UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUANA VILLWOCK SILVA

UMA SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM LIMITE DE CALADO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO

2017

LUANA VILLWOCK SILVA

UMA SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO VIAJANTE COM LIMITE DE CALADO

Trabalho de Conclusão de Curso como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação, do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marco Antônio de Castro Barbosa

PATO BRANCO

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 10 horas do dia 24 de novembro de 2017, na sala V007, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Marco Antonio de Castro Barbosa (orientador), Dalcimar Casanova e Luciene de Oliveira Marin para avaliar o trabalho de conclusão de curso com o título **Uma solução para o problema do caixeiro viajante com limite de calado**, da aluna **Luana Villwock Silva**, matrícula 01270710, do curso de Engenharia de Computação. Após a apresentação a candidata foi arguida pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Marco Antonio de Castro Barbosa
Orientador (UTFPR)

_____
Dalcimar Casanova
(UTFPR)

_____
Luciene de Oliveira Marin
(UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

VILLWOCK SILVA, Luana. Uma solução para o problema do caixeiro viajante com limite de calado. 2017. 61f. Monografia (Trabalho de Conclusão de Curso) - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

Problemas de otimização cada vez mais complexos têm surgido nos últimos anos nas mais diversas áreas do conhecimento e isso exige que profissionais da área de computação procurem soluções para os mesmos. Entretanto, determinados problemas são tão complexos que não existem algoritmos que possam resolvê-los otimamente em tempo polinomial. Assim, o estudo nessa área tem crescido e se tornado bastante relevante. Um problema clássico em programação matemática é o problema do caixeiro viajante. Esse problema e suas variações possuem várias aplicações práticas. Uma das suas variações é o problema abordado nesse trabalho, que é o problema do caixeiro viajante com limite de calado. Existem várias classes de métodos para a resolução desses problemas, um delas são as meta-heurísticas, que nem sempre encontram uma solução ótima, mas permitem encontrar soluções satisfatórias em tempo polinomial, ao contrário dos outros métodos. Com o objetivo de encontrar melhores resultados surgiram várias meta-heurísticas, neste trabalho foi abordado a meta-heurística colônia de formigas para resolver o problema citado anteriormente. Esse algoritmo consiste do uso de formigas artificiais que percorrem um grafo deixando feromônio, uma substância química que evapora com o tempo nas arestas. O objetivo dessas ações é que essas formigas convirjam para o mesmo caminho e esse caminho ser o melhor possível. Com isso obteve-se resultados bons, apesar de nem todos terem chegado na solução ótima.

Palavras-chave: Problema do Caixeiro Viajante, Problema do Caixeiro Viajante com Limite de Calado, Colônia de formigas, Meta-heurísticas

ABSTRACT

VILLWOCK SILVA, Luana. A solution for the traveling salesman problem with draft limits. 2017. 61f. Monografia (Trabalho de Conclusão de Curso) - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2017.

In the past few year complex optimization problems have arisen in diverse areas of knowledge requiring that computer professionals search for a solution to them. However, certain problems are so complex that are no algorithms tha can solve them optimally in polynomial time. Thus, the study in this area has grown and become quite relevant. A classic problem in mathematical programming is the traveling salesman problem. This problem and its variations have several applications. One of its varionts is the problem addressed in this work, which is the problem of the traveling salesman with draft limits. There are several types of methods to solve these problems, one of them are meta-heuristics, which do not always find an optimal solution, but allow finding satisfactory solutions in polynomial time, unlike other methods. In this paper, Ant Colony was approached to solve the previously mentioned problem. This algoritm consists of the use of artificial ants that walk through a graph leaving pheromone, a chemical substance that evaporates with time on the edges of the graph. The purpose of these actions is for these ants to converge on the same path and this path be the best possible. This has led to good results, although not always to the best solution.

Keywords: Traveling Salesman Problem, Traveling Salesman Problem with Draft Limits, Ant Colony, meta-heuristics

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

ACO	*Ant Colony Otimization*
AS	*Ant System*
BCP	*Branch-cut-and-price*
GRASP	*Greedy Randomized Adaptive Search Procedure*
PCV	Problema do Caixeiro Viajante
PCVA	Problema do Caixeiro Viajante Assimétrico
PCVLC	Problema do Caixeiro Viajante com Limite de Calado
SA	*Simulated Annealing*
SAT	Satisfazibilidade
VNS	*Variable Neighborhood Search*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

Cotidianamente, as mais diversas áreas do conhecimento, deparam-se com problemas extremamente complexos, muitos destes problemas são classificados como problemas de otimização combinatória, os quais, dentre os tipos de problemas, são os mais difíceis de serem solucionados (GOLDBARG; LUNA, 2004).

A quantidade de problemas da matemática, agronomia, biologia, logística, economia, computação e várias outras áreas do conhecimento vêm crescendo quase diariamente. Grande parte desses problemas são considerados NP-Completos. Esse termo foi introduzido no início dos anos 70, e simboliza o abismo da intratabilidade inerente que os projetistas de algoritmos encontram ao tentar solucionar problemas maiores e mais complexos (GAREY; JOHNSON, 1990).

Em Teoria da Computação existe uma diversidade de classes que definem os mais diversos tipos de problemas (SIPSER, 2007). A classe P é a classe de problemas que podem ser resolvidos deterministicamente em tempo polinomial. A classe de problemas NP é o conjunto de problemas que podem ser verificados não deterministicamente em tempo polinomial, ou seja, pode-se verificar rapidamente se uma dada instância de entrada para um dado problema é ou não uma solução viável para o problema. Um problema é dito NP-Completo se esse problema pertence à classe NP e todo problema NP possa ser transformado a esse problema em tempo polinomial. Os problemas desta classe são conhecidos como “os problemas mais difíceis da classe NP”. De acordo com o teorema de Cook-Levin, se um problema NP-Completo for resolvido em tempo polinomial, então todos os problemas NP podem ser resolvidos, mas se qualquer problema NP for intratável então todos os problemas NP-Completos também o serão (SIPSER, 2007). Ainda há a classe de problemas NP-Difícil, a qual define-se por qualquer problema de decisão membro ou não da classe NP, que para se transformar em um problema NP-Completo terá a propriedade que não poderá ser resolvido em tempo polinomial a menos que $P = NP$ (GAREY; JOHNSON, 1990).

Portanto a classe NP-Difícil pode ser vista como a classe em que se deseja encontrar uma solução para o problema enquanto que a classe NP-Completo deseja-

se saber se há uma solução viável para o problema sem, necessariamente, precisar encontrá-la.

Há um grande interesse em encontrar algoritmos mais eficientes para resolver problemas de otimização combinatória, várias técnicas têm sido utilizadas ao longo dos anos. Algoritmos exatos garantem encontrar para cada instância de tamanho finito de um problema uma solução ótima em tempo limitado. No entanto, essa solução não é efetiva para todos os problemas, como, por exemplo, a classe NP-Difícil que não possui solução em tempo polinomial. Dessa forma, utilizam-se métodos aproximados, nos quais se sacrifica a garantia de encontrar a melhor solução para se obter boas respostas em uma quantidade de tempo significativamente reduzido, pois a melhor solução levaria muito tempo (BLUM; ROLI, 2003).

Também há um tipo de método que orquestra uma interação de procedimentos de buscas locais com estratégias de alto nível, para criar um processo capaz de escapar de mínimos locais e realizar uma busca robusta no espaço de soluções de um problema, esse método é chamado de meta-heurística (GLOVER; KOCHENBERGER, 2003).

1.2 PROBLEMA

O Problema do Caixeiro Viajante (PCV) é um dos problemas NP-Completo mais tradicionais e conhecidos em programação matemática. Esse problema consiste em determinar o menor caminho que um Caixeiro Viajante, que saindo de sua cidade de origem deve percorrer, ao passar por uma série de cidades distintas apenas uma vez e retornando à cidade de origem.

O PCV possui um conjunto grande de variações de requisitos, como, por exemplo, o PCV Simétrico, o PCV Assimétrico, o PCV Generalizado, o PCV com Janelas de Tempo, o PCV com Gargalo, o Problema do Carteiro Chinês entre outros (GOLDBARG; LUNA, 2004).

Esse problema é muito estudado, pois possui inúmeras aplicações práticas como, por exemplo, roteamento de veículos, distribuição de energia e manipulação de itens em estoque. Vários outros problemas podem ser reduzidos ao PCV, de forma que se encontrada uma boa solução para ele, outros problemas também terão uma boa solução, conforme assegura o Teorema de Cook-Levin.

Em Rakke et al. (2012) foi apresentada uma nova variação do PCV no contexto marítimo de carga de navios, denominada como o Problema do Caixeiro Viajante com Limite de Calado (PCVLC). Calado é a expressão do transporte marítimo que indica a profundidade em que o navio está submerso, tecnicamente é a distância da lâmina de água até a quilha do navio. A Figura 1 ilustra o como o navio se comporta quando carregado ou não em relação ao limite de calado.

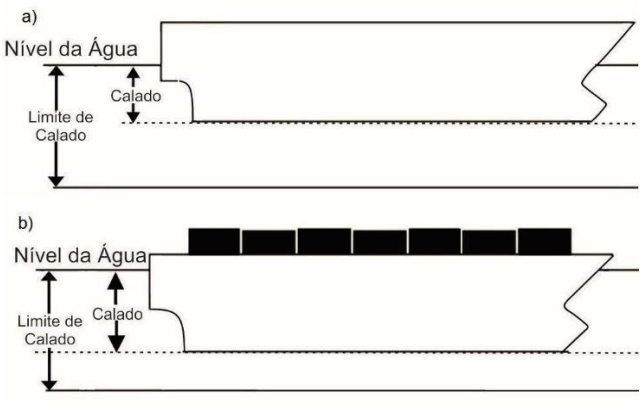

Figura 1 - Representação do calado e limite de calado. a) Navio descarregado b) Navio carregado
Fonte: Adaptado de Rakke et al. (2012, p. 2).

O PCVLC consiste em um navio saindo de um depósito completamente carregado, fazendo entregas em um número determinado de portos uma vez e voltando ao ponto inicial. Entretanto cada um dos portos possui um limite de calado, que varia de acordo com a carga do navio, de forma que o navio deve visitar alguns portos antes de outros.

Formalmente, o PCVLC é definido com um grafo direcionado $G = (V, A)$, onde $V = \{1, \ldots, n\}$ é o conjunto de vértices ou os portos a serem visitados e $= \{(i, j) \mid i, j \in V \; e \; i \neq j\}$ é o conjunto de arestas ou o conjunto de conexões entre os portos. O depósito tem uma demanda $d_1 = 0$ e os portos $i \in V \setminus \{1\}$ tem uma demanda inteira e não negativa d_i (RAKKE et al., 2012).

Como o PCVLC é um problema NP-Difícil, sua solução ótima através de algoritmos exatos não é sempre possível em tempo polinomial. De forma que neste trabalho será abordada a meta-heurística colônia de formigas e algoritmo guloso, com o objetivo de encontrar soluções ótimas ou próximas delas em tempo polinomial.

1.3 OBJETIVOS

Objetivo Geral

Implementar uma meta-heurística para solucionar o Problema do Caixeiro Viajante com Limite de Calado.

Objetivos Específicos

- Implementar uma solução utilizando o Método guloso;
- Implementar uma solução utilizando a meta-heurística colônia de formigas;
- Realizar experimentos computacionais para validação.

1.4 JUSTIFICATIVA

O PCVLC combina a complexidade de um problema de roteamento com problema de agendamento, o limite de calado define implicitamente restrições de precedência ao longo do percurso a ser realizado. Sendo o mesmo uma variação do PCV, ele é considerado NP-Difícil (BATTARRA et al., 2014).

Esse problema foi introduzido por Rakke et al. (2012), sendo propostas duas formulações para o problema e o uso algoritmo exato *branch-and-cut* para solucionar ambas. Posteriormente Battarra et al. (2014) propuseram mais três formulações matemáticas para resolver as instâncias de forma ótima com o uso de algoritmos exatos.

Todosijevic et al. (2014) propuseram duas variações da meta-heurística busca em vizinhança variável (VNS), nas quais foram encontrados resultados bons e com menor tempo computacional aos resultados já obtidos e, também, propuseram novas instâncias maiores obtendo resultados bons.

Machado e Neves (2015) propuseram a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) para resolução do problema. Recentemente em Morais et al. (2017) foi utilizada a meta-heurística *Simulated Annealing* (SA) como uma solução alternativa obtendo resultados ótimos para a maior parte das instâncias.

E neste trabalho decidiu-se por abordar a solução do algoritmo colônia de formigas, que até o presente momento não é do conhecimento dos autores, pelas buscas realizadas em publicações na área, que o mesmo já tenha sido abordado para o PCVLC. Optou-se por esse algoritmo, pois o mesmo imitia o comportamento de formigas as quais possuem um comportamento randômico e é baseado em uma solução probabilística.

1.5 ESTRUTURA DO TRABALHO

Esse trabalho está divido da seguinte forma:

- Seção 2: Referencial teórico, no qual encontra-se a teoria que é a base deste trabalho.
- Seção 3: Desenvolvimento da solução proposta, na qual é explicado detalhadamente o que foi feito para resolver o problema proposto.
- Seção 4: Resultados, na qual mostra os resultados das soluções propostas comparadas com o estado da arte.

2 REFERENCIAL TEÓRICO

2.1 TIPOS DE PROBLEMAS

Define-se problema como uma questão qualquer que necessita de uma resposta, geralmente o mesmo possui vários parâmetros ou variáveis, aos quais os valores não são especificados. Um problema é especificado por: (1) uma descrição de todos os seus parâmetros, e (2) uma declaração de quais propriedades da resposta, ou solução, são necessárias satisfazer (GAREY; JOHNSON, 1990).

Os problemas podem ser classificados como decisão, localização e otimização. Um problema de decisão caracteriza-se como aquele que dada uma entrada qualquer a resposta apresenta apenas duas possibilidades, sendo elas “sim” ou “não”. Pode-se dizer também que a solução pode ser representada por apenas um bit 0 ou 1 (GOODRICH; TAMASSIA, 2004). Um problema de localização se caracteriza por procurar um elemento que satisfaça determinada condição da solução. E um problema de otimização é aquele que além de exibir uma solução se preocupa com certa qualidade ótima da solução (TOSCANI; VELOSO, 2009).

2.2 CLASSES DE PROBLEMAS

Classe P

“A classe de complexidade P é o conjunto de todos os problemas de decisão (ou linguagens) L que podem ser aceitos em tempo polinomial no pior caso.” (GOODRICH; TAMASSIA, 2004, p.586). Formalmente:

$P = \{L$ | para uma máquina de Turing determinística de uma única fita M existe $L = L_m\}$, onde $L_m = \{x \in \sum^* | M$ aceita $x\}$ (GAREY; JOHNSON, 1990).

Toscani e Veloso (2009) definem a classe P como a classe dos problemas deterministicamente polinomiais, a qual consiste em problemas de decisão, que numa codificação razoável, podem ser resolvidos em tempo polinomial.

Pode-se dizer que a classe P é uma classe matematicamente robusta, pois a mesma é “invariante para todos os modelos de computação polinomialmente equivalentes à máquina de Turing determinística de uma única fita” (SIPSER, 2007,

p.273). Vale ressaltar que essa classe corresponde a maior parte da classe de problemas que de fato podem ser resolvidos por um computador, o que é extremamente relevante do ponto de vista prático (SIPSER, 2007).

Classe NP

"A classe de complexidade NP é o conjunto de todos os problemas de decisão (ou linguagens) L que podem ser aceitos não-deterministicamente em tempo polinomial." (GOODRICH; TAMASSIA, 2004, p.586). Formalmente:

$NP = \{L$ | para uma máquina de Turing não determinística de uma única fita M existe $L_m = L\}$ (GAREY; JOHNSON, 1990).

Existe uma forma determinística de se definir um problema NP, a qual diz que se um problema pode ser verificado através de uma máquina de Turing determinística, o mesmo pertence à classe NP, ou seja, dada uma entrada é possível verificar se a mesma corresponde a uma solução do problema em tempo polinomial (GAREY; JOHNSON, 1990; GOODRICH; TAMASSIA, 2004).

Relação de P versus NP

Quando se fala de classes P e NP não se pode deixar de citar um dos maiores problemas não resolvidos da ciência da computação e matemática contemporânea, que é se essas classes são iguais. Como já foi observado anteriormente a classe P pertence à classe NP, que diz que todo os problemas de decisão solucionados por um computador determinístico também são solucionados por um computador não determinístico (GAREY; JOHNSON, 1990; GOODRICH; TAMASSIA, 2004).

Existem várias razões para acreditar que P não é igual a NP, e é o que a maioria dos pesquisadores acreditam, algoritmos não determinísticos de tempo polinomial aparentam ser mais poderosos que os determinísticos, no entanto não foi possível provar cientificamente essa teoria, de forma que quando se trata dessas classes existem duas possibilidades, como ilustra a Figura 2.

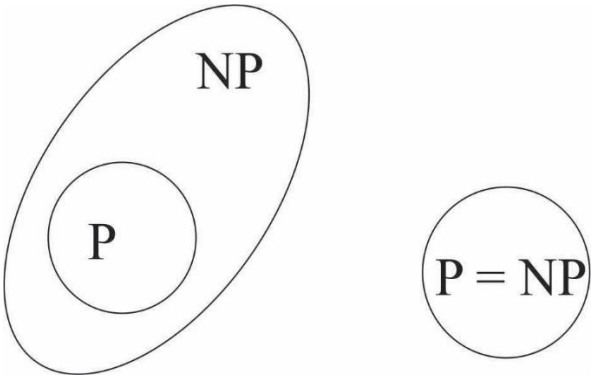

Figura 2 - Possibilidades da questão P versus NP
Fonte: Sipser (2007).

Classe NP-Completo

Se P difere de NP, então a distinção entre P e $NP - P$ é de extrema importância, pois dessa forma todos os problemas P poderiam ser resolvidos em tempo polinomial, enquanto os $NP - P$ seriam intratáveis. Até que seja provado que P é diferente de NP, não há motivos para mostrar que um problema qualquer pertence à classe NP-P. E por essa razão a teoria de NP-Completude foca em provar resultados através da hipótese fraca que "se P difere de NP, então o problema pertence a NP-P" (GAREY; JOHNSON, 1990).

Formalmente uma linguagem L é definida como NP-Completa, se ela pertence a NP, e para todas as outras linguagens L' que pertencerem a NP, L' será equivalente a L. Informalmente, um problema de decisão $\prod$ é NP-Completo se $\prod$ pertence a NP, e para todos os outros problemas de decisão $\prod'$ pertencentes a NP, $\prod'$ equivale a $\prod$ (GAREY; JOHNSON, 1990).

Como já citado anteriormente essa classe de problemas é considerada como "os problemas mais difíceis em NP", pois se existir uma solução em tempo polinomial para um único problema em NP-Completo, todos os problemas em NP poderão ser resolvidos, mas se um único problema em NP for intratável, então todos os problemas em NP-Completo também o serão (GAREY; JOHNSON, 1990).

A partir desse conceito, na prática torna-se mais simples identificar ou não a existência de algoritmo polinomial para resolver determinado problema, pois se for identificado que um problema é NP-Completo existe um forte evidencia da não-polinomialidade do mesmo (SIPSER, 2007).

Considerando as definições apresentadas e assumindo que P é diferente de NP, pode-se ter uma visão diferente do universo NP, como ilustra a Figura 3.

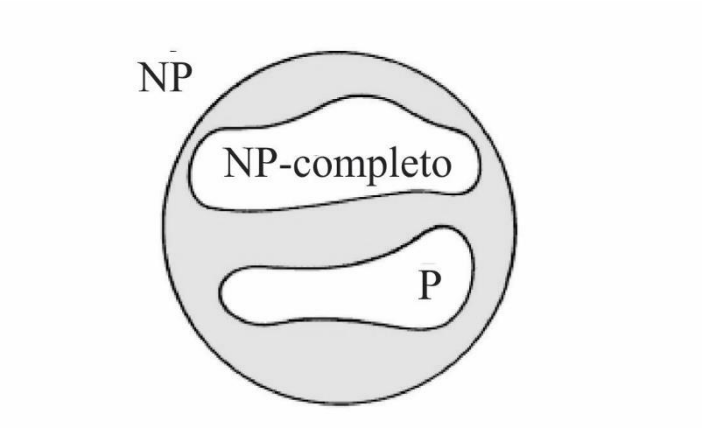

Figura 3 - Universo NP
Fonte: Garey e Johnson (1990).

Para provar que determinado problema $\prod$ é NP-Completo, é necessário mostrar que:

1. $\prod$ pertence a NP, e
2. Algum problema $\prod'$ NP-Completo conhecido é redutível a $\prod$ (GAREY; JOHNSON, 1990).

Mas antes de usar essa definição é preciso conhecer o primeiro problema NP-Completo o qual é provido pelo teorema de Cook-Levin.

Teorema de Cook-Levin

O primeiro problema NP-Completo é um problema de decisão de lógica booleana, chamado de problema de satisfazibilidade (SAT). Para entender esse problema é necessário definir uma fórmula booleana, a qual é uma expressão que envolve variáveis booleanas - geralmente representadas por 0 ou 1, ou ainda VERDADEIRO ou FALSO -, e operações - as quais são E, OU e NÃO (representadas pelos símbolos $\wedge$, $\vee$ e $\neg$. O problema da satisfazibilidade é testar se uma fórmula booleana é satisfazível. Uma fórmula booleana é satisfazível se alguma atribuição de 0s e 1s às variáveis fazem com que a fórmula tenha valor 1 (SIPSER, 2007). Alguns exemplos de variáveis booleanas podem ser vistos a seguir em (1), tal como uma expressão booleana em (2).

$$\begin{array}{lll} 0 \wedge 0 = 0 & 0 \vee 0 = 0 & \bar{0} = 1 \\ 0 \wedge 1 = 0 & 0 \vee 1 = 1 & \bar{1} = 0 \end{array} \qquad (1)$$

$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$

$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

$$\vartheta = (\bar{x} \wedge y) \vee (x \wedge \bar{z}) \tag{2}$$

A partir disso pode-se enunciar o teorema de Cook-Levin, o qual relaciona a complexidade de SAT com todos os outros problemas pertencentes à classe NP (SIPSER, 2007). Garey e Johnson (1990) determinam o Teorema de Cook-Levin como: Satisfazibilidade é NP-Completo.

Classe NP-Difícil

Outra classe importante a ser tratada nesse trabalho é a classe de problemas NP-Difícil. Essa classe caracteriza-se por qualquer problema pertencente ou não a classe NP e o mesmo não pode ser reduzido a um problema NP-Completo em tempo polinomial, a menos que P seja igual a NP (GAREY; JOHNSON, 1990).

Uma forma de provar que um problema $\prod$ é NP-Difícil é mostrar que ao tentar provar que o mesmo pertence à classe NP-Completo é possível apenas satisfazer a condição (2), que diz que todo problema em NP se reduz a $\prod$, mas não é possível satisfazer a condição (1) que $\prod$ pertence a NP (HOPCROFT et al., 2002).

Esse tipo de problema é aquele que para encontrar uma solução ótima exige muito tempo, ou seja, o mesmo precisa de tempo exponencial ou maior (HOPCROFT et al., 2002).

2.3 ESTRATÉGIA DE SOLUÇÕES DE PROBLEMAS

Há vários tipos de algoritmos clássicos que visam solucionar os problemas de forma ótima, na verdade existe uma quantidade tão imensa de algoritmos que a questão é porque existem tantos, e a resposta para essa pergunta é que cada vez que um problema muda, o algoritmo que o resolve também deve mudar (MICHALEWICZ; FOGEL, 2000).

A solução de determinado problema depende da sua complexidade, a partir desse ponto é possível determinar se o mesmo será solucionado por um método exato ou aproximado. Métodos exatos obtêm soluções ótimas. Para problemas NP Completos, esse tipo de solução é não polinomial (a não ser que $N = NP$). Métodos aproximados (ou heurísticos) geram soluções de boa qualidade em um tempo razoável para o uso prático, mas não garantem uma solução ótima como os exatos (TALBI, 2009).

Métodos Exatos

Dentro dessa classe de métodos é possível encontrar os seguintes algoritmos clássicos: programação dinâmica, a família *Branch and X* (*branch and bound, branch and cut, branch and price*), programação de restrição e a família A* de algoritmos de busca. Esses métodos enumerativos são vistos como algoritmos de busca em árvore. A busca ocorre no espaço de interesse e o problema é resolvido ao dividi-lo em problemas menores (TALBI, 2009).

Métodos exatos garantem encontrar uma solução ótima para cada instância finita de problemas de otimização combinatória. Mas para problemas NP-Difíceis não existe solução em tempo polinomial (BLUM; ROLI, 2003), de forma que métodos exatos podem ser aplicados em instâncias pequenas de problemas difíceis (TALBI, 2009).

Métodos Aproximados

Esses métodos têm como objetivo encontrar uma solução aproximada, pois nem sempre é possível encontrar uma solução exata para determinado algoritmo, como é o caso de problemas de otimização NP-Difícil. Nem sempre é necessário encontrar uma solução ótima para determinado problema, as vezes uma solução quase ótima pode ser suficiente para um problema prático e é mais fácil de se encontrar e esses métodos são usados nesses casos (SIPSER, 2007).

Em Talbi (2009) um algoritmo aproximado é definido como: um algoritmo aproximado tem um fator de aproximação ϵ se o seu tempo é polinomial e para qualquer instância de entrada, o mesmo gera uma solução ω que

$$\omega \leq \epsilon \cdot s, \qquad se\ \epsilon > 1 \tag{3}$$

$$\epsilon \cdot s \leq \omega, \qquad se\ \epsilon < 1 \tag{4}$$

onde s é a solução ideal global, e o fator ϵ define a garantia de desempenho relativo. O fator ϵ pode ser uma constante ou uma função do tamanho da instância de entrada.

Um algoritmo de aproximação ϵ gera uma garantia de desempenho absoluta ϵ se a seguinte equação é satisfeita:

$$(s - \epsilon) \leq \omega \leq (s + \epsilon) \tag{5}$$

Métodos heurísticos e Meta-heurísticos

Meta-heurísticas, em sua definição original, são métodos de solução que orquestram a interação entre procedimentos de melhoria local e estratégias de alto nível para criar um processo capaz de escapar de ótimos locais e realizando uma pesquisa robusta no espaço da solução (GLOVER; KOCHENBERGER, 2003). Ao contrário dos métodos exatos, meta-heurísticas permitem resolver instâncias de grande porte, fornecendo soluções satisfatórias em tempo razoável, entretanto não há garantia de encontrar as melhores soluções globais (TALBI, 2009).

A palavra heurística vem do grego antigo *heuriskein*, que significa a arte de descobrir novas estratégias (regras) para resolver problemas, o que defini uma heurística com um método para encontrar a solução de um problema de otimização. O sufixo *meta*, também provém do grego, e significa “metodologia de nível superior”, de forma que pode se definir meta-heurísticas como metodologias de alto nível que podem ser usadas como estratégias de orientação na elaboração de heurísticas subjacentes para resolver problemas de otimização específicos (TALBI, 2009).

Esses métodos têm ganhado mais e mais popularidade nos últimos anos. O uso de meta-heurísticas tem provado a eficiência e eficácia das mesmas para a solução de problemas grandes e complexos. A aplicação de meta-heurísticas está em várias áreas, sendo algumas delas: aprendizado de máquinas, mineração de dados em bioinformática, biologia computacional, finanças, processamento de imagens, simulações, problemas de roteamento, entre outras. (TALBI, 2009).

O conceito de solucionar problemas de otimização combinatória através de heurísticas foi introduzido por Polya em 1945. Em 1947 G. Dantzig criou o algoritmo *Simplex,* que pode ser visto como um algoritmo de busca local para problemas de programação linear. J. Edmonds foi o primeiro a apresentar a heurística gulosa na

literatura de otimização combinatória em 1971. As referências originais das seguintes meta-heurísticas são baseadas nessas aplicações para otimização e/ou problemas de aprendizado de maquinas: colônia de formigas, sistemas imunológicos artificiais, colônia de abelhas, algoritmos culturais, algoritmos coevolutivos, estratégia evolutiva de adaptar a matriz covariância, evolução diferencial, algoritmos de estimação de distribuição, programação evolutiva, estratégias de evolução, algoritmos genéticos, algoritmos do grande dilúvio, busca local guiada, programação genética, procedimento de busca adaptativa gulosa e randômica, busca local iterativa, método barulhento, enxame de partículas, *simulated annealing*, método de suavização, busca dispersa, limiar de aceitação, busca tabu e busca em vizinhança variável (TALBI, 2009).

As meta-heurísticas que serão usadas nesse trabalho serão tratadas a seguir:

Algoritmos Gulosos

O método guloso é uma heurística popular e simples de desenvolver, geralmente esses algoritmos são menos complexos quando comparados com algoritmos iterativos. Esse método consiste em iniciar com um conjunto de solução vazio e constrói-se então uma solução atribuindo valores para uma variável de decisão por vez, até que uma solução completa seja gerada. Entretanto essa abordagem nem sempre tende para uma solução ótima (GOODRICH; TAMASSIA, 2004; TALBI, 2009).

Em um problema de otimização, a solução pode ser definida pela presença ou ausência de um conjunto de elementos $E = \{e_1, e_2, \dots, e_n\}$, a função objetivo pode ser definida como $f: 2^E \rightarrow \mathbb{R}$, e o espaço de busca é definido como $F \subset 2^E$. Uma solução parcial s pode ser vista como um subconjunto $\{e_1, e_2, \dots, e_n\}$de elementose_i do conjunto de todos os elementos E. O conjunto que define a solução inicial é vazio. E a cada passo, uma heurística local é usada para selecionar um novo elemento para ser incluso no conjunto. Uma vez que um elemento e_i é selecionado para ser parte da solução, o mesmo nunca será substituído por outro elemento. Não há volta nas decisões tomadas. Tipicamente heurísticas gulosas são algoritmos determinísticos.

O Algoritmo 1 mostra o pseudocódigo de um algoritmo guloso (TALBI, 2009).

Algoritmo 1 – Pseudocódigo do algoritmo guloso

```
s = {}; /*Solução inicial (vazia)*/
enquanto Solução não está completa faça
    e_i = Heurística-Local(E \{e|e ∈ s});
    /*próximo elemento selecionado do conjunto E menos os
    elementos já selecionados */
    se s ∪ e_i ∈ F então
        /*testa a viabilidade da solução*/
        s = s ∪ e_i;
    fim
fim
```

Fonte: Talbi (2009)

Colônia de formigas

Colônia de formigas (ACO, do inglês *Ant Colony Otimazation*) foi introduzido por Dorigo (1992) através do algoritmo Sistema de Formigas (AS, do inglês *Ant System*) o qual foi inspirado no comportamento de formigas reais, e um número variado de algoritmos se desenvolveu posteriormente baseado nessas mesmas ideias (DORIGO; STÜTZLE, 2004). Aqui será descrito brevemente como um algoritmo pertencente à ACO funciona.

A ideia básica em um algoritmo ACO é imitar o comportamento cooperativo de formigas reais para solucionar problemas de otimização. ACO pode ser visto como um sistema multiagente, no qual cada agente é inspirado no comportamento de uma formiga (DORIGO, 1992).

O principal objetivo do comportamento de formigas é que simples formigas usam comportamentos coletivos para realizar tarefas complexas, tais como transportar alimento e encontrar o menor caminho para as fontes de alimento. Uma colônia de formigas é capaz de encontrar o melhor caminho entre dois pontos. A Figura 4 mostra um experimento feito por Goss et al. (1989) com uma colônia de formigas na Argentina, para demonstrar como é o comportamento das mesmas. É importante notar que essas formigas não conseguem enxergar muito bem. A colônia tem acesso à fonte de alimento por dois caminhos. Durante o percurso, as formigas deixam uma trilha química de feromônio no chão, que é uma substância olfativa e volátil, o objetivo dessa trilha é guiar as outras formigas até o ponto desejado. Quanto

maior a quantidade de feromônio em um caminho, maior é a probabilidade de que as formigas irão selecioná-lo (DORIGO; 1992).

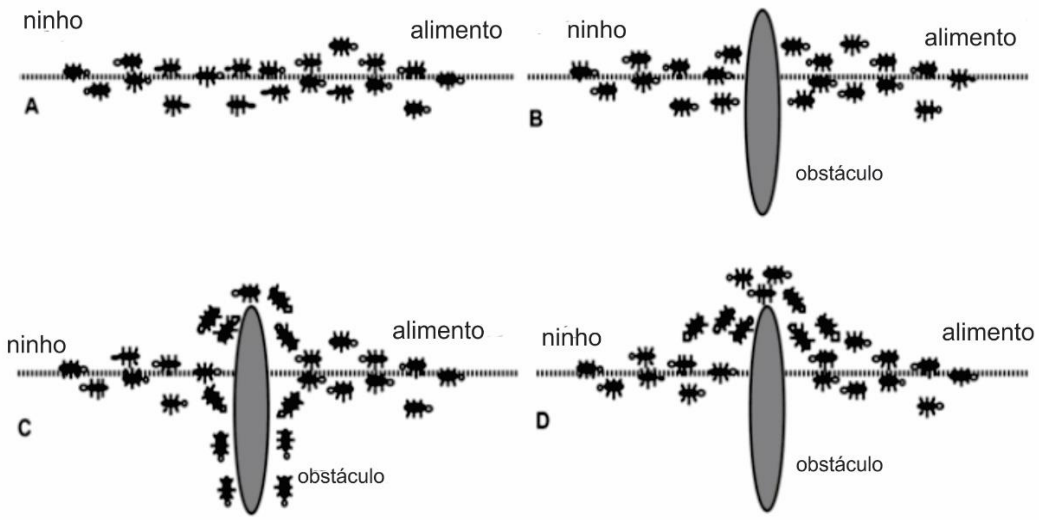

Figura 4 - Colônia de formigas procurando um caminho ideal entre o ninho e o alimento
Fonte: Adaptado de Talbi (2009).

Além disso, a ação do feromônio diminui com o passar do tempo (pela evaporação) e a quantidade deixada por uma formiga depende da quantidade de alimentos (reforço). Como mostrado na Figura 4, quando um obstáculo surge, existe uma probabilidade igual para a escolha de cada um dos lados do mesmo, como a trilha superior é menor e consequentemente requer menos tempo de viagem, o feromônio ficara mais concentrado nesse caminho, e quanto mais formigas passam por ele, maior será o nível de feromônio nele(TALBI, 2009).

Formigas artificiais utilizadas em ACO são procedimentos de construção de solução estocásticos que probabilisticamente constroem uma solução adicionando componentes de solução em soluções parciais levando em conta (i) informações heurísticas na instância do problema a ser resolvido, se disponível, e (ii) trilhas (artificiais) de feromônio, as quais mudam dinamicamente em tempo de execução, em reflexo da experiência adquirida pelos agentes (GLOVER; KOCHENBERGER, 2003).

O componente estocástico de ACO permite que as formigas criem uma grande variedade de diferentes soluções e, portanto, pode explorar um número muito maior de soluções do que heurísticas gulosas. Ao mesmo tempo, o uso de informações heurísticas, as quais são prontamente disponíveis por vários problemas, podem guiar as formigas através da solução mais promissora. Ainda mais importante, a experiência

de busca das formigas pode ser usada para influenciar a construção de soluções em futuras iterações do algoritmo (GLOVER; KOCHENBERGER, 2003).

O pseudocódigo dessa meta-heurística pode ser visto no algoritmo 2.

Algoritmo 2 – Pseudocódigo de um algoritmo ACO

```
Inicializa a trilha de feromônio;
enquanto Critério de finalização não é satisfeito faça
    para cada formiga faça
        Construção da solução usando a trilha de feromônio;
        Atualize a trilha de feromônio;
            Evaporação;
            Reforço;
    fim
fim
Saída: Melhor solução encontrada ou um conjunto de solução.
```

Fonte: Talbi (2009)

2.4 PROBLEMA DO CAIXEIRO-VIAJANTE

Um dos problemas mais clássicos e conhecidos da programação matemática é o problema do caixeiro-viajante. Esse problema consiste em encontrar a menor rota para uma série de cidades, visitando cada uma delas uma vez e retornando ao ponto inicial (GOLDBARG; GOLDBARG, 2012). Formalmente o problema consiste em dado um grafo não-direcionado G, que possui um custo não negativo associado a cada aresta deve se encontrar o circuito hamiltoniano de G com o menor custo. (APPLEGATE et al., 2006).

A importância desse problema é dividida em pelo menos três características:

- Grande aplicação prática (BELLMORE; NEMHAUSER, 1968; BUKARD, 1979; REINELT, 1994; LAPORTE, 1990; GUTIN; PUNNEN, 2007).
- Relação com outros problemas e muitas variantes (LAPORTE et al., 1995; GUTIN; PUNNEN, 2007)
- Grande dificuldade de solução exata (PAPADIMITRIOU; STEIGLITZ,1982; MILLER; PEKNY, 1991; GRÖTSCHEL; HOLLAND, 1991).

> O PCV é NP-Difícil (GAREY; JOHNSON, 1990), sendo um dos problemas de otimização combinatória mais intensamente pesquisados até o presente momento. A maior instância não trivial solucionada de forma exata evoluiu de 318 cidades na década de 1980 (CROWDER; PADBERG,1980) para 7.397 cidades em meados dos anos 1990 (APPLEGATE et al., 1995) e 24.978 cidades em 2004. A melhor marca foi obtida em abril de 2006, na solução de uma instância de 85.900 cidades (APPLEGATE et al., 2006). A maior instância hoje disponível é a "World TSP" com 1.904.711 cidades disponível em http://www.tsp.gatech.edu/world/ (GOLDBARG; GOLDBARG, 2012, p.543).

Existem vários problemas correlatos com o PCV e todos eles possuem aplicações reais, algum deles são o PCV Simétrico, PCV Generalizado, PCV com Backhauls, PCV Múltiplo, PCV com Gargalo, PCV Seletivo, PCV Estocástico, PCV com Coleta, PCV Remoto, problema do Carteiro Chinês, problema do Carteiro ao Vento, entre outros (GOLDBARG; LUNA, 2004).

O PCV e suas variações possuem várias aplicações, entre elas podem se destacar manipulação de itens em estoque (RAMESH, 1981), roteamento de veículos (BODIN et al., 1983) e de navios (FAGERHOLT; CHRISTIANSEN, 2000), programação de transporte entre células de manufatura (KUSIAK, 1985), otimização de perfurações de furos em placas de circuitos impressos (GRÖTSCHEL et al., 1989), trabalhos administrativos (LAPORTE et al., 1995), entre muitas outras (GOLDBARG; LUNA, 2004).

2.5 PROBLEMA DO CAIXEIRO-VIAJANTE COM LIMITE DE CALADO

O problema do caixeiro-viajante com limite de calado é uma variação do problema do caixeiro-viajante no contexto de transporte marítimo que foi introduzido por Rakke et al. (2012). O problema consiste em encontrar o circuito Hamiltoniano de menor custo, levando em consideração a estrutura dos portos pelos quais o navio passará. A estrutura dos portos está relacionada ao limite de calado, que é o ponto mais baixo da quilha de uma embarcação em relação ao à linha da água, e isto está fortemente ligado com a carga do navio. Determinados portos possuem um limite de calado menor, de forma que é necessário que o navio faça algumas entregas antes de passar por esses portos (BATTARRA et al., 2014).

Formalmente o problema é definido como: um grafo direcionado $G = (V, A)$, onde $V = \{1, \dots, n\}$ é o conjunto de portos a serem visitados e $= \{(i,j) | \, i,j \in V \; e \; i \neq j$

é o conjunto de conexões entre os portos, para as quais está associado um custo de rota $c_{ij} > 0, i \neq j$. O vértice 1 se refere ao depósito - onde o navio começa e termina sua rota e os outros portos constituem o conjunto $V \setminus \{1\}$, os quais são visitados apenas uma vez (BATTARRA et al., 2014).

Cada porto $i \in V \setminus \{1\}$ possui uma demanda inteira e não negativa d_i, a qual está associada ao limite de calado $l_i,\ i \in V$ e assume-se, sem perda de generalidade, que os portos são rotulados de forma que $l_i \geq l_{i-1}\ (i = 1, \ldots, n-1)$ (RAKKE et al., 2012). A carga inicial do navio é $Q = \sum_{i \in V} d_i$ e denota-se $d = min_{i \in V}\{d_i\}$, o navio não pode entrar no porto i, se ele tem uma carga maior que l_i ou o casco do navio pode ser aterrado (BATTARRA et al., 2014).

São encontradas na literatura cinco formulações matemáticas para o PCVLC.

Em Rakke et al. (2012) foram propostas duas formulações matemáticas, a primeira delas chama-se GG, e é baseada na formulação de Gavish e Graves para o problema da caixeiro-viajante assimétrico (PCVA) e a segunda chama-se SD que é uma extensão da formulação para o PCVA de Sherali e Driscoll.

A primeira formulação ($F1$), baseada na de Gavish e Graves para o PCVA, faz uso da variável binária x_{ij}, a qual é igual a 1 se e somente se a aresta (i,j) faz parte da solução e da variável y_{ij} a qual denota a carga a bordo do navio na aresta (i,j). O modelo de $F1$ é:

F1 minimiza

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \tag{6}$$

Sujeito a

$$\sum_{i \in V\ i \neq j} x_{ij} = 1, \qquad j \in V, \tag{7}$$

$$\sum_{j \in V\ i \neq j} x_{ij} = 1, \qquad i \in V, \tag{8}$$

$$\sum_{i \in V} y_{ij} - \sum_{i \in V} y_{ji} = d_j, \qquad j \in V\{\setminus 1\}, \tag{9}$$

$$\sum_{j \in V} y_{1j} = \sum_{i \in V} d_i \,, \tag{10}$$

$$\sum_{i \in V} y_{i1} = 0, \tag{11}$$

$$0 \leq y_{ij} \leq l_j x_{ij}, \qquad (i,j) \in A, \tag{12}$$

$$x_{ij} \in \{1,0\}, \qquad (i,j) \in A \tag{13}$$

A expressão (6) minimiza o custo da rota. Equações (7) e (8) garantem que cada porto será visitado apenas uma vez. A equação (9) assegura que a demanda de cada porto será satisfeita e impede a criação de subrotas. As equações (10) e (11) definem que o navio sai do depósito completamente carregado e volta para o mesmo vazio. E a equação (12) define o limite de calado de cada porto (RAKKE et al., 2012).

A segunda formulação ($F2$) estende do modelo de Gavish e Graves através da incorporação de restrições de subrotas de Miller, Tucker e Zemlin. Essas restrições incluem dois conjuntos adicionais de variáveis: u_i e t_{ij} os quais especificam a ordem do porto i e da aresta (i,j) no circuito respectivamente (RAKKE et al., 2012). O modelo de $F2$ é definido pelas equações de (6) a (13) e:

$$\sum_{j \in V \setminus \{i,1\}} t_{ij} + (n-1)x_{i1} = u_i, \qquad i \in V \setminus \{1\}, \tag{14}$$

$$\sum_{j \in V \{i,1\}} t_{ji} + 1 = u_i, \qquad i \in V \setminus \{1\}, \tag{15}$$

$$x_{ij} \leq t_{ij} \leq (n-2)x_{ij}, \qquad i,j \in V \setminus \{1\}, \tag{16}$$

$$u_j + (n-2)x_{ij} + (n-1)\big(1 - x_{ji}\big) \leq t_{ij} + t_{ji} \leq u_j - \big(1 - x_{ji}\big), \quad i,j \in V \setminus \{1\}, \tag{17}$$

$$1 + (1 - x_{1i}) + (n-3)x_{i1} \leq u_i \leq (n-1) - (n-3)x_{1i} - (1 - x_{i1}), \quad i \in V \setminus \{1\}, \tag{18}$$

$$t_{ij} \geq 0, \qquad (i,j) \in A, \tag{19}$$

$$u_i \geq 0, \qquad i \in V. \tag{20}$$

Nessa formulação, as equações de (14) a (18) previnem a criação de subrotas.

Os algoritmos *branch-and-cut* originados das fórmulas $F1$ e $F2$ são capazes de resolver eficientemente instâncias com um número limitado de portos com limite de calado. Entretanto, quando a porcentagem de portos com limite de calado aumenta, o algoritmo necessita de um esforço alto até para instâncias médias (BATTARRA et al., 2014).

Em Battarra et al. (2014) foram propostas outras três formulações, a primeira delas é baseada em variáveis de dois índices, a segunda em variáveis de três índices e a terceira pode ser vista como uma melhoria da decomposição de Dantzig Wolfe da segunda formulação, incluindo um conceito de rotas *ng* e resolvida através de um algoritmo *branch-cut-and-price*.

A terceira formulação ($F3$), proposta em Battarra et al. (2014), é baseada em dois conjuntos de variáveis binárias. A variável x_{ij} assume valor 1 se $(i,j) \in A$ faz parte da solução, 0 caso contrário. A variável y_{ik} assume valor 1 quando o navio entra no porto $i \in V$ carregando $k \in \{d_i, \dots, l_i\}$ unidades de carga, 0 caso contrário. $F3$ é determinada pelas equações de (6) a (8) e:

$$\sum_{i \in V \mid l_i \geq k} y_{ik} \leq 1, \qquad k \in \{d, \dots, Q-d\}, \tag{21}$$

$$\sum_{k=d_i} y_{ik} = 1, \qquad i \in V, \tag{22}$$

$$y_{iQ} = x_{0i}, \qquad i \in V \mid l_i = Q, \tag{23}$$

$$y_{id_i} = x_{i0}, \qquad i \in V, \tag{24}$$

$$x_{ij} + y_{ik} \leq 1 + y_{i,k-d_i}, \qquad (i,j) \in A, \qquad k \in \{d_j + d_i, \dots, \min(l_i, l_j + d_i)\} \tag{25}$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A \tag{26}$$

$$y_{ik} \in \{0,1\}, \qquad i \in V, k \in \{d_i, \dots, l_i\} \tag{27}$$

Nessa formulação, a equação (21) impõe que o navio visite no máximo um porto por valor de carga, entretanto essa equação não garante necessariamente a solução ótima, pois as equações (7), (8), (23) e (24) garantem que o navio faça o circuito

Hamiltoniano no qual a carga diminui monotonicamente. A equação (22) indica que cada um dos valores de carga deve ser designado a um porto. O início e o fim da rota estão ligados ao depósito imposto pelas equações (23) e (24). A equação (25) conecta as variáveis $\mathrm{x_{ij}}$ e $\mathrm{y_{ik}}$, se o navio passou pela aresta (i,j), então $x_{ij} = 1$ e i e j estão em posições consecutivas do circuito. As equações (22) a (25) asseguram que o fluxo diminua monotonicamente ao longo do circuito e, portanto, evite subrotas (BATTARRA et al., 2014).

A quarta formulação ($F4$) consiste na variável binária de três índices z_{ij}^{k}, a qual assume valor 1 se o navio passou pela aresta $(i,j) \in A$ carregando k unidades de carga (incluindo a demanda do porto j). Denotando $K_{ij} = \min\{l_j, l_i - d_i\}$, a formulação $F4$ é (BATTARRA et al., 2014):

$$(F4)\ \min \sum_{(i,j)\in A} \sum_{k=d_j}^{K_{ij}} c_{ij} z_{ij}^{k}, \tag{28}$$

$$\sum_{i \in V \mid i \neq j} \sum_{k=d_j}^{K_{ij}} z_{ij}^{k} = 1, \qquad i \in V, \tag{29}$$

$$\sum_{\substack{j \in V \mid j \neq i \\ l_j \geq k + d_j}} z_{ji}^{k} - \sum_{\substack{j \in V \mid j \neq i \\ l_j \geq k - d_i,\ d_j \geq k - d_i}} z_{ij}^{k-d_i} = 0, \qquad i \in V, \qquad k \in \{d_i, \ldots, l_i\}, \tag{30}$$

$$z_{i0}^{k} = 0, \qquad i \in V, \qquad k \in \{1, \ldots, Q\}, \tag{31}$$

$$z_{0j}^{k} = 0, \qquad j \in V \mid l_j = Q, \qquad k < Q, \qquad k \in \{d_i, \ldots, l_i\}, \tag{32}$$

$$z_{ij}^{k} \in \{0,1\}, \qquad (i,j) \in A, \qquad k \in \{d_j, \ldots, K_{ij}\}. \tag{33}$$

As funções (28) e (29) minimizam o custo da rota e garantem que o porto será visitado apenas uma vez respectivamente. A equação (30) preserva a carga do navio e as equações (31) e (32) garantem que o navio saia do depósito cheio e retorne vazio (BATTARRA et al., 2014).

A quinta formulação (F5) origina-se de uma interpretação alternativa do espaço de solução de $F2$. Nessa formulação define-se P como o conjunto de todos os caminhos possíveis, para cada $p \in P$ é introduzido uma variável λ_p indicando se p foi usado ou não na solução (BATTARRA et al., 2014). Em Battarra et al. (2014)

define-se q_{ij}^{kp} como um coeficiente binário que indica se a variável z_{ij}^{k} está associada ao caminho p, de forma que $F5$ é definida por:

$$\min \sum_{p \in P} \left(\sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} \sum_{k=d_j}^{K_{ij}} q_{ij}^{kp} c_{ij} \right) \lambda_p, \tag{34}$$

$$\sum_{p \in P} \left(\sum_{\substack{j \in V \\ j \neq i}} \sum_{k=d_j}^{K_{ij}} q_{ij}^{kp} \right) \lambda_p = 1, \qquad i \in V, \tag{35}$$

$$\lambda_p \in \{0,1\}, \qquad p \in P, \tag{36}$$

Uma restrição adicional r sobre a variável z_{ij}^{k} tendo o formato:

$$\sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} \sum_{k=d_j}^{l_i} \propto_{ij}^{kr} z_{ij}^{k} \geq b_r \tag{37}$$

Pode ser inclusa em $F5$ como:

$$\sum_{p \in P} \left(\sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} \sum_{k=d_j}^{l_i} \propto_{ij}^{kr} \right) \lambda_p \geq b_r \tag{38}$$

Nessa formulação, utiliza-se uma estratégia provada por Baldacci et al. (2011), na qual, para cada consumidor $i \in V$, seja $N_i \subseteq V$ um conjunto *ng* de i, a vizinhança do mesmo. Esse conjunto $|N_i|$ pode mostrar os portos mais próximos de i, e incluir o mesmo. As cardinalidades de todos os conjuntos *ng* são as mesmas, denotando T. Um caminho *ng* é um caminho como definido em P, onde cada ciclo começa e termina no vértice i deve conter pelo menos um vértice j tal que $i \notin N_j$. Isso pode ser interpretado como se o navio “esquecesse” de visitar o porto i quando passasse pelo porto j (BATTARRA et al., 2014).

Na formulação $F5$, se define um conjunto *ng* diferente N_i^k para cada carga k do navio após visitar cada porto i. Os vizinhos do porto i com carga k são então selecionados como T portos próximos que podem ser visitados imediatamente após

visitar i sem violar o limite de calado e preservar a mesma complexidade computacional. Essa modificação reforça o relaxamento, já que conjuntos *ng* originais gastam parte da memória para armazenar portos que violariam os limites de calado (BATTARRA et al., 2014).

De forma que a formulação $F5$,é definida pelas equações de (34) a (38), onde P é definida como o conjunto de caminhos *ng* e desigualdades traduzidas para as variáveis λ_p como mostrado nas equações 34 e 35 (BATTARRA et al., 2014).

Os algoritmos exatos apresentados em Battarra et al., (2014) resolveram as instâncias de forma ótima. Entretanto esses métodos de solução requerem um poder computacional grande, de forma que Todosijevic et al. (2014) propuseram duas soluções para o problema, utilizando-se de variações da meta-heurística busca em vizinhança variável (VNS).

VNS é uma meta-heurística proposta por Pierre Hansen e Nenad Mladenovic, que é baseada em um simples princípio: mudança sistemática de vizinhança durante a busca (MLADENOVIC; HANSEN, 1997).

Nas soluções propostas por Todosijevic et al. (2014), utilizou-se formas diferentes de escolha da próxima vizinhança, e ambas soluções propostas conseguiram resolver as instâncias propostas por Rakke et al. (2012) de forma ótima e com redução de tempo em relação aos resultados obtidos até então, e também foram criadas novas instâncias maiores, as quais obtiveram bons resultados.

Em Machado e Neves (2015) foi utilizada a meta-heurística *Greedy Randomized Adaptive Search Prodcedure* (GRASP) para o problema. Esse método de solução é iterativo, e cada iteração consiste basicamente em duas fases: construção e busca local (FEO; RESENDE, 1995).

A fase de construção constrói uma solução possível, então através da busca local, que é um procedimento que percorre um caminho em um grafo não-orientado, investiga-se a vizinhança da mesma até que um mínimo local seja encontrado. A melhor dentre todas as soluções é apresentada como o resultado do algoritmo (FEO; RESENDE, 1995).

Utilizando-se desse método e comparando os resultados com o que já havia sido encontrado na literatura, conseguiu-se encontrar para todas as instâncias resultados ótimos com tempo computacional baixo (MACHADO, NEVES,2015).

Recentemente foi proposta uma solução utilizando *Simulated Anneling* (SA) por Morais et al. (2017). SA é um algoritmo de busca local capaz de escapar do ótimo

local. Ele foi nomeado dessa forma em analogia com um processo termodinâmico, no qual um sólido cristalino é aquecido e então deixado resfriar lentamente até que o mesmo adquira uma forma mais cristalina possível, dessa forma o cristal está livre de defeitos, esse processo é chamado de recozimento, em inglês *annealing*, e esse algoritmo imita esse processo (GLOVER; KOCHENBERGER, 2003).

Os resultados obtidos em Morais et al. (2017) utilizando a meta-heurística SA mostram ser uma alternativa competitiva que apresenta bons resultados, os quais para a maior parte das instâncias foram encontrados resultados ótimos.

Neste trabalho será explorado a meta-heurística colônia de formigas e o algoritmo guloso com fim de comparação entre o que já foi obtido na literatura.

3 SOLUÇÕES PROPOSTAS PARA O PCVLC

Para tentar resolver o problema do PCVLC foi escolhido desenvolver um algoritmo colônia de formigas, mas para aumentar os critérios de comparação escolheu-se um algoritmo de desenvolvimento simples que é o método guloso.

3.1 MÉTODO GULOSO

O método guloso foi implementado com base na teoria abordada no capitulo 2, seção 2.5, adicionando as restrições do problema, que é o limite de calado. O algoritmo começa visitando os nós de todos os portos vizinhos ao depósito até encontrar aquele com a menor distância, repete esse processo para o nó encontrado até o circuito hamiltoniano ser finalizado. Restrições do problema são verificadas durante esse processo e são elas: se porto não foi visitado e se ele possui limite de calado menor ou igual a carga do navio. Caso essas condições sejam válidas verifica-se se a distância entre os portos é a menor dos portos não visitados, caso contrário avalia-se o próximo vizinho. O algoritmo 3 mostra o pseudocódigo do que foi implementado.

Algoritmo 3 – Pseudocódigo do algoritmo Guloso para o PCVLC

```
Inicializa as variáveis
enquanto não visitou todos os portos
        enquanto não fez comparações com todos os portos vizinhos
                se porto vizinho não foi visitado faça
                        se porto vizinho possui calado ≤ carga do navio faça
                                se a distância entre os portos é a menor
                                até o momento faça
                                        Porto visitado = Porto vizinho;
                                Fim
                        Fim
                Fim
        Atualiza distância percorrida;
        Fim
Fim
Saída: Distância percorrida
```

Fonte: Autoria Própria

A Figura 5 faz uma simulação do algoritmo para uma instância de 6 portos. Na parte “a” da Figura, nenhum porto foi visitado e o calado é igual a 5. A parte “b”, mostra que o algoritmo encontrou a menor distância, que é o porto 2, mas o limite de calado é menor que o calado atual do navio, assim, esse porto ainda não pode ser visitado. Para visitá-lo é necessário que o navio realize algumas entregas. Assim, nessa primeira interação é escolhido o porto 3, que é o que possui menor distância e limite de calado maior ou igual a carga. Nas partes “d” a “h” o algoritmo realiza o mesmo procedimento até fechar o circuito hamiltoniano.

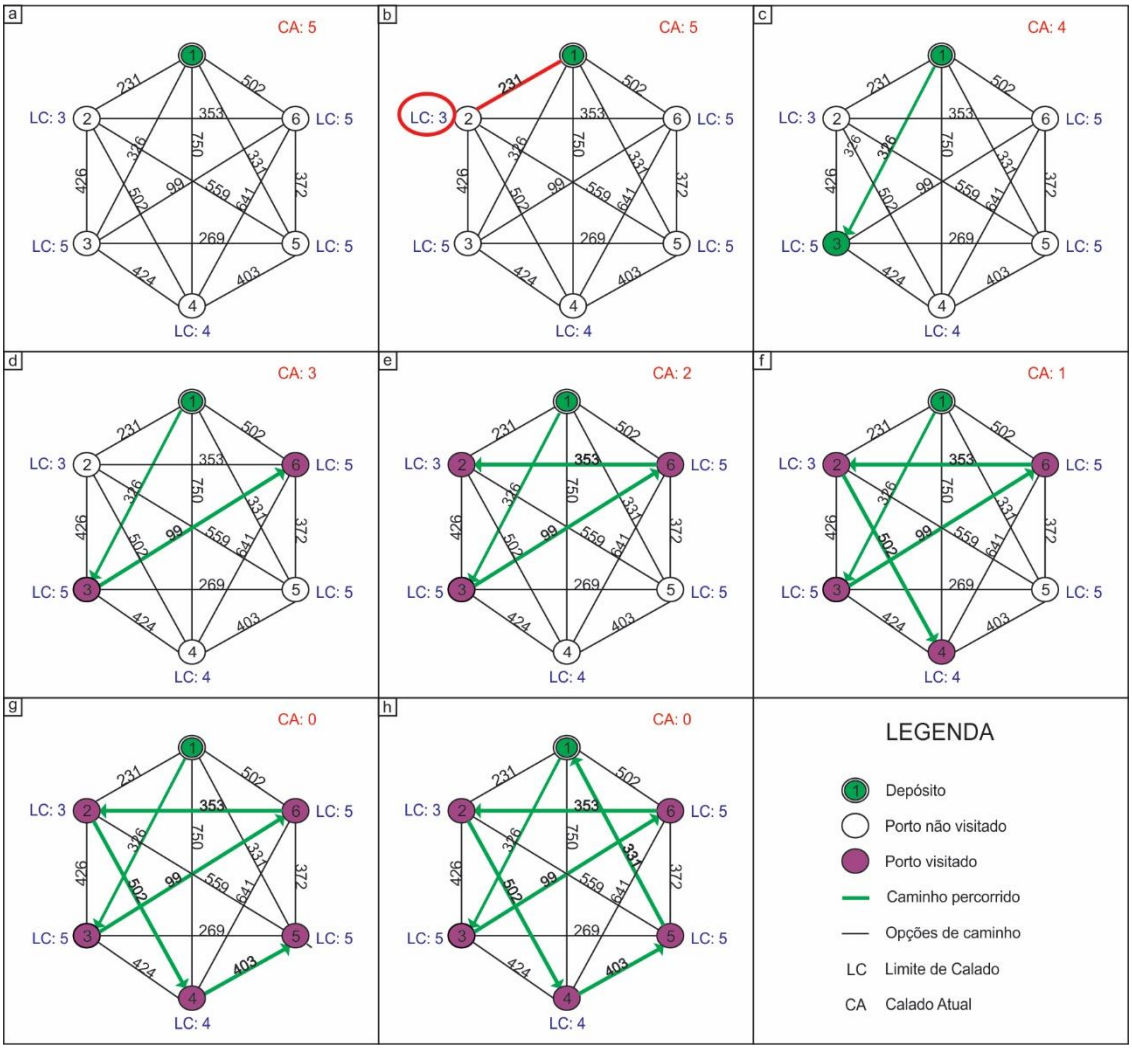

Figura 5 - Simulação do algoritmo guloso para uma instância pequena
Fonte: Autoria própria

3.2 COLÔNIA DE FORMIGAS

Segundo Dorigo e Stützle (2004) um algoritmo ACO pode ser imaginado como uma interação de três procedimentos: construção da solução pelas formigas, atualização de feromônio e ações *Daemon*.

A fase de construção de uma solução pelas formigas deve fazer com que a colônia de formigas visite os nós adjacentes ao nó atual de forma simultânea e assíncrona, movendo-se através dos vizinhos do grafo de construção do problema. As formigas devem mover-se aplicando uma política de decisão local estocástica que faz uso de trilhas de feromônios e informações heurísticas. De forma que as formigas criam de forma incremental soluções para o problema de otimização. Uma vez que uma formiga construiu ou está construindo uma solução, ela avalia a solução que será usada para a fase de evaporação do feromônio para decidir o quanto de feromônio irá depositar nas arestas (DORIGO; STÜTZLE, 2004).

O segundo procedimento para o desenvolvimento de um algoritmo colônia de formigas é a atualização de feromônio. Esse é o processo em que as trilhas de feromônio são modificadas. O valor das trilhas pode ou não aumentar, de acordo com o feromônio que as formigas depositam nas arestas que usam, ou diminuir, de acordo com a evaporação. De um ponto de vista prático, o depósito de feromônio aumenta a probabilidade de uma aresta pela qual muitas ou pelo menos uma formiga passou se tornar uma boa solução e ser usada novamente por várias formigas. Diferentemente da evaporação do feromônio que implementa uma forma de “esquecer”, a qual evita convergência rápida do algoritmo ao redor de uma solução não boa, isso faz com novas áreas sejam exploradas (DORIGO; STÜTZLE, 2004).

E para finalizar, tem se as ações *daemon,* que são procedimentos implementados para centralizar ações que não podem ser executadas por apenas uma formiga. Um exemplo de ação *daemon* é coletar informações globais que podem ser usadas para decidir se é vantajoso depositar feromônio adicional em determinado caminho. Esse procedimento é opcional, o algoritmo não depende dele para funcionar, mas ele pode melhorar resultados (DORIGO; STÜTZLE, 2004).

Existem vários estudos de algoritmos ACO para o PCV, neles a fase de construção é onde rotas são construídas aplicando os seguintes procedimentos construtivos para cada formiga: (i) escolhe, de acordo com algum critério, a cidade inicial a qual a formiga está posicionada; (ii) usa-se valores de feromônio e heurísticos

para construir a rota adicionando iterativamente cidades que ainda não foram visitadas, até todas as cidades serem visitadas; e (iii) volta-se para a cidade inicial. Depois que todas as formigas completaram sua rota, elas depositam feromônio nas rotas que seguiram (DORIGO; STÜTZLE, 2004).

Em todos os algoritmos ACO disponíveis para o PVC, as trilhas de feromônio estão associados com as arestas e então τ_{ij} refere-se à desejabilidade de se visitar a cidade j após a cidade i . E a informação heurística escolhida é $\eta_{ij} = \frac{1}{d_{ij}}$, isto é, a desejabilidade heurística de ir da cidade i diretamente para a cidade j é inversamente proporcional à distância entre as duas cidades (DORIGO; STÜTZLE, 2004).

Em AS, m formigas (artificiais) simultaneamente constroem uma solução para o PCV. Inicialmente as formigas são colocadas em cidades randomicamente escolhidas e a cada passo da construção, uma formiga k aplica uma regra de ação probabilística, chamada de regra randômica proporcional, para decidir qual cidade visitará depois (DORIGO; STÜTZLE, 2004). Em particular, a probabilidade de cada formiga k, que está na cidade i , escolher ir para a cidade j é:

$$p_{ij}^{k} = \frac{\left[\tau_{ij}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l \in \mathcal{N}_i^k}\left[\tau_{ij}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}, \qquad se\ j \in \mathcal{N}_i^k, \tag{39}$$

onde, α e β são parâmetros que determinam a influência relativa da trilha de feromônio e da informação heurística respectivamente. E $\mathcal{N}_i^k$ é a vizinhança plausível da formiga k quando começa na cidade i, isto é, o conjunto de cidades que a formiga k ainda não visitou. Por essa regra probabilística, a probabilidade de se escolher uma aresta (i, j) em particular cresce com o valor da trilha de feromônio τ_{ij} e da informação heurística η_{ij} (DORIGO; STÜTZLE, 2004).

Os dois parâmetros, α e β, são parâmetros ajustáveis que influenciam na escolha do próximo nó. Se $\alpha = 0$, as cidades mais próximas possuem mais chances de ser escolhidas, se $\beta = 0$, apenas o feromônio influencia na escolha do próximo nó, o que geralmente gera resultados ruins. Em geral quando $\alpha > 1$ o algoritmo entra em estado de estagnação, onde todas as formigas constroem e seguem o mesmo caminho, o que em geral não é bom (DORIGO; STÜTZLE, 2004).

A fase de construção PCVLC implementada nesse trabalho baseia-se no que já foi aplicado ao PCV, e descrito anteriormente, levando em consideração a restrição adicional, que é o limite de calado. Foram utilizados os mesmo conceitos de regra

probabilística, já citados, mas $\mathcal{N}_i^k$, que é a vizinhança plausível, foi definida não apenas pelos nós que já foram visitados, mas também por aqueles que possuíam limite de calado maior ou igual a carga do navio.

E para definir o que seria o conjunto $\mathcal{N}_i^k$ foi utilizado um subgrafo G' gerado baseado na carga do navio e dos nós que ainda não foram visitados, os quais estão armazenados na memória da formiga. Segundo Szwarcfiter (1986, p. 42) "um subgrafo $G_2\ (V_2, A_2)$ de um grafo $G_1\ (V_1, A_1)$ é um grafo tal que $V_2 \subset V_1$ e $A_2 \subset A_1$". O subgrafo G' gerado para o PCVLC consiste nos nós que possuem limite de calado maior ou igual à carga do navio e ainda não foram visitados, para evitar que a formiga repita um mesmo caminho.

Para encontrar os valores de α e β foram testadas variações dos mesmos dentro dos intervalos sugeridos por Dorigo e Stützle (2004), que $\alpha \leq 1$ e $2 \leq \beta \leq 5$ até encontrar um valor que se adequasse a respectiva instância, que foi definido como $\alpha = 0{,}4$ e $\beta = 3$. Esse valor foi determinado baseado no tempo que o algoritmo levou para convergir, o número de interações para o mesmo e a qualidade da solução. A variação para a instância *gr48_50_1* pode ser visto na Tabela 2, que se encontra na seção de resultados.

Outro ponto que é diferente do PCV é a construção de soluções, no qual geralmente insere formigas em cidades diferenciadas para iniciar a trajetória. Entretanto, essa prática não foi adotada neste trabalho, todas as formigas saem do mesmo ponto, que é o depósito, essa decisão foi tomada pois o PCVLC não é simétrico da mesma forma que o PCV, o que significa que ir de uma cidade i para j é diferente de ir de j para i, implicando que se uma formiga encontrar uma solução ótima saindo de um porto qualquer, que não seja o depósito, reconstruir essa solução posteriormente pode não ser possível, o que é possível no PCV.

Para o PCV em AS, depois que todas as formigas construíram suas rotas, a trilha de feromônio é atualizada. Isto é feito da seguinte maneira: (i) evapora-se o feromônio uniformemente de todas as arestas; e (ii) adiciona feromônio nas arestas em que as formigas visitaram (DORIGO; STÜTZLE, 2004). A evaporação de feromônio é feita através da seguinte equação:

$$\tau_{ij} = (1 - \sigma)\tau_{ij}, \qquad (i, j)\ \epsilon\ V, \tag{40}$$

onde $0 < \sigma \leq 1$ é taxa de evaporação. O parâmetro σ é usado para evitar acumulação ilimitada de feromônio e "esquecer" decisões ruins de caminhos que as

formigas possam ter tido. De fato, quando uma aresta não é escolhida pelas formigas, o valor de feromônio associado a essa aresta decresce exponencialmente ao número de iterações (DORIGO; STÜTZLE, 2004). Depois da evaporação as formigas depositam feromônio em todas as arestas que elas cruzaram de acordo com a seguinte equação:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \qquad (i,j)\ \epsilon\ V, \tag{41}$$

onde $\Delta\tau_{ij}^{k}$ é a quantidade de feromônio que uma formiga k deposita nas arestas que a ela visitou e é definido como segue:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{1}{C^{k}}, & se\ a\ aresta\ (i,j)\ pertence\ a\ T^{k}; \\ 0, & caso\ contrário. \end{cases} \tag{42}$$

onde C^{k} é a distância da rota T^{k} construída pela k -enésima formiga, a qual é composta pela soma das distâncias entre as arestas que a formiga percorreu. Pela equação (42) a rota de uma formiga é melhor quanto mais feromônio aquela rota recebe (DORIGO; STÜTZLE, 2004).

O ACO nem sempre garante desempenho bom quando comparado com outras meta-heurísticas. De forma que várias pesquisas tentaram melhorar esse algoritmo. Algumas dessas soluções implicam adicionar um valor maior nas trilhas da melhor solução encontrada desde o início do algoritmo, que é chamado de Sistema de Formiga Elitista, e essa alteração pode ser vista como uma ação *daemon*. Também existe a solução que o feromônio está associado a um *rank*, que decresce de acordo com as iterações, chamado de sistema de formigas baseado em *ranks*. Outro exemplo de melhoria é definir limites máximos e mínimos de fenômeno e essa solução é chamada de Sistema de Formigas MAX e MIN (DORIGO; STÜTZLE, 2004).

Para a solução do PCVLC foi utilizada a evaporação de feromônio usada em AS, no qual o foi definido $\sigma = 0{,}1$, que é o sugerido por Dorigo e Stützle (2004), pois permite que a evaporação não seja tão rápida permitindo que as formigas explorem vários caminhos.

E a ação *daemon* que foi utilizada é do Sistema de Formiga Elitista, sendo que o algoritmo reforça o melhor valor encontrado em todo o algoritmo. Quando é feita essa busca pela melhor solução até o momento, também é possível tirar algumas conclusões, que não afetam o resultado diretamente, mas ajudam a entender os

resultados obtidos. Um exemplo é que em uma das interações do algoritmo as formigas nunca escolheram o melhor caminho, de forma que é impossível que as formigas convirjam para esse valor e, assim, o algoritmo pode não apresentar o melhor resultado (DORIGO; STÜTZLE, 2004).

Baseado na teoria descrita anteriormente foi desenvolvido o pseudocódigo demonstrado no algoritmo 4.

Algoritmo 4 – Colônia de formigas para o PCVLC

```
Inicializa variáveis
Enquanto ((Iterações < Numero de portos * 10) ou
        (Convergência <= 70% dos portos com mesmo resultado))
        Enquanto todas as formigas não completarem um percurso
                Enquanto todos os portos não forem visitados
                        Escolha um porto a visitar;
                        Atualize as variáveis;
                Fim
        Fim
        Procure o melhor caminho local entre os feitos por todas as
        formigas;
        Se melhor caminho local é melhor que global:
                Atualiza melhor caminho global;
        Fim
        Realiza a evaporação do feromônio;
        Atualiza o a trilha de feromônio;
        Atualiza o Feromônio do melhor caminho global;
Fim
```

Fonte: Autoria Própria

Pode-se notar que nesse trabalho escolheu-se dois critérios de parada do algoritmo um valor alto de iterações, para garantir que as formigas pudessem explorar vários caminhos diferentes antes de convergir, em busca do melhor caminho. E se 70% dos caminhos das formigas fossem iguais, esse valor foi definido baseado nas iterações das formigas, pois notou-se que após essa quantidade de caminhos iguais, o restante dos caminhos tendia a convergir para esse caminho, e ao colocar essa restrição evita que o algoritmo realize muitas iterações, consequentemente diminui o tempo de execução do mesmo.

A escolha do porto em que a formiga irá visitar é descrita mais detalhamento através do algoritmo 5.

Algoritmo 5 – Escolha do porto para o ACO
Gera subgrafo com portos em que calado atual ≤ carga do navio e porto ainda não visitado; Sob o subgrafo calcula se a probabilidade do porto ser escolhido; Proporcionalmente a probabilidade aloca os portos entre valores de 0-100; Realiza o sorteio aleatório de um valor entre 0-100; Retorna o porto que possuía o valor associado ao resultado do sorteio;
Fonte: Autoria Própria

4 RESULTADOS

Para a obtenção de resultados dos algoritmos propostos nesse trabalho, algoritmo guloso e colônia de formigas, ambos foram executados nas instâncias propostas por Rakke et al. (2012), que são uma adaptação de instâncias clássicas do PCV disponibilizadas na biblioteca TSPLIB propostas por Reinelt (1991).

As instâncias variam de 14 a 48 portos e cada uma delas possui uma matriz, a qual se refere às distâncias entre cada porto, e vetores representando a demanda e o limite de calado dos portos. As instâncias podem ser identificadas pelo formato *nomeXX_XX_XX,* onde nome refere-se a quem as estudou, o primeiro número ao número de portos que a mesma possui, o segundo se refere a porcentagem de portos com limite de calado diferente do máximo, que é a quantidade de portos menos 1, e os valores dessas porcentagens são 10%, 25% ou 50%, e o último número se refere ao número da instância.

As Tabelas de 3 a 9 e gráficos das figuras de 6 a 12, mostram os resultados obtidos comparados com o estado da arte, entretanto antes de analisá-las é preciso salientar alguns pontos.

Primeiramente sabe-se que o algoritmo guloso sempre gera o mesmo resultado, assim, ele foi executado apenas uma vez. Já o algoritmo colônia de formigas a cada execução pode gerar um valor diferente dada a sua natureza randômica e probabilística. Um exemplo disso está na Tabela 1 que mostra que a instância *gr21_10_8* foi executada várias vezes.

Tabela 1 - Execuções de ACO com a instância *gr21_10_8*

Nº da execução	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
resultado	2962	2974	2962	2962	2964	3080	2962	2962	3065	2964

Os resultados gerados para as Tabelas de 3 a 9, foram obtidos da seguinte maneira: para cada instância, o algoritmo ACO foi executado 15 vezes. Dessas 15 execuções escolheu-se o resultado mais próximo do resultado ótimo, que significa a menor distância encontrada dessas execuções. Isto também mostra que o resultado varia de execução para execução e, assim, o algoritmo ACO pode gerar tantos resultados ótimos como ruins.

Ainda para obter os resultados de ACO foi necessário, primeiramente, encontrar valores de α e β, e esses valores, como já dito anteriormente, foram obtidos através da execução do algoritmo para a instância *gr48_50_1*, e analisado tanto o resultado como a quantidade de execuções e o tempo para convergência. A Tabela 2 apresenta os dados analisados.

Tabela 2 - Escolha dos parâmetros α e β

α	β	Iterações	Resultado	Tempo de convergência (s)
1	2	72	6796	0,825
0,9	2	73	6488	1,231
0,8	2	73	6152	1,209
0,7	2	86	6274	1,414
0,6	2	98	6263	1,607
0,5	2	129	6464	2,097
0,4	2	117	6436	1,914
0,3	2	185	6385	3,071
0,2	2	226	6148	3,669
0,1	2	468	6120	7,579
1	3	63	6306	0,646
0,9	3	61	6265	1,007
0,8	3	69	6182	1,22
0,7	3	67	6268	1,101
0,6	3	81	6262	1,284
0,5	3	101	6294	1,563
0,4	**3**	**124**	**6105**	**2,032**
0,3	3	200	6105	2,669
0,2	3	234	6157	4,809
0,1	3	418	6109	7,341
1	4	53	6262	0,624
0,9	4	62	6305	1,215
0,8	4	58	6297	0,978
0,7	4	67	6302	1,124
0,6	4	77	6268	1,433
0,5	4	101	6265	1,641
0,4	4	102	6109	1,68
0,3	4	187	6180	3,058
0,2	4	196	6184	3,401
0,1	4	481	6105	7,771
1	5	51	6305	0,589
0,9	5	74	6292	1,1

0,8	5	68	6227	1,118
0,7	5	73	6337	1,223
0,6	5	98	6293	1637
0,5	5	77	6254	1,281
0,4	5	113	6254	1,829
0,3	5	146	6184	2,065
0,2	5	203	6292	3,325
0,1	5	398	6109	6,389

Então foram definidos valores de $\alpha = 0{,}4$ e $\beta = 3$, pois dentro desses valores o resultado da instância se mostrou bom, o número de iterações não foi baixo, e isso permite que a formiga explore mais caminhos, e o tempo de convergência do algoritmo foi aceitável.

Segue então as tabelas de resultados de cada instância, sendo que os resultados dos algoritmos desenvolvidos nesse trabalho são comparados com os resultados do algoritmo *Branch cut and price* (BCP) proposto por Rakke et al. (2012), o qual é um algoritmo exato de forma que sempre chega no resultado ótimo, e o algoritmo GRASP proposto por Machado e Neves (2015).

Também é importante ressaltar que os resultados dos algoritmos do estado da arte e os desenvolvidos nesse trabalho foram executados em máquinas diferentes, sendo o desse trabalho bastante inferior ao estado da arte, de forma que a comparação de tempo em relação à execução do algoritmo não é diretamente comparável por não estarem em condições de igualdade. Para os resultados BCP foi utilizada uma máquina Intel® E5472 Xeon CPU, 2x3.0 GHz e 16GB de RAM, para os do GRASP uma máquina Intel® Core™ i7-3610QM, 2.30 GHZ e 12 GB de RAM, e para o Guloso e ACO uma máquina Intel® CoreTM i5-4200M, 2,5GHz e 8GB de RAM.

Tabela 3 - Resultados da instância *burma14*

Instância	BCP		GRASP		Guloso		ACO	
	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)
burma14_10_1	3416	0,38	3416	0,01	3814	1,236	3416	0,101
burma14_10_2	3323	0,72	3323	0,01	4048	0,096	3323	0,05
burma14_10_3	3323	0,41	3323	0,01	4820	0,067	3323	0,046
burma14_10_4	3751	0,66	3751	0,01	4048	0,018	3751	0,045
burma14_10_5	3323	0,38	3323	0,01	4863	0,111	3323	0,047
burma14_10_6	3323	0,65	3323	0,00	4048	0,366	3323	0,045
burma14_10_7	3323	0,47	3323	0,00	4048	0,015	3323	0,045
burma14_10_8	3346	0,37	3346	0,00	4291	0,016	3346	0,042

burma14_10_9	3416	0,52	3416	0,00	4048	0,062	3416	0,094
burma14_10_10	3323	0,66	3323	0,00	4048	0,022	3346	0,12
burma14_25_1	4036	0,34	4036	0,01	4217	0,008	4036	0,036
burma14_25_2	3465	0,37	3465	0,01	4841	0,012	3465	0,055
burma14_25_3	3336	0,31	3336	0,00	4144	0,057	3336	0,106
burma14_25_4	3696	0,36	3696	0,01	3839	0,076	3818	0,031
burma14_25_5	3346	0,45	3346	0,01	4048	0,022	3346	0,057
burma14_25_6	3610	0,32	3610	0,00	5086	0,021	3610	0,069
burma14_25_7	3346	0,37	3346	0,00	4291	0,12	3382	0,051
burma14_25_8	3371	0,45	3371	0,00	4214	0,031	3371	1,594
burma14_25_9	3834	0,38	3834	0,00	4296	0,044	3837	0,04
burma14_25_10	3928	0,35	3928	0,00	4080	0,009	3928	0,044
burma14_50_1	4412	0,32	4412	0,00	4687	0,01	4412	0,028
burma14_50_2	3748	0,33	3748	0,01	4250	0,039	3748	0,048
burma14_50_3	3870	0,33	3870	0,00	4644	0,066	3870	0,035
burma14_50_4	3323	0,31	3323	0,00	4107	0,026	3323	0,058
burma14_50_5	3524	0,43	3524	0,00	4164	0,033	3524	0,037
burma14_50_6	3846	0,32	3846	0,00	4426	0,014	3846	0,028
burma14_50_7	3408	0,37	3408	0,00	4040	0,02	3408	0,049
burma14_50_8	3506	0,38	3506	0,00	3988	0,011	3506	0,07
burma14_50_9	4519	0,34	4519	0,00	5466	0,012	4683	0,034
burma14_50_10	4467	0,32	4467	0,00	4716	0,029	4538	0,045

Tabela 4 - Resultados da instância *ulysses16*

Instância	**BCP**		**GRASP**		**Guloso**		**ACO**	
	Sol.	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**
ulysses16_10_1	6859	33,88	6859	0,02	9600	0,036	6859	0,042
ulysses16_10_2	6859	46,83	6859	0,02	9988	0,085	6900	0,082
ulysses16_10_3	6859	72,6	6859	0,02	9988	0,009	6900	0,085
ulysses16_10_4	6859	33,04	6859	0,02	9876	2,331	6909	0,072
ulysses16_10_5	6859	60,21	6859	0,02	9998	0,053	6875	0,1
ulysses16_10_6	6951	32,58	6951	0,02	9954	0,027	7037	0,081
ulysses16_10_7	6859	62,46	6859	0,02	9988	0,012	6859	0,114
ulysses16_10_8	6859	40,66	6859	0,02	10008	0,048	6889	0,069
ulysses16_10_9	6859	75,11	6859	0,02	9876	0,02	6890	0,078
ulysses16_10_10	6859	48,59	6859	0,02	9988	0,025	6859	0,072
ulysses16_25_1	6890	49,57	6890	0,02	8173	0,015	6890	0,097
ulysses16_25_2	6859	3,98	6859	0,02	10041	0,06	6859	0,072
ulysses16_25_3	6859	21,95	6859	0,02	9929	0,014	6875	0,071
ulysses16_25_4	7401	3,52	7401	0,02	9550	0,015	7401	0,059
ulysses16_25_5	7671	11,1	7671	0,02	10021	0,015	7671	0,115
ulysses16_25_6	7029	10,16	7029	0,02	9704	0,064	7029	0,08
ulysses16_25_7	7446	9,81	7446	0,02	9988	0,021	7446	0,085
ulysses16_25_8	6859	43,05	6859	0,02	10068	0,023	6859	0,094

ulysses16_25_9	6859	11,32	6859	0,02	9876	0,011	6909	0,103
ulysses16_25_10	7781	21,31	7781	0,02	9984	0,011	7820	0,089
ulysses16_50_1	7264	2,09	7264	0,02	10356	0,054	7264	0,059
ulysses16_50_2	7715	2,6	7715	0,02	10107	0,117	7725	0,06
ulysses16_50_3	9612	1,2	9612	0,02	10690	0,03	9618	0,049
ulysses16_50_4	7313	1,26	7313	0,02	10466	0,07	7313	0,065
ulysses16_50_5	6909	9,93	6909	0,02	9876	1,342	6909	0,062
ulysses16_50_6	7301	3,66	7301	0,02	10992	0,029	7301	0,065
ulysses16_50_7	8118	3,06	8118	0,02	11295	0,011	8197	0,052
ulysses16_50_8	7065	2,4	7065	0,02	10040	0,023	7065	0,057
ulysses16_50_9	6900	11,81	6900	0,02	8490	0,016	6951	0,077
ulysses16_50_10	7706	7,97	7706	0,02	10143	0,03	7706	0,108

Tabela 5 - Resultados da instância *gr17*

Instância	BCP		GRASP		Guloso		ACO	
	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)
gr17_10_1	2153	21,86	2153	0,02	2873	0,027	2221	0,1
gr17_10_2	2165	28,69	2165	0,02	2390	0,025	2165	0,109
gr17_10_3	2085	34,6	2085	0,02	2189	0,022	2085	0,093
gr17_10_4	2590	18,36	2590	0,02	2873	0,03	2661	0,112
gr17_10_5	2085	23,95	2085	0,02	2187	0,011	2085	0,116
gr17_10_6	2085	24,9	2085	0,02	2187	0,01	2085	0,072
gr17_10_7	2085	26,33	2085	0,02	2512	0,028	2090	0,106
gr17_10_8	2085	2,71	2085	0,02	2212	0,073	2088	0,081
gr17_10_9	2085	25,06	2085	0,02	2187	0,054	2085	0,081
gr17_10_10	2085	46,16	2085	0,02	2187	0,022	2085	0,077
gr17_25_1	2265	3,18	2265	0,02	2358	0,237	2265	0,061
gr17_25_2	2505	16,48	2505	0,02	3212	0,059	2514	0,064
gr17_25_3	2270	4,3	2270	0,02	2934	0,028	2270	0,063
gr17_25_4	2103	20,1	2103	0,02	3032	0,029	2207	0,097
gr17_25_5	2088	22,16	2088	0,02	3034	0,01	2120	0,077
gr17_25_6	2160	9,34	2160	0,02	2581	0,013	2219	0,094
gr17_25_7	2085	15,93	2085	0,02	2386	0,03	2180	0,093
gr17_25_8	2088	10,47	2088	0,02	2212	0,02	2088	0,087
gr17_25_9	2138	1,96	2138	0,02	2443	0,071	2233	0,085
gr17_25_10	2675	8,25	2675	0,02	2938	0,015	2675	0,19
gr17_50_1	2743	9,46	2743	0,02	3196	0,344	2743	0,054
gr17_50_2	2216	6,67	2216	0,02	2365	0,07	2216	0,051
gr17_50_3	3000	1,77	3000	0,02	3207	0,011	3000	0,058
gr17_50_4	2946	4,73	2946	0,02	3723	0,011	2946	0,071
gr17_50_5	2205	19,37	2205	0,02	2823	0,011	2243	0,27
gr17_50_6	2579	2,63	2579	0,02	3587	0,066	2579	0,111
gr17_50_7	2812	2,01	2812	0,02	3495	0,168	2812	0,134
gr17_50_8	3014	1,14	3014	0,02	3254	0,109	3027	0,051

gr17_50_9	3454	1,73	3454	0,02	3811	0,01	3454	0,056
gr17_50_10	2134	6,52	2134	0,02	2307	0,025	2134	0,053

Tabela 6 - Resultados da instância *gr21*

Instância	BCP		GRASP		Guloso		ACO	
	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)
gr21_10_1	2707	18,77	2707	0,02	3369	2,119	2707	0,012
gr21_10_2	3002	8,58	3002	0,02	3877	0,017	3002	0,136
gr21_10_3	2851	11,62	2851	0,02	3408	0,04	2863	0,289
gr21_10_4	2760	10,25	2760	0,02	3714	0,015	2760	0,221
gr21_10_5	2707	8,56	2707	0,02	3333	0,025	2707	0,195
gr21_10_6	2760	7,76	2760	0,02	3465	0,014	2760	0,164
gr21_10_7	3093	12,58	3093	0,02	3526	0,019	3093	0,149
gr21_10_8	2962	15,88	2962	0,03	3732	0,043	2962	0,175
gr21_10_9	2787	11,9	2787	0,02	3732	0,012	2962	0,148
gr21_10_10	2707	11,32	2707	0,02	3333	0,075	2707	0,152
gr21_25_1	2788	9,32	2788	0,02	4502	0,03	2788	0,155
gr21_25_2	2946	7,43	2946	0,02	3681	0,02	2946	0,135
gr21_25_3	3109	12,53	3109	0,03	3417	0,071	3186	0,106
gr21_25_4	2707	6,01	2707	0,02	4202	0,051	2707	0,187
gr21_25_5	3159	23,31	3159	0,04	3431	0,057	3159	0,128
gr21_25_6	3159	15,72	3159	0,02	4023	0,216	3160	0,13
gr21_25_7	2921	9,45	2921	0,02	3717	0,075	2921	0,139
gr21_25_8	3421	21,77	3421	0,02	4202	0,055	3474	0,12
gr21_25_9	2709	3,74	2709	0,02	4162	0,022	2709	1,853
gr21_25_10	2707	6,26	2707	0,02	3404	0,037	2707	0,153
gr21_50_1	3115	8,34	3115	0,02	3438	0,054	3115	0,152
gr21_50_2	4041	30,7	4041	0,02	4648	0,022	4041	0,087
gr21_50_3	3892	7,82	3892	0,03	4755	0,045	3892	0,159
gr21_50_4	3570	10,64	3570	0,02	4571	0,06	3574	0,078
gr21_50_5	4132	5,32	4132	0,02	4512	0,031	4236	0,11
gr21_50_6	3417	7,43	3417	0,03	4134	0,013	3417	0,098
gr21_50_7	4249	7,71	4249	0,02	4680	0,018	4249	0,129
gr21_50_8	3296	5,85	3296	0,02	4509	0,026	3296	0,129
gr21_50_9	4186	8,88	4186	0,02	4676	0,027	4186	0,095
gr21_50_10	3483	11,47	3483	0,02	4676	0,094	3516	0,102

Tabela 7 - Resultados da instância *ulysses22*

Instância	BCP		GRASP		Guloso		ACO	
	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)	Sol.	T. (s)
ulysses22_10_1	7013	35,76	7013	0,03	10562	0,025	7167	0,197
ulysses22_10_2	7013	26,54	7013	0,03	10945	0,012	7167	0,229

ulysses22_10_3	7013	28,06	7013	0,03	10548	0,036	7029	0,208
ulysses22_10_4	7013	19,77	7013	0,03	10548	0,041	7167	0,22
ulysses22_10_5	7013	39,26	7013	0,03	10899	1,532	7173	0,236
ulysses22_10_6	7250	40,55	7250	0,03	11383	1481	7250	0,244
ulysses22_10_7	7246	40,2	7246	0,03	10293	0,013	7356	0,258
ulysses22_10_8	7181	55,88	7181	0,03	11035	0,094	7193	0,225
ulysses22_10_9	7047	18,25	7047	0,03	9331	0,014	7047	0,209
ulysses22_10_10	7087	17,92	7087	0,03	10627	0,047	7087	0,229
ulysses22_25_1	7083	30,63	7083	0,03	10657	0,122	7083	0,211
ulysses22_25_2	7415	19,25	7415	0,03	10971	0,038	7415	0,215
ulysses22_25_3	8177	21,95	8177	0,03	10563	0,02	8247	0,202
ulysses22_25_4	7385	28,7	7385	0,03	10393	0,102	7399	0,226
ulysses22_25_5	7449	23,58	7449	0,03	10956	0,044	7468	0,237
ulysses22_25_6	7589	32,37	7589	0,03	11307	0,021	7589	0,174
ulysses22_25_7	7729	23,91	7729	0,03	10792	0,04	7729	0,227
ulysses22_25_8	7123	17,45	7123	0,03	10860	0,035	7123	0,221
ulysses22_25_9	7176	27,08	7176	0,03	8757	0,014	7270	0,221
ulysses22_25_10	7961	22,77	7961	0,03	8892	0,018	8190	0,165
ulysses22_50_1	8290	24,18	8290	0,03	10907	0,033	8443	0,202
ulysses22_50_2	7538	16,58	7538	0,03	11026	0,034	7620	0,191
ulysses22_50_3	8833	21,75	8833	0,03	11110	0,013	8840	0,132
ulysses22_50_4	9324	38,53	9324	0,03	9437	0,014	9356	0,123
ulysses22_50_5	8284	46,11	8284	0,03	10171	0,024	8321	0,446
ulysses22_50_6	7570	10,6	7570	0,03	8911	0,01	7570	0,198
ulysses22_50_7	7897	25,41	7897	0,04	10547	0,142	7897	0,205
ulysses22_50_8	9558	20,68	9558	0,03	10259	0,07	9601	0,193
ulysses22_50_9	9021	48,15	9021	0,03	10521	0,013	9240	0,157
ulysses22_50_10	7941	15,13	7941	0,03	11787	0,026	7981	0,17

Tabela 8 - Resultados da instância *fri26*

Instância	**BCP**		**GRASP**		**Guloso**		**ACO**	
	Sol.	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**
fri26_10_1	937	11,99	937	0,07	1096	0,16	959	0,37
fri26_10_2	937	11,35	937	0,03	1112	0,034	937	0,231
fri26_10_3	1009	14,54	1009	0,03	1160	0,059	1011	0,273
fri26_10_4	955	12,61	955	0,04	1096	0,014	1004	0,404
fri26_10_5	997	21,95	997	0,05	1284	0,017	997	0,378
fri26_10_6	937	8,54	937	0,05	1284	0,011	975	0,221
fri26_10_7	1039	14,24	1039	0,04	1231	0,015	1086	0,321
fri26_10_8	953	8,33	953	0,03	1112	0,039	974	0,321
fri26_10_9	937	11,18	937	0,03	1434	0,014	937	0,206
fri26_10_10	937	9,98	937	0,03	1112	0,012	937	0,242
fri26_25_1	1055	14,7	1055	0,04	1251	0,016	1055	0,23
fri26_25_2	1201	26,49	1201	0,04	1446	0,014	1208	0,26

fri26_25_3	1139	24,34	1139	0,04	1355	0,017	1139	0,308
fri26_25_4	1233	24,35	1233	0,04	1472	0,48	1233	0,262
fri26_25_5	1017	14,96	1017	0,05	1227	0,015	1038	0,256
fri26_25_6	1172	14,79	1172	0,04	1277	0,02	1172	0,229
fri26_25_7	1101	19,01	1101	0,03	1409	0,013	1101	0,323
fri26_25_8	955	6,46	955	0,04	1224	0,012	955	0,29
fri26_25_9	1081	10,24	1081	0,05	1295	0,049	1081	0,238
fri26_25_10	1093	9,22	1093	0,04	1145	0,017	1104	0,226
fri26_50_1	1273	16,64	1273	0,04	1486	0,073	1273	0,18
fri26_50_2	1045	22,21	1045	0,05	1209	0,012	1052	0,255
fri26_50_3	1035	12,52	1035	0,04	1236	0,016	1035	0,2
fri26_50_4	1185	12,13	1185	0,04	1497	0,013	1185	0,215
fri26_50_5	1185	12,57	1185	0,03	1371	1,578	1185	0,174
fri26_50_6	1158	9,9	1158	0,04	1366	0,031	1158	0,234
fri26_50_7	1150	14,74	1150	0,11	1296	0,014	1150	0,211
fri26_50_8	1441	22,36	1441	0,03	1779	0,065	1441	0,406
fri26_50_9	1267	28,57	1267	0,04	1449	0,021	1267	0,211
fri26_50_10	1048	11,96	1048	0,04	1097	0,019	1048	0,206

Tabela 9 - Resultados da instância *gr48*

Instância	**BCP**		**GRASP**		**Guloso**		**ACO**	
	Sol.	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**	**Sol.**	**T. (s)**
gr48_10_1	5046	813,29	5524	0,24	5889	0,014	5152	2,853
gr48_10_2	5542	341,27	5895	0,68	7518	0,015	5677	2,914
gr48_10_3	5300	3314,02	5754	0,29	6928	0,02	5491	2,404
gr48_10_4	5293	7057,36	5588	1,77	6236	0,062	5396	2,8
gr48_10_5	5679	496,29	6159	29,99	7109	0,017	5740	2,73
gr48_10_6	5610	41,04	5760	8,49	7503	0,079	5732	2,851
gr48_10_7	5063	22,79	5955	15,35	6201	0,053	5187	2,603
gr48_10_8	5103	188,77	5562	2,35	7508	0,071	5216	2,496
gr48_10_9	5153	892,87	5792	1,87	6373	0,036	5227	2,732
gr48_10_10	5055	359,47	6014	2,73	6204	0,039	5144	3,029
gr48_25_1	5524	1073,43	5524	0,24	6422	0,015	5607	2,644
gr48_25_2	5895	361,9	5895	0,68	7261	0,022	6016	2,861
gr48_25_3	5754	47,85	5754	0,29	7344	0,011	5849	3,139
gr48_25_4	5588	126,05	5588	1,77	6652	0,015	5617	2,52
gr48_25_5	6159	1793,96	6159	29,99	7226	0,035	6247	2,297
gr48_25_6	5760	3053,47	5760	8,49	6613	0,024	5851	2,259
gr48_25_7	5955	172,08	5955	15,35	7817	0,024	6028	2,544
gr48_25_8	5562	1070,86	5562	2,35	7124	0,013	5665	2,181
gr48_25_9	5792	184,24	5792	1,87	7277	0,021	5997	2,789
gr48_25_10	6014	730,82	6014	2,73	7253	0,024	6132	3,192
gr48_50_1	6069	157,47	6096	1,24	7372	0,015	6109	1,738
gr48_50_2	6629	21,44	6629	1,89	8187	0,017	6712	1,78

gr48_50_3	5896	132,45	5896	4,94	7621	0,038	6226	2,564
gr48_50_4	6404	20,75	6404	0,51	8516	0,018	6533	1,875
gr48_50_5	6617	19,41	6617	0,23	8351	0,016	6936	1,803
gr48_50_6	8533	66,09	8533	1,14	10541	0,014	8606	1,902
gr48_50_7	6166	1520,1	6166	1,33	7914	0,035	6262	2,423
gr48_50_8	6535	20,93	6535	9,73	9489	0,024	6576	1,745
gr48_50_9	7150	42,73	7150	6,36	9398	0,033	7176	2,004
gr48_50_10	6331	112,13	6331	0,59	7467	0,036	6442	2,243

Dado os resultados das Tabelas de 3 a 9, decidiu-se colocar esses valores em forma de gráfico, representado nas Figuras de 6 a 12, para que facilitasse a visualização de o quão perto ou longe o resultado está do ótimo.

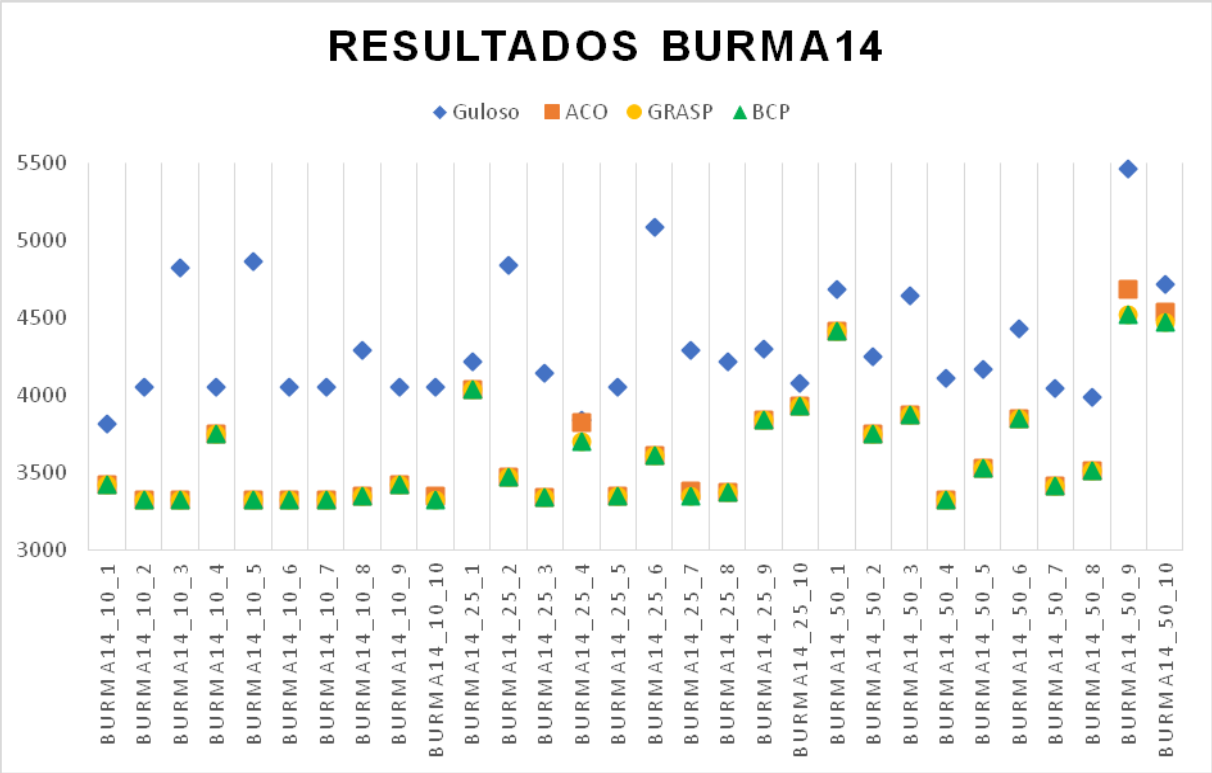

Figura 6 - Gráfico referente aos resultados da instância Burma14
Fonte: Autoria própria

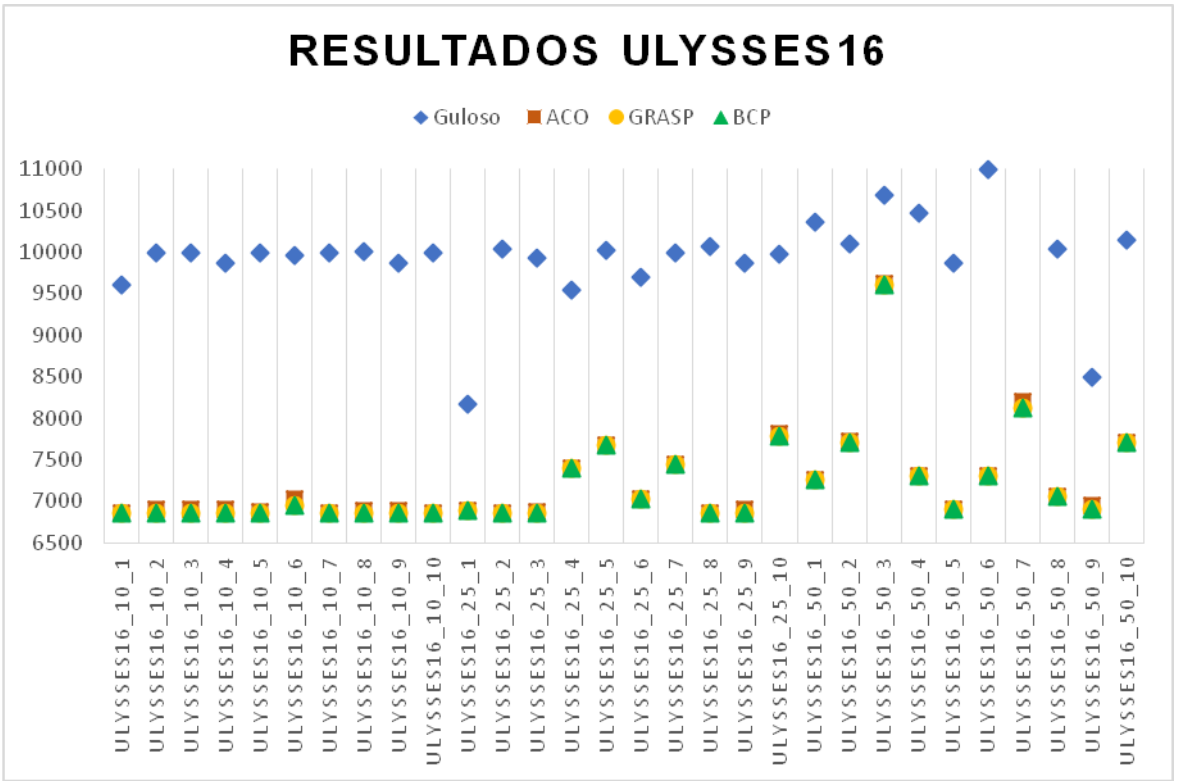

Figura 7 - Gráfico referente aos resultados da instância Ulysses16
Fonte: Autoria própria

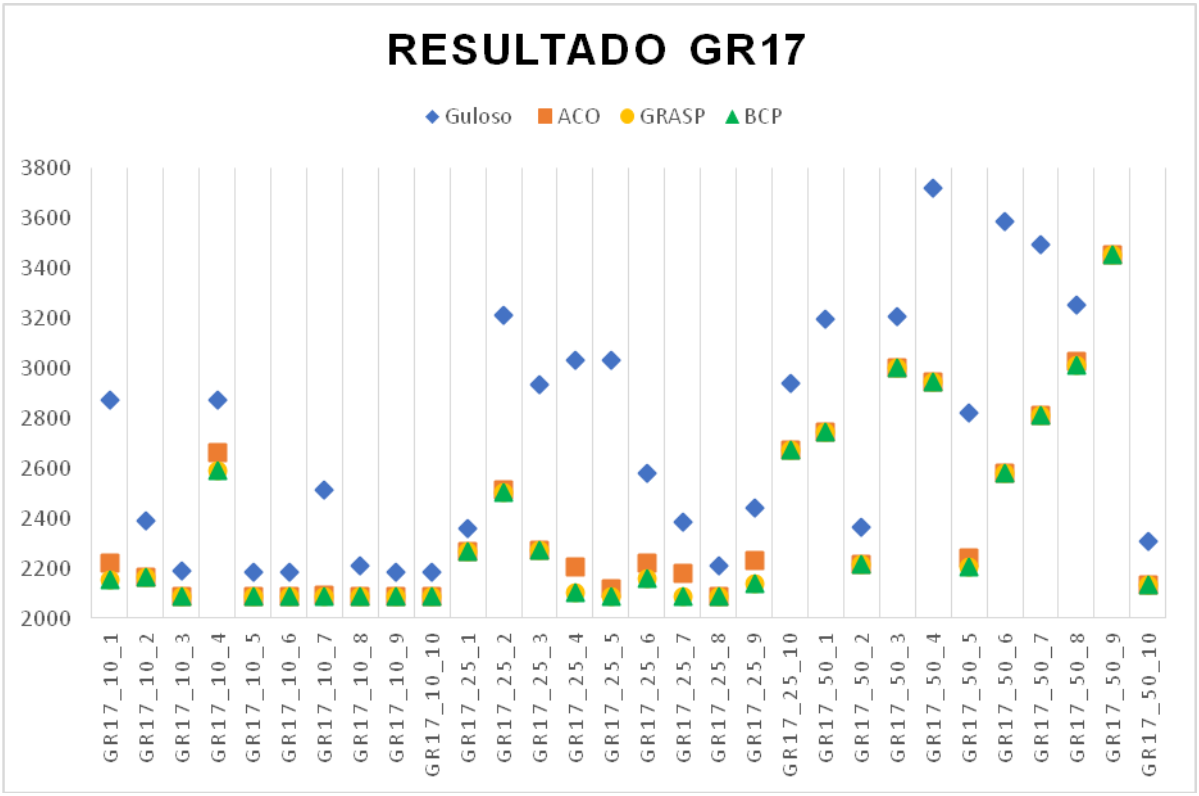

Figura 8 - Gráfico referente aos resultados da instância Gr17
Fonte: Autoria própria

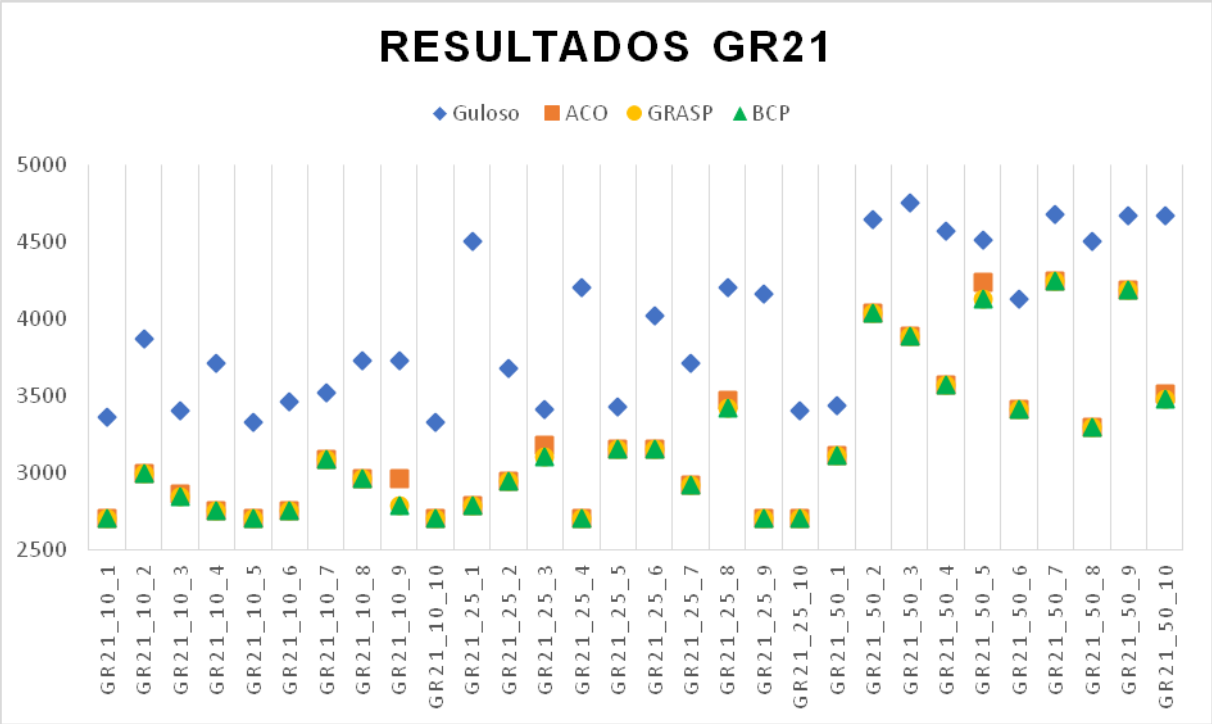

Figura 9 - Gráfico referente aos resultados da instância Gr21
Fonte: Autoria própria

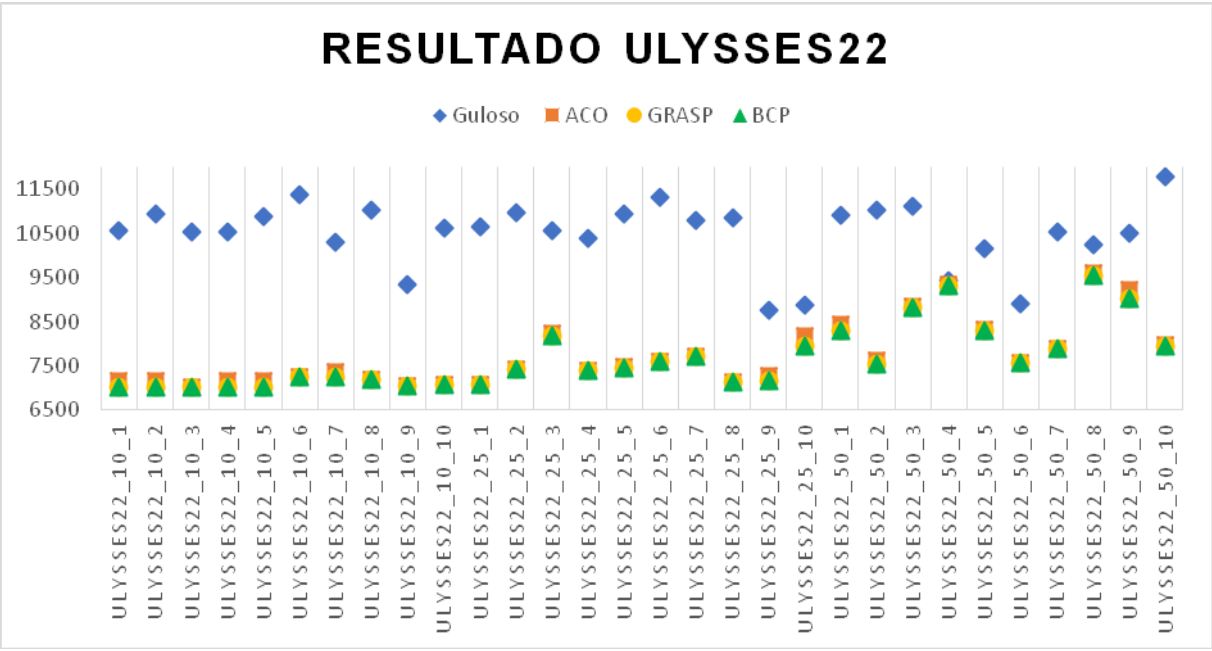

Figura 10 - Gráfico referente aos resultados da instância Ulysses22
Fonte: Autoria própria

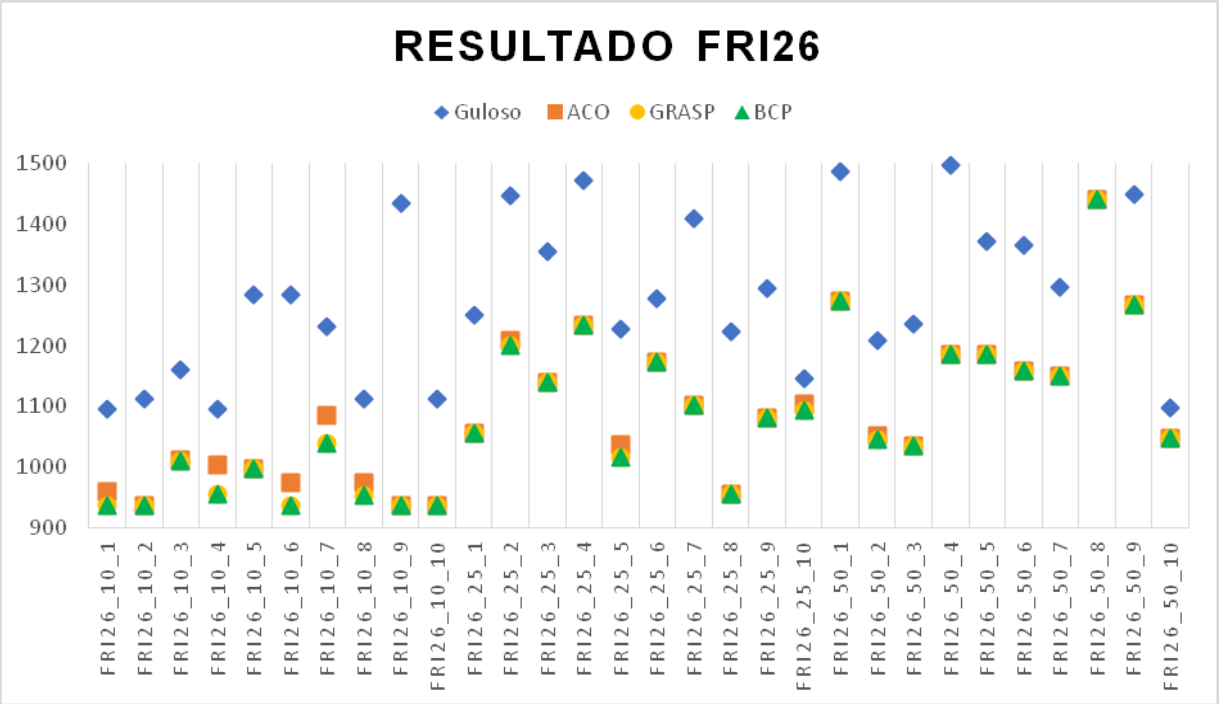

Figura 11 - Gráfico referente aos resultados da instância Fri26
Fonte: Autoria própria

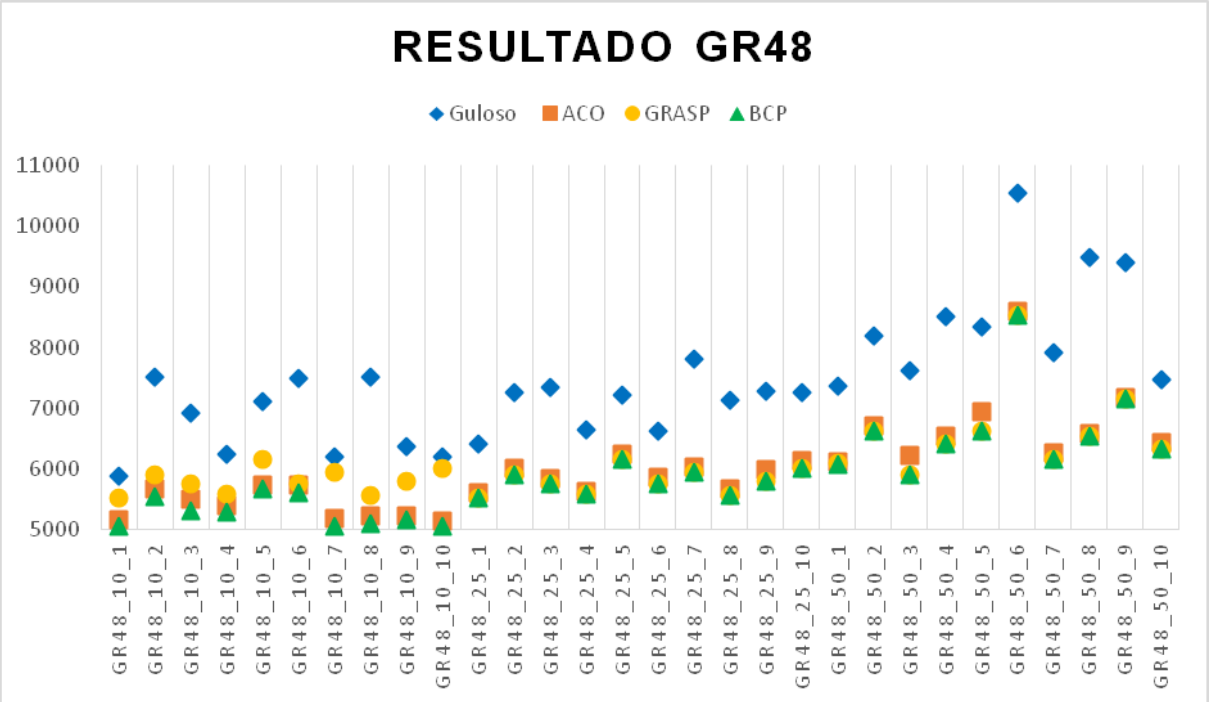

Figura 12 - Gráfico referente aos resultados da instância Gr48
Fonte: Autoria própria

Sabe-se que o resultado do algoritmo BCP é sempre ótimo, por se tratar de um algoritmo exato, pelas tabelas e pelos gráficos, nota-se que o algoritmo Guloso

sempre ficou bastante distante do resultado ótimo, enquanto o algoritmo ACO, apesar de nem sempre ter obtido resultados ótimos - em alguns casos como os resultados da maior instância *gr48*, nenhuma vez chegou a esse resultado, o mesmo desempenhou bem, na maioria das vezes chegando bastante próximo ao resultado ótimo, e em alguns casos teve um desempenho melhor que a solução do GRASP.

Para entender melhor os resultados do algoritmo ACO e encontrar padrões, foi gerada a Tabela 9, que apresenta a porcentagem de resultados ótimos de cada instância, dividido pelo resultado para cada uma das porcentagens que mostram o limite de calado diferenciado e o total.

Tabela 9 - Porcentagem de soluções ótimas em cada instância

	10	25	50	total
burma14	90%	70%	80%	80%
gr17	30%	70%	60%	53%
ulysses16	60%	40%	80%	60%
gr21	80%	70%	70%	73%
ulysses22	30%	50%	20%	30%
fri26	40%	70%	90%	66%
gr48	0%	0%	0%	0%

Nota-se, no entanto, que não existe um padrão especifico, em alguns casos instâncias maiores tiveram uma porcentagem de resultados ótimos maior que uma instância menor, por exemplo, é a comparação da instância *fri26* com *ulysses16,* em outros a porcentagem de portos com 50% de limite calado menor que o limite máximo de calado performou melhor que quando a porcentagem era 10%, um caso é a instância *fri26*.

Esses casos só reforçam a natureza randômica e probabilística do algoritmo, o qual pode gerar resultados completamente diferentes dependo da execução.

Também é importante salientar que o algoritmo ACO sempre buscava pelo melhor resultado global e as instâncias geralmente convergiam para esse valor, mas em vários casos notou-se que as formigas nunca nem chegavam a desbravar o melhor caminho, o que as impedia de convergirem para tal.

Apesar de o algoritmo ACO não ter gerado resultados sempre ótimos, como desejado, o mesmo se mostra uma alternativa bastante razoável. Pode-se dizer que talvez algumas alterações em cima do algoritmo possam gerar resultados melhores

em trabalhos posteriores, como, por exemplo, melhorar a forma da busca do algoritmo.

5 CONCLUSÃO

Introduzido por Rakke et al. (2012) o problema do caixeiro viajante com limite de calado é uma variação do problema do caixeiro viajante o qual tem ganhado visibilidade nos últimos tempos, entretanto os trabalhos em cima da mesma ainda são poucos. O PCVLC explora uma restrição ao PCV no contexto de transporte marítimo, que é limite de calado, o qual é um problema real, pois o mesmo pode implicar em um navio encalhado caso essa variável não seja considerada na geração de rotas.

Nesse trabalho foram implementadas duas soluções para PCV: o algoritmo guloso, com o objetivo principal de comparação, devido à natureza simples de sua implementação, e o maior foco do trabalho foi dado à meta-heurística colônia de formigas, cujo desempenho foi menor que o esperado, entretanto foram bons.

Sabe-se que o primeiro algoritmo de ACO, conhecido por Sistema de Formigas, também não alcançou os melhores resultados quando comparados ao estado da arte, mas a qualidade dos resultados inspirou extensões de trabalhos em cima de AS que tiveram desempenhos ótimos (DORIGO; STÜTZLE, 2004).

Apesar de ACO para o PCVLC não ter encontrado sempre a melhor solução, ele se mostrou uma alternativa viável, principalmente quando executado para instâncias maiores. Em alguns desses casos o algoritmo se comportou melhor do que apresentam dados existentes na literatura, que é o caso do algoritmo GRASP, que foi usado para comparação de resultados. Já o algoritmo guloso se comportou como esperado, gerando na maioria dos casos resultados distantes da solução ótima.

Ainda existe um leque de soluções a serem abordadas em cima desse problema, tanto meta-heurísticas diferentes das já abordadas até o momento, como aperfeiçoamentos das mesmas. Para ACO, durante o desenvolvimento desse trabalho surgiram algumas ideias de melhorias as quais ainda não foram implementadas, como refinar a busca local das formigas, influenciando-as a explorarem mais caminhos e consequentemente soluções melhores. Também sugere-se explorar ações *daemon* diferentes da busca pelo melhor caminho global para ver como a solução se comporta.

REFERÊNCIAS

APPLEGATE, David; BIXBY, Robert; CHVATAL, Vasek; COOK, William. **Finding cuts in the TSP (A Preliminary Report)**. [S.l.], 1995.

APPLEGATE, David L; BIXBY, Robert; CHVATAL, Vasek; COOK, William. **The traveling salesman problem: a computational study.** [S.l.]: Princeton university press, 2006.

BATTARRA, Maria; PESSOA, Artur Alves; SUBRAMANIAN, Anand; UCHOA, Eduardo. **Exact algorithms for the traveling salesman problem with draft limits.** European Journal of Operational Research, v. 235, n. 1, p. 115–128, 2014. Disponível em: <http://EconPapers.repec.org/RePEc:eee:ejores:v:235:y:2014:i:1:p:115-128>.

BELLMORE, Mandell; NEMHAUSER, George L. **The traveling salesman problem: a survey.** Operations Research, v. 16, n. 3, p. 538–558, 1968. Disponível em: <http://dx.doi.org/10-.1287/opre.16.3.538>.

BLUM, Christian; ROLI, Andrea. **Metaheuristics in combinatorial optimization: overview and conceptual comparison**. ACM Comput. Surv., ACM, New York, NY, USA, v. 35, n. 3, p. 268–308, set. 2003. ISSN 0360-0300. Disponível em: <http://doi.acm-.org/10.1145/937503.937505>.

BODIN, L.; GOLDEN, B.; ASSAD, A.; BALL, M**. Routing and scheduling of vehicles and crews: the state of the art.** Pergamon Press, v. 10, n. 8, 1983.

BUKARD, Rainer E. **Traveling salesman and assignment problems: a survey**. Annals of Discrete Mathematics, 1979.

CROWDER, Harlan; PADBERG, Manfred W. **Solving large-scale symmetric travelling salesman problems to optimality**. Management science, INFORMS, v. 26, n. 5, p. 495–509, 1980.

DORIGO, Marco. **Optimization, learning and natural algorithms**. PhD thesis, Politecnico diMilano, Italy, 1992.

DORIGO, Marco; STÜTZLE, Thomas. **Ant colony optimization**. London, England, 2004.

FAGERHOLT, Kjetil; CHRISTIANSEN, Marielle. **A combined ship scheduling and allocation problem**. Journal of the operational research society, Springer, v. 51, n. 7, p. 834–842, 2000.

FEO T.A.; RESENDE, M.G.C. **Greedy randomized adaptive search procedures.** J. of Global Optimization, v. 6, p. 109–133, 1995.

GAREY, Michael R.; JOHNSON, David S. **Computers and Intractability; a guide to the theory of NP-Completeness**. New York, NY, USA: [s.n.], 1990.

GLOVER, Fred; KOCHENBERGER, Gary A. (Ed.). **Handbook of metaheuristics**. Boston, Dordrecht, London: Kluwer Academic Publishers, 2003. (International series in operations research & management science). ISBN 1-4020-7263-5. Disponível em: <http://opac.inria.fr/record=b1099522>.

GOLDBARG, Marco; GOLDBARG, Elizabeth. **Grafos: Conceitos, algoritmos e aplicações**. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ldta, 2012.

GOLDBARG, Marco; LUNA, H. P. L. **Otimização combinatória e programação linear**. 2. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ldta, 2004.

GOODRICH, Michael T.; TAMASSIA, Roberto. **Projeto de algoritmos: fundamentos, análise e exemplos da Internet.** Porto Alegre, RS, Brasil: Bookman, 2004.

GOSS, S.; ARON, S.; DENEUBOURG, J. L.; PASTEELS, J. M. **Self-organized shortcuts in the Argentine ant**. Naturwissenschaften, 76:579–581, 1989.

GRÖTSCHEL, Martin; HOLLAND, Olaf. **Solution of large-scale symmetric travelling salesman problems**. Math. Program., Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 51, n. 2, p. 141–202, jul. 1991. ISSN 0025-5610. Disponível em: <http://dx-.doi.org/10.1007/BF01586932>.

GRÖTSCHEL, Martin; JÖNGER, M; REINELT, Gerhard. Via minimization with pin preassignments and layer preference**. ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik**, Wiley Online Library, v. 69, n. 11, p. 393–399, 1989.

GUTIN, G.; PUNNEN, A.P. **The traveling salesman problem and its variations**. [S.l.]: Springer, 2007. ISBN 978-0-306-48213-7.

HOPCROFT, John E.; MOTWANI, Rajeev; ULLMAN, Jeffrey D. **Introdução à teoria de autômatos, linguagens e computação**. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ldta, 2002.

KUSIAK, Andrew. **Planning of flexible manufacturing systems**. Robótica, Cambridge Univ Press, v. 3, n. 04, p. 229–232, 1985.

LAPORTE, G.; SRISKANDARAJAH, C.; ASEF-VAZIRI, A.; (MONTRÉAL, Québec) Centre de recherche sur les transports. **Some applications of the generalized traveling salesman problem**. Centre de recherche sur les transports, 1995. (Publication (Centre de recherche sur les transports (Montréal, Québec)))). Disponível em: <https:-//books.google.com.br/books?id=9NPGSAAACAAJ>.

LAPORTE, Silvano Martello Gilbert. **The selective travelling salesman problem. Discrete Applied Mathematics**, v. 26, p. 193–207, 1990.

MACHADO, Victor Mouffron Carvalho; OCHI, Luiz Satoru; NEVES, Tiago Araujo. **Grasp para o problema do caixeiro viajante com limite de calado**. In: Bastos Filho, C. J. A.; POZO, A. R.; LOPES, H. S. (Ed.). Anais do 12 Congresso Brasileiro de Inteligência Computacional. Curitiba, PR: ABRICOM, 2015. p. 1–6.

MICHALEWICZ, Zbigniew; FOGEL, David B. **How to solve it : modern heuristics**. New York: Springer, 2000. Réimpression : 2002. ISBN 3-540-66061-5. Disponível em: <http://opac.inria.fr/record=b1095540>.

MILLER, D.; PEKNY, J. **Exact solution of large asymmetric traveling salesman problems**. Science, v. 251, p. 754–761, 1991.

MLADENOVIC, Nenad; HANSEN, Pierre. **Variable neighborhood search**. Computers and Operations Research. V. 24, p. 11, p.1097–1100. doi:10.1016/s0305-0548(97)00031-2. 1997.

MORAIS, Wall Berg; ROSA, Marcelo; TEIXEIRA, Marcelo; BARBOSA, Marco Antonio. **O Problema do caixeiro viajante com limite de calado: uma abordagem usando simulated annealing**. XLIX Simpósio Brasileiro de Pesquisa Operacional. 2017

PAPADIMITRIOU, Christos H.; STEIGLITZ, Kenneth. **Combinatorial optimization**: algorithms and complexity. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

RAKKE, Jørgen Glomvik; CHRISTIANSEN, Marielle; FAGERHOLT, Kjetil; LAPORTE, Gilbert. **The traveling salesman problem with draft limits**. Comput. Oper. Res., Elsevier Science Ltd., Oxford, UK, UK, v. 39, n. 9, p. 2161–2167, set. 2012. ISSN 0305-0548. Disponível em: <http://dx.doi.org/10.1016/j.cor.2011.10.025>.

RAMESH, T. **Traveling purchaser problem**. OPSEARCH, n. 18, p. 78–91, 1981.

REINELT, Gerhard. **The traveling salesman**: computational solutions for TSP applications. Berlin, Heidelberg: Springer-Verlag, 1994. ISBN 3-540-58334-3.

REINELT, Gehard. **Tsplib - a traveling salesman problem library**. INFORMS Journal onComputing, v. 3. n.4, p. 376–384, 1991.

SIPSER, M. **Introdução à teoria da computação**. 2 ed. São Paulo, SP, Brasil: Thomson Learning, 2007.

SZWARCFITER, Jayme Luiz. **Grafos e algoritmos computacionais**. 2 ed. Rio de Janeiro, RJ, Brasil: Editora Campus LTDA, 1983.

TALBI, El-Ghazali. **Metaheuristics**: from design to implementation. [S.l.]: Wiley Publishing, 2009. ISBN 0470278587, 9780470278581.

TODOSIJEIC, Raca; MJIRDA, Anis; MLADENOVIC, Marko; SAID, Hanafi; GENDRON, Bernard. **A general variable neighborhood search variants for the travelling salesman problem with draft limits**. Optimization Letters, p. 1–10. 2014.

TOSCANI, L.V.; VELOSO, P.A.S. **Complexidade de algoritmos**. Série Livros Didáticos Informática UFRGS, v. 13. Bookman, 2009. ISBN 9788540701397. Disponível em: <https://books.google.com.br/books?id=Ond1ls37VHwC>.