Hojabri <i>et al.</i> , (2018)	25 a 200	5, 25 e 50%	1	<2	2
Ait Haddadene, Labadie e Prodhon (2016)	20 a 80	10%	1	<2	2
Rousseau, Gendreau e Pesant, (2013)	100 a 1.000	10%	1	<2	2
Liu, Tao e Xie (2019)	30 a 200	10, 30 e 50%: 30 clientes 5 a 20%: 100 e 200 clientes	1	<2	2

A partir da tabela, nota-se que as instâncias propostas neste trabalho possuem maior percentual de tarefas sincronizadas do que os *benchmarks* de problemas similares (para mais informações sobre esses trabalhos, consultar subseção 2.3.3 do capítulo de revisão da literatura). Ainda, a média de recursos demandados por tarefa é maior, assim como a quantidade máxima de recursos simultâneos. Isso significa que a interdependência entre rotas é maior para PRVJTSin-Het, uma vez que é maior a probabilidade de uma simples alteração em uma rota, como a remoção de uma única tarefa, impactar os cronogramas dos outros rebocadores.

Com intuito de identificar o impacto destes parâmetros na complexidade (e consequente dificuldade de solução) do problema, foram realizadas análises de sensibilidade variando seus valores, como apresentado a seguir.

6.1.3 Análise de sensibilidade dos parâmetros que diferenciam as instâncias propostas de outras instâncias da literatura

As instâncias propostas para o PRVJTSin-Het possuem características distintas daquelas já discutidas na literatura, não só por se tratar de um problema novo, mas porque algumas características da aplicação real são diferentes dos benchmarks encontrados, como por exemplo o percentual de clientes com restrição de sincronização. Para avaliar o impacto destes parâmetros na complexidade do problema, foram feitas modificações em seus valores de forma estruturada, gerando novas instâncias, derivadas das originais. Buscou-se alterar cada parâmetro de maneira isolada, isto é, evitando alterar os valores dos demais parâmetros.

Apenas as instâncias pequenas (de 10 tarefas) fizeram parte deste estudo, já que testes com as instâncias médias e grandes exigem maior tempo computacional, sem agregar maior valor às análises do que as menores. Os parâmetros variados para a análise de sensibilidade foram:

- Quantidade de classes: foram geradas instâncias com 1 classe de recursos. Para isso, todos os rebocadores de classe 2 passaram a fazer parte da classe 1 e a velocidade e os custos foram modificados para os valores médios da instância original. Na demanda das tarefas, foi mantida a quantidade total de rebocadores demandados (Q_i), de forma que eventuais demandas de classe maior foram transferidas para a classe inferior, fazendo $q_i^1 = Q_i$.
- Percentual de tarefas com restrição de sincronização: foram geradas instâncias com 10 a 100% de tarefas sincronizadas, variando de 10 em 10 pontos percentuais. Nos casos em que foi possível, manteve-se a quantidade total de rebocadores demandados, $Q_{total} = \sum_i Q_i$. Para os casos em que o novo percentual é maior que o original, buscou-se realocar rebocadores de tarefas cuja demanda é superior a 2 para as tarefas que, na instância original, demandam apenas 1 rebocador, evitando diminuir a quantidade máxima de 4 rebocadores por tarefa. Já para as alterações com diminuição na quantidade de tarefas sincronizadas, fez-se o oposto, de forma que na maioria dos casos não foi possível manter a quantidade máxima de rebocadores por tarefa, já que isso exigiria diminuir o valor de Q_{total} . As classes da demanda de rebocadores remanejada foram mantidas.

- Quantidade média de rebocadores por tarefa: gerou-se instâncias de 2,0 a 3,0 rebocadores por tarefa ($\bar{Q}_i$), com incrementos de 0,2. Para isto, priorizou-se aumentar as demandas de rebocadores de classe 1 (aumentando ou diminuindo seus valores) e, apenas nos casos em que não foi possível chegar ao valor almejado sem extrapolar o valor máximo de rebocadores por tarefa, mexeu-se na demanda de classe superior. Desta forma, não foi alterada a quantidade máxima de rebocadores por tarefa. Contudo, não foi possível manter fixo o percentual de tarefas sincronizadas, o que dificulta a avaliação do impacto deste fator isoladamente.

Os parâmetros descritivos das instâncias geradas estão apresentados a seguir, assim com os seus valores nas instâncias originais. As instâncias estão identificadas com seu ID original mais “-O” para as originais; “-C” para as geradas na variação da quantidade de classes; “-S” para variações na quantidade de tarefas sincronizadas; e “-Q” para alterações na demanda por tarefa. Em todas as tabelas desta seção, os valores em **negrito** indicam alterações em relação à instância original.

Tabela 6 – Parâmetros descritivos das instâncias usadas nos experimentos de análise de sensibilidade do número de classes. Os valores alterados estão em **negrito**.

ID – análise	n	% <i>sincr.</i>	Frota total m	Frota por classe		$\bar{Q}_i$	$\max_i$	Ocupação
				m_1	m_2			
1-O	10	80%	12	8	4	2,4	4	83%
1-C	10	80%	12	12	0	2,4	4	83%
2-O	10	100%	12	8	4	2,6	4	79%
2-C	10	100%	12	12	0	2,6	4	79%
3-O	10	90%	13	9	4	2,8	4	91%
3-C	10	90%	13	13	0	2,8	4	91%
4-O	10	90%	13	8	5	2,4	4	71%
4-C	10	90%	13	13	0	2,4	4	71%
5-O	10	90%	12	8	4	2,6	4	105%
5-C	10	90%	12	12	0	2,6	4	105%
6-O	10	90%	16	11	5	2,8	4	75%
6-C	10	90%	16	16	0	2,8	4	75%
7-O	10	100%	11	8	3	2,4	4	82%
7-C	10	100%	11	11	0	2,4	4	82%
8-O	10	80%	12	8	4	2,6	4	77%
8-C	10	80%	12	12	0	2,6	4	77%
9-O	10	90%	13	9	4	2,8	4	68%
9-C	10	90%	13	13	0	2,8	4	68%
10-O	10	80%	12	8	4	2,4	4	89%
10-C	10	80%	12	12	0	2,4	4	89%
11-O	10	70%	14	10	4	2,6	4	68%
11-C	10	70%	14	14	0	2,6	4	68%
12-O	10	100%	15	10	5	2,8	4	83%
12-C	10	100%	15	15	0	2,8	4	83%

Na Tabela 6, são apresentadas as instâncias geradas para análise do impacto das classes na complexidade da instância, destacando-se as modificações em negrito. Além da alteração na quantidade de rebocadores de cada classe, que aparece na tabela, deve-se ter em mente que a velocidade das embarcações também foi alterada para a média ponderada das velocidades da instância em geral. Ou seja:

$$v_{instâncias "-c"} = \frac{\sum_{s=1}^c m_s v^s}{m} \quad \text{Eq. 6-1}$$

As instâncias geradas por alteração no percentual de tarefas sincronizadas são apresentadas na Tabela 7, usando os mesmos parâmetros descritivos. É importante destacar que para manter a média de rebocadores por tarefa, a quantidade máxima de rebocadores por tarefa foi alterada em muitas das instâncias, como se observa na tabela. Os demais parâmetros foram mantidos.

Tabela 7 – Parâmetros descritivos das instâncias usadas nos experimentos de análise de sensibilidade do percentual de clientes com restrição de sincronização (ou seja, com quantidade de rebocadores demandados superior a um). Valores em negrito sinalizam alteração em relação à instância original.

ID – análise	n	% <i>sincr.</i>	Frota total <i>m</i>	Frota por classe		$\overline{Q}_i$	max Q_i	Ocupação
				<i>m</i> ₁	<i>m</i> ₂			
1-S	10	20%	12	8	4	2,4	8	83%
1-S	10	40%	12	8	4	2,4	5	83%
1-S	10	60%	12	8	4	2,4	4	83%
1-O	10	80%	12	8	4	2,4	4	83%
1-S	10	100%	12	8	4	2,4	4	83%
2-S	10	20%	12	8	4	2,6	10	79%
2-S	10	40%	12	8	4	2,6	7	79%
2-S	10	60%	12	8	4	2,6	5	79%
2-S	10	80%	12	8	4	2,6	4	79%
2-O	10	100%	12	8	4	2,6	4	79%
3-S	10	20%	13	9	4	2,8	10	91%
3-S	10	40%	13	9	4	2,8	7	91%
3-S	10	60%	13	9	4	2,8	6	91%
3-S	10	80%	13	9	4	2,8	4	91%
3-O	10	90%	13	9	4	2,8	4	91%
3-S	10	100%	13	9	4	2,8	4	91%
4-S	10	20%	13	8	5	2,4	9	71%
4-S	10	40%	13	8	5	2,4	6	71%
4-S	10	60%	13	8	5	2,4	5	71%
4-S	10	80%	13	8	5	2,4	4	71%
4-O	10	90%	13	8	5	2,4	4	71%
4-S	10	100%	13	8	5	2,4	4	71%
5-S	10	20%	12	8	4	2,6	10	105%
5-S	10	40%	12	8	4	2,6	5	105%
5-S	10	60%	12	8	4	2,6	5	105%
5-S	10	80%	12	8	4	2,6	5	105%
5-O	10	90%	12	8	4	2,6	4	105%
5-S	10	100%	12	8	4	2,6	3	105%
6-S	10	20%	16	11	5	2,8	11	75%

ID – análise	n	% <i>sincr.</i>	Frota total <i>m</i>	Frota por classe		$\overline{Q}_i$	max Q_i	Ocupação
				m_1	m_2			
6-S	10	40%	16	11	5	2,8	7	75%
6-S	10	60%	16	11	5	2,8	5	75%
6-S	10	80%	16	11	5	2,8	5	75%
6-O	10	90%	16	11	5	2,8	4	75%
6-S	10	100%	16	11	5	2,8	4	75%
7-S	10	20%	11	8	3	2,4	9	82%
7-S	10	40%	11	8	3	2,4	6	82%
7-S	10	60%	11	8	3	2,4	5	82%
7-S	10	80%	11	8	3	2,4	5	82%
7-O	10	100%	11	8	3	2,4	4	82%
8-S	10	20%	12	8	4	2,6	10	77%
8-S	10	40%	12	8	4	2,6	6	77%
8-S	10	60%	12	8	4	2,6	5	77%
8-O	10	80%	12	8	4	2,6	4	77%
8-S	10	100%	12	8	4	2,6	4	77%
9-S	10	20%	13	9	4	2,8	10	68%
9-S	10	40%	13	9	4	2,8	7	68%
9-S	10	60%	13	9	4	2,8	5	68%
9-S	10	80%	13	9	4	2,8	5	68%
9-O	10	90%	13	9	4	2,8	4	68%
9-S	10	100%	13	9	4	2,8	4	68%
10-S	10	20%	12	8	4	2,4	8	89%
10-S	10	40%	12	8	4	2,4	5	89%
10-S	10	60%	12	8	4	2,4	4	89%
10-O	10	80%	12	8	4	2,4	4	89%
10-S	10	100%	12	8	4	2,4	3	89%
11-S	10	20%	14	10	4	2,6	9	68%
11-S	10	40%	14	10	4	2,6	7	68%
11-S	10	60%	14	10	4	2,6	4	68%
11-O	10	70%	14	10	4	2,6	4	68%
11-S	10	80%	14	10	4	2,6	4	68%
11-S	10	100%	14	10	4	2,6	4	68%
12-S	10	20%	15	10	5	2,8	11	83%
12-S	10	40%	15	10	5	2,8	7	83%
12-S	10	60%	15	10	5	2,8	6	83%
12-S	10	80%	15	10	5	2,8	5	83%
12-O	10	100%	15	10	5	2,8	4	83%

Por fim, a Tabela 8 apresenta as instâncias geradas pela variação da média de rebocadores demandados por tarefa. Novamente, parâmetros que sofreram alteração com relação ao seu valor na instância original aparecem em negrito.

Nota-se que, para não alterar a demanda máxima de rebocadores por tarefa, em alguns casos o percentual de tarefas sincronizada sofreu alterações. Além disso, a ocupação varia junto com a demanda média, como se observa na tabela.

Tabela 8 – Parâmetros descritivos das instâncias usadas nos experimentos de análise de sensibilidade da quantidade média de rebocadores demandados por tarefa. Novamente, valores em negrito sinalizam alteração em relação à instância original.

ID – análise	n	% <i>sincr.</i>	Frota total m	Frota por classe		$\overline{Q}_i$	max Q_i	Ocupação
				m_1	m_2			
1-Q	10	50%	12	8	4	2	4	69%
1-Q	10	60%	12	8	4	2,2	4	76%
1-O	10	80%	12	8	4	2,4	4	83%
1-Q	10	100%	12	8	4	2,6	4	89%
1-Q	10	100%	12	8	4	2,8	4	96%
1-Q	10	100%	12	8	4	3	4	103%
2-Q	10	70%	12	8	4	2	4	61%
2-Q	10	70%	12	8	4	2,2	4	67%
2-Q	10	80%	12	8	4	2,4	4	73%
2-O	10	100%	12	8	4	2,6	4	79%
2-Q	10	100%	12	8	4	2,8	4	86%
2-Q	10	100%	12	8	4	3	4	92%
3-Q	10	50%	13	9	4	2	4	65%
3-Q	10	60%	13	9	4	2,2	4	71%
3-Q	10	60%	13	9	4	2,4	4	78%
3-Q	10	80%	13	9	4	2,6	4	84%
3-O	10	90%	13	9	4	2,8	4	91%
3-Q	10	90%	13	9	4	3	4	97%
4-Q	10	50%	13	8	5	2	4	59%
4-Q	10	70%	13	8	5	2,2	4	65%
4-O	10	90%	13	8	5	2,4	4	71%
4-Q	10	90%	13	8	5	2,6	4	77%
4-Q	10	90%	13	8	5	2,8	4	83%
4-Q	10	100%	13	8	5	3	4	89%
5-Q	10	60%	12	8	4	2	4	81%
5-Q	10	70%	12	8	4	2,2	4	89%
5-Q	10	80%	12	8	4	2,4	4	97%
5-O	10	90%	12	8	4	2,6	4	105%
5-Q	10	90%	12	8	4	2,8	4	113%
5-Q	10	100%	12	8	4	3	4	121%
6-Q	10	50%	16	11	5	2	4	54%
6-Q	10	60%	16	11	5	2,2	4	59%
6-Q	10	60%	16	11	5	2,4	4	64%
6-Q	10	80%	16	11	5	2,6	4	70%
6-O	10	90%	16	11	5	2,8	4	75%
6-Q	10	100%	16	11	5	3	4	80%
7-Q	10	80%	11	8	3	2	4	68%
7-Q	10	90%	11	8	3	2,2	4	75%
7-O	10	100%	11	8	3	2,4	4	82%
7-Q	10	100%	11	8	3	2,6	4	88%
7-Q	10	100%	11	8	3	2,8	4	95%
7-Q	10	100%	11	8	3	3	4	102%
8-Q	10	40%	12	8	4	2	4	59%
8-Q	10	60%	12	8	4	2,2	4	65%
8-Q	10	70%	12	8	4	2,4	4	71%
8-O	10	80%	12	8	4	2,6	4	77%
8-Q	10	90%	12	8	4	2,8	4	83%
8-Q	10	90%	12	8	4	3	4	88%
9-Q	10	50%	13	9	4	2	4	49%
9-Q	10	60%	13	9	4	2,2	4	53%
9-Q	10	70%	13	9	4	2,4	4	58%

ID – análise	n	% <i>sincr.</i>	Frota total <i>m</i>	Frota por classe		$\overline{Q}_i$	max Q_i	Ocupação
				m_1	m_2			
9-Q	10	80%	13	9	4	2,6	4	63%
9-O	10	90%	13	9	4	2,8	4	68%
9-Q	10	90%	13	9	4	3	4	73%
10-Q	10	70%	12	8	4	2	4	74%
10-Q	10	70%	12	8	4	2,2	4	81%
10-O	10	80%	12	8	4	2,4	4	89%
10-Q	10	100%	12	8	4	2,6	4	96%
10-Q	10	100%	12	8	4	2,8	4	104%
10-Q	10	100%	12	8	4	3	4	111%
11-Q	10	40%	14	10	4	2	4	52%
11-Q	10	50%	14	10	4	2,2	4	57%
11-Q	10	60%	14	10	4	2,4	4	62%
11-O	10	70%	14	10	4	2,6	4	68%
11-Q	10	80%	14	10	4	2,8	4	73%
11-Q	10	100%	14	10	4	3	4	78%
12-Q	10	60%	15	10	5	2	4	59%
12-Q	10	80%	15	10	5	2,2	4	65%
12-Q	10	90%	15	10	5	2,4	4	71%
12-Q	10	90%	15	10	5	2,6	4	77%
12-O	10	100%	15	10	5	2,8	4	83%
12-Q	10	100%	15	10	5	3	4	89%

A seguir, são apresentados os resultados do Gurobi para as instâncias geradas para análise de sensibilidade, usando o modelo matemático compacto (vide seção 2.4.3 desta tese). As instâncias geradas foram executadas utilizando o Gurobi Interactive Shell para Python, no IDE Spyder versão 3.6.6, em notebook iOS com processador de dois núcleos Intel® Core™ i5 @2.4-2.9GHz e 8GB @1600MHz DDR3L de memória RAM. Como o intuito desta análise é avaliar o impacto da variação dos valores dos parâmetros que diferenciam o PRVJTSin-Het de problemas já discutidos na literatura, é interessante avaliar o tempo computacional que o otimizador leva para chegar a uma solução ótima – ou o GAP final ao alcançar limite de tempo pré-estabelecido para estes testes, que foi igual a 1 hora (3.600 segundos).

A Tabela 9 apresenta os resultados da análise de sensibilidade do número de classes do problema, enquanto a **Figura 8** permite uma comparação visual entre os tempos de processamento para as instâncias originais e modificadas.

Como esperado, ao tornar a frota homogênea, o Gurobi chegou mais rapidamente à solução ótima na maioria das instâncias (apenas nas instâncias 4-C e 9-C houve aumento). Em média, houve uma redução de 52% do tempo de processamento, considerando as duas instâncias em que houve aumento do tempo. Excluindo estes dois casos, a redução média foi de 71%.

Figura 8 – Gráfico com resultados da análise de sensibilidade do número de classes das instâncias de 10 tarefas

Tabela 9 – Resultados dos experimentos de análise de sensibilidade do número de classes. Os valores alterados estão em negrito.

ID – análise	% <i>sincr.</i>	Frota total m	Frota por classe		$\bar{Q}_i$	$\max Q_i$	Ocupação	Resultado Gurobi (1h)	
			m_1	m_2				GAP final	Tempo de execução [s]
1-O	80%	12	8	4	2,4	4	83%	0%	4,64
1-C	80%	12	12	0	2,4	4	83%	0%	1,38
2-O	100%	12	8	4	2,6	4	79%	0%	0,96
2-C	100%	12	12	0	2,6	4	79%	0%	0,31
3-O	90%	13	9	4	2,8	4	91%	0%	4,78
3-C	90%	13	13	0	2,8	4	91%	0%	1,83
4-O	90%	13	8	5	2,4	4	71%	0%	0,63
4-C	90%	13	13	0	2,4	4	71%	0%	0,82
5-O	90%	12	8	4	2,6	4	105%	0%	3,20
5-C	90%	12	12	0	2,6	4	105%	0%	0,32
6-O	90%	16	11	5	2,8	4	75%	0%	2,09
6-C	90%	16	16	0	2,8	4	75%	0%	1,62
7-O	100%	11	8	3	2,4	4	82%	0%	6,76
7-C	100%	11	11	0	2,4	4	82%	0%	3,06
8-O	80%	12	8	4	2,6	4	77%	0%	3,39
8-C	80%	12	12	0	2,6	4	77%	0%	0,49
9-O	90%	13	9	4	2,8	4	68%	0%	1,58
9-C	90%	13	13	0	2,8	4	68%	0%	2,40
10-O	80%	12	8	4	2,4	4	89%	0%	3,10
10-C	80%	12	12	0	2,4	4	89%	0%	0,57
11-O	70%	14	10	4	2,6	4	68%	0%	0,95
11-C	70%	14	14	0	2,6	4	68%	0%	0,11
12-O	100%	15	10	5	2,8	4	83%	0%	7,46
12-C	100%	15	15	0	2,8	4	83%	0%	1,26

Tabela 10 – Resultados dos experimentos de análise de sensibilidade do percentual de clientes com restrição de sincronização. Valores em negrito sinalizam alteração em relação à instância original e linhas em tonalidade mais clara indicam instância inviável.

ID – análise	% <i>sinci</i>	Nome no gráfico	Frota total m	Frota por classe		$\bar{Q}_i$	max Q_i	Ocupação	Resultado Gurobi (1h)	
				m_1	m_2				GAP final	Tempo de execução [s]
1-S	20%	1-s-20%	12	8	4	2,4	8	83%	Inviável	Inviável
1-S	40%	1-s-40%	12	8	4	2,4	5	83%	0%	0,58
1-S	60%	1-s-60%	12	8	4	2,4	4	83%	Inviável	Inviável
1-O	80%	1-O-80%	12	8	4	2,4	4	83%	0%	4,64
1-S	100%	1-s-100%	12	8	4	2,4	4	83%	0%	0,98
2-S	20%	2-s-20%	12	8	4	2,6	10	79%	0%	0,25
2-S	40%	2-s-40%	12	8	4	2,6	7	79%	Inviável	Inviável
2-S	60%	2-s-60%	12	8	4	2,6	5	79%	0%	1,64
2-S	80%	2-s-80%	12	8	4	2,6	4	79%	Inviável	Inviável
2-O	100%	2-O-100%	12	8	4	2,6	4	79%	0%	0,96
3-S	20%	3-s-20%	13	9	4	2,8	10	91%	Inviável	Inviável
3-S	40%	3-s-40%	13	9	4	2,8	7	91%	0%	0,37
3-S	60%	3-s-60%	13	9	4	2,8	6	91%	Inviável	Inviável
3-S	80%	3-s-80%	13	9	4	2,8	4	91%	Inviável	Inviável
3-O	90%	3-O-90%	13	9	4	2,8	4	91%	0%	4,78
3-S	100%	3-s-100%	13	9	4	2,8	4	91%	0%	1,91
4-S	20%	4-s-20%	13	8	5	2,4	9	71%	Inviável	Inviável
4-S	40%	4-s-40%	13	8	5	2,4	6	71%	Inviável	Inviável
4-S	60%	4-s-60%	13	8	5	2,4	5	71%	Inviável	Inviável
4-S	80%	4-s-80%	13	8	5	2,4	4	71%	Inviável	Inviável
4-O	90%	4-O-90%	13	8	5	2,4	4	71%	0%	0,63
4-S	100%	4-s-100%	13	8	5	2,4	4	71%	0%	1,86
5-S	20%	5-s-20%	12	8	4	2,6	10	105%	Inviável	Inviável
5-S	40%	5-s-40%	12	8	4	2,6	5	105%	Inviável	Inviável
5-S	60%	5-s-60%	12	8	4	2,6	5	105%	Inviável	Inviável
5-S	80%	5-s-80%	12	8	4	2,6	5	105%	Inviável	Inviável
5-O	90%	5-O-90%	12	8	4	2,6	4	105%	0%	3,20
5-S	100%	5-s-100%	12	8	4	2,6	3	105%	0%	2,34
6-S	20%	6-s-20%	16	11	5	2,8	11	75%	0%	0,07
6-S	40%	6-s-40%	16	11	5	2,8	7	75%	0%	0,39
6-S	60%	6-s-60%	16	11	5	2,8	5	75%	0%	0,35
6-S	80%	6-s-80%	16	11	5	2,8	5	75%	0%	1,27
6-O	90%	6-O-90%	16	11	5	2,8	4	75%	0%	2,09
6-S	100%	6-s-100%	16	11	5	2,8	4	75%	0%	0,88
7-S	20%	7-s-20%	11	8	3	2,4	9	82%	Inviável	Inviável
7-S	40%	7-s-40%	11	8	3	2,4	6	82%	Inviável	Inviável
7-S	60%	7-s-60%	11	8	3	2,4	5	82%	Inviável	Inviável
7-S	80%	7-s-80%	11	8	3	2,4	5	82%	0%	9,24
7-O	100%	7-O-100%	11	8	3	2,4	4	82%	0%	6,76
8-S	20%	8-s-20%	12	8	4	2,6	10	77%	0%	0,45
8-S	40%	8-s-40%	12	8	4	2,6	6	77%	0%	0,60
8-S	60%	8-s-60%	12	8	4	2,6	5	77%	0%	2,02
8-O	80%	8-O-80%	12	8	4	2,6	4	77%	0%	3,39
8-S	100%	8-s-100%	12	8	4	2,6	4	77%	0%	0,62
9-S	20%	9-s-20%	13	9	4	2,8	10	68%	Inviável	Inviável

ID – análise	% <i>sinc_i</i>	Nome no gráfico	Frota total <i>m</i>	Frota por classe		$\bar{Q}_i$	max Q_i	Ocupação	Resultado Gurobi (1h)	
				<i>m₁</i>	<i>m₂</i>				GAP final	Tempo de execução [s]
9-S	40%	9-s-40%	13	9	4	2,8	7	68%	Inviável	Inviável
9-S	60%	9-s-60%	13	9	4	2,8	5	68%	Inviável	Inviável
9-S	80%	9-s-80%	13	9	4	2,8	5	68%	0%	4,40
9-O	90%	9-O-90%	13	9	4	2,8	4	68%	0%	1,58
9-S	100%	9-s-100%	13	9	4	2,8	4	68%	0%	2,16
10-S	20%	10-s-20%	12	8	4	2,4	8	89%	0%	0,63
10-S	40%	10-s-40%	12	8	4	2,4	5	89%	Inviável	Inviável
10-S	60%	10-s-60%	12	8	4	2,4	4	89%	0%	4,46
10-O	80%	10-O-80%	12	8	4	2,4	4	89%	0%	3,10
10-S	100%	10-s-100%	12	8	4	2,4	3	89%	0%	2,15
11-S	20%	11-s-20%	14	10	4	2,6	9	68%	0%	0,07
11-S	40%	11-s-40%	14	10	4	2,6	7	68%	Inviável	Inviável
11-S	60%	11-s-60%	14	10	4	2,6	4	68%	Inviável	Inviável
11-O	70%	11-O-70%	14	10	4	2,6	4	68%	0%	0,95
11-S	80%	11-s-80%	14	10	4	2,6	4	68%	0%	0,86
11-S	100%	11-s-100%	14	10	4	2,6	4	68%	0%	0,34
12-S	20%	12-s-20%	15	10	5	2,8	11	83%	0%	0,07
12-S	40%	12-s-40%	15	10	5	2,8	7	83%	0%	0,37
12-S	60%	12-s-60%	15	10	5	2,8	6	83%	0%	2,51
12-S	80%	12-s-80%	15	10	5	2,8	5	83%	0%	1,88
12-O	100%	12-O-100%	15	10	5	2,8	4	83%	0%	7,46

Por fim, são apresentadas as análises de sensibilidade da demanda média (em quantidade de rebocadores) das tarefas, na **Figura 10** e na Tabela 11.

Tempo de processamento do Gurobi para encontrar solução ótima das instâncias de 10 tarefas, com diferentes valores de demanda média por tarefa

Figura 10 – Gráfico com resultados da análise de sensibilidade da demanda média de rebocadores por tarefa, nas instâncias de 10 tarefas

Tabela 11 – Resultados da análise de sensibilidade da quantidade média de rebocadores demandados por tarefa. Valores em negrito sinalizam alteração em relação à instância original e instâncias inviáveis aparecem sinalizadas.

ID – análise	Nome no gráfico	% <i>sincr.</i>	Frota total m	Frota por classe		$\bar{Q}_i$	max Q_i	Ocupação	Resultados Gurobi (1h)	
				m_1	m_2				GAP	Tempo [s]
1-Q	1-Q-2,0	50%	12	8	4	2	4	69%	0%	0,27
1-Q	1-Q-2,2	60%	12	8	4	2,2	4	76%	0%	0,30
1-O	1-O-2,4	80%	12	8	4	2,4	4	83%	0%	4,64
1-Q	1-Q-2,6	100%	12	8	4	2,6	4	89%	0%	1,25
1-Q	1-Q-2,8	100%	12	8	4	2,8	4	96%	0%	1,67
1-Q	1-Q-3,0	100%	12	8	4	3	4	103%	0%	8,33
2-Q	2-Q-2,0	70%	12	8	4	2	4	61%	0%	0,10
2-Q	2-Q-2,2	70%	12	8	4	2,2	4	67%	0%	0,13
2-Q	2-Q-2,4	80%	12	8	4	2,4	4	73%	0%	0,30
2-O	2-O-2,6	100%	12	8	4	2,6	4	79%	0%	0,96
2-Q	2-Q-2,8	100%	12	8	4	2,8	4	86%	0%	1,51
2-Q	2-Q-3,0	100%	12	8	4	3	4	92%	0%	1,10
3-Q	3-Q-2,0	50%	13	9	4	2	4	65%	0%	0,11
3-Q	3-Q-2,2	60%	13	9	4	2,2	4	71%	0%	0,14
3-Q	3-Q-2,4	60%	13	9	4	2,4	4	78%	0%	0,22
3-Q	3-Q-2,6	80%	13	9	4	2,6	4	84%	0%	0,44
3-O	3-O-2,8	90%	13	9	4	2,8	4	91%	0%	4,78
3-Q	3-Q-3,0	90%	13	9	4	3	4	97%	0%	2,97
4-Q	4-Q-2,0	50%	13	8	5	2	4	59%	0%	0,21
4-Q	4-Q-2,2	70%	13	8	5	2,2	4	65%	0%	0,12
4-O	4-O-2,4	90%	13	8	5	2,4	4	71%	0%	0,63
4-Q	4-Q-2,6	90%	13	8	5	2,6	4	77%	0%	0,96
4-Q	4-Q-2,8	90%	13	8	5	2,8	4	83%	0%	0,54
4-Q	4-Q-3,0	100%	13	8	5	3	4	89%	inviável	inviável
5-Q	5-Q-2,0	60%	12	8	4	2	4	81%	0%	0,10
5-Q	5-Q-2,2	70%	12	8	4	2,2	4	89%	0%	0,10
5-Q	5-Q-2,4	80%	12	8	4	2,4	4	97%	0%	0,30
5-O	5-O-2,6	90%	12	8	4	2,6	4	105%	0%	3,20
5-Q	5-Q-2,8	90%	12	8	4	2,8	4	113%	0%	2,78
5-Q	5-Q-3,0	100%	12	8	4	3	4	121%	inviável	inviável
6-Q	6-Q-2,0	50%	16	11	5	2	4	54%	0%	0,10
6-Q	6-Q-2,2	60%	16	11	5	2,2	4	59%	0%	0,12
6-Q	6-Q-2,4	60%	16	11	5	2,4	4	64%	0%	0,27
6-Q	6-Q-2,6	80%	16	11	5	2,6	4	70%	0%	0,64
6-O	6-O-2,8	90%	16	11	5	2,8	4	75%	0%	2,09
6-Q	6-Q-3,0	100%	16	11	5	3	4	80%	0%	2,07
7-Q	7-Q-2,0	80%	11	8	3	2	4	68%	0%	0,19
7-Q	7-Q-2,2	90%	11	8	3	2,2	4	75%	0%	0,27
7-O	7-O-2,4	100%	11	8	3	2,4	4	82%	0%	6,76
7-Q	7-Q-2,6	100%	11	8	3	2,6	4	88%	0%	36,15
7-Q	7-Q-2,8	100%	11	8	3	2,8	4	95%	0%	32,42
7-Q	7-Q-3,0	100%	11	8	3	3	4	102%	inviável	inviável
8-Q	8-Q-2,0	40%	12	8	4	2	4	59%	0%	0,23
8-Q	8-Q-2,2	60%	12	8	4	2,2	4	65%	0%	0,64
8-Q	8-Q-2,4	70%	12	8	4	2,4	4	71%	0%	0,55
8-O	8-O-2,6	80%	12	8	4	2,6	4	77%	0%	3,39
8-Q	8-Q-2,8	90%	12	8	4	2,8	4	83%	0%	11,55
8-Q	8-Q-3,0	90%	12	8	4	3	4	88%	0%	13,40
9-Q	9-Q-2,0	50%	13	9	4	2	4	49%	0%	0,10
9-Q	9-Q-2,2	60%	13	9	4	2,2	4	53%	0%	0,10
9-Q	9-Q-2,4	70%	13	9	4	2,4	4	58%	0%	0,42

ID – análise	Nome no gráfico	% <i>sincr.</i>	Frota total <i>m</i>	Frota por classe		$\overline{Q}_i$	max Q_i	Ocupação	Resultados Gurobi (1h)	
				m_1	m_2				GAP	Tempo [s]
9-Q	9-Q-2,6	80%	13	9	4	2,6	4	63%	0%	1,98
9-O	9-O-2,8	90%	13	9	4	2,8	4	68%	0%	1,58
9-Q	9-Q-3,0	90%	13	9	4	3	4	73%	0%	1,53
10-Q	10-Q-2,0	70%	12	8	4	2	4	74%	0%	0,16
10-Q	10-Q-2,2	70%	12	8	4	2,2	4	81%	0%	0,68
10-O	10-O-2,4	80%	12	8	4	2,4	4	89%	0%	3,10
10-Q	10-Q-2,6	100%	12	8	4	2,6	4	96%	0%	1,49
10-Q	10-Q-2,8	100%	12	8	4	2,8	4	104%	0%	27,69
10-Q	10-Q-3,0	100%	12	8	4	3	4	111%	0%	23,43
11-Q	11-Q-2,0	40%	14	10	4	2	4	52%	0%	0,12
11-Q	11-Q-2,2	50%	14	10	4	2,2	4	57%	0%	0,13
11-Q	11-Q-2,4	60%	14	10	4	2,4	4	62%	0%	1,28
11-O	11-O-2,6	70%	14	10	4	2,6	4	68%	0%	0,95
11-Q	11-Q-2,8	80%	14	10	4	2,8	4	73%	0%	1,37
11-Q	11-Q-3,0	100%	14	10	4	3	4	78%	0%	14,11
12-Q	12-Q-2,0	60%	15	10	5	2	4	59%	0%	0,29
12-Q	12-Q-2,2	80%	15	10	5	2,2	4	65%	0%	0,33
12-Q	12-Q-2,4	90%	15	10	5	2,4	4	71%	0%	0,51
12-Q	12-Q-2,6	90%	15	10	5	2,6	4	77%	0%	0,48
12-O	12-O-2,8	100%	15	10	5	2,8	4	83%	0%	7,46
12-Q	12-Q-3,0	100%	15	10	5	3	4	89%	0%	11,81

Nestas instâncias, a quantidade máxima de rebocadores por tarefa foi mantida, mas houve alteração no percentual de tarefas sincronizadas. Três das instâncias geradas são inviáveis e estão sinalizadas na tabela.

Novamente, constata-se que o parâmetro variado afeta a velocidade com que o Gurobi conseguiu encontrar a solução ótima, uma vez que, na maioria dos casos, o aumento da demanda por tarefa resultou em aumento deste tempo.

6.2 TESTES COM PACOTE OTIMIZADOR DO MERCADO

6.2.1 Avaliação dos modelos matemáticos propostos

A fim de avaliar as modelagens matemáticas propostas para o problema, o Gurobi foi executado em todas as instâncias pequenas e médias (de 10 e 20 tarefas) com os três modelos descritos abaixo, usando a mesma notação do capítulo em que foram apresentados (capítulo 3):

M1: Tradicional completa: modelagem com variável de fluxo binária x_{ij}^k (indexada por rebocador), definida no grafo $G = (V, A)$ em que os arcos contemplam todos os pares de tarefas e arcos saindo do nó de origem e/ou chegando ao nó final ($A = \cup_{i \in J \cup N_{o+}} \cup_{j \in J \cup N_{o-}} (i, j)$);

M2: Tradicional sem arcos supérfluos: modelagem igual à anterior, ou seja, com variável de fluxo binária e indexada por rebocador, mas definida no grafo $G' = (V, A')$, suprimindo arcos ligando tarefas incompatíveis (ou seja, que representem $A' = \bigcup_{i \in J \cup N_o+} \bigcup_{j \in J \cup N_o-: b_j \leq b_i + p_i + \max T_{ij}^k(i, j)}$).

M3: Compacta: modelagem usando m nós de origem e indexação da variável de fluxo inteira e por classe de rebocador, x_{ij}^s , suprimindo conexões inviáveis entre pares de tarefas (ou seja, definida no grafo $G'' = (V, A'')$: $V = J \cup \mathcal{O} \cup \{m + n + 1\}$, $A'' = \bigcup_{i \in J \cup \mathcal{O}} \bigcup_{j \in J \cup \{n+m+1\}: b_j \leq b_i + p_i + \max T_{ij}^s(i, j)}$).

Complementando a descrição dos modelos matemáticos testados, a Tabela 12 apresenta o número total de variáveis, a quantidade de variáveis inteiras e o número de restrições de cada um. Na tabela, M1 refere-se à modelagem tradicional completa, M2 ao modelo sem arcos supérfluos e M3 à formulação compacta.

Tabela 12 – Características dos 3 modelos matemáticos testados, para comparação: quantidade de variáveis (total e inteiras) e de restrições (todas as restrições são lineares).

ID da Instância	Quantidade total de variáveis			Quantidade de variáveis inteiras (inclusive binárias)			Quantidade de restrições		
	M1	M2	M3	M1	M2	M3	M1	M2	M3
1	1.606	874	505	1.452	720	484	1.524	912	720
2	1.606	910	517	1.452	756	496	1.524	948	738
3	1.738	958	533	1.573	793	512	1.646	996	758
4	1.738	893	513	1.573	728	492	1.646	931	728
5	1.606	778	473	1.452	624	452	1.524	816	672
6	2.134	1.062	577	1.936	864	556	2.012	1.100	812
7	1.474	1.023	561	1.331	880	540	1.402	1.061	808
8	1.606	1.126	589	1.452	972	568	1.524	1.164	846
9	1.738	1.023	553	1.573	858	532	1.646	1.061	788
10	1.606	946	529	1.452	792	508	1.524	984	756
11	1.870	1.072	569	1.694	896	548	1.768	1.110	808
12	2.002	1.282	629	1.815	1.095	608	1.890	1.320	894
13	8.820	4.583	1.665	8.379	4.142	1.624	8.518	4.661	2.442
14	9.282	4.862	1.717	8.820	4.400	1.676	8.960	4.940	2.516
15	10.668	5.447	1.825	10.143	4.922	1.784	10.286	5.525	2.666
16	12.516	4.767	1.761	11.907	4.158	1.720	12.054	4.845	2.554
17	11.592	4.892	1.749	11.025	4.325	1.708	11.170	4.970	2.544
18	11.592	5.017	1.769	11.025	4.450	1.728	11.170	5.095	2.574
19	7.896	4.972	1.781	7.497	4.573	1.740	7.634	5.050	2.624
20	7.434	4.874	1.785	7.056	4.496	1.744	7.192	4.952	2.634
21	11.130	8.298	2.305	10.584	7.752	2.264	10.728	8.376	3.382
22	8.820	5.343	1.825	8.379	4.902	1.784	8.518	5.421	2.682
23	9.744	4.914	1.725	9.261	4.431	1.684	9.402	4.992	2.524
24	10.668	5.976	1.917	10.143	5.451	1.876	10.286	6.054	2.804

Como esperado, nota-se uma diminuição na quantidade de variáveis e de restrições conforme arcos foram suprimidos (de M1 para M2) e com a mudança na estratégia de modelagem (de M2 para M3), sem indexação por rebocador – mesmo com o surgimento de uma nova variável inteira. É interessante observar que, apesar da quantidade total de variáveis inteira ser menor nos modelos M3, o percentual em relação ao total de variáveis é maior (em média, 97% das variáveis de M3 são inteiras, contra 93% de M1 e 87% para modelos tipo M2). Outra conclusão que se pode tirar da tabela é que na modelagem compacta, o número de variáveis e restrições cresceu menos com o aumento das instâncias do que ao adotar a abordagem tradicional: ao dobrar o número de tarefas (instâncias 13 a 24 em relação às instâncias 1 a 12), a quantidade de variáveis e restrições aumentou mais de 4 vezes para M1 e M2, enquanto o crescimento do modelo M3 foi abaixo de 3.

Tabela 13 – Resultados da avaliação dos modelos matemáticos: tempo de execução e GAP final obtidos pelo Gurobi para cada modelagem e instância testadas.

ID da Instância	M1: Modelagem tradicional completa		M2: Modelagem tradicional sem arcos supérfluos		M3: Modelagem compacta	
	Tempo total (s)	GAP final (%)	Tempo total (s)	GAP final (%)	Tempo total (s)	GAP final (%)
1	3,44	0%	3,75	0%	0,74	0%
2	0,98	0%	1,12	0%	1,05	0%
3	1,42	0%	1,32	0%	0,34	0%
4	0,55	0%	0,55	0%	0,22	0%
5	0,89	0%	0,87	0%	0,38	0%
6	1,12	0%	1,07	0%	0,44	0%
7	3,30	0%	3,13	0%	20,72	0%
8	19,38	0%	18,28	0%	1,54	0%
9	4,39	0%	3,72	0%	0,55	0%
10	7,26	0%	7,39	0%	1,57	0%
11	0,79	0%	0,77	0%	0,34	0%
12	9,24	0%	5,81	0%	1,15	0%
13	—	—	—	—	3600,05	29%
14	3600,02	96%	3600,02	96%	3600,03	49%
15	3600,04	94%	3600,04	95%	3600,03	83%
16	46,16	0%	49,76	0%	8,98	0%
17	3600,04	91%	3600,02	91%	2923,57	0%
18	3600,53	95%	3600,03	87%	602,41	0%
19	—	—	—	—	3600,03	71%
20	—	—	—	—	—	—
21	3600,03	84%	3600,04	84%	3600,04	17%
22	—	—	—	—	3600,04	86%
23	—	—	—	—	3600,04	21%
24	—	—	—	—	3600,03	58%

Os resultados obtidos ao rodar o Gurobi usando as 3 modelagens são apresentados na Tabela 13, na qual os identificadores (IDs) das instâncias cujo ótimo foi encontrado

aparecem em negrito, assim como os resultados do modelo que melhor se saiu em cada instância. Casos em que o Gurobi não conseguiu encontrar nenhuma solução viável aparecem como “—” na tabela).

Tais resultados mostram que a modelagem compacta obteve desempenho superior às demais, pois além de ter resolvido mais instâncias, o tempo de execução do modelo compacto (ou o GAP final, para instâncias cuja solução ótima não foi obtida) foi menor do que os resultados obtidos pelas duas versões do modelo tradicional. Enquanto todas as instâncias de 10 tarefas foram resolvidas pelos três modelos, apenas três das instâncias de 20 tarefas tiveram solução ótima encontrada pelo otimizador, sendo duas destas apenas pelo modelo compacto. Nota-se ainda que não foi possível obter solução viável apenas para a instância 20.

Os testes documentados nesta subseção foram realizados via Gurobi Interactive Shell para Python, usando o ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) Spyder versão 3.6.6, em notebook iOS com processador de dois núcleos Intel® Core™ i5 @2.4-2.9GHz e 8GB @1600MHz DDR3L de memória RAM. Como dito anteriormente, estes testes foram executados com limitação no tempo de execução do Gurobi igual a 1 hora (3.600 segundos).

6.2.2 Obtenção de limitantes inferiores e superiores para as instâncias, usando o otimizador Gurobi

Com o objetivo de gerar parâmetros de referência que deem suporte à avaliação dos métodos de solução propostos para o problema, os seguintes testes foram realizados:

- Para avaliação do desempenho da heurística construtiva, o Gurobi foi executado em modo específico para encontrar solução viável rapidamente (parâmetro MIPFocus =1, conforme manual do otimizador) e com dois critérios de parada: até duas horas de execução ou até encontrar uma solução para o problema, o que ocorrer primeiro.
- Obtenção de UB e LB das instâncias, com execução do pacote otimizador em modo que balanceia os esforços para melhorar os limitantes, encontrar soluções viáveis e provar otimalidade (MIPFocus = 0). O único critério de parada adotado nestes testes foi o tempo de execução, também de até duas horas.

Ressalta-se que ambos os testes são terminados prematuramente caso o otimizador encontre solução ótima. Para instâncias em que não foi possível obter a solução ótima, o *Lower Bound* e o *Upper Bound* adotados foram os obtidos pelo Gurobi no tempo de execução pré-determinado, sendo o *Upper Bound* igual ao resultado da melhor solução

gerada pelo otimizador. O modelo matemático utilizado foi o compacto, que obteve desempenho superior aos demais (vide seção 6.2.1 deste capítulo).

A Tabela 14 resume os resultados obtidos nos dois testes. Para cada instância listada na primeira coluna, o tempo necessário para obtenção de uma solução viável é exibido na terceira coluna. Nas quarta e quinta colunas, são apresentados os melhores UB e LB obtidos nos experimentos realizados com o Gurobi. Na sexta coluna exibe-se o tempo total que o Gurobi precisou para obtenção de tais limites. Nos casos em que os valores de UB e LB são idênticos, ou seja, em que o otimizador encontrou solução ótima, o tempo apresentado é o tempo para chegar em uma solução ótima.

Tabela 14 – Resumo dos resultados obtidos com o Gurobi, mostrando o tempo para obtenção de uma solução viável (terceira coluna) e da melhor solução (última coluna), limitado a duas horas.

ID da Instância	Número de tarefas (n)	Tempo para obter solução viável [s]	Upper Bound	Lower Bound	Tempo para obter melhor solução [s]
1	10	0,06	\$ 247.755	\$ 247.755	1,2
2	10	0,07	\$ 294.849	\$ 294.849	0,4
3	10	0,04	\$ 121.142	\$ 121.142	0,6
4	10	0,05	\$ 283.856	\$ 283.856	0,5
5	10	0,03	\$ 1.102.162	\$ 1.102.162	0,5
6	10	0,05	\$ 164.211	\$ 164.211	0,5
7	10	0,12	\$ 395.429	\$ 395.429	1,3
8	10	0,12	\$ 612.340	\$ 612.340	1,2
9	10	0,08	\$ 243.396	\$ 243.396	0,9
10	10	0,11	\$ 195.177	\$ 195.177	2,1
11	10	0,20	\$ 990.055	\$ 990.055	0,3
12	10	0,12	\$ 290.179	\$ 290.179	1,1
13	20	105,82	\$ 3.053.070	\$ 1.967.902	7.200,0
14	20	8,32	\$ 1.387.019	\$ 757.250	7.200,0
15	20	1,60	\$ 1.524.964	\$ 273.671	7.200,1
16	20	0,26	\$ 1.776.306	\$ 1.776.306	7,7
17	20	0,41	\$ 3.452.716	\$ 3.452.695	3.863,6
18	20	1,59	\$ 3.095.897	\$ 3.095.897	1.437,4
19	20	2.900,88	\$ 3.753.510	\$ 942.286	7.200,1
20	20	Não obteve	--	---	--
21	20	2,33	\$ 225.346	\$ 225.346	5.112,4
22	20	606,55	\$ 4.610.724	\$ 791.720	7.200,5
23	20	496,78	\$ 4.179.453	\$ 3.504.885	1.643,1
24	20	29,34	\$ 2.130.427	\$ 803.239	7.200,1
25	30	23,04	\$ 1.437.540	\$ 838.097	7.200,2
26	30	22,19	\$ 3.762.463	\$ 3.514.718	7.200,2
27	30	3,43	\$ 135.879	\$ 124.905	7.200,1
28	30	2,53	\$ 1.205.845	\$ 929.018	7.200,2
29	30	3,50	\$ 265.499	\$ 204.458	7.200,2
30	30	3,88	\$ 502.044	\$ 275.886	7.200,3
31	30	3,80	\$ 63.141	\$ 63.141	1.525,1
32	30	33,18	\$ 678.747	\$ 136.190	7.200,3
33	30	3,31	\$ 337.694	\$ 81.151	7.200,2
34	30	16,10	\$ 87.058	\$ 87.058	591,0
35	30	26,79	\$ 1.485.108	\$ 221.609	7.200,2
36	30	3,37	\$ 609.180	\$ 89.376	7.200,1

Observando os resultados da Tabela 14, nota-se que para todas as instâncias menores (1 a 12), de 10 tarefas, foi possível encontrar solução ótima rapidamente – em média, o otimizador levou 0,9 segundo para resolver cada problema. Por sua vez, os resultados do primeiro teste mostram que foi possível obter uma solução viável em um décimo deste tempo.

Contudo, o desempenho do otimizador cai sensivelmente nas doze instâncias de 20 tarefas (IDs 13 a 24). Para estas, não foi possível obter solução ótima em sete casos, sendo que em um deles (instância 20) o Gurobi sequer chegou a uma solução viável, mesmo após duas horas de execução. Nas cinco instâncias em que foi possível obter solução ótima, o otimizador levou em média 2.400 segundos, ou 40 minutos. O tempo médio para obtenção de solução viável (para as 12 instâncias em que isso foi possível) foi aproximadamente 377,6 segundos, pouco mais de seis minutos. Cabe destacar que, apesar do número de tarefas ter apenas dobrado, o tempo médio para obtenção de uma solução viável subiu de 0,09 para 377,6, um aumento de mais de 4.000 vezes.

Para as instâncias 25 a 36, de 30 tarefas e ocupação mais baixa, foi possível obter solução viável em todos os casos, dentro de 12 segundos, em média. Contudo, apenas duas instâncias tiveram solução ótima encontrada, em 591 segundos (instância 34) e 1.525 segundos (31), o que equivale a uma média de 1.058 segundos (ou 17,6 minutos). Em comparação com as instâncias de 20 tarefas, as de 30 tiveram menos soluções ótimas encontradas, contudo o tempo de execução para achar solução viável foi mais baixo do que para as instâncias de 20 tarefas. Possivelmente, isso se deve à menor ocupação das instâncias maiores.

Os testes documentados nesta seção foram realizados via Gurobi Interactive Shell para Python, usando o ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) Spyder versão 3.6.6, em notebook com sistema operacional Windows 10 de 64 bits e processador de dois núcleos Intel® Core™ i7-6500U CPU @2.50-2.59GHz e 8GB de memória RAM. O otimizador rodou por até 2 horas (7.200 segundos).

6.2.3 Parâmetros para avaliação dos métodos de solução propostos

Para avaliar a qualidade das soluções não-ótimas obtidas pela heurística ou pelo próprio Gurobi, dois critérios foram utilizados: o tempo de execução e o valor da função objetivo, ou seja, o custo total da solução. Enquanto o tempo é comparado diretamente,

a qualidade do resultado de uma solução S é avaliada comparando o seu custo total, $z(S)$, ao LB do Gurobi, usando a seguinte definição de $desvio_{LB}(S)$:

$$desvio_{LB}(s) = \frac{z(s) - LB}{LB} \quad \text{Eq. 6-2}$$

Além do desvio com relação ao LB, o desvio com relação ao UB também é útil para avaliar a qualidade de soluções. O próprio Gurobi o adota em suas saídas, denominando-o de GAP, conforme definição abaixo:

$$GAP(s = UB) = \frac{UB - LB}{UB} \quad \text{Eq. 6-3}$$

Tabela 15 – Melhores Upper e Lower Bounds obtidos pelo Gurobi para cada instância, destacando em negrito os IDs das instâncias cuja solução ótima foi encontrada.

ID da Instância	n tarefas	Limitante superior (UB) [\$]	Limitante inferior (LB) [\$]	$desvio_{LB}(UB)$	$GAP(UB)$	Tempo de execução [s]
1	10	\$ 247.755	\$ 247.755	0%	0%	1,2
2	10	\$ 294.849	\$ 294.849	0%	0%	0,4
3	10	\$ 121.142	\$ 121.142	0%	0%	0,6
4	10	\$ 283.856	\$ 283.856	0%	0%	0,5
5	10	\$ 1.102.162	\$ 1.102.162	0%	0%	0,5
6	10	\$ 164.211	\$ 164.211	0%	0%	0,5
7	10	\$ 395.429	\$ 395.429	0%	0%	1,3
8	10	\$ 612.340	\$ 612.340	0%	0%	1,2
9	10	\$ 243.396	\$ 243.396	0%	0%	0,9
10	10	\$ 195.177	\$ 195.177	0%	0%	2,1
11	10	\$ 990.055	\$ 990.055	0%	0%	0,3
12	10	\$ 290.179	\$ 290.179	0%	0%	1,1
13	20	\$ 3.053.070	\$ 1.967.902	55%	36%	7.200,0
14	20	\$ 1.387.019	\$ 757.250	83%	45%	7.200,0
15	20	\$ 1.524.964	\$ 273.671	457%	82%	7.200,1
16	20	\$ 1.776.306	\$ 1.776.306	0%	0%	7,7
17	20	\$ 3.452.716	\$ 3.452.695	0%	0%	3.863,6
18	20	\$ 3.095.897	\$ 3.095.897	0%	0%	1.437,4
19	20	\$ 3.753.510	\$ 942.286	298%	75%	7.200,1
20	20	NA	\$ 151.633	NA	NA	7.200,0
21	20	\$ 225.346	\$ 225.346	0%	0%	5.112,4
22	20	\$ 4.610.724	\$ 791.720	482%	83%	7.200,5
23	20	\$ 4.179.453	\$ 3.504.885	19%	16%	7.200,0
24	20	\$ 2.130.427	\$ 803.239	165%	62%	7.200,1
25	30	\$ 1.437.540	\$ 838.097	72%	42%	7.200,2
26	30	\$ 3.762.463	\$ 3.514.718	7%	7%	7.200,2
27	30	\$ 135.879	\$ 124.905	9%	8%	7.200,1
28	30	\$ 1.205.845	\$ 929.018	30%	23%	7.200,2
29	30	\$ 265.499	\$ 204.458	30%	23%	7.200,2
30	30	\$ 502.044	\$ 275.886	82%	45%	7.200,3
31	30	\$ 63.141	\$ 63.141	0%	0%	1.525,1
32	30	\$ 678.747	\$ 136.190	398%	80%	7.200,3
33	30	\$ 337.694	\$ 81.151	316%	76%	7.200,2
34	30	\$ 87.058	\$ 87.058	0%	0%	591,0
35	30	\$ 1.485.108	\$ 221.609	570%	85%	7.200,2
36	30	\$ 609.180	\$ 89.376	582%	85%	7.200,1

Na Tabela 15 são apresentados os melhores LB e UB obtidos pelo Gurobi para o modelo matemático compacto, assim com o GAP e $desvio_{LB}(UB)$. O desvio (em relação ao LB) da melhor solução encontrada pelo Gurobi, apresentado na quarta coluna, tem valor médio de 2.446% nas instâncias de 20 tarefas. Isto pode dever-se à complexidade do problema ou à baixa qualidade do LB utilizado.

Os testes documentados nesta seção foram realizados via Gurobi Interactive Shell para Python, usando o ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) Spyder versão 3.6.6, em notebook Windows com processador de dois núcleos Intel® Core™ i7-6500 CPU @2.50-2.59GHz e 8GB de memória RAM. Como dito anteriormente, estes testes foram executados com limitação no tempo de execução do Gurobi igual a 2 horas.

6.3 RESULTADOS DOS MÉTODOS DE SOLUÇÃO PROPOSTOS

A seguir são apresentados os resultados dos métodos de solução propostos, tanto para geração de solução inicial (heurística construtiva executada até que se obtenha uma solução viável) como para melhoria desta solução. Todos os métodos foram implementados em programa desenvolvido pela autora em linguagem C++11 (ou 2011 *Standard*), no IDE Microsoft Visual Studio Enterprise© 2017 versão 15.9.18. O notebook em que os testes foram executados possui sistema operacional Windows 10, com processador de dois núcleos Intel® Core™ i7-6500 CPU @2.50-2.59GHz e 8GB de memória RAM.

6.3.1 Obtenção de solução viável usando heurística construtiva

A Tabela 16 mostra os valores adotados para os parâmetros da HC, para obtenção dos resultados apresentados nesta tese. Para chegar nestes valores, foram realizados diversos experimentos computacionais nas instâncias de 10 a 30 tarefas.

Tabela 16 – Parâmetros da HCMR, adotado após testes de calibração.

Parâmetro	Valor após calibração
<i>tamListaRestritaJ</i>	3
<i>tamListaRestritaM</i>	16

Na Tabela 17, a solução obtida pela heurística construtiva é comparada com a primeira solução obtida pelo Gurobi, em desvio em relação ao LB e em tempo computacional exigido.

Tabela 17 – Resultados da heurística construtiva e comparação com desempenho do Gurobi, no experimento usado para encontrar apenas uma solução factível.

ID da Instância	n tarefas	<i>desvio_{LB}</i> (s)			Tempo de execução [s]		
		Gurobi	HC médio	HC melhor	Gurobi	HC médio	HC melhor
1	10	456%	256%	256%	0,06	0,02	0,02
2	10	174%	138%	138%	0,07	0,05	0,04
3	10	2.218%	92%	92%	0,04	0,02	0,02
4	10	330%	285%	285%	0,05	0,01	0,01
5	10	182%	121%	80%	0,03	0,44	0,18
6	10	933%	1185%	1185%	0,05	0,03	0,03
7	10	1.032%	778%	778%	0,12	0,01	0,01
8	10	570%	254%	254%	0,12	0,02	0,02
9	10	1.127%	324%	324%	0,08	0,03	0,03
10	10	973%	1202%	1202%	0,11	0,01	0,01
11	10	97%	8%	8%	0,20	0,10	0,09
12	10	857%	910%	910%	0,12	0,04	0,04
13	20	151%	195%	162%	105,82	42,80	27,73
14	20	536%	571%	571%	8,32	0,26	0,25
15	20	375%	2447%	2447%	1,60	0,68	0,64
16	20	95%	155%	155%	0,26	0,29	0,19
17	20	109%	104%	62%	0,41	2,33	1,64
18	20	70%	68%	57%	1,59	383,24	78,54
19	20	146%	625%	625%	2.900,88	0,26	0,23
20	20	NA	4988%	4583%	7.200,02	179,09	31,73
21	20	5.105%	1867%	1867%	2,33	0,88	0,86
22	20	78%	1052%	1052%	606,55	0,35	0,32
23	20	NA	104%	85%		8,25	0,95
24	20	173%	817%	817%	29,34	1,83	1,75
25	30	494%	181%	181%	23,04	2,79	2,69
26	30	48%	112%	81%	22,19	65,15	11,32
27	30	7.067%	794%	794%	3,43	9,49	9,40
28	30	934%	826%	710%	2,53	3,14	1,95
29	30	1.602%	1347%	1347%	3,50	1,10	1,03
30	30	1.777%	1009%	1009%	3,88	6,47	6,36
31	30	20.170%	21685%	21685%	3,80	0,14	0,13
32	30	1.567%	4343%	4343%	33,18	0,58	0,56
33	30	3.572%	6879%	6879%	3,31	1,04	1,03
34	30	9.076%	10211%	10211%	16,10	0,21	0,20
35	30	877%	4299%	4299%	26,79	2,79	2,74
36	30	2.767%	16434%	16434%	3,37	2,37	2,33

A HC foi executada 5 vezes em cada instância, para avaliação da consistência dos resultados, levando em conta a aleatoriedade do método guloso; e para permitir a comparação de desempenho com o otimizador. Na tabela, o método que obteve melhor desempenho para o resultado em questão (LB ou tempo de execução) está destacado

em negrito. As instâncias cujo LB correspondem ao ótimo estão com seus IDs em itálico, e os resultados medidos em relação ao ótimo aparecem em itálico e realçados em cinza mais escuro.

Nota-se que todas as instâncias são resolvidas com sucesso pela heurística construtiva. A qualidade das soluções, no entanto, não é boa para as instâncias cujo ótimo é conhecido (destacadas em itálico) – e tampouco parece ser boa para as demais, dado que o desvio em relação ao LB é alto. Por outro lado, a HC construiu soluções melhores que o Gurobi em 17 instâncias, além de ter resolvido duas instâncias que o otimizador não resolveu dentro de 2 horas. Em média, a HC gerou soluções com com acréscimo de 463% do LB para as instâncias de 10 tarefas; 1083% para 20 tarefas e 5.677% para as maiores; *versus* os aumentos de 746%, 684% e 4.163% resultantes das soluções do próprio Gurobi. Ou seja, para as instâncias médias e grandes, o otimizador se saiu melhor.

Não obstante, os resultados foram produzidos pela HC mais rapidamente que pelo otimizador, visto que em 26 das 36 instâncias a heurística foi mais rápida, na média das 5 execuções (se comparado ao melhor resultado dentre as 5 execuções, este número sobe para 31). Se para as instâncias menores a diferença de tempo de execução é pequena (média de 0,09 segundos para o Gurobi contruir solução viável, contra 0,06 segundos para a HC), para as de 20 tarefas o Gurobi demorou mais de 10 vezes o tempo gasto pela heurística gulosa (987 segundos contra 51).

6.3.2 Heurística construtiva com múltiplos recomeços

A Tabela 17 mostra os valores dos parâmetros da HCMR adotados para obtenção dos resultados apresentados nesta tese. Para chegar nestes valores, foram realizados diversos experimentos computacionais em uma amostra de instâncias de 10 e de 20 tarefas, que não estão documentados aqui mas podem ser consultados em trabalho apresentado anteriormente, durante a pesquisa (CAVALCANTI; MENDES, 2019).

Assim como nos testes para obtenção da primeira solução viável, a HCMR foi executada 5 vezes em cada instância e os principais resultados são apresentados na Tabela 19. Cabe dizer que o número relativamente baixo de execuções deveu-se ao alto tempo necessário para os testes, já que cada instância demora a ser processada e que vários métodos foram desenvolvidos.

Tabela 18 – Parâmetros da HCMR, adotado após testes de calibração.

Parâmetro	Valor após calibração
<i>tamListaRestritaJ</i>	3
<i>tamListaRestritaM</i>	16
<i>minIt</i>	$m \times n^2$
<i>tamPop</i>	$m \times n^2$

Tabela 19 – Resultados da Heurística Construtiva com Múltiplos Recomeços (HCMR) nas instâncias propostas para o PRVJTSin-Het.

ID	Melhor Solução (HCMR*)	Solução Média: $z(HCMR)$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} (HCMR*)	Desvio _{LB} (HCMR)	GAP (HCMR*)	GAP (HCMR)
1	\$ 263.921	\$ 295.224	10,81	10,20	7%	19%	6%	16%
2	\$ 342.128	\$ 355.385	13,58	12,71	16%	21%	14%	17%
3	\$ 173.978	\$ 173.978	23,59	21,98	44%	44%	30%	30%
4	\$ 290.769	\$ 297.767	24,85	23,11	2%	5%	2%	5%
5	\$ 1.301.602	\$ 1.577.197	0,78	0,60	18%	43%	15%	30%
6	\$ 164.536	\$ 165.005	76,87	75,08	0%	0%	0%	0%
7	\$ 708.630	\$ 1.163.447	14,74	14,35	79%	194%	44%	66%
8	\$ 613.318	\$ 613.318	36,91	35,78	0%	0%	0%	0%
9	\$ 315.080	\$ 315.080	40,98	39,86	29%	29%	23%	23%
10	\$ 208.519	\$ 208.519	33,82	30,69	7%	7%	6%	6%
11	\$ 990.904	\$ 991.303	26,07	25,57	0%	0%	0%	0%
12	\$ 729.436	\$ 729.436	104,64	98,37	151%	151%	60%	60%
13	\$ 5.420.726	\$ 6.093.703	0,09	0,06	175%	210%	64%	68%
14	\$ 2.933.964	\$ 2.933.964	53,30	46,63	287%	287%	74%	74%
15	\$ 3.391.505	\$ 3.876.245	26,30	22,68	1139%	1316%	92%	93%
16	\$ 1.933.621	\$ 1.936.467	4.591,97	4.333,06	9%	9%	8%	8%
17	\$ 5.160.087	\$ 5.362.548	29,33	25,40	49%	55%	33%	36%
18	\$ 4.221.805	\$ 4.911.880	1,15	0,53	36%	59%	27%	37%
19	\$ 4.641.593	\$ 5.266.380	22,82	21,31	393%	459%	80%	82%
20	\$ 6.982.422	\$ 8.402.559	0,11	0,06	5158%	5441%	98%	98%
21	\$ 869.511	\$ 869.511	9.571,79	8.758,78	286%	286%	74%	74%
22	\$ 7.492.736	\$ 8.222.571	0,37	0,22	846%	939%	89%	90%
23	\$ 5.737.230	\$ 6.356.878	0,93	0,54	64%	81%	39%	45%
24	\$ 3.102.971	\$ 3.102.971	59,56	56,17	286%	286%	74%	74%
25	\$ 2.358.457	\$ 2.358.457	0,16	0,03	181%	181%	64%	64%
26	\$ 7.682.956	\$ 7.682.956	0,34	0,07	119%	119%	54%	54%
27	\$ 1.116.654	\$ 1.116.654	48,22	9,64	794%	794%	89%	89%
28	\$ 6.394.471	\$ 6.394.471	13,18	2,64	588%	588%	85%	85%
29	\$ 1.963.081	\$ 1.963.081	47,29	9,46	860%	860%	90%	90%
30	\$ 2.616.013	\$ 2.616.013	92,88	18,58	848%	848%	89%	89%
31	\$ 534.923	\$ 534.923	295,25	59,05	747%	747%	88%	88%
32	\$ 2.601.539	\$ 2.601.539	112,47	22,49	1810%	1810%	95%	95%
33	\$ 1.337.668	\$ 1.337.668	587,65	117,53	1548%	1548%	94%	94%
34	\$ 1.896.650	\$ 1.896.650	542,06	108,41	2079%	2079%	95%	95%
35	\$ 3.143.194	\$ 3.143.194	473,05	94,61	1318%	1318%	93%	93%
36	\$ 2.270.048	\$ 2.270.048	1.320,60	264,12	2440%	2440%	96%	96%

Na Tabela 19, as instâncias cujas soluções ótimas são conhecidas estão sublinhadas e em itálico. As colunas 2, 6 e 8 dizem respeito à melhor solução encontrada pelo método, $HCMR^*$, enquanto as colunas 3, 7 e 9 referem-se à solução média das cinco execuções do método, $\overline{HCMR}$. O tempo da execução mais demorada da HCMR é apresentado na coluna 4 e a média do tempo total das execuções aparece logo ao lado, na quinta coluna.

6.3.3 Busca Local com múltiplos recomeços

Os resultados da BLMR são apresentados na Tabela 21, seguindo o formato adotado na seção 6.3.2: a melhor solução é representada por $BLMR^*$ e a média das soluções, por $\overline{BLMR}$. Os valores dos parâmetros adotados na calibração também são documentados aqui, na Tabela 20.

Tabela 20 – Parâmetros da BLMR, adotado após testes de calibração.

Parâmetro	Valor após calibração
<i>tamListaRestritaJ</i>	3
<i>tamListaRestritaM</i>	16
<i>maxItLocais</i>	100
<i>maxItGlobais</i>	10

Tabela 21 – Resultados da Busca Local com Múltiplos Recomeços (BLMR) nas instâncias propostas para o PRVJTSin-Het.

ID	Melhor Solução ($BLMR^*$)	Solução Média: $\overline{z(BLMR)}$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} ($BLMR^*$)	Desvio _{LB} ($\overline{BLMR}$)	GAP ($BLMR^*$)	GAP ($\overline{BLMR}$)
1	<u>\$ 259.467</u>	<u>\$ 287.936</u>	<u>43,49</u>	<u>31,65</u>	<u>5%</u>	<u>16%</u>	<u>5%</u>	<u>14%</u>
2	<u>\$ 338.794</u>	<u>\$ 358.876</u>	<u>27,85</u>	<u>23,98</u>	<u>15%</u>	<u>22%</u>	<u>13%</u>	<u>18%</u>
3	<u>\$ 203.623</u>	<u>\$ 206.644</u>	<u>140,71</u>	<u>140,38</u>	<u>68%</u>	<u>71%</u>	<u>41%</u>	<u>41%</u>
4	<u>\$ 284.643</u>	<u>\$ 294.706</u>	<u>98,42</u>	<u>55,37</u>	<u>0%</u>	<u>4%</u>	<u>0%</u>	<u>4%</u>
5	<u>\$ 2.513.737</u>	<u>\$ 2.913.944</u>	<u>0,50</u>	<u>0,25</u>	<u>128%</u>	<u>164%</u>	<u>56%</u>	<u>62%</u>
6	<u>\$ 164.394</u>	<u>\$ 164.449</u>	<u>229,01</u>	<u>156,87</u>	<u>0%</u>	<u>0%</u>	<u>0%</u>	<u>0%</u>
7	<u>\$ 681.684</u>	<u>\$ 954.158</u>	<u>28,59</u>	<u>23,54</u>	<u>72%</u>	<u>141%</u>	<u>42%</u>	<u>59%</u>
8	<u>\$ 612.343</u>	<u>\$ 664.106</u>	<u>63,25</u>	<u>55,37</u>	<u>0%</u>	<u>8%</u>	<u>0%</u>	<u>8%</u>
9	<u>\$ 314.824</u>	<u>\$ 315.048</u>	<u>75,72</u>	<u>72,57</u>	<u>29%</u>	<u>29%</u>	<u>23%</u>	<u>23%</u>
10	<u>\$ 233.741</u>	<u>\$ 261.088</u>	<u>57,35</u>	<u>56,83</u>	<u>20%</u>	<u>34%</u>	<u>16%</u>	<u>25%</u>
11	<u>\$ 990.612</u>	<u>\$ 1.029.778</u>	<u>163,02</u>	<u>147,88</u>	<u>0%</u>	<u>4%</u>	<u>0%</u>	<u>4%</u>
12	<u>\$ 292.652</u>	<u>\$ 594.579</u>	<u>78,95</u>	<u>53,17</u>	<u>1%</u>	<u>105%</u>	<u>1%</u>	<u>51%</u>
13	\$ 5.364.702	\$ 5.882.733	0,11	0,08	173%	199%	63%	67%
14	\$ 3.793.023	\$ 3.861.683	2.985,77	2.606,10	401%	410%	80%	80%
15	\$ 6.448.234	\$ 6.448.234	2.817,25	2.455,84	2256%	2256%	96%	96%
16	<u>\$ 5.615.208</u>	<u>\$ 5.916.294</u>	<u>7,95</u>	<u>4,00</u>	<u>216%</u>	<u>233%</u>	<u>68%</u>	<u>70%</u>
17	\$ 6.097.067	\$ 6.273.050	7.410,07	3.705,11	77%	82%	43%	45%
18	<u>\$ 4.263.742</u>	<u>\$ 4.684.358</u>	<u>6.753,79</u>	<u>3.376,97</u>	<u>38%</u>	<u>51%</u>	<u>27%</u>	<u>34%</u>
19	\$ 6.313.324	\$ 6.460.523	970,38	725,61	570%	586%	85%	85%

ID	Melhor Solução (BLMR*)	Solução Média: $z(BLMR)$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} (BLMR*)	Desvio _{LB} (BLMR)	GAP (BLMR*)	GAP (BLMR)
20	\$ 7.091.980	\$ 8.512.923	0,05	0,05	4577%	5514%	98%	98%
21	<u>\$ 1.032.123</u>	<u>\$ 1.798.768</u>	<u>4.348,14</u>	<u>3.761,96</u>	<u>358%</u>	<u>698%</u>	<u>78%</u>	<u>87%</u>
22	\$ 9.122.912	\$ 9.122.912	0,05	0,04	1052%	1052%	91%	91%
23	\$ 6.765.335	\$ 6.828.383	3.003,21	2.528,91	93%	95%	48%	49%
24	\$ 6.030.506	\$ 6.066.316	5.202,46	4.976,96	651%	655%	87%	87%
25	\$ 2.358.457	\$ 2.358.457	0,20	0,19	181%	181%	64%	64%
26	\$ 5.699.129	\$ 6.519.265	6.102,63	4.452,31	62%	85%	38%	46%
27	\$ 1.116.654	\$ 1.116.654	1,14	1,12	794%	794%	89%	89%
28	\$ 6.002.763	\$ 6.626.168	8.760,67	6.009,87	546%	613%	85%	86%
29	\$ 2.878.532	\$ 2.890.977	5.602,95	4.404,29	1308%	1314%	93%	93%
30	\$ 2.599.441	\$ 2.716.125	11.986,70	11.529,80	842%	885%	89%	90%
31	<u>\$ 7.887.783</u>	<u>\$ 8.127.924</u>	<u>1.763,85</u>	<u>891,60</u>	<u>12392%</u>	<u>12773%</u>	<u>99%</u>	<u>99%</u>
32	\$ 4.275.566	\$ 4.732.781	9.267,75	8.864,41	3039%	3375%	97%	97%
33	\$ 3.136.316	\$ 3.916.802	25.816,90	20.173,70	3765%	4727%	97%	98%
34	<u>\$ 2.523.557</u>	<u>\$ 3.346.133</u>	<u>8.519,07</u>	<u>8.236,29</u>	<u>2799%</u>	<u>3744%</u>	<u>97%</u>	<u>97%</u>
35	\$ 8.535.996	\$ 8.884.307	14.829,60	12.379,35	3752%	3909%	97%	98%
36	\$ 5.123.585	\$ 7.649.821	33.173,00	26.997,10	5633%	8459%	98%	99%

6.3.4 GRASP

Os valores dos parâmetros adotados na calibração estão documentados na Tabela 22. Em seguida, são apresentados os resultados das GRASP na Tabela 23, na qual a melhor solução é representada por $GRASP^*$ e a média das soluções, por $\overline{GRASP}$.

Tabela 22 – Parâmetros de calibração da GRASP.

Parâmetro	Valor após calibração
<i>tamListaRestritaJ</i>	3
<i>tamListaRestritaM</i>	16
<i>tamPop</i>	10
<i>recomecosSemMelhoria</i>	100
<i>maxItGlobais</i>	10

Tabela 23 – Resultados da heurística GRASP nas instâncias propostas para o PRVJTSin-Het.

ID	Melhor Solução (GRASP*)	Solução Média: $z(\overline{GRASP})$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} (GRASP*)	Desvio _{LB} ($\overline{GRASP}$)	GAP (GRASP*)	GAP ($\overline{GRASP}$)
1	<u>\$ 259.425</u>	<u>\$ 333.820</u>	<u>56,34</u>	<u>53,78</u>	<u>5%</u>	<u>35%</u>	<u>4%</u>	<u>26%</u>
2	<u>\$ 300.980</u>	<u>\$ 344.463</u>	<u>83,46</u>	<u>79,72</u>	<u>2%</u>	<u>17%</u>	<u>2%</u>	<u>14%</u>
3	<u>\$ 159.417</u>	<u>\$ 169.124</u>	<u>193,74</u>	<u>172,98</u>	<u>32%</u>	<u>40%</u>	<u>24%</u>	<u>28%</u>
4	<u>\$ 288.122</u>	<u>\$ 295.811</u>	<u>118,58</u>	<u>110,56</u>	<u>2%</u>	<u>4%</u>	<u>1%</u>	<u>4%</u>
5	<u>\$ 1.520.990</u>	<u>\$ 1.641.980</u>	<u>71,23</u>	<u>68,89</u>	<u>38%</u>	<u>49%</u>	<u>28%</u>	<u>33%</u>
6	<u>\$ 164.337</u>	<u>\$ 164.508</u>	<u>601,32</u>	<u>571,66</u>	<u>0%</u>	<u>0%</u>	<u>0%</u>	<u>0%</u>
7	<u>\$ 695.568</u>	<u>\$ 806.314</u>	<u>36,41</u>	<u>31,99</u>	<u>76%</u>	<u>104%</u>	<u>43%</u>	<u>51%</u>

ID	Melhor Solução (GRASP*)	Solução Média: $z(GRASP)$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} (GRASP*)	Desvio _{LB} (GRASP)	GAP (GRASP*)	GAP (GRASP)
8	\$ 612.823	\$613.153	108,20	102,91	0%	0%	0%	0%
9	\$ 312.217	\$314.072	184,38	165,58	28%	29%	22%	23%
10	\$ 208.519	\$208.519	94,24	91,09	7%	7%	6%	6%
11	\$ 990.622	\$990.884	180,30	175,47	0%	0%	0%	0%
12	\$ 729.436	\$729.436	322,62	305,90	151%	151%	60%	60%
13	\$ 5.472.643	\$ 5.945.015	823,48	617,73	178%	202%	64%	67%
14	\$ 2.933.964	\$ 2.933.964	3.611,86	3.231,24	287%	287%	74%	74%
15	\$ 3.984.331	\$ 4.068.773	6.032,25	5.134,52	1356%	1387%	93%	93%
16	\$ 1.937.178	\$ 1.937.178	9.631,62	9.130,43	9%	9%	8%	8%
17	\$ 5.316.427	\$ 5.482.863	7.686,67	7.498,59	54%	59%	35%	37%
18	\$ 5.208.935	\$ 5.539.431	1.472,56	1.470,39	68%	79%	41%	44%
19	\$ 5.877.042	\$ 5.877.042	1.464,91	1.452,54	524%	524%	84%	84%
20	\$ 8.244.251	\$ 8.862.405	543,77	447,72	5337%	5745%	98%	98%
21	\$ 1.205.356	\$ 1.504.908	5.249,58	4.971,92	435%	568%	81%	85%
22	\$ 9.065.312	\$ 9.065.312	253,86	253,86	1045%	1045%	91%	91%
23	\$ 6.515.161	\$ 6.515.161	397,31	397,31	86%	86%	46%	46%
24	\$ 3.102.971	\$ 3.102.971	8.280,03	7.786,98	286%	286%	74%	74%
25	\$ 2.351.430	\$ 2.354.944	7.664,59	5.076,88	181%	181%	64%	64%
26	\$ 5.697.681	\$ 6.964.865	3.518,77	2.692,93	62%	98%	38%	50%
27	\$ 1.043.372	\$ 1.080.013	12.351,70	10.463,68	735%	765%	88%	88%
28	\$ 5.628.868	\$ 6.886.964	5.214,24	4.473,34	506%	641%	83%	87%
29	\$ 1.937.594	\$ 1.950.127	9.154,44	8.355,28	848%	854%	89%	90%
30	\$ 2.234.775	\$ 2.425.394	14.268,00	12.384,60	710%	779%	88%	89%
31	\$ 462.412	\$ 1.917.171	3.849,44	3.167,79	632%	2936%	86%	97%
32	\$ 2.601.234	\$ 2.601.387	3.628,41	3.333,09	1810%	1810%	95%	95%
33	\$ 1.334.244	\$ 2.570.539	12.341,30	10.484,02	1544%	3068%	94%	97%
34	\$ 1.553.721	\$ 2.640.469	4.267,83	3.909,40	1685%	2933%	94%	97%
35	\$ 3.143.194	\$ 3.143.194	8.118,40	6.915,48	1318%	1318%	93%	93%
36	\$ 2.270.048	\$ 5.767.689	18.900,30	15.203,45	2440%	6353%	96%	98%

6.3.5 RVTMR

A tabela a seguir mostra os valores dos parâmetros adotados na calibração da RVTMR, cujos resultados aparecem na Tabela 24. Os resultados das 5 execuções da RVTMR são apresentados na Tabela 25.

Tabela 24 – Parâmetros da RVTMR, adotado após testes de calibração.

Parâmetro	Valor após calibração
<i>tamListaRestritaJ</i>	3
<i>tamListaRestritaM</i>	16
<i>recomecosSemMelhoria</i>	500
<i>maxItGlobais</i>	4
<i>maxString</i>	40% <i>n</i>
<i>numConfigPorModo</i>	max{50%, 1}

Tabela 25 – Resultados da heurística de Remoção de Vetor de Tarefas com Múltiplos Recomeços (RVTMR) nas instâncias propostas para o PRVJTSin-Het.

ID	Melhor Solução (RVTMR*)	Solução Média: $z(RVTMR)$	Tempo máximo [s]	Tempo médio [s]	Desvio _{LB} (RVTMR*)	Desvio _{LB} (RVTMR)	GAP (RVTMR*)	GAP (RVTMR)
1	<u>\$254.690</u>	<u>\$ 431.849</u>	<u>159,31</u>	<u>147,69</u>	<u>3%</u>	<u>74%</u>	<u>3%</u>	<u>43%</u>
2	<u>\$298.240</u>	<u>\$ 462.254</u>	<u>63,03</u>	<u>49,98</u>	<u>1%</u>	<u>57%</u>	<u>1%</u>	<u>36%</u>
3	<u>\$187.522</u>	<u>\$ 214.299</u>	<u>367,63</u>	<u>183,82</u>	<u>55%</u>	<u>77%</u>	<u>35%</u>	<u>43%</u>
4	<u>\$306.176</u>	<u>\$ 533.868</u>	<u>179,86</u>	<u>123,65</u>	<u>8%</u>	<u>88%</u>	<u>7%</u>	<u>47%</u>
5	<u>\$ 1.750.100</u>	<u>\$ 2.103.395</u>	<u>9,16</u>	<u>5,60</u>	<u>59%</u>	<u>91%</u>	<u>37%</u>	<u>48%</u>
6	<u>\$164.317</u>	<u>\$ 553.196</u>	<u>709,42</u>	<u>523,31</u>	<u>0%</u>	<u>237%</u>	<u>0%</u>	<u>70%</u>
7	<u>\$490.146</u>	<u>\$ 1.521.067</u>	<u>60,14</u>	<u>33,43</u>	<u>24%</u>	<u>285%</u>	<u>19%</u>	<u>74%</u>
8	<u>\$701.758</u>	<u>\$ 1.298.284</u>	<u>102,40</u>	<u>83,39</u>	<u>15%</u>	<u>112%</u>	<u>13%</u>	<u>53%</u>
9	<u>\$312.217</u>	<u>\$ 477.744</u>	<u>196,68</u>	<u>184,53</u>	<u>28%</u>	<u>96%</u>	<u>22%</u>	<u>49%</u>
10	<u>\$207.416</u>	<u>\$ 642.975</u>	<u>300,08</u>	<u>265,10</u>	<u>6%</u>	<u>229%</u>	<u>6%</u>	<u>70%</u>
11	<u>\$990.336</u>	<u>\$ 1.018.866</u>	<u>774,73</u>	<u>393,18</u>	<u>0%</u>	<u>3%</u>	<u>0%</u>	<u>3%</u>
12	<u>\$884.608</u>	<u>\$ 1.333.246</u>	<u>295,93</u>	<u>156,70</u>	<u>205%</u>	<u>359%</u>	<u>67%</u>	<u>78%</u>
13	\$ 6.965.648	\$ 6.965.648	0,08	0,08	254%	254%	72%	72%
14	\$ 5.081.716	\$ 5.081.716	0,06	0,06	571%	571%	85%	85%
15	\$ 6.971.273	\$ 6.971.273	0,07	0,07	2447%	2447%	96%	96%
16	<u>\$ 4.387.562</u>	<u>\$ 4.436.884</u>	<u>16,64</u>	<u>16,64</u>	<u>147%</u>	<u>150%</u>	<u>60%</u>	<u>60%</u>
17	\$ 7.152.033	\$ 7.152.033	0,12	0,12	107%	107%	52%	52%
18	<u>\$ 5.213.113</u>	<u>\$ 5.213.113</u>	<u>0,11</u>	<u>0,11</u>	<u>68%</u>	<u>68%</u>	<u>41%</u>	<u>41%</u>
19	\$ 6.828.066	\$ 6.828.066	0,04	0,04	625%	625%	86%	86%
20	\$ 7.040.214	\$ 7.040.214	0,05	0,05	4543%	4543%	98%	98%
21	<u>\$ 4.432.316</u>	<u>\$ 4.432.316</u>	<u>0,87</u>	<u>0,87</u>	<u>1867%</u>	<u>1867%</u>	<u>95%</u>	<u>95%</u>
22	\$ 9.122.912	\$ 9.122.912	0,04	0,04	1052%	1052%	91%	91%
23	\$ 7.590.733	\$ 7.590.733	0,04	0,04	117%	117%	54%	54%
24	\$ 7.366.309	\$ 7.366.309	0,29	0,29	817%	817%	89%	89%
25	\$ 2.358.457	\$ 2.358.457	0,18	0,18	181%	181%	64%	64%
26	\$ 6.959.438	\$ 7.197.609	0,11	0,10	98%	105%	49%	51%
27	\$ 1.116.654	\$ 1.116.654	1,01	1,00	794%	794%	89%	89%
28	\$ 8.291.652	\$ 8.605.612	3,59	1,30	793%	826%	89%	89%
29	\$ 2.890.489	\$ 2.938.751	0,59	0,29	1314%	1337%	93%	93%
30	\$ 3.058.600	\$ 3.058.600	0,72	0,71	1009%	1009%	91%	91%
31	<u>\$ 5.255.603</u>	<u>R\$10.151.130</u>	<u>483,68</u>	<u>164,54</u>	<u>8224%</u>	<u>15977%</u>	<u>99%</u>	<u>99%</u>
32	\$ 6.050.815	\$ 6.050.815	0,56	0,54	4343%	4343%	98%	98%
33	\$ 5.663.910	\$ 5.663.910	1,06	1,04	6879%	6879%	99%	99%
34	<u>\$ 8.976.519</u>	<u>\$ 8.976.519</u>	<u>0,22</u>	<u>0,20</u>	<u>10211%</u>	<u>10211%</u>	<u>99%</u>	<u>99%</u>
35	\$ 9.749.556	\$ 9.749.556	0,56	0,54	4299%	4299%	98%	98%
36	\$ 9.834.116	R\$11.768.039	199,20	113,96	10903%	13067%	99%	99%

Como nas outras tabelas, as instâncias cujas soluções ótimas são conhecidas estão sublinhadas e em itálico. As colunas 2, 6 e 8 dizem respeito à melhor solução encontrada pelo método, enquanto as colunas 3, 7 e 9 referem-se à solução média das cinco execuções. O tempo da execução mais demorada da RVTMR e a duração média das execuções são apresentadas na quarta e na quinta colunas, respectivamente.

6.4 ANÁLISE DOS RESULTADOS

Para garantir a viabilidade das soluções, os resultados obtidos tiveram verificados os seguintes aspectos:

- Início de execução da tarefa ocorre dentro de sua janela de atendimento, para todas as tarefas.
- Número total de embarcações demandadas é igual ao número de embarcações alocadas a cada tarefa.
- Das embarcações alocadas a uma tarefa, o número de rebocadores da classe superior é maior ou igual à demanda desta mesma classe; para todas as tarefas.
- Em uma dada rota com pelo menos duas tarefas, o tempo decorrido entre final de uma tarefa e início da tarefa subsequente é maior ou igual ao tempo que a embarcação associada àquela rota leva para percorrer o arco em questão; para todas as rotas.
- Para qualquer uma das rotas, o instante de início da primeira atividade executada é maior ou igual à data de liberação da embarcação associada mais o tempo de deslocamento do arco em questão.

Tal verificação foi feita utilizando um programa em c++, capaz de verificar os pontos mencionados para o conjunto de arquivos de saída gerados nos testes computacionais. Para conferir e visualizar os resultados um-a-um, foi usada uma rotina em VBA que gera diagramas de espaço-tempo e de Gantt. A macro utilizada foi a desenvolvida por Mendes (2005), com algumas adaptações para torná-la capaz de também gerar gráficos de soluções parciais. O diagrama permite a visualização da propriedade da interdependência entre as rotas, graças ao impacto causado pela inserção de tarefas na solução parcial. Com apoio dos gráficos, é possível verificar que o movimento realizado funciona bem, respeitando as janelas de atendimento de todas as tarefas impactadas, sem formação de subciclos.

A Figura 11 foi produzida pela planilha citada, ilustrando a representação de uma solução da instância 6 por diagramas de Gantt. Ela mostra uma solução completa e viável obtida pela Heurística Construtiva. Cada uma das 16 linhas do tempo representa a rota de uma embarcação, identificada por seu ID à esquerda. As linhas do tempo estão numeradas de 0 até 35, representando os dias do horizonte de planejamento. Cada barra colorida posicionada ao longo das linhas do tempo representam o tempo em que o rebocador fica ocupado, ou executando uma atividade ou em deslocamento.

As barras que representam execução de tarefa são identificadas por seu respectivo ID no círculo central à barra. Por exemplo, o rebocador 1 realiza a tarefa 3, iniciando-a no 3º dia do horizonte e finalizando-a no 24º dia. Já as barras não identificadas com um círculo representam o tempo de deslocamento de cada rebocador entre atividades consecutivas ou do local de início até a primeira atividade. Ou seja, o rebocador 1 inicia seu trajeto assim que fica disponível, no dia 1, e chega na plataforma de início da atividade 3 pouco depois, ainda no dia 1. Então, ele fica ocioso na plataforma até que a tarefa possa ser iniciada, porque a janela de tempo desta atividade inicia-se apenas no dia 3.

Figura 11 – Ilustração de solução obtida pela Heurística Construtiva para a instância 6, no formato de diagramas de Gantt

Ainda na figura, barras de mesma coloração representam embarcações de mesma classe: os rebocadores 1 a 11 são da classe inferior, enquanto os demais (12 a 16) pertencem à classe superior.

Na Tabela 26, são apresentados lado a lado os resultados dos quatro métodos de solução testados, junto aos resultados obtidos com pelo otimizador Gurobi, usando o modelo compacto. Os resultados sublinhados correspondem às instâncias cujo ótimo é conhecido. Em negrito e em tom mais escuro, aparecem os resultados do método que

se sobressaiu aos demais. Também em **negrito**, mas em tom mais claro, são destacados os melhores resultados obtidos pelos métodos propostos neste trabalho, ou seja, ignorando o otimizador.

Observando a tabela, nota-se que o Gurobi obteve as melhores soluções em 35 instâncias, ou seja, em todas as que chegou a alguma solução dentro do prazo estipulado, de duas horas. Para a instância 20, que o Gurobi não resolveu, a heurística RVTMR gerou a melhor solução dentre os métodos. Comparando as soluções geradas apenas pelas heurísticas propostas, constata-se que para instâncias médias a HC com múltiplos recomeços (HCMR) teve a maior quantidade de resultados melhores que os demais métodos; enquanto a RVTMR superou as demais nas instâncias pequenas e o GRASP, nas grandes. Contudo, se ao invés de avaliar os métodos pela quantidade de instâncias em que sua solução foi a melhor das quatro, o critério adotado for a média dos desvios em relação ao LB, constata-se que a GRASP obtém melhores resultados nas instâncias pequenas e grandes, enquanto a HCMR se sobrepõe às demais nas instâncias de 20 tarefas.

Com relação ao tempo de execução, as instâncias pequenas não justificam a aplicação dos métodos, uma vez que o pacote otimizador produz resultados ótimos rapidamente. Para os outros tamanhos, a RVTMR foi mais rápida na maioria das instâncias.

Tabela 26 – Melhores resultados obtidos para 5 execuções dos quatro métodos propostos e comparação com o Gurobi

ID	Gurobi			HCMR			BLMR			GRASP			RVTMR	
	GAP (UB Gurobi)	Desvio _{LB} (UB Gurobi)	Tempo máximo [s]	Desvio _{LB} (HCMR best)	GAP(HC MR best)	Tempo máximo [s]	Desvio _{LB} (BLMR best)	GAP(BL MR best)	Tempo máximo [s]	Desvio _{LB} GRASP best)	GAP(GR ASP best)	Tempo máximo [s]	Desvio _{LB} (RVTMR best)	GAP(RV TMR best)
<u>1</u>	<u>0%</u>	<u>0%</u>	<u>10,81</u>	<u>7%</u>	<u>6%</u>	<u>43,49</u>	<u>5%</u>	<u>5%</u>	<u>56,34</u>	<u>5%</u>	<u>4%</u>	<u>159,31</u>	<u>3%</u>	<u>3%</u>
<u>2</u>	<u>0%</u>	<u>0%</u>	<u>13,58</u>	<u>16%</u>	<u>14%</u>	<u>27,85</u>	<u>15%</u>	<u>13%</u>	<u>83,46</u>	<u>2%</u>	<u>2%</u>	<u>63,03</u>	<u>1%</u>	<u>1%</u>
<u>3</u>	<u>0%</u>	<u>0%</u>	<u>23,59</u>	<u>44%</u>	<u>30%</u>	<u>140,71</u>	<u>68%</u>	<u>41%</u>	<u>193,74</u>	32%	<u>24%</u>	<u>367,63</u>	<u>55%</u>	<u>35%</u>
<u>4</u>	<u>0%</u>	<u>0%</u>	<u>24,85</u>	<u>2%</u>	<u>2%</u>	<u>98,42</u>	<u>0%</u>	<u>0%</u>	<u>118,58</u>	<u>2%</u>	<u>1%</u>	<u>179,86</u>	<u>8%</u>	<u>7%</u>
<u>5</u>	<u>0%</u>	<u>0%</u>	<u>0,78</u>	18%	<u>15%</u>	0,50	<u>128%</u>	<u>56%</u>	<u>71,23</u>	<u>38%</u>	<u>28%</u>	<u>9,16</u>	<u>59%</u>	<u>37%</u>
<u>6</u>	<u>0%</u>	<u>0%</u>	<u>76,87</u>	<u>0%</u>	<u>0%</u>	<u>229,01</u>	<u>0%</u>	<u>0%</u>	<u>601,32</u>	<u>0%</u>	<u>0%</u>	<u>709,42</u>	<u>0%</u>	<u>0%</u>
<u>7</u>	<u>0%</u>	<u>0%</u>	<u>14,74</u>	<u>79%</u>	<u>44%</u>	<u>28,59</u>	<u>72%</u>	<u>42%</u>	<u>36,41</u>	<u>76%</u>	<u>43%</u>	<u>60,14</u>	24%	<u>19%</u>
<u>8</u>	<u>0%</u>	<u>0%</u>	<u>36,91</u>	<u>0%</u>	<u>0%</u>	<u>63,25</u>	<u>0%</u>	<u>0%</u>	<u>108,20</u>	<u>0%</u>	<u>0%</u>	<u>102,40</u>	<u>15%</u>	<u>13%</u>
<u>9</u>	<u>0%</u>	<u>0%</u>	<u>40,98</u>	<u>29%</u>	<u>23%</u>	<u>75,72</u>	<u>29%</u>	<u>23%</u>	<u>184,38</u>	28%	<u>22%</u>	<u>196,68</u>	28%	<u>22%</u>
<u>10</u>	<u>0%</u>	<u>0%</u>	<u>33,82</u>	<u>7%</u>	<u>6%</u>	<u>57,35</u>	<u>20%</u>	<u>16%</u>	<u>94,24</u>	<u>7%</u>	<u>6%</u>	<u>300,08</u>	<u>6%</u>	<u>6%</u>
<u>11</u>	<u>0%</u>	<u>0%</u>	<u>26,07</u>	<u>0%</u>	<u>0%</u>	<u>163,02</u>	<u>0%</u>	<u>0%</u>	<u>180,30</u>	<u>0%</u>	<u>0%</u>	<u>774,73</u>	<u>0%</u>	<u>0%</u>
<u>12</u>	<u>0%</u>	<u>0%</u>	<u>104,64</u>	<u>151%</u>	<u>60%</u>	<u>78,95</u>	<u>1%</u>	<u>1%</u>	<u>322,62</u>	<u>151%</u>	<u>60%</u>	<u>295,93</u>	<u>205%</u>	<u>67%</u>
13	36%	55%	0,09	175%	64%	0,11	173%	63%	823,48	178%	64%	0,08	254%	72%
14	45%	83%	53,30	287%	74%	2.985,77	401%	80%	3.611,86	287%	74%	0,06	571%	85%
15	82%	457%	26,30	1139%	92%	2.817,25	2256%	96%	6.032,25	1356%	93%	0,07	2447%	96%
<u>16</u>	<u>0%</u>	<u>0%</u>	<u>4.591,97</u>	<u>9%</u>	<u>8%</u>	<u>7,95</u>	<u>216%</u>	<u>68%</u>	<u>9.631,62</u>	<u>9%</u>	<u>8%</u>	<u>16,64</u>	<u>147%</u>	<u>60%</u>
17	<u>0%</u>	<u>0%</u>	29,33	49%	33%	7.410,07	77%	43%	7.686,67	54%	35%	0,12	107%	52%
<u>18</u>	<u>0%</u>	<u>0%</u>	<u>1,15</u>	36%	<u>27%</u>	<u>6.753,79</u>	<u>38%</u>	<u>27%</u>	<u>1.472,56</u>	<u>68%</u>	<u>41%</u>	<u>0,11</u>	<u>68%</u>	<u>41%</u>
19	75%	298%	22,82	393%	80%	970,38	570%	85%	1.464,91	524%	84%	0,04	625%	86%
20	NA	NA	0,11	5158%	98%	0,05	4577%	98%	543,77	5337%	98%	0,05	4543%	98%
<u>21</u>	<u>0%</u>	<u>0%</u>	<u>9.571,79</u>	286%	<u>74%</u>	<u>4.348,14</u>	<u>358%</u>	<u>78%</u>	<u>5.249,58</u>	<u>435%</u>	<u>81%</u>	<u>0,87</u>	<u>1867%</u>	<u>95%</u>
22	83%	482%	0,37	846%	89%	0,05	1052%	91%	253,86	1045%	91%	0,04	1052%	91%
23	16%	19%	0,93	64%	39%	3.003,21	93%	48%	397,31	86%	46%	0,04	117%	54%
24	62%	165%	59,56	286%	74%	5.202,46	651%	87%	8.280,03	286%	74%	0,29	817%	89%
25	42%	72%	0,16	181%	64%	0,20	181%	64%	7.664,59	181%	64%	0,18	181%	64%

ID	Gurobi			HCMR			BLMR			GRASP			RVTMR	
	GAP (UB Gurobi)	Desvio _{LB} (UB Gurobi)	Tempo máximo [s]	Desvio _{LB} (HCMR best)	GAP(HC MR best)	Tempo máximo [s]	Desvio _{LB} (BLMR best)	GAP(BL MR best)	Tempo máximo [s]	Desvio _{LB} GRASP best)	GAP(GR ASP best)	Tempo máximo [s]	Desvio _{LB} (RVTMR best)	GAP(RV TMR best)
26	7%	7%	0,34	119%	54%	6.102,63	62%	38%	3.518,77	62%	38%	0,11	98%	49%
27	8%	9%	48,22	794%	89%	1,14	794%	89%	12.351,70	735%	88%	1,01	794%	89%
28	23%	30%	13,18	588%	85%	8.760,67	546%	85%	5.214,24	506%	83%	3,59	793%	89%
29	23%	30%	47,29	860%	90%	5.602,95	1308%	93%	9.154,44	848%	89%	0,59	1314%	93%
30	45%	82%	92,88	848%	89%	11.986,70	842%	89%	14.268,00	710%	88%	0,72	1009%	91%
<u>31</u>	<u>0%</u>	<u>0%</u>	<u>295,25</u>	<u>747%</u>	<u>88%</u>	<u>1.763,85</u>	<u>12392%</u>	<u>99%</u>	<u>3.849,44</u>	<u>632%</u>	<u>86%</u>	<u>483,68</u>	<u>8224%</u>	<u>99%</u>
32	80%	398%	112,47	1810%	95%	9.267,75	3039%	97%	3.628,41	1810%	95%	0,56	4343%	98%
33	76%	316%	587,65	1548%	94%	25.816,90	3765%	97%	12.341,30	1544%	94%	1,06	6879%	99%
<u>34</u>	<u>0%</u>	<u>0%</u>	<u>542,06</u>	<u>2079%</u>	<u>95%</u>	<u>8.519,07</u>	<u>2799%</u>	<u>97%</u>	<u>4.267,83</u>	<u>1685%</u>	<u>94%</u>	<u>0,22</u>	<u>10211%</u>	<u>99%</u>
35	85%	570%	473,05	1318%	93%	14.829,60	3752%	97%	8.118,40	1318%	93%	0,56	4299%	98%
36	85%	582%	1.320,60	2440%	96%	33.173,00	5633%	98%	18.900,30	2440%	96%	199,20	10903%	99%

7 CONCLUSÕES

Este trabalho aborda um problema enfrentado por empresas petrolíferas que operam em alto mar, utilizando rebocadores especiais para transporte, posicionamento, fixação e operação de equipamentos cruciais à operação. Apesar das tarefas serem planejadas com certa antecedência, a limitação no tamanho da frota e o impacto financeiro do atraso na execução de uma atividade na plataforma tornam a tomada de decisão caras à equipe gestão. Nesta pesquisa, tal problema é abordado sob a ótica da Pesquisa Operacional, com a qual é possível modelar matematicamente a tomada de decisão, para gerar um cronograma das atividades de cada rebocador no horizonte de planejamento pré-estabelecido, em função das atividades previstas.

O modelo matemático proposto pertence à classe de Problemas de Roteirização de Veículos, em que há Janelas de Tempo para atendimento dos clientes e restrições de Sincronização de operações, o que significa que o cliente demanda a visita simultânea de mais de um veículo (nesta aplicação, rebocadores AHTS). Ademais, a frota é heterogênea e a demanda de cada cliente especifica quantos recursos de cada tipo são necessários, sendo que veículos mais potentes podem substituir os menos potentes, a um custo maior de transporte. Na aplicação que motiva este trabalho, quase todas as tarefas que dependem da frota de AHTS necessitam a atuação de mais de um rebocador, podendo exigir até quatro ao mesmo tempo.

Com a revisão da literatura pertinente (capítulo 2), constata-se que o problema em questão, aqui denominado PRVJTSin-Het, generaliza problemas comprovadamente NP-difíceis. Isto justifica a escolha de um método de solução heurístico, que seja capaz de encontrar boas soluções rapidamente. Contudo, como apontam outros autores que estudaram problemas de roteirização com operações sincronizadas, a interdependência entre as rotas dos veículos dificulta a aplicação de métodos de melhoria baseados em busca em vizinhança, pois pequenas modificações na sequência de atendimento podem gerar a necessidade de recalcular toda a solução. Por conta disto, espera-se que quanto mais recursos simultâneos sejam exigidos e quanto maior a proporção de clientes (ou tarefas) que exigem mais de um veículo, maior o tempo computacional despendido por este tipo de heurística.

Uma vez que o PRVJTSin-Het é novo na literatura, tendo sido abordado apenas pelo do grupo de pesquisa do qual a autora deste trabalho faz parte, uma das contribuições deste trabalho está na proposição de modelos matemáticos eficientes para o

PRVJTSin-Het. Por isto, dois modelos matemáticos são propostos nesta tese e testados para a obtenção de soluções ótimas ou de bons limitantes inferiores: um baseado na abordagem clássica para este tipo de problema; e outro mais compacto, inspirado em uma estratégia de modelagem encontrada na literatura recente do Problema de Coleta e Entrega. Ainda, para o primeiro modelo matemático, é proposta uma versão que realiza cálculos de pré-processamento para reduzir o grafo de definição do problema, diminuindo a quantidade de arcos e, conseqüentemente, de variáveis de decisão.

Os resultados dos testes realizados com o Gurobi nas instâncias pequenas e médias mostram que a versão compacta do modelo é mais eficiente que a modelagem clássica, tendo não só chegado às soluções ótimas mais rapidamente (nos casos em que foi possível), como resolvido mais instâncias que os demais. Com isso, esta tese contribui para a área propondo não só um problema novo de otimização combinatória, como uma estratégia de modelagem eficiente. Ainda, este modelo pode ser adaptado para os problemas relacionados ao PRVJTSin-Het, com intuito de melhorar *Lower Bounds* de *benchmarks* da literatura.

Com relação aos métodos de solução propostos, a complexidade da heurística construtiva proposta reflete a dificuldade de gerar algoritmos que resolvam com sucesso todas as instâncias do PRVJTSin-Het. De fato, outros autores apontam para a dificuldade de gerar soluções para problemas de roteirização com sincronização de recursos, mas a comparação entre as instâncias encontradas na literatura para tais problemas e as propostas neste trabalho indica que a complexidade aqui é maior. Não obstante, a heurística construtiva desenvolvida é capaz de gerar soluções viáveis para todas as instâncias propostas, rapidamente; enquanto o Gurobi, pacote de otimização usado para obtenção de limitantes inferiores para o problema, não é capaz de resolver uma delas. Por outro lado, para as instâncias que o otimizador consegue resolver, constata-se que a qualidade das soluções produzidas pela HC não é boa.

Para obtenção de soluções de melhor qualidade, nesta tese são propostos quatro métodos: múltiplos recomeços da heurística construtiva; busca local com movimento único, de remoção de uma tarefa; GRASP usando este mesmo movimento, mas critério de parada por estagnação da solução; e RVTMR, um método baseado nas heurísticas do tipo destrói-e-repara, o que ao invés de gerar soluções inteiras a cada iteração do método, apostam na remoção de parte suficientemente grande da solução, seguida de processo de reconstrução que use uma lógica diferente daquela adotada na produção

de uma solução viável. Contudo, os testes computacionais mostram que nenhum dos métodos obteve resultados satisfatórios no conjunto de instância do PRVJTSin-Het.

Apesar das heurísticas propostas para a geração de boas soluções não apresentarem o desempenho esperado, espera-se que sua documentação nesta tese ajude pesquisadores interessados no problema a seguir caminhos diferentes dos trilhados aqui, ou inspire novas ideias. Tendo em vista o sucesso obtido com o novo modelo matemático proposto nesta tese, uma possibilidade promissora de continuação desta pesquisa é combinar métodos otimizantes com heurísticas.

Por fim, sugere-se o teste das seguintes abordagens para melhorar os resultados obtidos nesta pesquisa, deixadas para estudos futuros:

- para métodos de solução combinando a construtiva com heurística de melhoria, permitir que a heurística construtiva rompa algumas restrições do problema na geração de soluções iniciais, como utilizar mais embarcações do que há na instância e/ou não atender todas as tarefas demandadas;
- adotar a solução obtida pela heurística construtiva como partida para o método *branch-and-bound*, usado pelo pacote otimizador Gurobi;
- testar o GRASP e a BLMR partindo das soluções obtidas pela HCMR;
- alterar o critério de aceitação de soluções piores ao longo da heurística de melhoria, como feito pela meta-heurística *Simulated Annealing*, em que nas iterações iniciais são aceitas soluções com maior acréscimo na função objetivo; enquanto nas iterações finais, são aceitas apenas soluções com baixo acréscimo na função objetivo;
- incorporar indicadores que computam o tempo navegando, o tempo aguardando e o tempo em uso (executando tarefa) da frota, para que os movimentos de remoção e inserção possam levar em conta a ociosidade de cada rota ou possíveis desbalanços entre as ociosidades de rebocadores diferentes;
- outras estruturas de vizinhança também podem ser implementadas, usando os movimentos desenvolvidos para este trabalho, de remoção e inserção de tarefas em rotas interdependentes;
- uma vez que outras regras de destruição e construção da solução tenham sido criadas, testar a meta-heurística *Adaptative Large Neighbourhood Search* (ALNS) no problema, uma vez que o framework deste algoritmo também utiliza operadores de remoção e inserção;

- utilizar *Constraint Programming* para obtenção de solução inicial para o problema;
- os movimentos de modificação da solução são complexos neste tipo de problema, e sua implementação depende de estruturas de dados auxiliares, que possivelmente podem ser melhoradas. Por exemplo, soluções equivalentes são consideradas como diferentes pelos métodos usados nesta pesquisa, caso o mesmo agrupamento de tarefas seja atendido por rebocadores da mesma classe, mas de ID diferentes. Seria interessante desenvolver um algoritmo que entenda que tais soluções são essencialmente iguais, reduzindo o número de combinações a avaliar na vizinhança de uma solução.

REFERÊNCIAS

- AAS, Bjørnar; HALSKAU, Øyvind; WALLACE, Stein W. The role of supply vessels in offshore logistics. v. 11, p. 302–325, 2009. DOI 10.1057/mel.2009.7. Disponível em: <http://dx.doi.org/10.1057/mel.2009.7>.
- AFIFI, Sohaib; DANG, Duc-cuong; MOUKRIM, Aziz. A Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows and Synchronization Constraints. 7997., 2013. **Anais [...]**. Berlim: Springer Berlin Heidelberg, 2013. p. 259–265. DOI 10.1007/978-3-642-44973-4. Disponível em: <https://hal.archives-ouvertes.fr/hal-00916972>.
- AIT HADDADENE, Syrine Roufaida; LABADIE, Nacima; PRODHON, Caroline. A GRASP x ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. **Expert Systems with Applications**, v. 66, p. 274–294, 2016. DOI 10.1016/j.eswa.2016.09.002. Disponível em: <http://dx.doi.org/10.1016/j.eswa.2016.09.002>.
- ALTAY, Nezihi; GREEN, Walter G. OR/MS research in disaster operations management. **European Journal of Operational Research**, v. 175, n. 1, p. 475–493, nov. 2006. DOI 10.1016/j.ejor.2005.05.016. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0377221705005047>. Acesso em: 28 fev. 2013.
- ARENALES, M. *et al.* Otimização discreta. In: ARENALES, M. *et al.* (org.). **Pesquisa Operacional**. 8ª impress ed. Rio de Janeiro: Ed. Campus/Elsevier, 2007. p. 163–287.
- ARTIGUES, Christian; MICHELON, Philippe; REUSSER, Stéphane. Insertion techniques for static and dynamic resource-constrained project scheduling. **European Journal of Operational Research**, v. 149, n. 2, p. 249–267, 2003. DOI 10.1016/S0377-2217(02)00758-0.
- ARTIGUES, Christian; ROUBELLAT, François. Polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. **European Journal of Operational Research**, v. 127, n. 2, p. 297–316, 2000. DOI 10.1016/S0377-2217(99)00496-8.
- BALLOU, Ronald H. **Gerenciamento da Cadeia de Suprimentos: Logística Empresarial**. 5. ed. [s.l.]: Bookman, 2009.
- BERTELS, Stefan; FAHLE, Torsten. A hybrid setup for a hybrid scenario: Combining heuristics for the home health care problem. **Computers and Operations Research**, v. 33, n. 10, p. 2866–2890, 2006. DOI 10.1016/j.cor.2005.01.015.
- BISKUP, Dirk; FELDMANN, Martin. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. **Computers and Operations Research**, v. 28, n. 8, p. 787–801, 2001. DOI 10.1016/S0305-0548(00)00008-3.
- BŁAZEWCZ, J. *et al.* Scheduling tasks on two processors with deadlines and additional resources. **European Journal of Operational Research**, v. 26, n. 3, p. 364–370, 1986. DOI 10.1016/0377-2217(86)90138-4.

BRADLEY, Stephen; HAX, Arnoldo; MAGNANTI, Thomas. Mathematical Programming: An Overview. **Applied Mathematical Programming**. [s.l.: s.n.], 1977.

BREDSTRÖM, David; RÖNNQVIST, Mikael. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. **European Journal of Operational Research**, v. 191, n. 1, p. 19–29, 2008. DOI 10.1016/j.ejor.2007.07.033.

BRUCKER, Peter. **Scheduling Algorithms**. 5^a ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. Disponível em: <http://link.springer.com/10.1007/978-3-662-03612-9>.

BRUCKER, Peter; KNUST, Sigrid. **Complex Scheduling**. 2^a ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Disponível em: <http://link.springer.com/10.1007/978-3-642-23929-8>.

CAVALCANTI, Luísa Brandão; MENDES, André Bergsten. A Greedy Heuristic for the Vehicle Routing Problem with Time Windows, Synchronization Constraints and Heterogeneous Fleet. *In*: C., Paternina-Arboleda; VOSS, Stefan (org.). **International Conference on Computational Logistics. Lecture notes in computer science**. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019. v. 11756. p. 265–280. DOI 10.1007/978-3-319-11421-7. Disponível em: <http://link.springer.com/10.1007/978-3-319-11421-7>

CAVALCANTI, Luísa Brandão; MENDES, André Bergsten. Roteirização de veículos com operações sincronizadas e janelas de tempo: uma aplicação à indústria do petróleo. 2018. **Anais [...]**. Rio de Janeiro: SOBRAPO, 2018. DOI ISSN 1518-1731. Disponível em: <https://proceedings.science/sbpo/papers/roteirizacao-de-veiculos-com-operacoes-sincronizadas-e-janelas-de-tempo%3A-uma-aplicacao-a-industria-do-petroleo>.

CHENG, T. C.E.; SIN, C. C.S. A state-of-the-art review of parallel-machine scheduling research. **European Journal of Operational Research**, v. 47, n. 3, p. 271–292, 1990. DOI 10.1016/0377-2217(90)90215-W.

CHOPRA, Sunil; MEINDL, Peter. **Supply Chain Management: Strategy, Planning, and Operation - Global Edition**. 5. ed. Harlow: Pearson Education Limited, 2013.

CHRISTIAENS, Jan; VANDEN BERGHE, Greet. Slack Induction by String Removals for Vehicle Routing Problems. **Transportation Science**, v. 54, n. 2, p. 417–433, 15 jan. 2020. DOI 10.1287/trsc.2019.0914. Disponível em: <http://pubsonline.informs.org/doi/10.1287/trsc.2019.0914>.

DANIELS, Richard L; HOOPEs, Barbara J; MAZZOLA, Joseph B. Scheduling Parallel Manufacturing Cells with Resource Flexibility. **Management Science**, v. 42, n. 9, p. 1260–1276, 1996. DOI 10.2307/2634436.

DANIELS, Richard L; MAZZOLA, Joseph B. Flow Shop Scheduling with Resource Flexibility. **Operations Research**, v. 42, n. 2, p. 504–522, 1994.

DEES, William A; SMITH, Robert J. 18th Design Automation Conference. 1981. **Anais [...]**. Nashville, TN, USA: IEEE, 1981. p. 382–390. DOI 10.1109/DAC.1981.1585385. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1585385>.

DELORME, Maxence; IORI, Manuel; MARTELLO, Silvano. Bin packing and cutting stock problems: Mathematical models and exact algorithms. **European Journal of Operational Research**, v. 255, n. 1, p. 1–20, 2016. DOI 10.1016/j.ejor.2016.04.030. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2016.04.030>.

DESROCHERS, M. *et al.* Towards a model and algorithm management system for vehicle routing and scheduling problems. **Decision Support Systems**, v. 25, n. 2, p. 109–133, 1999. DOI 10.1016/S0167-9236(98)00090-6. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167923698000906>.

DOHN, Anders; KOLIND, Esben; CLAUSEN, Jens. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. **Computers and Operations Research**, v. 36, n. 4, p. 1145–1157, 2009. DOI 10.1016/j.cor.2007.12.011. .

DREXL, M. Synchronization in Vehicle Routing--A Survey of VRPs with Multiple Synchronization Constraints. **Transportation Science**, v. 46, n. 3, p. 297–316, 2012. DOI 10.1287/trsc.1110.0400.

DURON, Cyril; PROTH, Jean Marie; WARDI, Yorai. Insertion of a random task in a schedule: A real-time approach. **European Journal of Operational Research**, v. 164, n. 1, p. 52–63, 2005. DOI 10.1016/j.ejor.2003.11.024.

EDIS, Emrah B.; OGUZ, Ceyda; OZKARAHAN, Irem. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. **European Journal of Operational Research**, v. 230, n. 3, p. 449–463, 2013. DOI 10.1016/j.ejor.2013.02.042. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2013.02.042>.

EN-NAHLI, Laila *et al.* Local Search Analysis for a Vehicle Routing Problem with Synchronization and Time Windows Constraints in Home Health Care Services. **IFAC-PapersOnLine**, v. 49, n. 12, p. 1210–1215, 2016. DOI 10.1016/j.ifacol.2016.07.674. Disponível em: <http://dx.doi.org/10.1016/j.ifacol.2016.07.674>.

EVEBORN, Patrik; FLISBERG, Patrik; RÖNNQVIST, Mikael. Laps Care-an operational system for staff planning of home care. **European Journal of Operational Research**, v. 171, n. 3, p. 962–976, 2006. DOI 10.1016/j.ejor.2005.01.011.

FAGERHOLT, Kjetil. Ship scheduling with soft time windows: An optimisation based approach. **European Journal of Operational Research**, v. 131, n. 3, p. 559–571, 2001. DOI 10.1016/S0377-2217(00)00098-9.

FEO, Thomas A.; RESENDE, Mauricio G.C. Greedy Randomized Adaptive Search Procedures. **Journal of Global Optimization**, v. 6, n. 2, p. 109–133, 1995. DOI 10.1007/BF01096763.

FESTA, P.; RESENDE, M. G. C. GRASP: basic components and enhancements. **Telecommunication Systems**, v. 46, n. 3, p. 253–271, 5 mar. 2011. DOI 10.1007/s11235-010-9289-z. Disponível em: <http://link.springer.com/10.1007/s11235-010-9289-z>.

FRANCIS, Peter; ZHANG, Guangming; SMILOWITZ, Karen. Improved modeling and

solution methods for the multi-resource routing problem. **European Journal of Operational Research**, v. 180, n. 3, p. 1045–1059, 2007. DOI 10.1016/j.ejor.2006.03.054.

FURTADO, Maria Gabriela S.; MUNARI, Pedro; MORABITO, Reinaldo. Pickup and delivery problem with time windows: A new compact two-index formulation. **Operations Research Letters**, v. 45, n. 4, p. 334–341, 2017. DOI 10.1016/j.orl.2017.04.013. Disponível em: <http://dx.doi.org/10.1016/j.orl.2017.04.013>.

GLOVER, Fred; LAGUNA, Manuel. Tabu Search. In: DZ., Du; P.M, Pardalos (org.). **Handbook of Combinatorial Optimization, Vol. 3**. 3. ed. Boston, MA: Springer US, 1998. p. 2093–2229. DOI 10.1007/978-1-4613-0303-9_33. Disponível em: http://link.springer.com/10.1007/978-1-4613-0303-9_33.

GLOVER, Fred; LAGUNA, Manuel; MARTI, Rafael. Control and Cybernetics Fundamentals of scatter search and path relinking. **Control and cybernetics**, v. 29, n. 3, p. 653–684, 2000.

GRÖFLIN, Heinz; KLINKERT, Andreas. Feasible insertions in job shop scheduling, short cycles and stable sets. **European Journal of Operational Research**, v. 177, n. 2, p. 763–785, 2006. DOI 10.1016/j.ejor.2005.12.025.

GRÖFLIN, Heinz; KLINKERT, Andreas; DINH, Nguyen Pham. Feasible job insertions in the multi-processor-task job shop. **European Journal of Operational Research**, v. 185, n. 3, p. 1308–1318, 2008. DOI 10.1016/j.ejor.2005.10.077.

HOJABRI, Hossein *et al.* Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. **Computers and Operations Research**, v. 92, p. 87–97, 2018. DOI 10.1016/j.cor.2017.11.011. Disponível em: <https://doi.org/10.1016/j.cor.2017.11.011>.

HOMBERGER, Jörg; GEHRING, Hermann. Two evolutionary metaheuristics for the vehicle routing problem with time windows. **INFOR Journal**, v. 37, n. May, 1999. DOI 10.1080/03155986.1999.11732386.

JOHNSON, David S.; PAPADIMITRIOU, Christos H.; YANNAKAKIS, Mihalís. How easy is local search? 100., 1985. **Anais [...]**. [s.l.]: IEEE, 1985. p. 39–42. DOI 10.1109/SFCS.1985.31. Disponível em: <http://ieeexplore.ieee.org/document/4568125/>.

KARLSSON, Jenny; RÖNNQVIST, Mikael; BERGSTRÖM, Johan. An optimization model for annual harvest planning. **Canadian Journal of Forest Research**, v. 34, n. 8, p. 1747–1754, 2004. DOI 10.1139/x04-043. Disponível em: <http://www.nrcresearchpress.com/doi/abs/10.1139/x04-043>.

KHANAFER, Ali; CLAUTIAUX, François; TALBI, El-ghazali. New lower bounds for bin packing problems with conflicts. **European Journal of Operational Research**, v. 206, n. 2, p. 281–288, 2010. DOI 10.1016/j.ejor.2010.01.037. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2010.01.037>.

KONTORAVDIS, George; BARD, Jonathan F. A GRASP for the vehicle routing problem with time windows. **ORSA journal on Computing**, v. 7, n. 1, p. 10–23, 1995.

KOVACS, Attila A. *et al.* Adaptive large neighborhood search for service technician routing and scheduling problems. **Journal of Scheduling**, v. 15, n. 5, p. 579–600, 2012. DOI 10.1007/s10951-011-0246-9.

LI, Yanzhi; LIM, Andrew; RODRIGUES, Brian. Manpower allocation with time windows and job-teaming constraints. **Naval Research Logistics**, v. 52, n. 4, p. 302–311, 2005. DOI 10.1002/nav.20075.

LIU, J *et al.* A new decision support model in multi-criteria decision making with intuitionistic fuzzy sets based on risk preferences and criteria reduction. **Journal of the Operational Research Society**, v. 64, n. 8, p. 1205–1220, 2013. DOI 10.1057/jors.2012.180. Disponível em: <http://www.palgrave-journals.com/doifinder/10.1057/jors.2012.180>.

LIU, Ran; TAO, Yangyi; XIE, Xiaolei. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. **Computers and Operations Research**, v. 101, p. 250–262, 2019. DOI 10.1016/j.cor.2018.08.002.

LOURENÇO, Helena R.; MARTIN, Olivier C.; STÜTZLE, Thomas. Iterated Local Search. *In*: GLOVER, Fred; KOCHENBERGER, G.A. (org.). **Handbook of Metaheuristics**. 1. ed. Boston: Kluwer Academic Publishers, 2003. p. 320–353. DOI 10.1007/0-306-48056-5_11. Disponível em: https://link.springer.com/chapter/10.1007/0-306-48056-5_11.

LU, Quan; DESSOUKY, Maged M. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. **European Journal of Operational Research**, v. 175, n. 2, p. 672–687, 2006. DOI 10.1016/j.ejor.2005.05.012.

MARTÍ, Rafael; RESENDE, Mauricio G.C.; RIBEIRO, Celso C. Multi-start methods for combinatorial optimization. **European Journal of Operational Research**, v. 226, n. 1, p. 1–8, 2013. DOI 10.1016/j.ejor.2012.10.012.

MENDES, André Bergsten. **PROGRAMAÇÃO DE FROTA DE APOIO A OPERAÇÕES “OFFSHORE” SUJEITA À REQUISIÇÃO DE MÚLTIPLAS EMBARCAÇÕES PARA UMA MESMA TAREFA**. 2007. 224 f. Escola Politécnica da Universidade de São Paulo, 2007.

MOKOTOFF, Ethel. Parallel machine scheduling problems: a survey. **Asia-Pacific Journal of Operational Research**, v. 18, n. 2, p. 193–242, 2001. Disponível em: [g:%5Cpapers%5Cbasic%5C.pdf](#).

PARRAGH, Sophie N.; DOERNER, Karl F. Solving routing problems with pairwise synchronization constraints. **Central European Journal of Operations Research**, v. 26, n. 2, p. 443–464, 2018. DOI 10.1007/s10100-018-0520-4.

PINEDO, Michael L. **Scheduling: Theory, algorithms, and systems**. 4^a ed. Nova Iorque: Springer New York, 2016.

PISINGER, David; ROPKE, Stefan. A general heuristic for vehicle routing problems. **Computers and Operations Research**, v. 34, n. 8, p. 2403–2435, 2007. DOI 10.1016/j.cor.2005.09.012.

PRINS, Christian. A GRASP x Evolutionary Local Search Hybrid for the Vehicle Routing Problem. In: PEREIRA, Francisco Babbista; TAVARES, Jorge (org.). **Bio-inspired Algorithms for the Vehicle Routing Problem**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 35–53. DOI 10.1007/978-3-540-85152-3_2. Disponível em: https://doi.org/10.1007/978-3-540-85152-3_2.

PRINS, Christian. A simple and effective evolutionary algorithm for the vehicle routing problem. **Computers & Operations Research**, v. 31, n. 12, p. 1985–2002, out. 2004. DOI 10.1016/S0305-0548(03)00158-8. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0305054803001588>.

RAHIMIAN, Erfan; AKARTUNALI, Kerem; LEVINE, John. A hybrid Integer Programming and Variable Neighbourhood Search algorithm to solve Nurse Rostering Problems. **European Journal of Operational Research**, v. 258, n. 2, p. 411–423, 2016. DOI 10.1016/j.ejor.2016.09.030. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2016.09.030>.

REDJEM, Rabeh; MARCON, Eric. Operations management in the home care services: a heuristic for the caregivers' routing problem. **Flexible Services and Manufacturing Journal**, v. 28, n. 1–2, p. 280–303, 2016. DOI 10.1007/s10696-015-9220-8. Disponível em: <http://dx.doi.org/10.1007/s10696-015-9220-8>.

RESENDE, Mauricio G.C.; RIBEIRO, Celso C. GRASP: Greedy randomized adaptive search procedures. **Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Second Edition**. [s.l.: s.n.], 2014. DOI 10.1007/978-1-4614-6940-7_11.

RIBEIRO, Celso C.; HANSEN, Pierre. **Essays and Surveys in Metaheuristics**. Boston, MA: Springer US, 2002. v. 15. Disponível em: <http://link.springer.com/10.1007/978-1-4615-1507-4>. (Operations Research/Computer Science Interfaces Series).

ROUSSEAU, Louis Martin; GENDREAU, Michel; PESANT, Gilles. The synchronized dynamic vehicle dispatching problem. **Infor**, v. 51, n. 2, p. 76–83, 2013. DOI 10.3138/infor.51.2.76.

SANTOS, L.F.O.M. *et al.* A variable neighborhood search algorithm for the bin packing problem with compatible categories. **Expert Systems with Applications**, v. 124, 2019. DOI 10.1016/j.eswa.2019.01.052.

SCHOLL, Armin; KLEIN, Robert; JÜRGENS, Christian. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. **Computers & Operations Research**, v. 24, n. 7, p. 627–645, 1997. DOI 10.1016/S0305-0548(96)00082-2.

SCHRIMPF, Gerhard *et al.* Record Breaking Optimization Results Using the Ruin and Recreate Principle. **Journal of Computational Physics**, v. 159, n. 2, p. 139–171, 2000. DOI 10.1006/jcph.1999.6413.

SEIXAS, Michel Povlovitsch *et al.* A heuristic approach to stowing general cargo into platform supply vessels. **Journal of the Operational Research Society**, v. 67, n. 1, p. 148–158, 2016. DOI 10.1057/jors.2015.62.

SHAW, Paul. Using constraint programming and local search methods to solve vehicle

routing problems. 1998, London, UK: UK: Springer-Verlag, 1998. p. 417–431.

SHYSHOU, Aliaksandr; GRIBKOVSKAIA, Irina; BARCELÓ, Jaume. A simulation study of the fleet sizing problem arising in offshore anchor handling operations. **European Journal of Operational Research**, v. 203, n. 1, p. 230–240, 2010. DOI 10.1016/j.ejor.2009.07.012. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2009.07.012>.

SLOWINSKI, Roman. Two approaches to problems of resource allocation among project activities—a comparative study. **Journal of the Operational Research Society**, v. 31, n. 8, p. 711–723, 1980. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-0019047699&partnerID=tZOtx3y1>.

SOLOMON, Marius M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. **Operations Research**, v. 35, n. 2, p. 254–265, abr. 1987. DOI 10.1287/opre.35.2.254. Disponível em: <http://pubsonline.informs.org/doi/abs/10.1287/opre.35.2.254>.

WERNER, Frank; WINKLER, Andreas. Insertion techniques for the heuristic solution of the job shop problem. **Discrete Applied Mathematics**, v. 58, n. 2, p. 191–211, 1995. DOI 10.1016/0166-218X(93)E0127-K.

APÊNDICE A - Versões anteriores da Heurística Construtiva

O desenvolvimento da heurística construtiva ocupou a maior parte da pesquisa que resultou na presente tese. Prova disso é que no exame de qualificação, o algoritmo apresentado não resolvia todas as instâncias e exigia tempo computacional elevado. Nesta seção, são apresentadas versões descartadas do algoritmo, para documentar ideias que não deram certo ou obtiveram desempenho inferior à versão final da heurística construtiva. Assim, busca-se fornecer informações que auxiliem outros pesquisadores interessados no desenvolvimento de métodos eficazes no problema de roteirização de tarefas com sincronização de recursos.

A.1. Heurística construtiva de duas fases

A abordagem adotada na fase inicial desta pesquisa usava duas fases de construção na solução, conforme se observa no procedimento global apresentado no Pseudocódigo 1 (ou 9?). No algoritmo, a função `construtivaAditiva`, corresponde à primeira fase da heurística, em que o encaixe das tarefas nas rotas é sempre feito nas últimas posições, o que simplifica bastante o algoritmo, ganhando tempo computacional. Já na segunda fase, o movimento usado é o mesmo da versão final da heurística construtiva, denominado `construtivaPorInsercao`, em que as tarefas podem ser inseridas em quaisquer posições das rotas de conjuntos de rebocadores compatíveis com sua demanda.

Duas versões dos algoritmos `construtivaAditiva` e `construtivaPorInsercao`, foram testadas, com ou sem randomização na escolha de tarefas e soluções parciais geradas a cada iteração do método. Na versão sem aleatoriedade, a tarefa escolhida a cada iteração é sempre a primeira da lista que ainda não tenha sido selecionada; enquanto na randômica, a tarefa é sorteada entre aquelas que ocupam as primeiras posições da lista. Uma vez selecionada a tarefa, parte-se para a designação das rotas às quais a tarefa será adicionada, seja ao fim delas (válido para as duas funções, `construtivaAditiva` e `construtivaPorInsercao`) ou em posição intermediária (apenas para o algoritmo `construtivaPorInsercao`). Assim como na versão final da HC, este processo é feito buscando diminuir o **custo incremental**, denotado por Δz . Para isso, todos os subconjuntos de navios que podem atender a tarefa são avaliados, calculando-se para cada um deles o incremento de custo de deslocamento e de penalidade que seria gerado caso a tarefa fosse a ele designada. Então, na versão com aleatoriedade, a escolha é feita sorteando-se entre as melhores opções de alocação, ou seja, aquelas

de menor custo incremental na função objetivo do problema, enquanto na versão não-aleatória simplesmente adota-se a alocação de menor custo. Uma vez alocada a tarefa, parte-se para a seleção da próxima tarefa da lista ordenada, repetindo o processo até que não haja mais tarefa em tal lista. Se em qualquer iteração a tarefa não puder ser encaixada nas rotas parciais, ela é guardada em um conjunto separado de tarefas, chamado *backlog*, para tratamento posterior.

A inicialização das estruturas de dados é feita logo antes do algoritmo, nas 6 primeiras linhas do Pseudocódigo 12. Todas as tarefas do problema são armazenadas em um conjunto J^A , enquanto o *backlog*, denotado por J^B , é inicializado vazio e serve para guardar as tarefas não adicionadas às rotas. As rotas de cada embarcação do conjunto são inicializadas com o depósito inicial da embarcação correspondente. Então, a primeira fase da heurística é executada enquanto houver tarefa em J^A , conforme *loop* das linhas 8 e 9. Após tentar fazer a adição de todas as tarefas, as rotas recebem o depósito final na sua última posição (linhas 10 e 11). Caso haja alguma tarefa no *backlog*, tenta-se inserir (uma a uma) as tarefas não roteirizadas, usando o algoritmo *construtivaPorInsercao*.

Pseudocódigo 12–Algoritmo da heurística construtiva desenvolvida nas etapas iniciais da pesquisa, posteriormente descartada

```

1  Inicializa conjunto de tarefas a alocar,  $J^A := J$ , e backlog  $J^B := \emptyset$ 
2  Ordena  $J^A$  segundo regra de ordenação selecionada
3  Cria  $ES_j = 0$  e  $tConclusao_j = 0 \ \forall j \in J$ ; e  $ES = \bigcup_{j=1}^n ES_j$ ,  $tConclusao = \bigcup_{j=1}^n tConclusao_j$ 
4  Inicializa  $m$  conjuntos ordenados  $rota_k$ , fazendo  $rota_k = \{O_k^+\}$  para  $1 \leq k \leq m$ 
5  Inicializa  $n$  conjuntos  $Sucessores(j)$ , com  $1 \leq j \leq n$ 
6  Cria  $rotas = \bigcup_{k=1}^m rota_k$  e  $Sucessores = \bigcup_{j=1}^n Sucessores(j)$ ;
7  Algoritmo hConstrutivaGulosa:
8  Enquanto  $J^A \neq \emptyset$  faça:
9      construtivaAditiva( $J^A, J^B, ES, tConclusao, rotas, Sucessores$ )
10     Para toda  $rota_k \in rotas$  faça:
11          $rota_k = rota_k \cup O_k^-$ 
12     Se  $J^B \neq \emptyset$ 
13         construtivaPorInsercao( $J^B, ES, tConclusao, rotas, Sucessores$ )
14  Fim

```

A primeira fase da heurística construtiva é apresentada no Pseudocódigo 13. A heurística começa com a seleção de uma tarefa *job* dentre as tarefas de J^A para adição em *rotas*, na linha 2. Para cada um dos modos possíveis para sua execução, ou seja, para cada $M_{job,g} \in \hat{M}_{job}$, verifica-se se a janela de tempo de *job* seria respeitada caso a tarefa fosse adicionada às posições finais das rotas do conjunto $M_{job,g}$. Caso positivo, a adição é considerada viável e o modo é guardado no conjunto *listaCandidatos* (linhas 7 e 8). Então, nas linhas 10 a 13, se houver pelo menos um modo viável para adição

da tarefa, o conjunto `listaCandidatos` é ordenado segundo o acréscimo de custo na função objetivo, um dos modos da lista é escolhido e a adição é realizada. Por outro lado, se a `listaCandidatos` estiver vazia, a tarefa é guardada no *backlog*, conforme linhas 14 e 15.

Pseudocódigo 13 – Algoritmo da primeira fase da heurística construtiva, aditiva

```

1  Algoritmo construtivaAditiva( $J^A, J^B, ES, tConclusao, rotas, Sucessores$ ):
2    job  $\leftarrow$  escolheTarefaEm( $J^A$ )
3     $J^A = J^A \setminus \text{job}$ 
4    Cria listaCandidatos  $:= \emptyset$ 
5     $g \leftarrow 1$ 
6    Enquanto  $g \leq |\hat{M}_{job}|$  faça
7      Se (tentaAdição(job,  $M_{job,g}$ , rotas) é viável)
8        listaCandidatos = listaCandidatos  $\cup M_{job,g}$ 
9         $g \leftarrow g + 1$ 
10   Se (|listaCandidatos| > 0)
11     Ordena listaCandidatos não decrescentemente em  $\Delta z$ 
12     gSelecionado  $\leftarrow$  escolheAlocacaoEm(listaCandidatos)
13     fazAdiçãoECalculaParametros(job, gSelecionado, rotas)
14   Senão
15      $J^B = J^B \cup \text{job}$ 
16 Fim

```

Neste e nos próximos algoritmos, os procedimentos que requerem cálculos mais elaborados são realizados por **funções** ou **métodos**, que utilizam como dados de entrada informações contidas em seus argumentos, além de dados definidos na instância do problema. O argumento de uma função é apresentado entre os parênteses que sucedem sua chamada ou definição.

Nos pseudocódigos, as funções estão ressaltadas em negrito e têm coloração diferente do restante do algoritmo. A função **escolheTarefaEm**(conjunto de tarefas) seleciona uma tarefa dentre o conjunto de tarefas dado no seu argumento. Assim, na versão da heurística com aleatoriedade, essa função sorteia uma tarefa entre as primeiras posições da lista ordenada de tarefas, J^A ; enquanto para o mesmo procedimento sem randomização, a tarefa selecionada é a primeira da lista. Similarmente, a função **escolheAlocacaoEm**(conjunto de modos) é responsável pela seleção de um modo, ou seja, de um subconjunto de embarcações dentre todos os modos compatíveis com a tarefa e viáveis.

É importante lembrar que no momento da alocação de uma tarefa j , um grupo de embarcações M_{jg} compatível com j pode ser inviável porque a tarefa j não cabe nas rotas parciais, porque uma ou mais embarcações chegariam no ponto inicial de j depois do limitante superior de sua janela de tempo, b_j^H . Por isso, a função

tentaAdicao(tarefa, modo, rotas) verifica justamente se a tarefa em questão pode ser adicionada às rotas do modo especificado no argumento.

Por último, a função **fazAdiçãoECalculaParametros**(tarefa, modo, rotas), apresentada no Pseudocódigo 14, adiciona a tarefa ao final das rotas das embarcações que pertencem ao modo, calculando o ES_{job} resultante.

Conforme se nota no Pseudocódigo 14, cada vez que a tarefa job é adicionada a uma rota k do modo escolhido, calcula-se qual o instante mais cedo em que a embarcação pode chegar ao ponto de início de job , s_{job}^k , usando o instante mínimo para conclusão da tarefa anterior, $tConclusao_{prec_{job}^k}$, ou a data de liberação da embarcação, r^k , caso a tarefa precedente seja o depósito inicial (linhas 10 e 11, respectivamente), além do tempo de deslocamento entre o nó anterior e o da nova tarefa, $t_{prec_{job}^k, job}^k$. Caso a chegada da embarcação seja após o ES_{job} calculado até o momento, o mesmo é atualizado. Desta forma, ao fim do *loop* que percorre todas as embarcações, o mínimo instante de início da tarefa recém adicionada é igual ao instante em que a última embarcação do conjunto chega ao local de início do job ou ao início da janela de tempo a_{job}^H , o que ocorrer depois.

Pseudocódigo 14 – Algoritmo que adiciona uma tarefa no final das rotas parciais de uma solução e calcula o *Earliest Starting time* (ES) e a data de conclusão, $tConclusao$

```

1  Algoritmo fazAdiçãoECalculaParametros( $job$ ,  $M_{job,g}$ ,  $rotas$ ):
2     $ES_{job} \leftarrow a_{job}^H$ 
3     $k \leftarrow 1$ 
4    Enquanto  $k \leq m$  faça:
5      Se ( $k \in M_{job,g}$ )
6         $prec_{job}^k \leftarrow$  última tarefa de  $rota_k$ 
7        Se ( $sucessores(prec_{job}^k) \cap job = \emptyset$ )
8           $sucessores(prec_{job}^k) = sucessores(prec_{job}^k) \cup job$ 
9           $rota_k = rota_k \cup job$ 
10         Se ( $prec_{job}^k = O_k^+$ ) então  $s_{job}^k = r^k + t_{prec_{job}^k, job}^k$ 
11         Senão  $s_{job}^k = tConclusao_{prec_{job}^k} + t_{prec_{job}^k, job}^k$ 
12         Se ( $s_{job}^k > ES_{job}$ )  $ES_{job} \leftarrow s_{job}^k$ 
13        $k \leftarrow 1 + k$ 
14      $tConclusao_{job} = ES_{job} + p_{job}$ 
15  Fim

```

Ao término da primeira fase, o programa resultante é composto pelo conjunto **rotas** e pelo vetor **ES**. Caso tal solução seja parcial, ou seja, se houver uma ou mais tarefas no *backlog* J^B , a segunda fase da heurística é executada. O procedimento da Fase 2 é similar ao da primeira fase: a cada iteração, uma tarefa do *backlog* é selecionada e tenta-se encaixá-la nas rotas parciais, mas desta vez em posições intermediárias.

O Pseudocódigo 15 apresenta o processo adotado para a segunda fase da HC apresentada na qualificação desta pesquisa. Novamente, na versão randômica da heurística, as funções `escolheTarefaEm(conjunto de tarefas)` e `escolheAlocacaoEm(conjunto de candidatos)` fazem um sorteio entre as primeiras posições dos conjuntos ordenados especificados em seus respectivos argumentos. Os métodos `calculaFolgas(rotas,tConclusao,sucessores,ES)` e `calculaA(job,rotas)` são utilizados para montar uma estrutura de dados auxiliar ao movimento de inserção, conforme explicado anteriormente. O método `geraCombinacoesPosicoes(conjunto de embarcações, matriz A)` cria, para o conjunto de rotas das embarcações dadas no argumento, todas as combinações entre posições de inserção que, examinadas isoladamente, seriam viáveis, ou seja, posições ocupadas por tarefas com folga suficiente para encaixe da nova tarefa. Em seguida, para cada uma dessas combinações, a função `tentaInsercao(conjunto com modo e posição de inserção nas rotas do modo, tarefa)` testa a viabilidade da inserção da tarefa em tais posições, verificando se haveria formação de subciclo e se a janela de atendimento da tarefa seria respeitada.

Pseudocódigo 15 –Algoritmo da segunda fase da heurística construtiva, que usa inserção.

```

1  Algoritmo construtivaPorInsercao( $J^B$ , ES, tConclusao, rotas, Sucessores):
2     $J^C := J^B$  e  $J^B = \emptyset$ 
3    calculaFolgas(rotas, tConclusao, Sucessores, ES,  $J^C$ )
4    Enquanto ( $J^C \neq \emptyset$ )
5      job ← escolheTarefaEm( $J^C$ )
6       $J^C = J^C \setminus \text{job}$ 
7      listaCandidatos :=  $\emptyset$ 
8      A = calculaA(job, rotas)
9      g ← 1
10     Para todo  $M_{job,g} \in \hat{M}_{job}$  faça:
11       listaCombinacoes ← geraCombinacoesPosicoes( $M_{job,g}$ , A)
12       Para todo comb ∈ listaCombinacoes faça:
13         Se tentaInsercao(listaCombinacoes[comb], job, Sucessores, rotas) = viável
14           listaCandidatos = listaCandidatos  $\cup$  listaCombinacoes[comb]
15     Se ( $| \text{listaCandidatos} | > 0$ )
16       Ordena listaCandidatos não decrescentemente em  $\Delta z$ 
17       gSelecionado ← escolheAlocacaoEm(listaCandidatos)
18       fazInsercaoECalculaParametros(job, gSelecionado, rotas, Sucessores, ES, tConclusao)
19       calculaFolgas(rotas, tConclusao, Sucessores, ES,  $J^C$ ):
20     Senão
21        $J^B = J^B \cup \text{job}$ 
22  Fim

```

Caso positivo, o candidato à alocação (conjunto de posições de inserção em um conjunto de rotas compatível com a tarefa a inserir) é guardado na `listaCandidatos`. Após o cálculo de todas as combinações de posições viáveis de cada modo de atendimento, dentre todos os modos compatíveis com a tarefa selecionada, uma opção

viável para inserção da tarefa é selecionada desde que a lista de candidatos não esteja vazia. Se a lista estiver vazia, a tarefa é guardada no *backlog*, conforme linhas 20 e 21. Senão, a função `fazInsercaoECaculaParametros(tarefa, conjunto de posições de inserção num conjunto de rotas, rotas parciais, estrutura Sucessores)` calcula o menor instante de início e fim possível para a tarefa inserida, e o novo conjunto de rotas após inserção da nova tarefa é acatado como solução parcial corrente. Em seguida, a função `propagaAtrasos(tarefa inserida, conjunto de rotas)` calcula o novo instante de início de todas as tarefas afetadas pela inserção, para todas as rotas impactadas. O Pseudocódigo 16 apresenta o método que tenta fazer a inserção de uma tarefa *job* em determinadas posições de um modo. Após inicialização do ES'_{job} com o limitante superior de sua janela, na linha 3, é calculada a menor data de chegada que cada um dos rebocadores do modo conseguiria atender, partindo da tarefa na posição anterior à posição em que a tarefa *job* seria inserida, entre as linhas 4 e 9. Toda vez que uma data de chegada s'^k_{job} é calculada, ela é comparada com o atual valor de ES'_{job} e, se for maior, o ES'_{job} é atualizado (linha 9).

Pseudocódigo 16 – Algoritmo da função que verifica se uma tarefa pode ser inserida simultaneamente em determinadas posições de um conjunto de rotas

```

18 Algoritmo tentaInsercao(listaCombinacoes[comb], job, Sucessores, rotas):
19    $M_{job,g} \leftarrow$  conjunto de rebocadores especificado em listaCombinacoes[comb]
20    $ES'_{job} \leftarrow a_{job}^H$ 
21   Para todo  $k \in M_{job,g}$  faça:
22     posInsr  $\leftarrow$  posição de inserção na  $rota_k$  especificada em listaCombinacoes[comb]
23      $prec'^k_{job} \leftarrow$  tarefa na posição (posInsr-1) da  $rota_k$ 
24     Se  $prec'^k_{job} = O_k^+$  então  $s'^k_{job} = r^k + t^k_{prec'^k_{job}, job}$ 
25     Senão  $s'^k_{job} = tConclusao_{prec'^k_{job}} + t^k_{prec'^k_{job}, job}$ 
26     Se  $s'^k_{job} > ES'_{job}$  então  $ES'_{job} \leftarrow s'^k_{job}$ 
27     Se  $ES'_{job} \geq b_{job}^H$  então retorna inviável
28     Senão, faça:
29        $Sucessores' = atualizaSucessores(Sucessores, listaCombinacoes[comb], job)$ 
30        $J^{Suc} = listaSucessoresDiretosEIndiretos(job, Sucessores')$ 
31       Para todo  $j \in J^{Suc}$  faça:
32         Se  $j = job$  então retorna inviável
33     Retorna viável
34 Fim

```

Ao fim do loop que verifica todos os instantes de chegada das embarcações do modo, verifica-se o atendimento da janela de tempo da tarefa inserida e, se a mesma tiver sido violada, a função retorna a informação “inviável” (linha 10). Caso contrário, nas linhas 12 e 13 a estrutura de $Sucessores'$ é atualizada e todos os sucessores diretos e indiretos da tarefa recém inserida são guardados no conjunto J^{Suc} . Então, caso a tarefa inserida esteja entre as tarefas deste conjunto, considera-se a inserção inviável. Por fim, caso

o método não tenha sido interrompido nas linhas 10 ou 15, a inserção é considerada viável.

O funcionamento do método que faz a inserção é similar ao que tenta inserção, com a diferença que no primeiro as condições de viabilidade não são cheçadas e o valor $detConclusao'_{job}$ também é calculado, simplesmente somando o valor de $ES'_{job} + p_{job}$. Além disto, todos os sucessores diretos de job , ou seja, aqueles contidos em $sucessores'(job)$ têm seu ES' recalculados, considerando o $tConclusao'_{job}$, o tempo de deslocamento o seu ES antes da inserção. Sempre que houver necessidade de mudar o valor de ES'_j de uma tarefa j , o atraso é propagado para os seus sucessores diretos e indiretos, conforme linha 24 do pseudocódigo abaixo, que gera mais execuções do *loop* entre as linhas 14 e 24. Só depois da propagação de todos os atrasos, as estruturas *Sucessores*, *ES*, *tConclusao* recebem os valores de *Sucessores'*, *ES'*, *tConclusao'*, finalizando o método que faz a inserção.

Três versões randomizadas desta heurística de duas fases foram testadas: com aleatoriedade apenas na escolha da próxima tarefa a encaixar na solução; com aleatoriedade apenas na escolha das posições de inserção (solução parcial gerada após inclusão da tarefa da vez nas rotas já programadas); e com aleatoriedade nas duas decisões. Constatou-se que a versão com aleatoriedade em ambas escolhas se sobressaiu em todas as instâncias, por isso as duas outras versões foram descartadas. Por outro lado, a versão sem aleatoriedade apresentou bom desempenho em alguns casos, o que levou à sua adoção na versão final do algoritmo.

A.2. Heurística construtiva de uma fase, com aleatoriedade em todas as execuções

Descartada a heurística de duas fases, partiu-se para construção de um método único, que considera a possibilidade de inserção em posições intermediárias das rotas desde a primeira interação. Também foram testadas outras regras de ordenação e versões que trabalhavam com apenas uma regra ou com todas elas, escolhendo aleatoriamente a regra no início do processo construtivo.

Esta versão da heurística é apresentada a seguir, no Pseudocódigo 17. Inicialmente, as tarefas são ordenadas segundo um critério pré-definido e as rotas das embarcações são inicializadas nos respectivos nós de início. Então, uma-a-uma, tenta-se adicionar as tarefas às rotas da seguinte forma: para cada modo compatível com a tarefa, verifica-se a viabilidade e o custo incremental de adicionar a tarefa nas últimas posições das

rotas do modo, sem desrespeitar sua janela de atendimento. Caso haja mais de um modo em que a adição da tarefa é viável, utiliza-se um critério associado ao custo para selecionar apenas um modo; por outro lado, caso não seja possível adicionar a tarefa a nenhum dos modos compatíveis, ela é guardada em um *backlog*, J^B .

Pseudocódigo 17—Algoritmo da heurística construtiva proposta após a qualificação, apresentada no L Simpósio Brasileiro de Pesquisa Operacional (CAVALCANTI; MENDES, 2018)

```

1  Algoritmo HC_PRV-JTSMM( $J, M, nExp$ ):
2  Enquanto nenhum critério de parada for atingido:
3      Ordena tarefas de  $J$  segundo uma regra de ordenação e guarda em  $S$ 
4  Enquanto houver tarefa em  $S$ :
5      Escolhe uma tarefa de  $S$  segundo critério de seleção, removendo-a de  $S$ 
6      Para todo modo compatível com a tarefa:
7          Para toda configuração deste modo:
8              Avalia o custo de adicionar a tarefa no fim das rotas de configuracao
9              Se viável, guarda modo e custo em listaDeCandidatos
10         Se houver elemento em listaDeCandidatos:
11             Ordena listaDeCandidatos não-crescentemente em custo
12             Escolhe modo* da listaDeCandidatos segundo critério de seleção
13             Adiciona tarefa na última posição das rotas de modo*
14         Senão adiciona a tarefa no conjunto  $S^B$ 
15     Para toda tarefa de  $S^B$ :
16         Para toda configuração compatível com a tarefa:
17             Para todo modo desta configuração:
18                 Gera todas as possibilidades viáveis de inserir a tarefa no modo
19                 Para toda possibilidade de inserção da tarefa no modo:
20                     Avalia o custo de inserir a tarefa nas posições especificadas
21                     Guarda possibilidade e custo em listaDeCandidatos2
22         Se houver elemento em listaDeCandidatos2:
23             Ordena listaDeCandidatos2 não-crescentemente em custo
24             Escolhe elemento* da listaDeCandidatos segundo critério de seleção
25             Insere tarefa nas posições especificadas pelo elemento*
26         Remove tarefa de  $S^B$ 
27 FIM

```

Após este processo, caso haja tarefa no *backlog*, tenta-se inserir uma-a-uma esta(s) tarefa(s) em posições intermediárias das rotas das embarcações, considerando todas as configurações e modos compatíveis com a tarefa e as possíveis posições de inserção das rotas. Por exemplo, se uma tarefa puder ser inserida nas posições 2 e 3 da rota da embarcação 1 e nas posições 1 e 2 da embarcação 4, deve-se avaliar a inserção simultânea nas posições: 2 da rota 1 e 1 da rota 4; 2 da rota 1 e 2 da rota 4; 3 da rota 1 e 1 da rota 4; e 3 da rota 1 e 2 da rota 4. Novamente, caso haja mais de uma possibilidade para adicionar a tarefa nas rotas, seleciona-se a de menor custo incremental. Após tentar inserir no cronograma todas as tarefas de J^B , se a solução gerada estiver completa, o processo é terminado; caso contrário, o processo todo é reexecutado, reiniciado a partir da linha 3 do pseudocódigo. Os critérios de parada da HC_PRV-JTSMM são o número máximo de recomeços ou a construção de uma solução completa, o que ocorrer antes.