Universidade Federal
do Rio de Janeiro
Escola Politécnica

CLUSTERIZAÇÃO DE UNIDADES OFFSHORE PARA O ROTEAMENTO DE EMBARCAÇÕES DE SUPRIMENTO

Rafael Pedro Longhi

Projeto de Graduação apresentado ao Curso de Engenharia de Petróleo da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Virgílio José Martins Ferreira Filho

Rio de Janeiro
Março de 2014

CLUSTERIZAÇÃO DE UNIDADES OFFSHORE PARA O ROTEAMENTO DE EMBARCAÇÕES DE SUPRIMENTO

Rafael Pedro Longhi

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE PETRÓLEO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE PETRÓLEO.

Examinado por:

_____

Prof. Virgílio José Martins Ferreira Filho, D. Sc..
(Orientador)

_____

Prof.ª Laura Silvia Bahiense da Silva Leite, D. Sc.

_____

Prof. Edilson Fernandes de Arruda**,** D. Sc.

RIO DE JANEIRO, RJ – BRASIL

MARÇO de 2014

Longhi, Rafael Pedro

Clusterização de unidades offshore para o roteamento de embarcações de suprimento/ Rafael Pedro Longhi – Rio de Janeiro: UFRJ/ Escola Politécnica, 2014.

XI, 85 p.: il.; 29,7 cm.

Orientador: Virgílio José Martins Ferreira Filho

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Petróleo, 2014.

Referências Bibliográficas: p. 42.

1.Clusterização 2. Indústria do Petróleo 3. Pesquisa operacional

I. Ferreira Filho, Virgílio José Martins. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Petróleo. III. Clusterização de unidades offshore para o roteamento de embarcações de suprimento

Dedico este trabalho a meus pais, meu irmão Luan, meus amigos e professores da UFRJ

Agradecimentos

Agradeço à equipe do LORDE (Laboratório de Otimização de Recursos, de Simulação Operacional e de Apoio a Decisões na Indústria do Petróleo) pela infraestrutura necessária para a realização deste trabalho. Agradeço também à ANP, que, por meio do PRH-21, ofereceu suporte à realização deste trabalho.

Agradeço ao meu amigo e professor Virgílio José Martins Ferreira Filho, por seu tempo dedicado e por seus conselhos como orientador, além de sua importante ajuda em nossa formação pessoal e profissional.

Agradeço também aos outros professores envolvidos no projeto, Professor Edilson Fernandes de Arruda e Professora Laura Silvia Bahiense da Silva Leite, por seus conselhos e ajuda nos momentos mais difíceis da elaboração deste projeto.

Agradeço também pelo empenho e ajuda aos outros alunos que me ajudaram neste projeto, Juliana, Rennan, Sayuri e Tales. Além da ajuda na elaboração de toda a metodologia, acabamos criando uma grande amizade que pretendo levar para a minha vida.

Agradeço aos meus pais, Vilnei e Dilena, pelo amor e por tudo que me proporcionaram durante minha vida.

Agradeço ao meu irmão Luan pela companhia em todos os momentos.

Agradeço a todos meus familiares e amigos que me apoiaram durante minha vida pessoal e acadêmica e que me permitiram realizar esse Projeto.

Resumo do Projeto de Graduação apresentado à escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Petróleo.

CLUSTERIZAÇÃO DE UNIDADES OFFSHORE PARA O ROTEAMENTO DE EMBARCAÇÕES DE SUPRIMENTO

Rafael Pedro Longhi

Março de 2014

Orientador: Virgílio José Martins Ferreira Filho

Curso: Engenharia de Petróleo

Os custos envolvidos na implementação de projetos da indústria do petróleo são muito elevados, por isso, torna-se necessário otimizar ao máximo as atividades realizadas. Ao longo de um projeto no ambiente offshore, vários equipamentos e suprimentos são demandados e, para o transporte destes para as unidades, há a necessidade da utilização de barcos de apoio, cujas rotas necessitam ser planejadas. Hoje em dia, na Bacia de Campos, é comum a utilização de métodos empíricos, o que torna as operações pouco otimizadas. Este projeto consiste na pesquisa por novas metodologias que possam ser empregadas. Pretende-se otimizar as rotas dos barcos que as atendam, diminuindo-se a distância total que necessita-se percorrer, gerando menores custos graças ao menor consumo de diesel, por exemplo. Ao fim do trabalho, uma metodologia estudada será implementada e novas rotas serão sugeridas.

Palavras-chave: Clusterização, Indústria do Petróleo, Pesquisa Operacional

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

OFFSHORE UNITS CLUSTERING FOR SERVICE VESSEL ROUTING

Rafael Pedro Longhi

March/ 2014

Advisor: Virgílio José Martins Ferreira Filho

Course: Petroleum Engineering

In petroleum industry, the costs are usually big, so activities must be optimized. During an offshore project, a large number of equipment and supplies are demanded. To transport it to offshore units, service vessels are necessary, and its routes should be planned. Empiric methods are commonly used in service vessel routes elaboration at the moment, so the operations are not optimized. This project consists on a research for new methods that can be used. The objective is to optimize the service vessel routes by minimizing the total distance that should be traveled. This can minimize diesel use and decrease costs. At the end of this project a new methodology will be implemented and new routes will be suggested.

Keywords: Clustering, Petroleum Industry, Operational research

Sumário

Lista de figuras

Lista de tabelas

1 – Introdução

Apesar dos esforços iniciais empreendidos na exploração petrolífera brasileira apontarem algumas acumulações em bacias terrestres, tais como a Bacia do Recôncavo Baiano, a produção petrolífera nacional só se impulsionou após a descoberta de acumulações *offshore*.

A pesquisa na Bacia de Campos iniciou-se em 1971, empreendida em um contexto de procura por acumulações de óleo no litoral brasileiro, dada a inexistência de grandes acumulações na parte terrestre do Brasil. Apenas em 1974 foi descoberta a primeira evidência de óleo e em 1977 a produção iniciou-se, inserida em um contexto de aumento dos preços do petróleo, tornando a produção *offshore* economicamente viável.

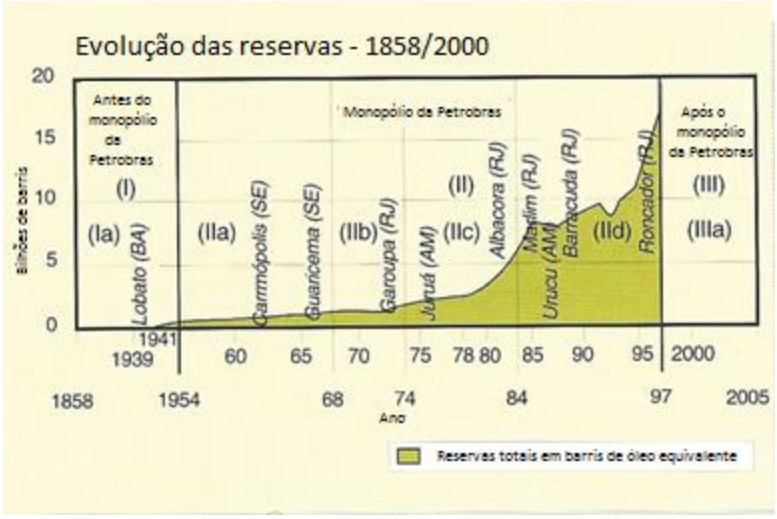

Figura 1. Evolução das reservas brasileiras. Fonte: Piquet [1]

Desde então, o cenário da exploração petrolífera brasileira alterou-se consideravelmente. O Recôncavo Baiano deixou de ter destaque na produção nacional e aos poucos a Bacia de Campos passou a assumir o seu lugar, com produções cada vez maiores. Como pode ser observado na Figura **1**, desde a descoberta do primeiro campo petrolífero na Bacia de Campos, o Campo de Garoupa, aumentaram consideravelmente as reservas brasileiras, ou seja, o petróleo possível de se produzir nos reservatórios. Segundo ANP[2], a produção de petróleo na Bacia de Campos em 2013 chegou a 102.744.128 m³, representando 85,66% da produção nacional.

Para sustentar toda essa produção, um grande número de plataformas e sondas é necessário. Cada uma dessas unidades possui demandas que variam desde comida para os funcionários até equipamentos e dutos mais específicos para alguma operação. Como toda a operação nessa bacia ocorre em mar, deve-se realizar todo este abastecimento através de barcos, tendo-se que lidar com limitações devido à falta de tempo disponível no porto ou ao número de barcos, por exemplo.

Apesar do grande número de unidades a que se deve atender, os únicos portos disponíveis para as operações da Petrobras são os portos de Macaé, Rio de Janeiro e Vitória. Destes, o grande destaque é o Porto de Imbetiba, em Macaé, que concentra grande parte da operação. Para a operação de abastecimento das unidades da Petrobras, que representam 91,7% dos campos da Bacia de Campos, segundo Leite[3], o Porto de Macaé teve que movimentar 650 mil toneladas de carga de convés, ou seja, 54% de toda a carga de convés da Petrobras.

O número de atendimentos demandados é bastante grande também. Existem em torno de 60 unidades de produção operando atualmente na Bacia de Campos. Além delas, várias sondas podem operar esporadicamente na região devido à perfuração e manutenção de poços. Como algumas unidades necessitam de mais de uma visita por semana, torna-se necessário aproximadamente 100 atendimentos a unidades por semana. A variação do número de visitas necessário por semana ao longo de um período registrado em 2013 pode ser observado na Figura **2**. Nota-se que ao longo das semanas, o número altera-se, devido provavelmente a entrada em operação de novas unidades de produção e sobretudo sondas ou devido a alterações na quantidade de produtos demandadas, que exigem mais ou menos visitas a uma unidade.

Como se não bastasse ter de se lidar com um grande número de unidades ao mesmo tempo, a operação na Bacia de Campos ainda apresenta outro grande complicador: a distância das unidades até a costa. Com o objetivo de encontrar reservas com maior potencial de produção, regiões com lâmina d'água cada vez mais profundas foram sendo exploradas, conforme pode ser observado na Figura **3**. Lâminas d'água maiores significam operações em regiões cada vez mais distantes da costa, cujo tempo

demandado de viagem entre o porto e as unidades é maior. Isso tudo torna a operação de abastecimento bastante complexa.

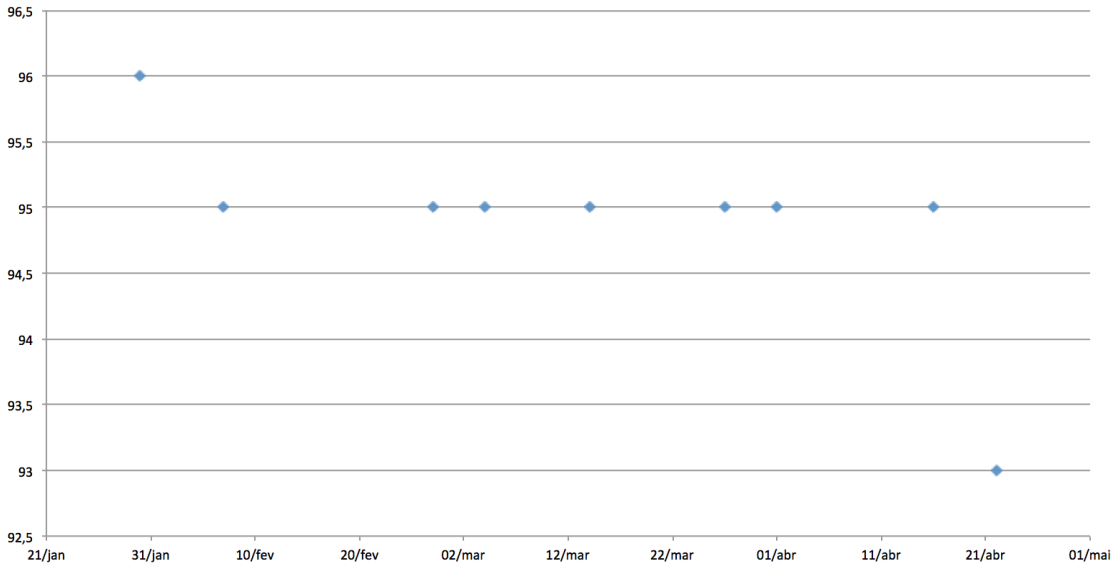

Figura 2. Número de visitas por semana registradas ao longo de um período de tempo em 2013.

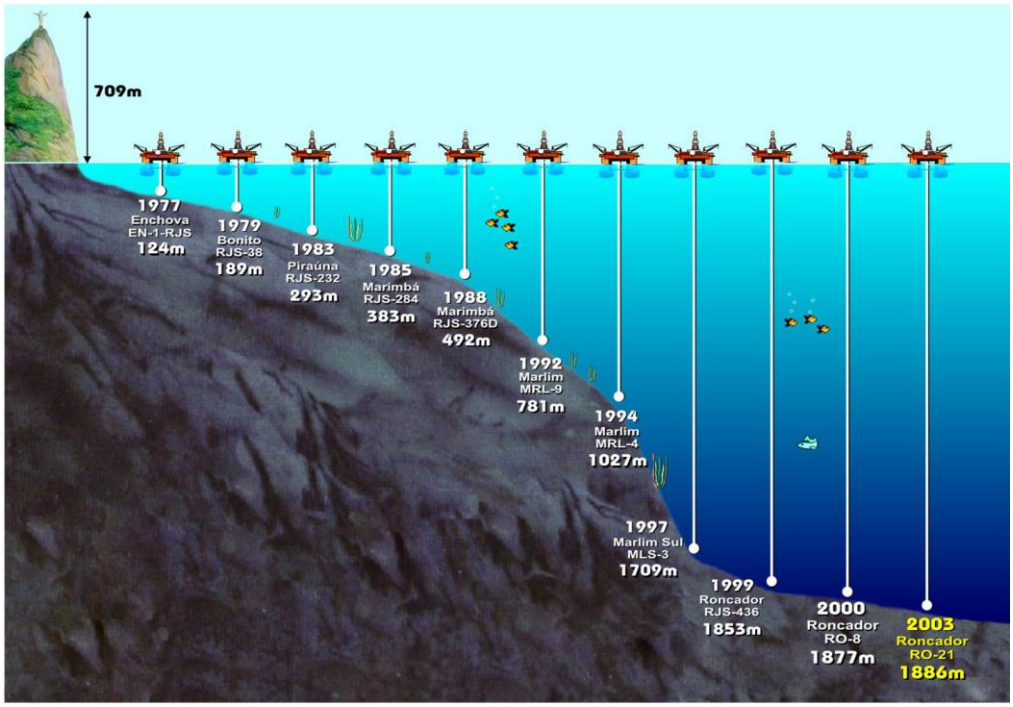

Figura 3. Evolução das lâminas d'água de poços exploratórios da Petrobras com o passar dos anos.

Dada a complexidade das operações de abastecimento a unidades da Bacia de Campos, esperava-se que o planejamento atualmente adotado fosse o melhor possível. Entretanto, os problemas de infraestrutura tornam tudo ainda mais complexo. O principal porto que atende às unidades, o Porto de Imbetiba, opera próximo de sua

capacidade máxima e a quantidade de barcos disponível é limitada. Atrasos e não atendimento a unidades são habituais.

Segundo Leite[3], hoje em dia muitos são os problemas encontrados no cotidiano das atividades. Existem *clusters* que necessitam de mais viagens semanais do que as planejadas, é grande o número de revisitas a unidades durante uma viagem, as unidades não sabem quando vão receber uma viagem e há 17% de visitas sem operação. Qualquer parada na produção em unidades *offshore* representa grandes perdas de dinheiro e o atual panorama de instabilidades no abastecimento facilita este tipo de problema. Motiva-se, portanto, estudos mais profundos que possam vir a melhorar este panorama.

Além de todo este panorama, as perspectivas para o futuro são ainda mais complexas. A produção de regiões com lâminas d'água menores deve chegar a maturidade e diminuir. O foco será, portanto, na produção de regiões cada vez mais longe da costa. Assim, o tempo de viagem dos barcos de apoio aumentará em média. Além disso, com a entrada de operação dos campos do Pré-Sal, muitas novas unidades deverão ser atendidas, sobrecarregando ainda mais os portos que prestam serviço às empresas petrolíferas.

Pretende-se, então, neste estudo, buscar uma programação de embarques e desembarques para o Porto de Macaé com o objetivo de torná-lo mais eficiente. Assim, espera-se conseguir atender a um maior número de unidades, com um menor custo e maior confiabilidade nas operações.

Este projeto de graduação tem como objetivo propor novas formas de clusterização das unidades da Bacia de Campos. O objetivo central proposto será de tentar minimizar ao máximo a distância total percorrida pelos barcos. Como rotas maiores representam maiores gastos com diesel para os barcos, então daí resultam configurações com menores custos. Da mesma maneira, com menores rotas, haverá um aproveitamento maior da frota de barcos de apoio, diminuindo o tamanho total da frota necessário. Isso reduzirá problemas de falta de atendimento devido à falta de barcos no porto e, no futuro, possibilitará inclusive o redimensionamento da frota.

Neste trabalho, pretende-se realizar um pré-processamento em que se procura otimizar as rotas percorridas pelos barcos, diminuindo-se a distância total percorrida. Não são consideradas na análise questões referentes ao tempo em que cada atividade ocorre, ou às janelas de tempo. Isso não será discutido a fundo aqui, mas fará parte de outros trabalhos referentes ao mesmo projeto de pesquisa.

Os resultados esperados são grandes grupamentos de clientes a serem usados em fases posteriores de resolução do problema. Graças a este pré-processamento, o trabalho posterior não precisará tratar o problema como um todo de uma só vez, possibilitando a obtenção de soluções em menor tempo computacional, tornando esta metodologia mais fácil de ser aplicada de forma operacional, quando são necessárias decisões rápidas e eficazes.

Este projeto de graduação está organizado da seguinte forma: primeiramente, no capítulo 2, é exposto um panorama geral de como ocorre o processo produtivo ao longo da cadeia de produção do petróleo. O objetivo é apresentar todas as principais demandas ao longo das etapas desde a perfuração até a produção, para se que possa saber as dimensões do problema.

Em seguida, no capítulo 3, é realizada uma análise mais específica da Bacia de Campos. O objetivo é especificar como ocorre atualmente o abastecimento de unidades nessa região, para que se possa identificar os problemas, o que pode ser melhorado e as condições operacionais que não podem ser alteradas.

Com a análise da cadeia produtiva de petróleo já realizada e as condições específicas da Bacia de Campos mais bem conhecidas, no capítulo 4, determina-se qual é o problema que deve ser resolvido, especificando-se portanto os objetivos do estudo.

A partir do problema determinado, em uma fase seguinte do projeto, no capítulo 5, é apresentada uma revisão da bibliografia sobre o tema. A metodologia utilizada para resolver o problema é apresentada no capítulo 6. Por fim, serão apresentados os resultados encontrados, acompanhados de uma breve discussão. O código desenvolvido é apresentado no Apêndice 1.

2 - As demandas na indústria do petróleo

O objetivo dessa seção é percorrer todas as fases de implementação e atividade de um projeto de produção de petróleo *offshore*. Serão explicadas todas as fases enfrentadas, desde a exploração, até a produção. Com isso, é possível apresentar um panorama dos equipamentos e suprimentos necessários ao longo de um projeto.

Pouco tempo após um bloco ser adquirido em um leilão da ANP, inicia-se sua fase exploratória. Esta fase tem como objetivo aumentar os conhecimentos sobre o campo, através de estudos que vão desde sísmicas até perfurações de poços exploratórios. Ao fim, espera-se que regiões onde possivelmente existiriam acumulações de petróleo tornem-se reservas prováveis e provadas, podendo compor o ativo da empresa proprietária do bloco.

Inicialmente, estudos de gravimetria e magnetometria podem ser realizados. Apesar de dados coletados com essas técnicas não apresentarem grande eficácia, são estudos fáceis de serem realizados, por vezes até através de aeronaves, não sendo, portanto, necessária uma grande preocupação logística.

Dentre os estudos realizados nesta etapa, destacam-se as sísmicas. Sísmicas caracterizam-se por serem operações que causam alguma perturbação acústica na água, para o caso *offshore*, ou na terra, para o caso *onshore*, com o objetivo de serem registradas cada reflexão do sinal ocorrida na passagem entre formações geológicas diferentes. O objetivo é que seja possível realizar um processamento dos tempos em que cada sinal é recebido, reconstruindo toda a sequência de deposição de sedimentos ocorrida na região estudada. A partir de estudos como esses, torna-se possível a identificação das possíveis regiões onde podem se localizar os reservatórios.

Sísmicas são realizadas por barcos dedicados a esta atividade. Segundo Thomas[4], para esta atividade, estes barcos apresentam linhas em sua parte posterior com hidrofones e canhões de ar comprimido. O maior desafio logístico nesta atividade reside na operação com estas linhas, que costumam se estender por centenas de

metros, tornando sua operação bastante complexa. Entretanto, em geral, as operações são relativamente rápidas, não sendo necessária uma operação muito complexa de abastecimento.

Com a região com possíveis acumulações de petróleo definida, as primeiras perfurações exploratórias são realizadas. Para a perfuração de um poço, torna-se necessária a contratação de uma sonda, que permanecerá na mesma posição por várias semanas realizando a perfuração do poço. Torna-se necessária, portanto, o constante abastecimento destas unidades para que haja o suprimento adequado de equipamentos e suprimentos. Em caso de necessidade de interrupção das atividades, os custos são muito grandes, já que alugueis de sondas são bastante caros.

A perfuração de um poço consiste na utilização de brocas rotativas que desgastam a formação a sua frente, tornando possível o aprofundamento do poço. A rotação da broca é possível graças a uma coluna conhecida como coluna de perfuração, que transmite a rotação desde a superfície até a broca. Há, portanto, a necessidade de robustos equipamentos na superfície, que sejam capazes de fornecer tanta energia quanto necessária para rotacionar toda a coluna de produção. Grandes quantidades de diesel são necessárias nessa etapa, além de equipamentos de manutenção.

Para a remoção do cascalho gerado assim como resfriamento da broca, utiliza-se o fluido de perfuração. Este segue por dentro da coluna de produção até a broca, passa pelos orifícios da broca e retorna com o cascalho para a superfície pelo espaço entre os tubos e o poço, o espaço anular. O fluido de perfuração também tem como finalidade a manutenção do estado de tensões no poço, já que a perfuração retira camadas rochosas de seu lugar e o fluido deve compensar essa retirada de material exercendo forças contra as paredes do poço, de peso tal que evite o seu colapso, quando ocorre desmoronamento do material rochoso por falta de peso, ou colapso do poço, quando o fluido avança e danifica a formação. Conforme se avança na perfuração, as geopressões (pressões encontradas em subsuperfície) variam de modo a exigir mudança na densidade do fluido para prosseguimento da perfuração. Torna-se necessário o constante fornecimento de fluidos e aditivos para torná-lo mais espesso conforme geopressões maiores são enfrentadas. Em caso de fluido a base de água, a operação é mais facilitada e a base do fluido pode ser a própria água do mar,

entretanto em caso de base óleo, até mesmo isso deve ser trazido por meio de barcos de apoio. Esta operação é em geral realizada por barcos fluideiros, que se dedicam à entrega de equipamentos para este tipo de operação.

A medida que aditivos são utilizados para tornar o fluido mais espesso, para atender a situações em que se necessita de um fluido com densidade grande o suficiente para suportar grandes pressões no fundo, pode ocorrer que a pressão gerada por este mesmo fluido resulte em danos em regiões mais acima no poço, já que a pressão gerada seria grande demais para as formações mais superficiais. Assim, é muito difícil, se não impossível, manter uma pressão no interior do poço que evite ao mesmo tempo o fluxo vindo da formação no fundo e não frature as partes mais rasas. Para evitar este problema, são utilizados revestimentos para proteção do poço e permitir que a operação continue. No final de cada revestimento, é assentada uma sapata, há a cimentação da região anular e prossegue-se posteriormente a perfuração com um diâmetro menor que passe pelo interior do revestimento anterior. Por isso, na fase de perfuração, é necessário um constante suprimento de tubos que possibilitem a instalação destas colunas e revestimentos. Eles são demandados em grandes quantidades e o espaço necessário nos barcos para seu transporte é bastante grande. Às vezes, devido a limitações de espaço na própria sonda, torna-se necessário ainda que barcos atuem como armazéns temporários, permanecendo próximos da sonda.

O principal equipamento de segurança dessa etapa é o *Blow Out Preventer* – BOP. Denomina-se como *kick* o fluxo de fluidos não desejados das formações para o espaço anular do poço, e *blowout* o fluxo incontrolável de fluidos da formação para a superfície, decorrente de um *kick* não controlado. O BOP é assentado na cabeça do poço visando isolar o mesmo do ambiente marinho, evitando contaminações no mar e maior segurança das operações. Permite também a comunicação da plataforma com o espaço central e anular do poço, de vital importância na atividade de "matar um poço" (injeção de lama pesada na operação para controle de *kick* e circulação dos fluidos) e medição das pressões em diversos pontos do poço. Trata-se de um equipamento bastante pesado e em geral já é levado pela própria sonda. Entretanto, em caso de necessidade de troca, a operação de transporte seria bastante complexa, além de exigir uma grande urgência.

Por fim, chega-se à fase de completação do poço. Nesta fase, são instalados os equipamentos no interior do poço que possibilitem a sua produção. Além disso, o BOP é trocado por uma Árvore de Natal, que caracteriza-se por ser um conjunto de válvulas colocado na cabeça do poço, sendo responsável por controlar a produção e prevenir acidentes. Nota-se que nesta fase uma série de equipamentos é demandada, gerando uma grande demanda por operações de transporte para a unidade.

Nota-se, portanto, que as atividades de perfuração apresentam uma grande heterogeneidade de demandas. Muitas operações são realizadas e cada uma demanda diferentes suprimentos, muitos deles em grande quantidade. A operação de uma sonda nunca pode ser interrompida, na medida em que custos de aluguel de sondas são bastante grandes. Isso, somado ao fato de sondas permanecerem em uma mesma posição por tempos relativamente pequenos, motiva que rotas de abastecimento a unidades de produção e de sondas sejam diferentes.

Com a declaração de comercialidade do campo, entra-se na fase de produção. Ainda nesta fase, outros poços vão sendo perfurados, até que se complete o planejamento do campo. Assim como explicado anteriormente, sondas são demandadas e a operação de perfuração é realizada quantas vezes forem necessárias.

Para poços já em produção, a operação é menos complexa. Em geral, ao longo da vida produtiva de um campo, as demandas de uma unidade de produção não se alteram substancialmente. São necessários suprimentos para os funcionários, equipamentos para substituição e diesel para os equipamentos, por exemplo. As únicas exceções ocorrem em casos de necessidades de intervenção do poço, quanto de novo é demandada uma sonda e repete-se operações semelhantes a sua perfuração.

3- A operação na Bacia de Campos

Como observado na seção anterior, as demandas das unidades durante suas operações são bastante grandes e diversas, sendo necessário uma grande quantidade de barcos para seu abastecimento e suporte. O objetivo dessa seção é especificar como isso ocorre na prática, explicando-se como as operações são organizadas na Bacia de Campos.

3.1 - Procedimentos gerais realizados

O primeiro passo para o abastecimento das unidades é o transporte da carga desde os armazéns até o porto. Em geral, a maioria dos armazéns localiza-se já em Macaé e esta fase não apresenta muitas dificuldades nem custos elevados.

Em geral as demandas são levadas por terra até um dos 3 portos utilizados pela Petrobras: o Porto de Imbetiba, o Porto de Vitória e o Porto do Rio de Janeiro. Dentre estes, o que concentra a maior parte das atividades é o Porto de Imbetiba, em Macaé, na medida em que este é um porto próprio da Petrobras e encontra-se relativamente próximo das unidades. A área para armazenamento da produção antes do embarque é bastante limitada, por isso todo o procedimento de chegada da carga e embarque nos barcos deve ser realizada de forma bastante rápida.

Com a chegada do barco ao porto, inicia-se o processo de embarque da carga. A interface entre o porto e os barcos é realizada através de berços, que são prolongamentos construídos para o acesso aos barcos. Eles devem ter uma infraestrutura básica para as operações, tais como guindastes, linhas de fluidos e área disponível. A quantidade máxima de barcos que podem ser atendidos ao mesmo tempo depende da quantidade de berços disponíveis no porto. O Porto de Imbetiba possui 6 berços disponíveis.

A quantidade de carga transportada é medida em número de lingadas, ou seja, o número de operações de levantamento do guindaste para a colocação de carga no deck. O tempo total gasto dependerá do tempo por lingada característico do porto.

Com o barco carregado, este realiza rotas percorrendo todas as unidades que devem ser atendidas por este barco. Em geral, em uma mesma rota não existem sondas e plataformas, devido às diferenças que existem nas demandas entre esses dois tipos de unidade além das constantes mudanças de posição das sondas, que ocasionariam dificuldades no planejamento das rotas para um horizonte maior.

Segundo Leite[3], as demandas das unidades não apresentam padrões certos, variando bastante de uma unidade para a outra, mesmo que estas sejam plataformas do mesmo tipo e muito parecidas. Em geral, cada plataforma ou sonda apresenta demandas bastante específicas e características, que refletem a maneira em que a produção é realizada e os problemas específicos enfrentados em cada tipo de campo. Durante o ano, pequenas variações podem ser visualizadas no padrão de demandas, entretanto a pequena diferença entre estes valores faz com que possa ser desconsiderado este aspecto.

Em média, segundo Leite[3], as demandas de cada unidade são como mostrado na Tabela **1**. Em geral, as demandas já são registradas em área de deck demandada. Isso é feito porque, na prática, o limite de capacidade de um barco não ocorre por causa do peso total da carga exceder o limite do barco, mas sim por não haver área suficiente para novas cargas. Inclusive, toda a análise feita neste trabalho quanto ao rompimento da capacidade do barco será feita exclusivamente baseada na área de deck demandada.

Tabela 1. Demandas médias de uma unidade

	Backload		Load	
	Média da área de deck(m^2)	Desvio padrão	Média da área de deck(m^2)	Desvio padrão
Unidades especiais	35	25	22	20
Unidades de produção	156	112	133	54
Sondas	192	159	195	171

Para a maioria das demandas cotidianas das unidades o transporte ocorre como o descrito. A única exceção ocorre para o transporte de diesel. Para este caso, existe um

navio reservatório de diesel mais próximo das unidades que funciona como um *hub*. O diesel é levado até ele e armazenado. Barcos menores fazem o transporte do combustível dele até as unidades. Como este tipo de atividade ocorre de maneira bastante independente do transporte dos outros suprimentos, então este será desconsiderado em nossa análise.

As unidades apresentam janelas de tempo fixas, dentro das quais deve ocorrer o descarregamento e recolhimento de carga nelas. Isso é necessário pois o atendimento a unidades por barcos altera por completo a rotina da plataforma ou da sonda. Por exemplo, equipes de funcionários devem prestar suporte ao descarregamento de carga, não se pode haver trocas de turnos de funcionários e atividades com mergulhadores são suspensas. Caso o barco chegue fora da janela de tempo ou não consiga cumprir todo seu trabalho dentro desta, pode ser necessário que este retorne outra hora ou espere até poder realizar suas atividades. Na realidade, esta condição é bastante crítica, sendo responsável na prática por diversos casos de não atendimento à unidade ou necessidade de novas visitas posteriormente.

Hoje em dia, o Porto de Imbetiba encontra-se no limite de sua capacidade, com disponibilidade de berços ao longo do dia bastante pequena, haja vista a grande quantidade de atividades de carga e descarga que já ocorrem diariamente.

3.2 - Recursos disponíveis para o transporte

O objetivo desta subseção é descrever o que há de disponível para levar os suprimentos em terra para o mar, quantificando e caracterizando todos os tipos de embarcações disponíveis.

Segundo Leite[3], os barcos podem ser usados para várias atividades. Muitos deles prestam serviços à atividade das unidades, atuando, por exemplo, no reboque e atracação de plataformas, atracação de unidades flutuantes de produção e plataformas, inspeções de unidades *offshore*, suporte para *offloading* de FPSOs, auxílio em operações de emergência onde há a necessidade de recolhimento de óleo em mar ou

resgate, suporte em projetos de construção *offshore*, operações de estimulação de poços, serviços de mergulho e instalações de dutos.

Considerando-se as operações de serviço, alguns tipos de barco destacam-se e devem ser definidos. Os barcos do tipo *Line Handling* (LH) caracterizam-se por atuarem auxiliando a manipulação de linhas e dutos. Os barcos responsáveis pelo lançamento e instalação de dutos no mar são chamados de *Pipe Laying Support Vessel* (PLSV). Os barcos do tipo *Tug Supply (*TS) caracterizam-se por serem rebocadores, apesar de também poderem exercer as mesmas funções dos barcos responsáveis por levarem suprimentos às unidades. Os barcos do tipo *Anchor Handling Tug Supply*(AHTS) caracterizam-se por serem navios de reboque e de manuseio de âncoras, podendo exercer funções de ancoragem, auxílio à construção e instalação e amarração de unidades no mar, por exemplo. Em caso de operações com mergulhadores, existem os barcos do tipo *Diving Support Vessel* (DSV), que contam com estruturas e equipamentos necessários para o mergulho. Para regiões mais profundas, onde há a necessidade da utilização de ROVs, que são robôs submarinos responsáveis por possibilitar operações submarinas à distância, são utilizados ROV *support vessels* (RSV). Para as operações de limpeza e estimulação de poços, são utilizados *Well Stimulation Support Vessels* (WSSV). Para o suporte nas operações com equipamentos subsea, são utilizados *Subsea Equipament Support Vessels* (SESV). Os barcos do tipo *Oil Spill Response Vessel* (OSRV) caracterizam-se por serem embarcações que ficam em constante *stand by*, atuando na contenção e combate a derramamentos de óleo caso algum acidente ocorra. Da mesma maneira, em caso de incêndio em alguma unidade, existem os barcos do tipo *Fire Fighting*, que ficam na maior parte do tempo em espera, deslocando-se até alguma unidade apenas em caso de incêndio, combatendo-o com jatos com altas vazões de água.

Apesar de sua importância, os barcos que prestam serviço não serão o foco deste trabalho. Devido a seu caráter esporádico ou atuação apenas em períodos de emergência não existe motivos para fazer a programação destas embarcações.

Além das operações de serviço, destacam-se também os barcos que prestam serviços de logística. Nesse tipo de operação, estuda-se o transporte de carga para unidades, o

offloading de cargas vindos de unidades, o transporte de resíduos, o transporte de pessoas, a estocagem e o transporte em terra.

Dentre os barcos que prestam serviços de logística, destacam-se os barcos do tipo PSV, UT e P. Os barcos PSV (Plataform Supply Vessel) caracterizam-se por serem barcos com grandes espaços em convés responsáveis pelo transporte de cargas até unidades, de unidades ou entre unidades. Já os barcos do tipo UT (Utility) transportam cargas de deck de forma emergencial, mas possuem restrições de peso e tamanho de carga. Os barcos do tipo *Passenger boat* (P) caracterizam-se por serem barcos relativamente rápidos, utilizados para trocas de tripulação, além de transporte de pequenas quantidades de combustível, água, dentre outros equipamentos.

Dentre essas atividades, o trabalho focará na atividade dos barcos que prestem serviço de logística, sobretudo nos barcos do tipo PSV, devido ao seu grande número e periodicidade das atividades, que proporcionam maior dificuldade e complexidade no planejamento e programação das rotas.

Tanto cargas de sondas quando de unidades de produção devem ser transportadas por PSVs. Apesar de ambos os tipos de carga terem como objetivo manter o funcionamento das unidades levando equipamentos e suprimentos, elas são bastante diferentes entre si. As demandas de sondas são em geral bastante heterogêneas, variando de acordo com a operação que está sendo executada, enquanto que as demandas de unidades de produção são em geral constantes já que as atividades executadas nesses tipos de unidades não costumam variar de acordo com o tempo.

As cargas transportadas pelos barcos PSV podem ser levadas sob o deck ou em compartimentos abaixo deste. Segundo Leite[3], a carga de deck é bastante heterogênea, podendo-se haver desde sacos e tanques, até containers e containers refrigerados para comida, além de equipamentos de diversos tamanhos. De acordo com Leite[3], esta carga pode ser subdividida em carga geral, comida e água para consumo humano, dutos, risers, químicos e resíduos. Abaixo do deck, existe uma outra infinidade de cargas que podem ser transportadas em compartimentos e tanques, tais como água industrial, diesel, fluidos (salmoras e lamas de perfuração) e volume seco (cimento, barita e bentonita).

Os PSVs podem ser divididos em 3 classes de acordo com o peso máximo que são capazes de transportar. Barcos do tipo PSV 1500 conseguem transportar até 1500 toneladas, os do tipo PSV 3000 transportam até 3000 toneladas e os do tipo PSV 4500 transportam até 4500 toneladas.

As características médias de cada barco, segundo Leite[3] estão expressas nas Tabela 2 e Tabela 3. Constata-se, portanto, a grande capacidade e menor velocidade de barcos do tipo PSV comparado com uma menor capacidade e maior velocidade dos barcos do tipo *Utility*. Por isso, em geral, os barcos do tipo PSV são utilizados no cotidiano, já que, com eles, é possível formar rotas maiores, com mais unidades, resultando-se em um custo total menor. Já os barcos do tipo *Utility* são utilizados, sobretudo, em situações de emergência, em que pequenas cargas precisam chegar ao destino em pouco tempo.

Tabela 2. Características gerais dos barcos PSVs

Tipo de barco	Área de deck(m^2)	Área útil de deck(m^2)	Comprimento (m)	Largura (m)	Calado máximo (m)	Resistência do deck (t/m^2)	Capacidade de água (m^3)	Vazão de água (m^3/h)	Velocidade (Knot)
PSV 1500	394	295	62	15	4.7	4,0	565	96	10,0
PSV 3000	575	431	70	16	5,7	5,1	972	68	10,0
PSV 4500	880	660	87	18	6,2	5,0	1379	81	10,2
UT	183	137	50	9	2,9	2,3	85	42	20,2

Tabela 3. Características de consumo de diesel dos barcos PSVs

Tipo de barco	Capacidade de diesel (m^3)	Vazão de diesel (m^3/h)	Consumo em serviço (ton/dia)	Consumo parado (ton/dia)	Consumo no porto (ton/dia)
PSV 1500	485	93	11,4	4,3	1,1
PSV 3000	652	70	13,3	4,3	1,3
PSV 4500	1832	91	16,7	5,7	1,3
UT	66	42	21,8	2,4	0,8

3.3 - Atual política adotada na Bacia de Campos

Segundo Leite[3], para uma melhor organização, são atribuídos códigos aos tipos de viagens realizadas na Bacia de Campos de acordo com o tipo de unidade atendida ou grau de emergência. Viagens do tipo P são viagens regulares para unidades de produção. Viagens do tipo R são as viagens regulares para sondas ou de transporte de emergência para plataformas. Viagens do tipo RF são viagens regulares para sondas para transporte de comida, bebidas e água industrial. Viagens do tipo E são viagens de emergência. Viagens do tipo S são viagens regulares para embarcações especiais. Por fim, viagens do tipo D são viagens montadas sob demanda, sobretudo para o transporte de *risers.*

Segundo Leite[3], em 2011, os clusters da Bacia de Campos organizaram-se como descrito na Tabela **4**, baseando-se nos códigos definidos.

Tabela 4. Organização das viagens na Bacia de Campos em 2011

Código	Unidades atendidas	Tipo de carga	Barco	Número de clusters	Viagens por semana por cluster	Número de unidades por cluster
P	Unidades de produção	Carga de deck e água	PSV 3000	3	2	7 a 8
			PSV 4500	3	1	5 a 8
R	Sondas	Carga de deck e água	PSV 1500 PSV 3000	4	7	3 a 8
R	Unidades de produção	Carga de deck emergencial				11 a 17
RF	Sondas	Comida, água e água industrial	PSV 3000 PSV 4500	2	1	11 a 15
E	Todos os tipos de unidades	Carga de deck emergencial	UT	-	7	-
S	Unidades especiais	Carga de deck e água	PSV 3000 PSV 4500	1	1	-
D	Todos os tipos de unidade	Risers	PSV	Sob demanda	Sob demanda	-

Apesar de um aparente bom planejamento, vários problemas podem ser detectados na Bacia de Campos. Existem muitas situações em que uma unidade é visitada mais de uma vez em uma mesma viagem. Da mesma maneira, há um costume de classificar as

unidades como emergenciais não havendo a real necessidade disso, com o objetivo de apenas apressar a visita dos barcos às unidades.

Existem tentativas de melhora deste panorama, entretanto estas esbarram em alguns problemas. Segundo Valente et al.[5], a complexidade do problema leva a procedimentos empíricos, ao passo de que os avanços em tecnologia da informação são adotados lentamente pelas empresas e os responsáveis pela tomada de decisão acabam não aceitando novas técnicas e resiste em mudar as maneiras como trabalham hoje em dia.

4 - Descrição do problema

Percebe-se, portanto, que o abastecimento às unidades *offshore* da Bacia de Campos enfrenta um problema bastante complexo. Sofre-se com práticas empíricas e a desorganização da programação é bastante grande, levando-se a não atendimento a unidades, atrasos e a casos de visitas desnecessárias. Enquanto isso, tem-se um porto sobrecarregado, com perspectivas de aumento de unidades atendidas no futuro, com a entrada em operação de mais unidades e o desenvolvimento do Pré-Sal. Há, portanto, um cenário de perda de dinheiro devido à falta de uma otimização do porto e de necessidade de mudanças haja vista a maior utilização do porto no futuro. Portanto, cria-se a necessidade de estudos que possam indicar caminhos para a melhora das operações, o que será estudado neste trabalho.

Assim, o objetivo da pesquisa consiste em encontrar formas de otimizar o roteamento e a programação do porto e das atividades a serem realizadas. Pretende-se encontrar rotas que otimizem a distância total percorrida, ao mesmo tempo em que consigam otimizar o tempo de carregamento e descarregamento no porto e respeitem a todas as janelas de tempo e outras restrições das unidades atendidas.

Existem, portanto, dois problemas principais que devem ser observados: otimizar a programação do porto para que este deixe de ser um gargalo para as atividades e melhorar a confiabilidade das entregas nas unidades, garantindo-se que os barcos cheguem até elas dentro da janela de tempo. Nesse trabalho, o tamanho e características da frota de barcos já existente são considerados dados, na medida em que esta frota já encontra-se atualmente contratada pela Petrobras. Apesar disso, considerações finais do trabalho poderão sugerir mudanças nesta frota a longo prazo, visando otimizar sua utilização.

Depara-se, portanto, com um problema de roteamento, com frota heterogênea e janelas de tempo. Apesar de existir mais de um porto, não há a necessidade de trabalhar com um problema multi-depósito, na medida em que as operações de cada porto são razoavelmente isoladas, de forma que uma unidade nunca será atendida por barcos provenientes de mais de um porto.

Entretanto, o tempo computacional exigido para resolver o problema como um todo, considerando-se todas as unidades, seria grande demais. Por esse motivo, uma alternativa consiste em trabalhar apenas um conjunto menor de unidades por vez. Para dividir o conjunto de todas as unidades em subconjuntos, menores, é realizado um pré-processamento, usando um algoritmo de clusterização.

Para resolver o problema completo de programação do porto e definição das rotas, montou-se uma equipe de pesquisa e a resolução do problema foi dividida entre seus integrantes. Parte é responsável pela solução de um problema de clusterização, com o objetivo de realizar um pré-processamento para as fases seguintes, facilitando-se o problema e diminuindo o tempo computacional total necessário. Outra equipe é responsável pelo roteamento, programando-se o tempo de cada atividade, com o objetivo de montar a programação do porto e observar o atendimento dentro das janelas de tempo. Por fim, uma equipe de simulação trabalha com o objetivo de avaliar a confiabilidade do sistema, simulando-se de forma estocástica o tempo em que cada atividade ocorre, para verificar qual a probabilidade de chances de não atendimento às unidades. Este trabalho tem foco na parte da pesquisa responsável pela clusterização.

O objetivo da clusterização é o de separar as unidades em grupos, de acordo com um objetivo determinado. Serão desconsiderados qualquer relação com o tempo, janelas de tempo e programação do porto.

O objetivo principal pretendido será encontrar a melhor forma de se agrupar e ordenar as unidades dentro de grupos de forma a obter a configuração com a menor distância total percorrida pelo barco. Isso representará, portanto, um menor tempo de viagem total demandado e um menor consumo de diesel possível.

Apesar de isso não ser o objetivo final pretendido para a pesquisa, é um pré-processamento necessário. O roteamento feito posteriormente para encontrar as rotas que otimizem o atendimento às janelas de tempo não é capaz de analisar todas as unidades de uma única vez. Processando apenas uma parte de todas as unidades por vez é impossível a construção de rotas entre unidades de grupos diferentes. Por isso,

precisa-se encontrar formas de dividir as unidades e fazer a cada vez processamento de unidades mais próximas e com boas rotas possíveis é algo interessante. Caso isso não fosse realizado, seria possível que, para atender ao critério de atendimento a janelas de tempo, grandes distâncias fossem percorridas e as rotas acabassem em um padrão de “zig-zag”.

De certa forma, o trabalho empreendido nesta fase de clusterização também auxiliará na análise do impacto que a obrigação de atender as unidades dentro de suas janelas de tempo causa ao problema. Apesar de janelas de tempo não serem possíveis de serem alteradas cotidianamente, caso o estudo detecte que sua alteração impactaria muito positivamente, uma redefinição destas pode ser sugerida a longo prazo.

Portanto, com o objetivo já definido, encontrar uma clusterização cuja configuração apresente a menor distância total percorrida, será objetivo das seções posteriores deste trabalho realizar uma revisão sobre a bibliografia relacionada ao tema, selecionar o melhor método possível e aplicá-lo.

5 - Revisão bibliográfica

O problema estudado engloba uma série de assuntos bastante diversos. Vários campos podem ser pesquisados para o trabalho proposto, tais como os problemas de coleta e entrega, roteamento e clusterização. Existem também algumas especificidades do problema já estudadas pela literatura, tais como a condição de multi-depósito (mais de um porto, no caso), a limitação dada por janelas de tempo e a existência de uma frota heterogênea. Há, portanto, bastante material na literatura, entretanto dificilmente existe algum material que reúna de forma unificada todos estes assuntos.

Exemplos de como a operação ocorre em diferentes partes do mundo são citados em vários artigos e várias proposições são realizadas.

Aas et al[6] fazem uma análise partindo de um problema de roteamento real de suprimentos de bases *onshore* para as unidades *offshore* na Noruega e propõe uma versão simplificada de um problema de roteamento de veículos formulada com modelos de programação inteira mista. É considerado o problema de coletas e entregas e a restrição à capacidade dos barcos, assim como ocorre no caso estudado. O objetivo é encontrar formas de determinar rotas com o comprimento mínimo. Apesar de bastante similar ao problema estudado, este modelo ainda carece de algumas especificidades, tais como a necessidade do atendimento às janelas de tempo.

Artigos dos mesmos autores foram publicados em anos posteriores, refletindo mais análises sobre casos semelhantes. Em 2008, Aas et al.[7] realizaram uma análise sobre o roteamento de barcos de apoio, e concluíram que em outras partes do mundo, assim como no caso brasileiro, também não é dada a devida importância ao tema, os operários possuem poucos conhecimentos em logística e em geral as decisões são tomadas com base na experimentação. Em 2010, Aas&Wallace[8] apresentaram um outro caso semelhante, só que da Statoil em Kristiansund. Em 2009, Aas et al.[9], apresentaram um artigo focado em capacidades e configuração da frota, o que não chega a ser interessante na análise proposta por este trabalho, já que admite-se que a frota já foi previamente contratada e não passará por mudanças.

Outros autores também realizaram estudos sobre problemas semelhantes em diversas partes do mundo. Tanto Romero et al.[10] quanto Gribkovskaia et al.[11] realizaram também a montagem de modelos quantitativos para a movimentação de barcos e helicópteros para o caso do abastecimento à unidades *offshore.*

Kaiser[12] realizou uma análise sobre a condição até então do Golfo do México e propôs um modelo para quantificar e prever níveis de serviço nas unidades. Este era baseado em um sistema determinístico com tempo invariante e linear, diferentemente dos outros autores, este teve como foco em seu artigo exatamente o objetivo central proposto neste trabalho, ou seja, otimizar o atendimento às janelas de tempo e elevar o nível de serviço. Ainda em 2010 Kaiser&Snyder[13] também publicaram um estudo sobre o número de viagens necessárias para as unidades *offhsore* do Golfo do México, propondo sua otimização.

O problema encontrado na Bacia de Campos pode ser analisado também como um problema de roteamento de veículos periódico. Muitos artigos analisam problemas deste tipo. Por exemplo, Fagerholt & Lindstad[14] propõem um modelo de roteamento periódico utilizando programação inteira mas sua análise tem como foco o custo financeiro devido a unidades fechadas durante a noite.

Halvorsen-Weare & Fagerhold[15] utilizaram modelos de programação robusta para determinar a frota ótima, a programação e as rotas, considerando este um problema periódico. São consideradas janelas de tempo, frota heterogênea, assim como o problema analisado neste trabalho. Entretanto, diferentemente do caso brasileiro, existe um tempo pré-determinado de abertura do porto, além de que a instabilidade climática presente no ambiente analisado pelo artigo faz com que soluções determinísticas não sejam aceitáveis. A função objetivo tinha como objetivo minimizar o custo.

Shyshou et al.[16] também trabalharam com problemas de roteamento periódico e montaram uma heurística para resolver problemas do tipo, além de realizar uma análise sobre a composição ótima da frota. Panarenka[17] foi o responsável por definir uma heurística de busca local com otimização de velocidade, tendo como objetivo otimizar custos e aspectos ambientais.

Muitos autores inclusive estudaram a problemática da Bacia de Campos. Christiansen et al.[18] discutem alguns aspectos do problema de programação que são encontrados na Bacia de Campos, apontando problemas, como por exemplo a resistência na adoção de sistemas baseados na otimização. Batista[19] focou sua análise no Porto de Imbetiba, propondo um modelo para simular as operações neste porto.

Leite[3] também fez uma grande análise sobre toda a cadeia de transporte de cargas para unidades na Bacia de Campos. Em seu texto, é detalhada toda a sequência de operações praticada na Bacia de Campos, definindo-se quais os principais problemas enfrentados e quais as principais melhorias demandadas. Além disso, é realizada uma revisão completa da literatura sobre o tema, compilando-se resumos de vários artigos que debatem temas relacionados. Também podem ser encontrados muitos dados referentes às operações na Bacia de Campos, além de uma extensa análise sobre estes.

Além de uma extensa parte descritiva das atividades, Leite[3] também propõe uma nova modelagem para o problema estudado. A heurística adotada é composta por quatro fases: na primeira fase é realizada uma clusterização segundo o método de Clarke & Wright. Em seguida, em uma segunda fase, é criada uma grade de tempo das atividades. Em uma terceira fase, a demanda histórica é alocada nas viagens de acordo com a grade de tempo, ainda sem considerar a limitação devido à capacidade do barco. Por fim, é realizada uma simulação para determinar o tempo necessário para se chegar a cada unidade, o tamanho necessário para a frota e sua composição. O objetivo central deste método é otimizar as viagens, diminuindo-se o número total de viagens e o tempo perdido em cada uma. Deseja-se também garantir o nível de serviço, respeitando-se janelas de tempo.

Muitos destes artigos apresentam uma visão geral sobre como gerenciar e projetar o transporte de cargas à unidades *offshore*. Entretanto, o foco deste trabalho é a definição de uma estratégia de clusterização eficiente. Por isso, foi dada grande importância à pesquisa por um método eficiente e que se adeque às condições do problema estudado. Diferentemente dos artigos até aqui analisados, não se procurou um método já utilizado no ambiente de produção de petróleo, mas sim um algoritmo de clusterização genérico, mas que fosse suficientemente prático e eficiente. Após

uma extensa análise, definiu-se, por fim, que o método utilizado seria o definido por Koskosidis&Powell[20].

O artigo de Koskosidis&Powell[20] propõe a clusterização de entidades com demandas pré-determinadas entre veículos com capacidade também pré-determinada. O artigo propõe formas de levar em conta não só a distância viajada, mas também o nível de serviço. A metodologia insere-se no contexto de problemas CCP (*Capacitated Clustering Problem*), podendo ser encarado como um subproblema de um VRP (Problema de roteamento de veículos).

A formulação trabalha com o seguinte conjunto de parâmetros, variáveis e conjuntos especificados a seguir.

I: Conjunto de consumidores i
J: Conjunto de consumidores candidatos a *seed* j
K: conjunto de veículos k
X_k: conjunto de consumidores i atendidos pelo veículo k
c_{ij}: custo de viagem entre i e j
q_i: demanda de carga do consumidor i
V: capacidade do veículo
y_{ij}: variável binária igual a 1 caso a unidade i pertença ao *seed* j
g_i: variável binária igual a 1 caso o consumidor j seja um *seed*

A função objetivo do método pode ser visualizada em [1]. Ela consiste em minimizar a distância das unidades até *seeds*, que são as unidades centrais de cada *cluster*. Ou seja, tem-se como objetivo tornar os *clusters* homogêneos, de forma que todas as unidades encontrem-se nas proximidades.

$$\sum\sum\left[\begin{pmatrix}custo\\(proximidade)\\entre\\i\ e\ seed\ j\end{pmatrix}\begin{pmatrix}unidade\\i\ faz\ parte\\do\ cluster\\do\ seed\ j\end{pmatrix}\right] \quad [1]$$

$$\min F(y,g) = \sum_i \sum_j c_{ij} y_{ij}$$

Neste modelo, tem-se como restrições que as demandas de cada cluster não podem exceder a capacidade do barco que o atende [2], a condição de que cada unidade poderá ser associada a somente um *cluster* [3], a condição de que cada unidade deve estar obrigatoriamente ligada a algum *seed* [4], ou seja, a algum *cluster* oficialmente existente, e a condição de que deve-se existir exatamente um número definido de *seeds* ou seja, de *clusters* [5]. Por fim, dentre as restrições, inclui-se também a definição das variáveis binárias [6].

$$\sum \left[\begin{pmatrix} \text{demanda} \\ \text{da} \\ \text{unidade } i \end{pmatrix} \begin{pmatrix} \text{binário:} \\ \text{unidade } i \\ \text{faz parte} \\ \text{do cluster} \end{pmatrix} \right] \leq \begin{pmatrix} \text{capacidade} \\ \text{do veículo} \end{pmatrix} \qquad \sum_{i \in \mathcal{I}} q_i y_{ij} \leq V \qquad \forall\, j \in \mathcal{J} \qquad [2]$$

$$\sum_{\substack{\text{para} \\ \text{todos} \\ \text{seeds}}} \begin{pmatrix} \text{binário:} \\ \text{unidade } i \\ \text{está atrelada} \\ \text{a seed } j \end{pmatrix} = 1 \qquad \sum_{j \in \mathcal{J}} y_{ij} = 1 \qquad \forall\, i \in \mathcal{I} \qquad [3]$$

$$\begin{pmatrix} 1 = i \text{ faz parte do cluster } j \\ 0 = i \text{ não é atendido por } j \text{ porque cluster } j \text{ não existe} \end{pmatrix} \leq \begin{pmatrix} 1 = j \text{ é seed} \\ 0 = j \text{ não é} \end{pmatrix} \qquad y_{ij} \leq g_j \qquad \forall\, i \in \mathcal{I}, j \in \mathcal{J} \qquad [4]$$

$$\sum (J \text{ é um seed}) = \begin{pmatrix} \text{número de seeds} \\ \text{desejado} \end{pmatrix} \qquad \sum_{j \in \mathcal{J}} g_j = K \qquad [5]$$

$$(y_{ij},\, g_j) = (0,1)\ \forall\, i \in \mathcal{I}, j \in \mathcal{J} \qquad [6]$$

Segundo Koskosidis&Powell[20], existe ainda uma forma de implementar de forma iterativa a heurística de clusterização apresentada. Esta maneira de aplicar o método será utilizada neste trabalho e sua descrição encontra-se na próxima seção.

6 - Metodologia adotada

O método proposto por Koskosidis&Powell[20] também pode ser adotado de uma forma iterativa. Segundo Koskosidis&Powell[20] esta forma de implementação garante soluções mais rápidas para problemas mais complexos, tais como o problema estudado aqui, que envolve a clusterização de dezenas de unidades. Devido a isso e a maior facilidade de adoção desta forma de implementação em linguagens de programação convencionais, preferiu-se trabalhar desta maneira.

A abordagem iterativa do método consiste em três fases. É importante frisar que neste trabalho estas fases foram implementadas em uma ordem diferente do que a proposta no artigo, apenas por critérios de simplicidade e facilitação na obtenção de resultados nas fases anteriores à finalização do trabalho. Na realidade, como se trata de um método iterativo que consiste na repetição de um conjunto de ações repetidamente a ordem que cada ação é realizada perde sua importância. Entretanto, devido a esta troca de ordem nos procedimentos, precisou-se dar uma maior atenção à saída, que desta vez foi registrada em uma das fases intermediárias e não no fim do programa. Nos próximos parágrafos, será descrito cada fase do método na ordem em que foram implementadas neste trabalho.

A primeira fase consiste em tentar melhorar uma solução inicial já utilizada. Parte-se de uma solução inicial para que o processo seja otimizado e chegue-se mais facilmente a uma boa solução, apesar do método admitir qualquer entrada, ainda que uma clusterização gerada aleatoriamente. Para este trabalho, foram obtidos conjuntos de dados de clusterizações já realizadas no passado e tentou-se melhorá-las. Na prática, em caso de utilização desta metodologia no dia-a-dia das operações, esta solução inicial pode partir de clusterizações anteriores, acrescidas de modificações das propriedades das unidades que possam existir.

Esta primeira fase consiste em fazer todas as trocas de posições possíveis entre duas unidades, com o objetivo de se encontrar a configuração com o menor custo, neste caso, a menor distância total percorrida pelos barcos. Particularmente neste trabalho, foi realizada uma modificação no algoritmo proposto no artigo original. Na busca por melhores soluções após a realização de todas as trocas em pares possíveis e obtenção

de uma solução de custo mínimo, repete-se este processo com esta nova solução mínima, até uma situação em que novas trocas não resultem em melhores resultados, ou até se chegar a um número máximo de repetições pré-determinado.

Neste momento do programa, chega-se à melhor solução possível da iteração. A partir daqui, serão realizadas perturbações na solução com o objetivo de prepará-la para o processamento na iteração seguinte. Não se garante, portanto, que soluções melhores serão encontradas. Por isso, a solução de cada iteração é registrada neste ponto e os dados finais de saída do programa consequentemente também sairão daqui.

Ao realizar trocas em pares, não são percorridas todas as soluções possíveis. Para uma maior eficiência do método, novos mínimos devem ser visitados. Assim, passa-se à próxima fase do algoritmo, que consiste em aumentar o horizonte de soluções possíveis de se visitar induzindo-se grandes alterações na clusterização.

O primeiro passo desta fase consiste na determinação de *seeds*. Para isso, a unidade mais próxima do centro geométrico dos clusters antigos transforma-se nos *seeds* que serão utilizados nesta etapa. A partir destes, novos *clusters* serão criados ao seu redor. Ou seja, tem-se como objetivo a criação de novos *clusters*, que serão formados ao redor do centro dos *clusters* antigos.

Com os *seeds* determinados, passa-se a fase de definição do *cluster* a que cada unidade pertencerá. O objetivo é atribuir cada unidade ao *cluster* cujo *seed* encontra-se mais próximo. A ordem em que isso é feito com as unidades é determinada por uma função denominada *Regret*. O objetivo desta função é determinar quais são as unidades em que a atribuição a *seeds* mais próximos é mais emergencial e em que a escolha por *clusters* mais longínquos representaria maiores perdas. Estes deverão ser clusterizados primeiramente e, por isso, não enfrentarão problemas relacionados à impossibilidade de atribuição a algum *cluster* devido a restrições, como, por exemplo, as relacionadas à capacidade do barco que atenderá a este *cluster*.

A função *Regret* é definida então como a diferença de custos, ou seja de distâncias, entre o seed mais próximo e o segundo *seed* mais próximo, como descrito a seguir:

$$Regret(i) = C_{ij^2} - C_{ij^1}$$

Onde j^2 é o segundo *seed* mais próximo da unidade i e j^1 é o primeiro *seed* mais próximo. Portanto, esta função representa a penalidade pela escolha do segundo seed mais próximo ao invés do primeiro. Deseja-se saber, portanto, quais são as unidades que estão bem próximas de um *cluster* mais muito longe de outros, devendo, portanto, ser atribuída preferencialmente ao *cluster* mais próximo para que seja evitada a criação de rotas com distâncias totais muito grandes.

Seguindo-se a ordem de prioridades definida pela função *Regret*, atribui-se cada unidade a um *cluster*. Caso haja disponibilidade de atendimento ao *cluster* mais próximo, este é escolhido. Caso isso não seja possível devido a restrições tais como de capacidade do barco, número máximo de unidades por *seed* ou janelas de tempo muito diferentes, então este é atribuído a clusters cada vez mais longes. Caso seja impossível realizar o procedimento com todos os *clusters*, um novo *cluster* é criado.

Com a nova clusterização definida, o processo é repetido, e trocas de posições de unidades em pares são realizadas tentando-se melhorar a configuração atual.

Nas Figura 4 e Figura 5, são apresentados fluxogramas resumindo o procedimento realizado pelo algoritmo.

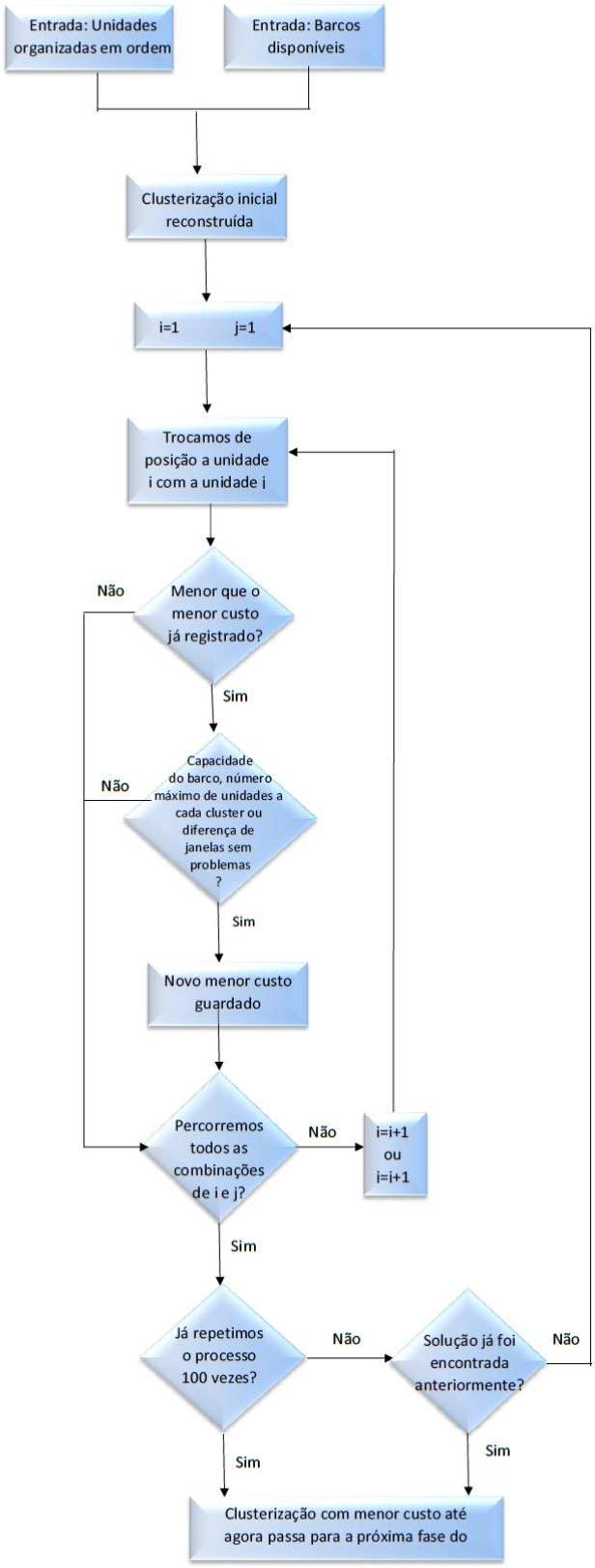

Figura 4. Fluxograma da primeira parte do método.

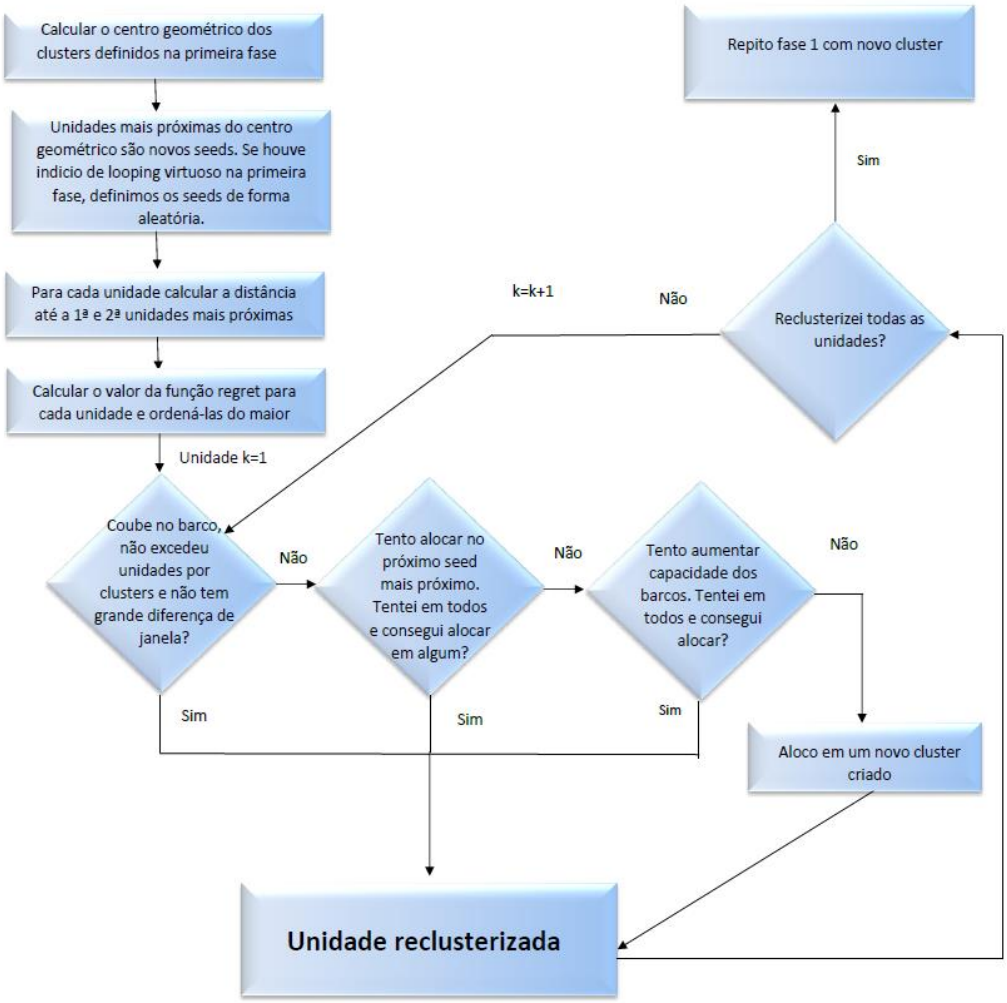

Figura 5. Fluxograma da segunda parte do método.

7 – Experimentação computacional

Com a metodologia apresentada no Capítulo 6 vários casos diferentes foram executados no *software* Matlab. Inicialmente, optou-se por rodar casos com o menor número de restrições possíveis, para avaliar as reais potencialidades do método. Nas simulações seguintes, restrições foram sendo adicionadas, até se chegar ao caso real.

Os resultados expostos estão em dimensões de latitude e longitude. Na realidade, cada latitude representa 110 Km e longitudes variam de acordo com a distância até o Equador, de forma que, perto do Equador, esta comporta-se igual ás latitudes, representando 110Km, enquanto que, quanto mais próximo do polo, menor é esta distância. No problema estudado neste trabalho, trabalha-se com distâncias relativamente pequenas em relação ao raio da Terra e a Bacia de Campos pode ser considerada também relativamente próxima ao Equador. Por isso, diferenças nas distâncias que representem intervalos entre diferentes latitudes e longitudes foram desconsideradas. Optou-se, assim, por rodar todo o programa registrando distâncias em unidades de latitude e longitude, considerando-se, conforme exposto, a correspondência direta destas distâncias com as distâncias reais. Para realizar a conversão para a distância em quilômetros, basta-se multiplicar os resultados por 110.

Dentre as entradas desejadas do usuário, está o número de iterações máximo que se deseja fazer. Quanto maior o número de iterações, mais próximo do ótimo estará o resultado, entretanto, da mesma maneira maior será o tempo computacional demandado. Para todas as simulações, optou-se por utilizar um limite de 1000 iterações. O tempo demandado para rodar estes casos é relativamente grande para operações em campo, entretanto, para fins de pesquisa foi considerado ideal, sendo em torno de 2 horas.

Inicialmente, na primeira simulação, optou-se por não utilizar as restrições de diferença máxima de janelas de tempo dentro de um mesmo *cluster* e número máximo de unidades por *cluster*. Apesar de a falta destas restrições implicarem na inviabilidade do algoritmo de roteamento que será utilizado em fases posteriores da pesquisa, isso possibilitaria a avaliação do real potencial da metodologia adotada até

aqui. O programa foi rodado 1000 vezes, captando-se as soluções encontradas a cada iteração. Como saídas do programa foram fornecidas as 10 melhores soluções encontradas organizadas em uma planilha do software Microsoft Excel. O tempo demandado foi de 1 hora e 33 minutos, rodando-se o algoritmo no software Matlab em um computador com processador Intel Core 2 Duo 2 GHz, 3 GB de memória RAM e sistema operacional Windows 8.

A variação entre os resultados encontrados por iteração e o resultado vindo dos dados de entrada pode ser visualizados na Figura 6. Tratando-se de uma variação, os menores resultados são aqueles que possuem soluções com distâncias totais percorridas menores, sendo, portanto, os melhores resultados. Variações negativas implicam em soluções com distância menores do que os dados de entrada, enquanto que valores positivos referem-se a soluções piores.

Destaca-se neste gráfico o grande horizonte de soluções visitado. A não ocorrência de *loopings* de soluções iguais indica que as estratégias utilizadas para varrer a maior quantidade de soluções possível foram efetivas.

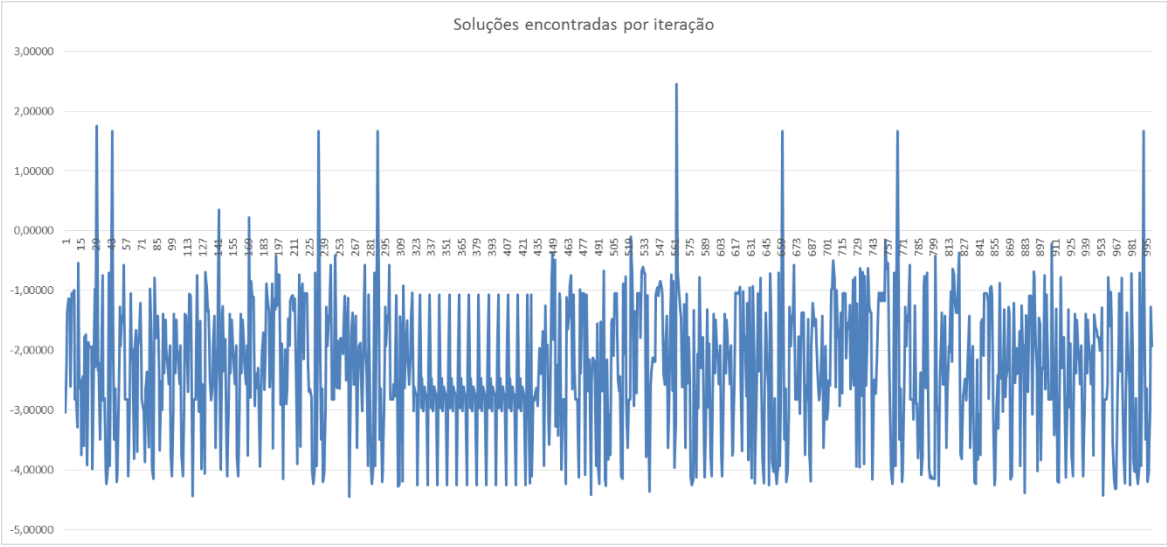

Figura 6. Resultados encontrados por iteração, para o caso sem restrições de janela de tempo e número máximo de unidades por *cluster*

Das 1000 iterações, 990 tiveram resultados cuja variação de distâncias em relação à solução vinda dos dados de entrada indicou resultados melhores, como pode ser observado no gráfico presente na Figura 7. Percebe-se, portanto, uma grande

qualidade no método, na medida em que soluções piores só representaram 1% do total.

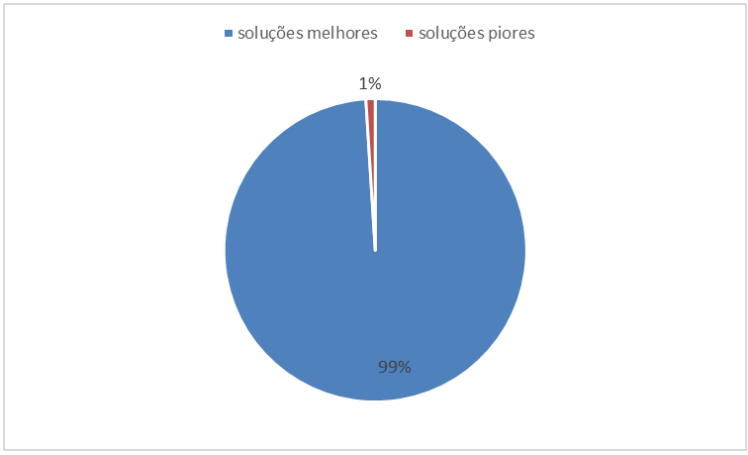

Figura 7. Qualidade dos resultados de cada iteração

Entretanto, o que efetivamente será importante na saída do programa é a melhor das soluções e não a solução encontrada em cada iteração individualmente. Por isso, na Figura 8 é apresentada a variação de distâncias totais percorridas entre a melhor solução encontrada até cada iteração e a distância registrada nos dados de entrada. Percebe-se que em pouco mais de 100 iterações conseguiu-se chegar a uma solução quase tão boa quanto a encontrada até a iteração 1000, o que mostra que caso seja utilizado em campo, este programa pode ter seu tempo de processamento diminuído sem grandes prejuízos ao resultado encontrado.

Figura 8. Melhores soluções encontradas até cada iteração.

Ao final, obteve-se um resultado cuja variação foi de 4,4545 unidades de latitude e longitude. Ou seja, houve uma redução de aproximadamente 490 Km no total de distância percorrida para o abastecimento às unidades. Isso representaria uma grande diminuição em consumo de combustível e tempo demandado para o processo. Como essas rotas acabam sendo repetidas diversas vezes dentro de um mesmo mês para o constante abastecimento às unidades, a economia é ainda mais significativa.

Entretanto, levando-se em conta restrições de processamento por parte do roteamento que é realizado posteriormente, não se pode trabalhar com um número ilimitado de unidades a cada cluster. Para isso, foram simulados vários casos, com diferentes restrições do número de unidades por cluster. Foram feitas simulações com 6, 5 e 4 unidades no máximo por cluster.

Foi utilizada a mesma máquina da simulação anterior. Para o caso com limite de 6 unidades, demandou-se 1 hora e 34 minutos. Para o caso com limite de 5 unidades, demandou-se 1 hora e 50 minutos. Por fim, para o caso com limite de 4 unidades demandou-se 1 hora e 39 minutos. Para comparar o comportamento para cada um desses casos, plotou-se em um único gráfico a evolução da melhor solução ao longo das iterações para todos os casos. O resultado pode ser observado na Figura 9.

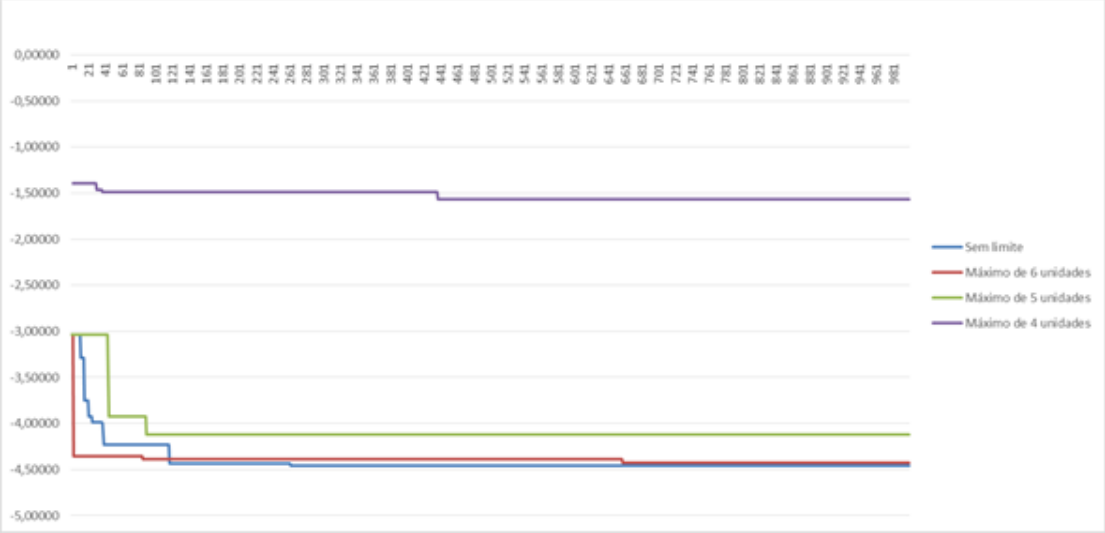

Figura 9. Evolução das melhores soluções ao longo das iterações para diferentes limites de unidades por cluster

Na primeira solução encontrada nas primeiras iterações, considera-se aquela encontrada pelo simples rearranjo da solução inicial. Não são aplicadas, portanto, todas as restrições que garantem a viabilidade do programa, tais como o limite de unidades que podem existir em um mesmo cluster. Por isso, esta solução deve ser desconsiderada. Para o caso em que se limitou a no máximo 4 unidades por cluster, as restrições foram tão grandes que não se encontrou nenhuma melhor solução do que a solução inicial. Por isso, trabalhou-se com a segunda melhor solução encontrada e a primeira solução encontrada foi desconsiderada.

Como era de se esperar, conforme a restrição torna-se mais rígida, são encontradas soluções cada vez piores. A melhor solução encontrada após 1000 iterações no caso sem restrições diminuiu a distância percorrida em 4,4545 unidades de latitude, enquanto que para o caso em que limita-se a no máximo 4 unidades por *cluster* conseguiu-se diminuir apenas 1,5704. Ou seja, houve um aumento de 2,8841 unidades de latitude e longitude, ou seja, 317,251 Km acrescidos à rota. Isso se deve, sobretudo à necessidade de criação de um maior número de clusters ou rotas com unidades longes entre si para satisfazer a restrição.

Apesar disso, torna-se necessário este procedimento para fins de processamento de fases posteriores da pesquisa. Por isso, a partir deste momento, todas as simulações serão feitas com a restrição de no máximo 4 unidades por *cluster*. Entretanto, caso na prática seja possível mais unidades serem ser processadas por vez, então restrições menos rígidas poderão ser utilizadas e resultados melhores poderão ser encontrados.

Optou-se por aplicar a restrição relativa às janelas de tempo apenas ao fim do procedimento e por isso, as partes do código referentes a esta restrição no meio do procedimento tiveram a restrição relaxada. Assim, evitou-se que isto interferisse no método e prejudicasse a visita a um conjunto maior de soluções. O procedimento adotado foi, então, percorrer ao fim do procedimento todo o histórico de melhores soluções encontradas, eliminando-se aquelas que não atendessem à restrição.

Considerando-se ainda a restrição de no máximo 4 unidades por cluster, simulou-se a aplicação de diferentes restrições quanto às janelas de tempo no final do processamento: diferença máxima de 20 horas, 25 horas, 30 horas, 40 horas e 50

horas. Os tempos de processamento demandados na mesma máquina referida anteriormente para cada um dos casos foram, respectivamente, 1 hora e 39 minutos, 1 hora e 44 minutos, 1 hora e 42 minutos, 1 hora e 36 minutos e 1 hora e 40 minutos. A melhor solução encontrada para cada um desses casos foi plotada em um gráfico, exposto na Figura **10**.

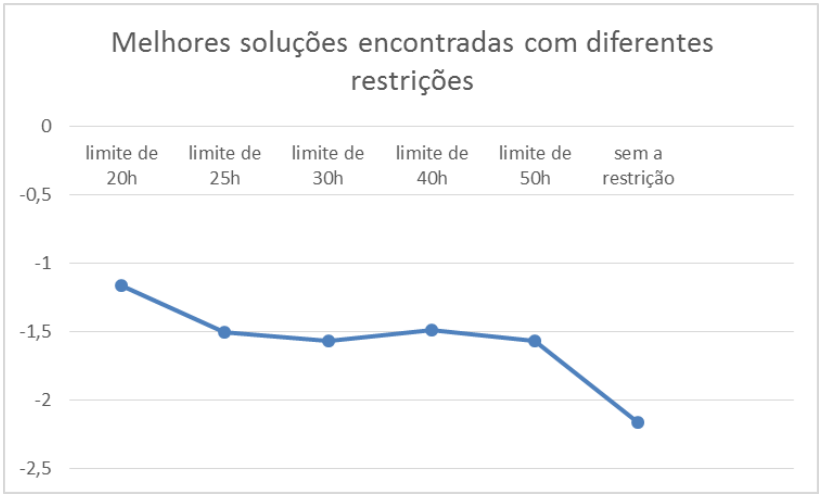

Figura 10. Melhores soluções encontradas com diferentes graus de restrições de diferenças de janelas de tempo.

Como era de se esperar, nota-se que as soluções são piores para os casos em que existe uma restrição mais rígida, ou seja, para aqueles casos em que se impõe uma diferença máxima entre janelas de tempo dentro de um mesmo cluster menor. Apesar de isso piorar a solução que será resultante do programa, trata-se de um procedimento necessário. Janelas de tempo existentes hoje em dia são muito complexas de serem alteradas, na medida em que sua alteração exigiria negociações com cada plataforma ou sonda existente, exigindo mudanças no cotidiano dessas unidades.

Entretanto, o maior problema consiste na alteração constante dessas janelas, já que a rotina das unidades precisaria ser alterada constantemente. Caso defina-se uma nova estratégia definitiva, que não se pretende alterar por um longo período de tempo, os problemas não seriam muito grandes. Por isso, fica como sugestão a alteração das rotas para a configuração atual, construindo-se novas janelas de tempo baseando-se nos tempos de visita a cada unidade definidos aqui. Caso isso fosse adotado, ao invés de haver uma diminuição de 1,16 unidades de latitude e longitude, haveria uma diminuição de 2,16 unidades. Ou seja, ao invés de uma diminuição de 127,6 Km nas

rotas, haveria uma diminuição de 237,6 Km. Ou seja, haveria uma diminuição de 110 Km extras a cada rota, o que se refletiria em economia de combustível e de tempo demandado para percorrer cada rota.

Durante o ano, com a instalação de novas unidades e mudanças de posições de sondas, talvez a configuração escolhida aqui fique, de novo, ineficiente. Sugere-se, então, renegociações de janelas de tempo de tempos em tempos pré-determinados. A periodicidade dessas ações não deve ser pequena o suficiente para causar problemas na rotina das unidades, nem grande o suficiente para fazer com que as rotas tornem-se cada vez mais ineficientes.

Por fim, preocupou-se em adequar a configuração atual à frota já atualmente existente. Durante todos os procedimentos realizados até agora, permitiu-se o aumento da capacidade dos barcos de apoio em caso de ser impossível a alocação de alguma unidade em nenhum *cluster*. Repetiu-se a simulação feita anteriormente, com as restrições de apenas 4 unidades no máximo por *cluster* e 20 horas de diferença no máximo entre janelas de tempo dentro de um mesmo *cluster*, só que agora sem a possibilidade de aumento da área de deck dos barcos, ou seja, sem a modificação na frota existente. Para este procedimento, o tempo de execução demandado foi de 1 hora e 26 minutos. A evolução da solução mínima encontrada até cada iteração, ainda sem a restrição de janelas de tempo, pode ser observada na Figura 11.

Ao contrário do que se esperaria, retirar a possibilidade de aumentar a capacidade de barcos não surtiu efeito negativo. Pelo contrário, provavelmente esta nova configuração possibilitou a visitação a um ambiente completamente novo de soluções que inclusive melhorou o resultado final. Diante desses resultados positivos, define-se que estas restrições podem ser aplicadas no programa final.

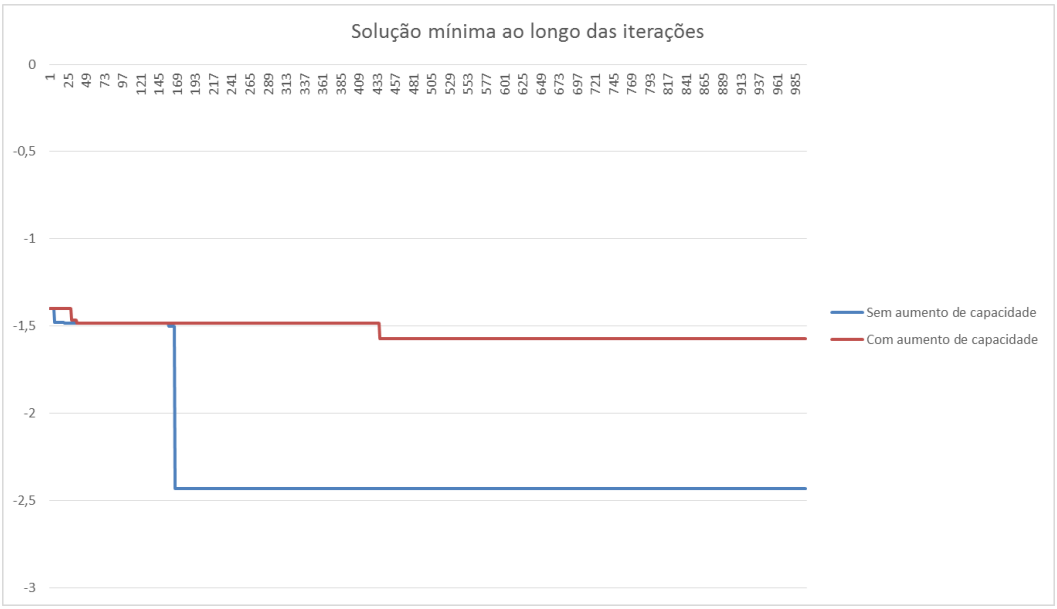

Figura 11. Solução mínima ao longo das iterações desconsiderando a restrição de janelas de tempo para o caso em que se pode mudar a capacidade do barco e o caso em que não se pode

Filtrando dentre os melhores resultados aqueles que cumprem a restrição de diferenças de janelas de tempo, chega-se a uma redução de unidades de latitude de 1,33. Neste caso, ao invés de uma economia de 4,32308 unidades de latitude, ou seja 475,53 Km, haverá uma economia de 1,33, ou seja 146.62Km. Como se pode perceber, a restrição das janelas de tempo prejudica bastante a solução e, por isso, uma nova configuração com novas janelas de tempo baseada na solução ainda é sugerida. Aqui, entretanto, será considerada como configuração final aquela em que as janelas de tempo são consideradas, já que provavelmente estas se adequam mais à realidade atual e são de mais fácil implantação.

Portanto, considerando-se o caso em que não se permite o aumento de capacidade dos barcos, restringe-se a capacidade dos *clusters* a 4 unidades/*cluster* e a diferenças de janelas de tempo dentro de um mesmo *cluster* não ultrapassem 20 horas, obtém-se a configuração de clusterização expressa na Figura **12**.

Com o objetivo de analisar a eficiência do método empregado, é apresentada na Figura **13** a clusterização originalmente utilizada para a mesma data estudada.

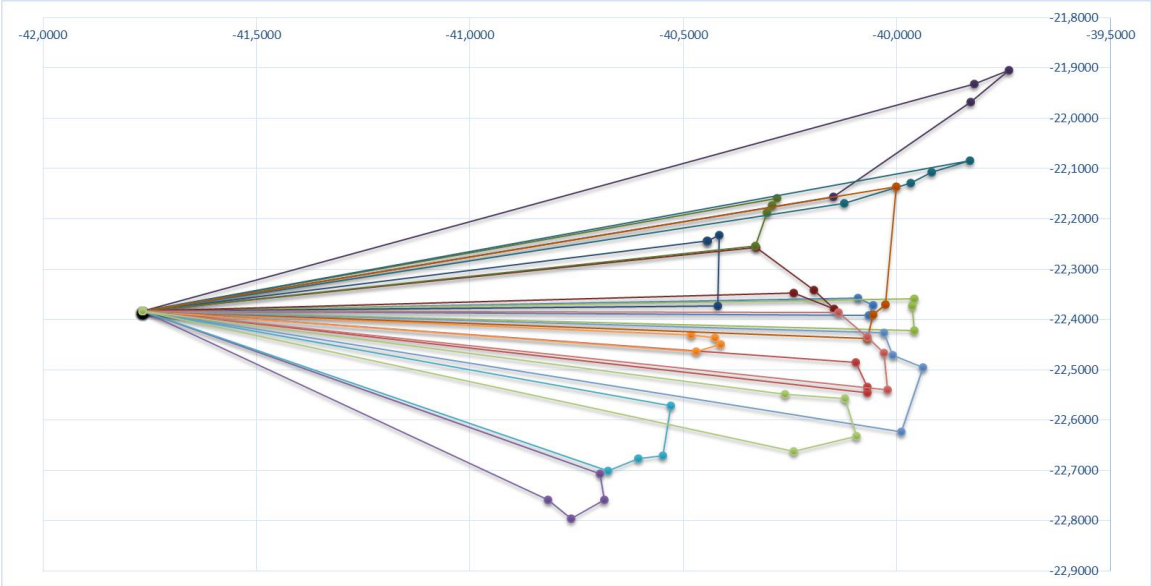

Figura 12. Mapa com clusterização resultante do método empregado

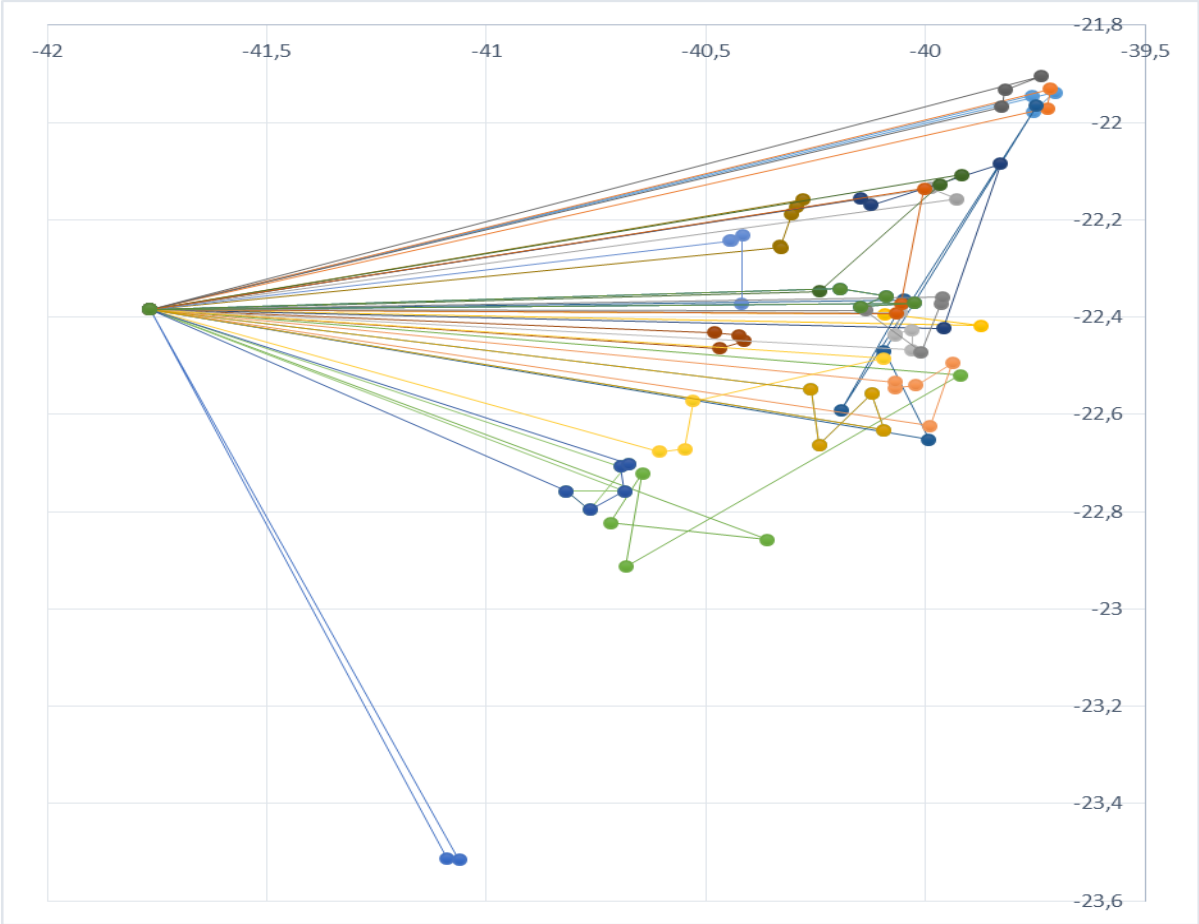

Figura 13. Configuração originalmente utilizada.

A partir das Figura **12** e Figura **13**, nota-se que o método utilizado tornou as rotas mais organizadas. Existem muito menos sobreposições de rotas e idas e vindas entre unidades.

Existe uma grande facilidade de aplicação desta nova metodologia na prática. As rotas mostraram-se viáveis sob o critério de janelas de tempo e não há a necessidade de mudança nas características da frota. Entretanto, a metodologia aplicada até agora não teve como objetivo realizar a programação do porto e de cada atividade realizada. Isso será objetivo de fases distintas da pesquisa, não contempladas neste trabalho.

8 - Conclusões

Dos experimentos apresentados no capítulo 7, pode-se observar a obtenção de resultados bastante favoráveis. O método, sem nenhuma restrição resultou em uma economia de aproximadamente 490 Km na distância total percorrida para a realização das rotas. A utilização dessas rotas na prática seria um pouco complexa, na medida em que exigiria mudanças nas janelas de tempo já existentes. Entretanto, ainda assim, diante de resultados tão positivos, recomenda-se que as atividades atuais sejam revistas e adote-se configurações como esta.

Aplicando-se as restrições, ainda assim houve uma redução de 146,62 Km. Neste caso, a aplicação destas rotas na prática é ainda mais facilitada, na medida em que o critério de janelas de tempo foi observado, impedindo-se que dentro de um mesmo cluster houvesse unidades com janelas muito diferentes e que exigissem períodos de espera por parte do barco irrealistas. Por isso, recomenda-se a adoção dessas rotas a curto prazo.

Ainda assim, deseja-se realizar mais alguns passos na pesquisa que refinarão ainda mais a solução. Já está sendo montado por outra equipe de pesquisa um algoritmo que processa a saída vindo da seção de clusterização e aplica uma metodologia de roteamento. O objetivo é avaliar o problema de um ponto de vista dinâmico, considerando-se janelas de tempo e o tempo no porto.

Paralelamente, outra equipe de pesquisa está realizado estudos para efetuar simulações probabilísticas da situação estudada. Espera-se que, com isso, deixe-se de trabalhar com o problema de um ponto de vista determinístico e passe-se a interpretá-lo de um ponto de vista estocástico. Assim, poderão ser definidos quais os reais níveis de atendimento que serão alcançados com a nova metodologia proposta.

Referências bibliográficas

[1] Piquet, Rosélia. 2003. **Petróleo, royalties e região**, Garamond.

[2] ANP. 2013. **Anuário Estatístico Brasileiro do Petróleo, Gás Natural e Biocombustíveis 2013.** Disponível em http://www.anp.gov.br/?pg=66833#Se__o_2. Acessado em 23/02/2014.

[3] Leite, Ricardo Penna. 2012. **Maritime transporte of deck cargo to Petrobras fields in Campos Basin: an emprirical analysis, identification and quantification of improvement points.** PUC-RJ.

[4] Thomas, José Eduardo. **Fundamentos de engenharia de petróleo.** Editora Interciência

[5] VALENTE, A. M.; NOVAES, A. G.; PASSAGLIA, E. ; VIEIRA, H**.. Gerenciamento de transporte e frotas.** Cengage Learning, São Paulo, 2ª edição, 2008.

[6] AAS, B.; GRIBKOVSKAIA, I.; ØYVIND HALSKAU ; SHLOPAK, A.. **Routing of supply vessels to petroleum installations**. International Journal of Physical Distribution and Logistics Management, 37(2):164–179, 2007.

[7] AAS, B.; BUVIK, A. ; CAKIC, D.. **Outsourcing of logistics activities in a complex supply chain: a case study from the Norwegian oil and gas industry**. International Journal of Procurement Management, 1(3):280–296, 2008.

[8] AAS, B.; WALLACE, S.. **Management of Logistics Planning. International Journal of Information Systems and Supply Chain Management**, 3(3):1–17, 2010.

[9] AAS, B.; WALLACE, S. W. ; ØYVIND HALSKAU. **The role of supply vessels in offshore logistics. Maritime Economics & Logistics**, 11:302–325, 2009.

[10] Romero, M., Sheremetov, L. and Soriano, A. (2007). **A genetic algorithm for the pickup and delivery problem: an application to the helicopter offshore transportation**, em Castillo, O. (Ed.,) **Theoretical Advances and Applications of Fuzzy Logic and Soft Computing**, Springer, Berlim.

[11] Gribkovskaia, I., Laporte, G. and Shlopak, A. (2007**), A tabu search heuristic for a routing problem arising in servicing of offshore oil and gas platforms**. Journal of the Operational Research Society, Vol. 59, pp. 1449-59.

[12] KAISER, M. J.. **An integrated systems framework for service vessel forecasting in the Gulf of Mexico**. Energy, 35(7):2777 – 2795, 2010.

[13] KAISER, M. J.; SNYDER, B.. **An empirical analysis of offshore service vessel utilization in the US Gulf of Mexico.** International Journal of Energy Sector Management, 4(2):152–182, 2010.

[14] FAGERHOLT, K.; LINDSTAD, H**.. Optimal policies for maintaining a supply service in the Norwegian Sea**. Omega, 28(3):269 – 275, 2000.

[15] HALVORSEN-WEARE, E. E.; FAGERHOLT, K.. **Robust Supply Vessel Planning**.

[16] SHYSHOU, A.; FAGERHOLT, K.; GRIBKOVSKAIA, I. ; LAPORTE, G.. **A large neighbourhood search heuristic for a periodic supply vessel planning problem arising in offshore oil and gas operations.**

[17] PANAMARENKA, K.. **Minimization of emissions in periodic supply vessel planning through speed optimization.**. Tese de mestrado, Molde University College, 2011.

[18] CHRISTIANSEN, M.; FAGERHOLT, K. ; RONEN, D.. **Ship Routing and Scheduling: Status and Perspectives**. Transportation Science, 38(1):1–18, 2004.

[19] BATISTA, B. C. D**.. Análise das operacões com embarcacões de apoio offhore na Bacia de Campos-RJ**. Tese de mestrado, Universidade Federal do Rio de Janeiro, 2005.

[20] KOSKOSIDIS, Y. A.; POWELL, W. B.. **Clustering algorithms for consolidation of customer orders into vehicle shipments**. 1992.

Apêndice 1: Código implementado

Esta seção tem como objetivo apresentar a parte a implementação do código empreendido neste estudo. Foi implementado o método iterativo disponível em Koskosidis&Powell[20], com algumas alterações incluídas devido a particularidades do problema estudado. Na entrada de dados do problema, foram utilizados dados reais referentes a configurações de clusterização já utilizadas no passado. Devido à confidencialidade destes dados, estes não serão expostos aqui.

```
clear all
clc
```

Figura 14. Trecho 1 do código implementado.

As primeiras linhas do programa têm como utilidade apagar qualquer variável utilizada em simulações anteriores. Assim, não há nenhum problema em valores anteriormente utilizados alterarem o comportamento do programa.

```
    final_tentativa_seed=1000;
    diferenca_maxima_de_janelas_de_tempo=30;
    diferenca_maxima_de_janelas_permitida=20; %na output
    numero_maximo_de_unidades_por_clusters=4;
```

Figura 15. Trecho 2 do código implementado.

Nas próximas linhas, são concentradas as principais entradas que precisamos do usuário.

O parâmetro ***final_tentativa_seed*** está relacionado a quantas vezes deseja-se rodar o programa. Quanto maior a quantidade de vezes que são rodadas, maior a possibilidade de as soluções encontradas serem melhores. Entretanto, também é maior a quantidade de tempo demandado para chegar-se ao resultado. Cabe ao usuário determinar qual é seu tempo ótimo, baseado na quantidade de tempo desejada e precisão na solução requerida. Caso este algoritmo seja utilizado para estudos, o tempo demandado para o programa poderá ser maior, entretanto, caso este seja utilizado no dia-a-dia operacional em que demanda-se soluções mais rápidas, então o tempo demandado pelo programa não poderá ser muito grande.

Outro parâmetro de entrada do programa definido pelo usuário é o ***diferenca_maxima_de_janelas_de_tempo***. Nesse caso, deseja-se definir qual a máxima diferença de tempo entre inícios das janelas de tempo de unidades de um mesmo cluster para as novas configurações obtidas após as fases de trocas de posições entre unidades e reconstrução dos *clusters* a partir dos *seeds*. Isso é essencial, na medida em que deve-se pensar na viabilidade da solução apresentada para as fases posteriores da pesquisa que tentarão criar novas programações para o porto e considerarão as janelas de tempo em sua análise. Dependendo das configurações finais obtidas nessa fase, é possível que haja clusterizações em que a distância total percorrida seja pequena, mas na prática o barco não consiga atender as unidades dentro da janela de tempo e novos clusters pouco otimizados tenham de ser criados. Logo, o ideal é limitar diferenças entre janelas de tempo de um mesmo cluster para que se impeça que sejam criados clusters com janelas de tempo impossíveis de serem cumpridas.

Também foi definido o parâmetro ***diferenca_maxima_de_janelas_permitida***, o qual tem a mesma função do ***diferenca_maxima_de_janelas_de_tempo***, mas atua apenas na saída do programa, selecionando dentre as respostas do programa quais são aquelas que realmente são viáveis segundo o critério da viabilidade de atendimento às janelas de tempo. A vantagem de impor a restrição desta maneira é que não interfere-se na mecânica interna do algoritmo, possibilitando a análise de um horizonte muito maior de soluções sem que o programa mostre-se precocemente inviável. A presença desses dois parâmetros ao mesmo tempo possibilita o direito de escolha ao usuário sobre qual seria a melhor forma de trabalho.

Por fim, o parâmetro ***numero_maximo_de_unidades_por_clusters*** tem como finalidade limitar o número máximo de unidades que podem existir a cada cluster. A primeira finalidade disso refere-se à viabilidade operacional, na medida em que *clusters* com muitas unidades demandariam tempos de viagens grandes demais, o que seria inviável na prática. Da mesma forma, a presença desta restrição também tem como finalidade atender à limitação de unidades processadas por vez nos algoritmos presentes em outras fases da pesquisa.

```
    passo=0;
```

Figura 16. Trecho 3 do código implementado.

Em seguida, é inicializada a variável ***passo***. Ela irá variar ao longo das iterações quando houver soluções diferentes. O objetivo é ajudar a montar matrizes e vetores que representam o histórico de soluções encontradas.

```
%:::::::::::::::Definições:::::::::::::::::
macae_long=-41,7671;
macae_lat=-22,3838;
```

Figura 17. Trecho 4 do código implementado.

Passa-se a seguir, para a fase de definição das propriedades das unidades e do porto. Como a posição do porto nunca mudará, a latitude e longitude referentes a ele são definidas diretamente no código. Resta-se obter os dados referentes as unidades, que serão adquiridos de uma planilha de entrada gerada a partir de um conjunto de dados obtido junto a Petrobras.

A planilha de entrada utilizada deve tem em sua segunda coluna o nome de cada unidade. Nas colunas posteriores, há a latitude da unidade, longitude, demandas, início da janela de tempo e tempo de serviço respectivamente. Além disso, são especificados os barcos utilizados.

Será utilizada como solução inicial a configuração já utilizada anteriormente pela Petrobras em alguma clusterização adotada no passado. Na realidade, seria possível partir de qualquer solução, entretanto, a utilização de uma configuração já utilizada aumenta a velocidade de chegada à solução.

```
[numeros_unidade_1,strings_unidade,tabelatoda_unidade] = xlsread('C:\local onde
encontra-se o arquivo de input\dados_input.xlsm',1);
```

Figura 18. Trecho 5 do código implementado.

Para a leitura da planilha de entrada, o primeiro passo consiste em localizar o arquivo onde esta localiza-se. Em matlab, ao se obter valores vindos de uma planilha Excel, é gerado uma matriz que contém todos os elementos numéricos, além de todos os

elementos de *strings* e a tabela completa. No caso deste estudo, as *strings* representarão apenas as legendas e os nomes das unidades, enquanto que os valores numéricos representarão a maior parte de nossa entrada.

```
numeros_unidade=numeros_unidade_1(:,1:6);
```

Figura 19. Trecho 6 do código implementado.

No trecho de código presente na figura, mostra-se como foram obtidos dados numéricos vindos da tabela de entrada. Cada linha representará uma unidade e cada coluna uma propriedade. Como nem tudo todas as propriedades serão utilizadas neste momento, seleciona-se apenas algumas colunas do que veio do Excel. Quanto as linhas, ao se selecionar todas as linhas da entrada, automaticamente trabalha-se com todas as unidades presentes, independente de quantas unidades existam. Assim, são obtidos os 6 parâmetros que definem cada unidade: latitude, longitude, demanda, início da janela, fim da janela e tempo de serviço.

```
[linhas_numeros_unidade, coluna_numeros_unidade]=size(numeros_unidade);
nome_input=strings_unidade(4:linhas_numeros_unidade,2);
latitude_input=numeros_unidade(4:linhas_numeros_unidade,1);
longitude_input=numeros_unidade(4:linhas_numeros_unidade,2);
demandas_input=numeros_unidade(4:linhas_numeros_unidade,3);
inicio_janela_input=numeros_unidade(4:linhas_numeros_unidade,4);
fim_janela_input=numeros_unidade(4:linhas_numeros_unidade,5);
tempo_servico_input=numeros_unidade(4:linhas_numeros_unidade,6);
```

Figura 20. Trecho 7 do código implementado.

O passo seguinte consiste em organizar os dados vindos da tabela de entrada, já que trabalha-se com uma matriz com ordem das colunas diferente da obtida diretamente do Excel. Para uma melhor organização, o primeiro passo consiste em obter cada propriedade em matrizes separadas, onde cada linha da matriz representa uma propriedade diferente. O objetivo é que posteriormente seja possível montar uma matriz da forma que será utilizada no trabalho, com as colunas na ordem correta.

Entretanto, como na tabela de entrada utilizada não há registrados em qual cluster cada unidade se encontra, mas apenas a ordem em que estas aparecem na configuração de clusters, então deve-se também criar uma rotina para tentar recriar os clusters. O primeiro passo consiste em determinar o limite que define o fim de cada cluster e marca o início do outro, neste caso a capacidade do barco que atende ao cluster. Seguindo-se procedimentos normalmente utilizados nesse processo, optou-se por começar montando *clusters* com os maiores barcos disponíveis e, apenas ao fim de sua disponibilidade, com barcos menores. Resta obter os dados referentes aos barcos disponíveis e criar uma rotina para poder remontar os clusters.

```
numeros_barco=numeros_unidade_1(2:29,10:11);

[linhas_demanda_input, coluna_demanda_input]=size(demandas_input);
[linhas_barcos_input, coluna_barcos_input]=size(numeros_barco);
```

Figura 21. Trecho 8 do código implementado.

A partir da planilha com os dados de entrada, são obtidos os dados de capacidade referentes a cada barco disponível como mostrado na figura. Em seguida, são obtidos o tamanho de algumas matrizes, com o objetivo de descobrir o número total de unidades envolvidas e o número total de barcos, o que será utilizado em vários momentos do código.

```
for percorre_input=1:linhas_demanda_input
    if percorre_input==1
        demanda_total_input=demandas_input(percorre_input,1);
    else
        demanda_total_input=demanda_total_input+demandas_input(percorre_input,1);
    end
    if demanda_total_input>numeros_barco(barco_escolhido,2)
        barco_escolhido=barco_escolhido-1;
        demanda_total_input=demandas_input(percorre_input,1);
        cluster_sendo_mexido=cluster_sendo_mexido+1;
        numero_cluster_input(percorre_input,1)=cluster_sendo_mexido;
        capacidade_barco_input(percorre_input,1)=numeros_barco(barco_escolhido,2);
    else
        numero_cluster_input(percorre_input,1)=cluster_sendo_mexido;
        capacidade_barco_input(percorre_input,1)=numeros_barco(barco_escolhido,2);
    end

end
```

Figura 22. Trecho 9 do código implementado.

Passa-se, a seguir, para a rotina de reconstrução dos cluster originais propriamente dita. O objetivo consiste em percorrer a coluna de demandas de cada unidade até o final, registrando o somatório de demanda até cada unidade. Se este somatório rompe

a capacidade do maior barco disponível, então significa que deve-se começar um novo *cluster* a partir desta unidade. Zera-se o somatório e continua-se o procedimento. Para saber a qual cluster cada unidade pertence, foi criado um identificador numérico para que cada cluster diferente seja representado por um número.

Os barcos maiores encontram-se nas últimas posições da tabela de barcos. Por isso, durante a seleção de qual barco atenderá o cluster, começa-se seleção com os barcos que estão no fim da tabela e, conforme barcos vão sendo alocados, percorre-se a tabela do fim até o início.

```
nome_unidade_input=[1:linhas_demanda_input]';
```

Figura 23. Trecho 10 do código implementado.

Deseja-se também ter na registrados dentro do programa o nome das unidades incluídas na clusterização. Não será possível, entretanto, adicionar estes nomes na mesma matriz que será modificada ao longo do programa. Como no matlab não existem maneiras simples de se concatenar strings e números em uma mesma matriz, então serão utilizados identificadores numéricos ao invés do nome real da unidade. O objetivo é que, ao final de todo o programa, seja possível recuperar os nomes das unidades a partir dessas *tags* recorrendo-se à ordem dos nomes das unidades que registrou-se na entrada.

```
BC_input=[numero_cluster_input,demandas_input, latitude_input,
longitude_input,capacidade_barco_input,nome_unidade_input,inicio_janela_input,fim_
janela_input,tempo_servico_input ];
```

Figura 24. Trecho 11 do código implementado.

Com isso, finalmente, pode-se definir a matriz que será alterada ao longo do código. Cada linha representará uma unidade e as colunas representarão as propriedades das unidades na seguinte ordem: tag com um número representativo do cluster a que a unidade pertence, demanda da unidade, latitude da unidade, longitude da unidade, capacidade do barco que atende à unidade, tag representativa do nome da unidade, início da janela de tempo, fim da janela de tempo e tempo de serviço.

A seguir, passa-se para a primeira fase do algoritmo: realizar trocas de posição das unidades em pares. Ou seja, não quantidade de unidades em cada cluster permanecerá inalterada e apenas será trocada a posição de uma única unidade com outra. Percorre-se todas as combinações possíveis de trocas. Não significa, aqui, que foram percorridas todas as configurações possíveis de clusterização, na medida em que para que fizéssemos isso seria necessário mais de uma troca simultânea, enquanto que faremos apenas uma troca por vez. Por isso, o procedimento será continuamente repetido com os resultados, até que chegue-se a um número máximo de procedimentos realizados ou não consiga-se encontrar novas soluções.

```
BC=BC_input;

BC_nova=BC;

BC_utilizada=BC;
```

Figura 25. Trecho 12 do código implementado.

Durante o código serão utilizadas 3 matrizes diferentes que representarão em uma matriz como a discutida anteriormente. A primeira delas representa a matriz original em que estamos baseando nossas trocas(BC), não será alterada mesmo que achamos uma solução melhor. A segunda matriz, BC_nova, será a matriz alterada a cada iteração, tendo seu custo analisado para que se descubra se o custo resultante é maior ou menor do que o menor custo até agora registrado. Por fim, a Matriz BC_utilizada será a matriz que guardará o menor custo até o momento.

Em sua inicialização, essas três matrizes terão o valor igual à configuração vinda da planilha com os dados de entrada. Posteriormente, após a primeira iteração, estes valores serão provenientes do resultado iteração imediatamente anterior.

```
[linhas,colunas]= size(BC)
```

Figura 26. Trecho 13 do código implementado.

A partir do momento que já tem-se matriz que será utilizada ao longo das iterações, torna-se possível definir quantidade de linhas e colunas presentes nesta. Isso é

essencial, na medida em que estes valores serão utilizados em várias iterações ao longo do código.

```
custo_original=99999999;
custo_escolhida=custo_original;
```

Figura 27. Trecho 14 do código implementado.

O objetivo pretendido é minimizar o custo. Para isso, a cada iteração, este será calculado e comparado com o menor custo registrado até agora. Caso esta configuração tenha um custo menor, este novo custo passa a ser registrado como o novo menor custo e todas as iterações posteriores têm seus custos comparados com esta. Este procedimento segue desta maneira até que uma nova configuração com custo ainda menor é encontrada.

Para dar início a este processo, deve-se inicializar as variáveis que representarão os custos. Como deseja-se minimizar o custo, define-se inicialmente as variáveis ***custo_original*** e ***custo_escolhida*** como valores muito grandes, que possam não interferir no processo. Assim, já na primeira iteração, os valores serão substituídos pelo custo desta iteração, dando início à rotina discutida no parágrafo anterior.

```
numero_rodada=1;

coluna_config=1;
existe_config_igual=0;
achou_nova_solucao=0;
total_sol_iguais_todos=zeros(500,1);

janelas_muito_diferentes=0;
```

Figura 28. Trecho 15 do código implementado.

A seguir, é inicializada uma variável que marcará o número da iteração atual. Além disso, serão inicializadas algumas variáveis binárias que serão utilizadas ao longo do código e definirão se alguma situação ocorre ou não. Nestas variáveis binárias, será definido aqui a condição inicialmente esperada, de forma que estas variáveis só sejam alteradas se alguma condição diferente for provada ao longo do código.

```
for tentativa_seed=1:final_tentativa_seed
```

Figura 29. Trecho 16 do código implementado.

O programa ocorrerá de forma iterativa, de forma que cada iteração tenha como resultado um novo custo. No final, todos os custos serão comparados e os resultados finais serão aquelas configurações que tiverem os menores custos. Neste ponto do código, define-se, então, o início deste *looping*. Dentro dele estarão compreendidos tanto a parte de trocas de unidades entre *clusters*, quanto a definição de novos seeds e reclusterização.

```
    distancias_ate_seeds=0;
    num_cluster_novo=0;
    demanda_novo=zeros(14,1);
    latitude_novo=0;
    longitude_novo=0;
    capacidade_novo=0;
    nome_unidade_novo=0;
    inicio_janela_novo=0;
    fim_janela_novo=0;
    tempo_servico_novo=0;
    soma_demanda=0;
    soma_demanda_proximo_cluster=0;
    num_cluster_novo_sobra=0;
    demanda_novo_sobra=0;
    latitude_novo_sobra=0;
    longitude_novo_sobra=0;
    capacidade_novo_sobra=0;
    nome_unidade_novo_sobra=0;
    inicio_janela_novo_sobra=0;
    fim_janela_novo_sobra=0;
    tempo_servico_novo_sobra=0;
```

Figura 30. Trecho 17 do código implementado.

Como o programa passará por este ponto quantas vezes for pedido pelo usuário na variável ***final_tentativa_seed***, então é essencial que sejam zeradas as variáveis que serão alteradas ao longo do código para que valores da iteração anterior não interfiram nesta.

```
if tentativa_seed~=1
        BC=BC_reclusterizada;
        BC_nova=BC;
        BC_utilizada=BC;
    end
```

Figura 31. Trecho 18 do código implementado.

Anteriormente, já havia sido definido os valores das matrizes ***BC***, ***BC_nova*** e ***BC_utilizada*** para a primeira iteração, atribuindo a elas a matriz proveniente da tabela com os dados de entrada. Entretanto, para as iterações posteriores, serão utilizados os dados provenientes da matriz resultado da iteração anterior. Nesses casos, as matrizes com que se trabalhará na atual iteração são inicializadas com os valores da iteração anterior, após a fase de reclusterização.

```
BC_reclusterizada=0;
```

Figura 32. Trecho 19 do código implementado.

Após os valores proveniente da iteração anterior serem registrados, não há mais a necessidade de guarda-los e a variável ***BC_reclusterizada*** pode ser zerada. O objetivo de se fazer isso é que resultados provenientes da iteração anterior não sejam confundidos com resultados dessa iteração. Por exemplo, caso algum problema no código ocorra e algumas unidades não sejam reclusterizadas, esse procedimento facilita a detecção deste erro, na medida em que o número de linhas da matriz diminuiria e dados de iterações antigas não seriam misturados com o da atual.

```
while tentativa<=100
       BC_antiga=BC_utilizada;
```

Figura 33. Trecho 20 do código implementado.

Como discutido anteriormente, são realizadas todas as trocas em pares possíveis, registra-se a configuração com menor custo total e repete-se o processo com ela. Para que não exista muita perda de tempo computacional desnecessariamente, imita-se essa repetição até um valor determinado ou até ser impossível encontrar novas soluções. Este é o trecho do código é o responsável por realizar esta limitação.

Este *looping* será o responsável por definir a quantidade de vezes que o procedimento é realizado e limitar essa repetição a 100 vezes. Da mesma forma, caso seja detectado que não houve diminuição do custo, este *looping* será encerrado com a atribuição do valor 100 automática à variável ***tentativa***. Para saber se isso acontece, a configuração

inicial, ***BC_antiga***, é guardada e constantemente compara-se os novos valores encontrados com esta, definindo-se, portanto, se houve ou não novas soluções.

```
if tentativa_seed==1
    percentagem=(tentativa_seed/final_tentativa_seed)*100
    else
        if (tentativa_seed/final_tentativa_seed)*100~=percentagem
            percentagem=(tentativa_seed/final_tentativa_seed)*100
        end
    end
```

Figura 34. Trecho 21 do código implementado.

Para nos auxiliar a observar a evolução de nosso programa, é definida uma variável denominada ***percentagem***, a qual definirá o progresso do programa. Basicamente, como foi definido que o final do programa ocorre quando chega-se à iteração de número definido pelo parâmetro ***final_tentativa_seed***, então verificamos quantas destas iterações já foram feitas, o que pode ser verificado pela variável ***tentativa_seed.***

```
custo_original=0;
    i=1;

     while i<= linhas
            if (i==1) || (BC_utilizada(i,1)~=BC_utilizada(i-1,1))
                 custo_original=custo_original+((((BC_utilizada(i,3)-
macae_lat)^2)+((BC_utilizada(i,4)-macae_long)^2))^0.5);
            elseif (i==linhas) || (BC_utilizada(i,1)~=BC_utilizada(i+1,1))
                 custo_original=custo_original+((((BC_utilizada(i,3)-
BC_utilizada(i-1,3))^2)+((BC_utilizada(i,4)-BC_utilizada(i-
1,4))^2))^0.5)+((((BC_utilizada(i,3)-macae_lat)^2)+((BC_utilizada(i,4)-
macae_long)^2))^0.5);
            else
                custo_original=custo_original+((((BC_utilizada(i,3)-BC_utilizada(i-
1,3))^2)+((BC_utilizada(i,4)-BC_utilizada(i-1,4))^2))^0.5);
            end
            i=i+1;
        end
```

Figura 35. Trecho 22 do código implementado.

Calcula-se, então, o custo para a configuração original, ainda sem nenhuma troca realizada, seja esta configuração proveniente da entrada vinda da tabela ou proveniente da última iteração. Esse procedimento não será refeito a cada troca de posição de unidades para evitar a demanda de tempo computacional desnecessariamente e, nesses casos, apenas será recalculada a parcela do custo referente às posições trocadas. Para evitar propagação de erros por arredondamento, o

cálculo do custo presente neste trecho do código é refeito também a cada início de nova iteração.

Admite-se que o custo para cada viagem será proporcional à distância percorrida, então, na prática, calcula-se qual foi a distância percorrida. Como trabalha-se sob distâncias relativamente pequenas em relação ao raio da Terra, foi considerado que as latitudes e longitudes poderiam ser interpretadas como os eixo x e y perpendiculares sob um plano. Ou seja, para calcular a distância de um ponto ao outro calcula-se apenas a hipotenusa dos catetos formados pelas distâncias nos eixos x e y respectivamente entre duas unidades.

A única exceção nesse cálculo refere-se à primeira e à última unidade. Para a primeira unidade, deve-se considerar a distância do porto até unidade, utilizando-se, portanto, os parâmetro de latitude e longitude do porto definidos anteriormente no código. Para a última unidade, além de calcular o custo da penúltima unidade até esta, deve-se calcular também a distância necessária para se fechar a rota até o porto.

```
if tentativa==1
            custo_antiga=custo_original;
            custo_tentativa1=custo_original;
               custo_escolhida=custo_original;
            if numero_rodada==1
                custo_primeira_rodada=custo_original;
            end
        end
```

Figura 36. Trecho 23 do código implementado.

Com o custo inicial calculado, é essencial guardar estes valores iniciais para futuras conferências. A variável ***custo_antiga*** reflete o custo da variável reclusterizada na última iteração (para a segunda ***tentativa_seed*** em diante) ou o custo da clusterização proveniente da tabela de entrada (para a primeira tentativa_seed). Guardar este valor é essencial para saber se após todas as trocas da iteração analisada realmente consegue-se encontrar uma solução melhor. Caso não haja nenhuma configuração proveniente das trocas em que o custo é menor do que o da iteração anterior, então não encontra-se uma nova configuração, a configuração acaba sendo exatamente igual à anterior, a variável tentativa é atribuída como 100 e este looping acaba.

No caso de ser a primeira de todas as iterações, o cálculo dos custos representa o custo da configuração original montada a partir dos dados de entrada. Como deseja-se no final encontrar custos menores do que este como solução, este valor é guardado para futuras comparações, construindo a variável ***custo_primeira_rodada.***

A variável ***custo_escolhida*** será o custo mínimo encontrado até agora nas operações de troca de posições. A cada nova tentativa, o custo será comparado com o encontrado no ***custo_escolhida***. Somente será elegida como candidato a mínimo aqueles que tiverem um custo menor do que o da tentativa escolhida até agora, ou seja, aqueles que o custo for menor que a variável ***custo_escolhida***.

```
for i1 = 1:linhas
    aux2=BC(i1,2);
    aux3=BC(i1,3);
    aux4=BC(i1,4); %Variaveis auxiliares para guardar os dados durante a troca
    aux6=BC(i1,6);
    aux7=BC(i1,7);
    aux8=BC(i1,8);
    aux9=BC(i1,9);

    for i2 = 1:linhas
        if i1==i2
        else
            BC_nova(i1,2)=BC(i2,2);
            BC_nova(i1,3)=BC(i2,3);
            BC_nova(i1,4)=BC(i2,4);
            BC_nova(i1,6)=BC(i2,6);
            BC_nova(i1,7)=BC(i2,7);
            BC_nova(i1,8)=BC(i2,8);
            BC_nova(i1,9)=BC(i2,9);

            BC_nova(i2,2)=aux2;
            BC_nova(i2,3)=aux3;
            BC_nova(i2,4)=aux4;
            BC_nova(i2,6)=aux6;
            BC_nova(i2,7)=aux7;
            BC_nova(i2,8)=aux8;
            BC_nova(i2,9)=aux9;
```

Figura 37. Trecho 24 do código implementado.

Chega-se, então, finalmente à fase em que ocorrerão as trocas de posição entre unidades. Para trocar a posição de uma unidade na clusterização, simplesmente deve-se trocar uma linha com a outra na matriz que representa a configuração da clusterização. As únicas colunas que não deverão ser trocadas são aquelas que representam a *tag* do *cluster* da linha envolvida assim como a coluna que representa a capacidade do barco que atende ao *cluster* desta linha. Isso acontece pois, na verdade, essas variáveis não estão associadas à unidade, mas sim ao *cluster*.

Há a necessidade da criação de variáveis auxiliares que guardem os valores trocados, para que, ao ocorrer a primeira das duas substituições, não se perca os valores do elemento cujo valor foi sobrescrito.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Primeira linha trocada%%%%%%%%%
            if (i1==1) || (BC_nova(i1,1)~=BC_nova(i1-1,1)) %primeiro

                if (i1==linhas) || (BC_nova(i1,1)~=BC_nova(i1+1,1)) %primeiro e
ultimo
                    custo=custo-((((BC(i1,3)-macae_lat)^2)+((BC(i1,4)-
macae_long)^2))^0.5)+((((BC_nova(i1,3)-macae_lat)^2)+((BC_nova(i1,4)-
macae_long)^2))^0.5); %antes
                    custo=custo-((((BC(i1,3)-macae_lat)^2)+((BC(i1,4)-
macae_long)^2))^0.5)+((((BC_nova(i1,3)-macae_lat)^2)+((BC_nova(i1,4)-
macae_long)^2))^0.5); %depois

                else %apenas primeiro
                    custo=custo-((((BC(i1,3)-macae_lat)^2)+((BC(i1,4)-
macae_long)^2))^0.5)+((((BC_nova(i1,3)-macae_lat)^2)+((BC_nova(i1,4)-
macae_long)^2))^0.5); %antes
                    custo=custo-((((BC(i1+1,3)-BC(i1,3))^2)+((BC(i1+1,4)-
BC(i1,4))^2))^0.5)+((((BC_nova(i1+1,3)-BC_nova(i1,3))^2)+((BC_nova(i1+1,4)-
BC_nova(i1,4))^2))^0.5); %depois
                end


            elseif (i1==linhas) || (BC_nova(i1,1)~=BC_nova(i1+1,1)) %último
                    custo=custo-((((BC(i1,3)-BC(i1-1,3))^2)+((BC(i1,4)-BC(i1-
1,4))^2))^0.5)+((((BC_nova(i1,3)-BC_nova(i1-1,3))^2)+((BC_nova(i1,4)-BC_nova(i1-
1,4))^2))^0.5); %antes
                    custo=custo-((((BC(i1,3)-macae_lat)^2)+((BC(i1,4)-
macae_long)^2))^0.5)+((((BC_nova(i1,3)-macae_lat)^2)+((BC_nova(i1,4)-
macae_long)^2))^0.5); %depois


            else %meio
                custo=custo-((((BC(i1,3)-BC(i1-1,3))^2)+((BC(i1,4)-BC(i1-
1,4))^2))^0.5)+((((BC_nova(i1,3)-BC_nova(i1-1,3))^2)+((BC_nova(i1,4)-BC_nova(i1-
1,4))^2))^0.5); %antes
                custo=custo-((((BC(i1+1,3)-BC(i1,3))^2)+((BC(i1+1,4)-
BC(i1,4))^2))^0.5)+((((BC_nova(i1+1,3)-BC_nova(i1,3))^2)+((BC_nova(i1+1,4)-
BC_nova(i1,4))^2))^0.5); %depois
            end
```

Figura 38. Trecho 25 do código implementado.

Para economizar tempo computacional, a cada troca o valor do custo como um todo não será recalculado. Afinal, existem dezenas de unidades, mas apenas a distância das rotas ao redor de duas unidades será alterado. Por isso, serão modificados apenas o custo ao redor das duas unidades cujas posições foram trocadas.

Deve-se atentar para as exceções que existem. Por exemplo, caso a unidade trocada seja a primeira de um *cluster* o cálculo deverá levar em conta não uma unidade, mas sim a distância do porto a esta. Se for o último, o cálculo também deverá ser diferente por levar em conta que deverá ser também somada a distância desta unidade até o porto.

```
%%%%%%%%%%Segunda linha trocada%%%%%%%%%%%%%%%%%%%%

             if (i2==1) || (BC_nova(i2,1)~=BC_nova(i2-1,1)) %primeiro
                 if (i2+1==i1) %o exatamente posterior ja foi mudado
                    custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %antes

                 else
                     if (i2==linhas) || (BC_nova(i2,1)~=BC_nova(i2+1,1)) %primeiro
e ultimo
                     custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %antes
                     custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %depois

                     else %apenas primeiro
                     custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %antes
                     custo=custo-((((BC(i2+1,3)-BC(i2,3))^2)+((BC(i2+1,4)-
BC(i2,4))^2))^0.5)+((((BC_nova(i2+1,3)-BC_nova(i2,3))^2)+((BC_nova(i2+1,4)-
BC_nova(i2,4))^2))^0.5); %depois
                     end
                    end

            elseif (i2==linhas) || (BC_nova(i2,1)~=BC_nova(i2+1,1)) %último
                if (i2-1==i1) %o exatamente anterior ja foi mudado
                     custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %depois
                else %apenas ultimo
                     custo=custo-((((BC(i2,3)-BC(i2-1,3))^2)+((BC(i2,4)-BC(i2-
1,4))^2))^0.5)+((((BC_nova(i2,3)-BC_nova(i2-1,3))^2)+((BC_nova(i2,4)-BC_nova(i2-
1,4))^2))^0.5); %antes
                     custo=custo-((((BC(i2,3)-macae_lat)^2)+((BC(i2,4)-
macae_long)^2))^0.5)+((((BC_nova(i2,3)-macae_lat)^2)+((BC_nova(i2,4)-
macae_long)^2))^0.5); %depois
                end

            else %meio
                if (i2-1==i1) %o anterior ja foi mudado
                     custo=custo-((((BC(i2+1,3)-BC(i2,3))^2)+((BC(i2+1,4)-
BC(i2,4))^2))^0.5)+((((BC_nova(i2+1,3)-BC_nova(i2,3))^2)+((BC_nova(i2+1,4)-
BC_nova(i2,4))^2))^0.5); %depois
                elseif (i2+1==i1) %o posterior ja foi mudado
                     custo=custo-((((BC(i2,3)-BC(i2-1,3))^2)+((BC(i2,4)-BC(i2-
1,4))^2))^0.5)+((((BC_nova(i2,3)-BC_nova(i2-1,3))^2)+((BC_nova(i2,4)-BC_nova(i2-
1,4))^2))^0.5); %antes
                else
                     custo=custo-((((BC(i2,3)-BC(i2-1,3))^2)+((BC(i2,4)-BC(i2-
1,4))^2))^0.5)+((((BC_nova(i2,3)-BC_nova(i2-1,3))^2)+((BC_nova(i2,4)-BC_nova(i2-
1,4))^2))^0.5); %antes
                     custo=custo-((((BC(i2+1,3)-BC(i2,3))^2)+((BC(i2+1,4)-
BC(i2,4))^2))^0.5)+((((BC_nova(i2+1,3)-BC_nova(i2,3))^2)+((BC_nova(i2+1,4)-
BC_nova(i2,4))^2))^0.5); %depois
                end
             end
```

Figura 39. Trecho 26 do código implementado.

Repete-se o mesmo procedimento para os custos referentes à segunda unidade envolvida na troca. Da mesma maneira, deve-se prestar atenção nas exceções que existem caso esta unidade seja a primeira e/ou a última de um *cluster*. Entretanto,

agora, deve-se também prestar atenção nos casos em que a primeira troca realizada foi em uma linha vizinha a esta. Nesses casos, já houve a substituição do custo entre a primeira unidade trocada e a posição atualmente trocada. Por isso, ao se fazer o procedimento de alteração no custo, ao subtrair o custo antigo e adicionar o custo novo, deve-se levar em conta que o custo antigo não é o original, mas sim já foi alterado.

```
if custo<custo_escolhida
            a=1;
            demanda=zeros(linhas,1);

            while a<=linhas
                if (a==1) || (BC_nova(a,1)~= BC_nova(a-1,1))
                     demanda(a,1)=0+BC_nova(a,2);
                else
                    demanda(a,1)=demanda(a-1,1)+BC_nova(a,2);
                end
                a=a+1;
            end

            a=1;
```

Figura 40. Trecho 27 do código implementado.

Como descrito anteriormente, somente poderá ser candidato a mínimo aqueles cujo custo forem menores do que o da iteração(***tentativa***) anterior, representado pela variável ***custo_escolhida***.

Caso isso ocorra, deve-se ainda verificar se a nova configuração não rompe a capacidade dos barcos. Para isso, cria-se uma variável ***demanda***, em que cada linha i representará a capacidade demandada de cada barco somando-se as demandas até a unidade i.

Para que isso seja possível, inicialmente é criado um vetor inicialmente composto por zeros com dimensão igual ao número de unidades clusterizadas(***linhas***). Em seguida, percorre-se todas as unidades, realizando o somatório da capacidade demandada desde o início do *cluster* até aquela unidade e imprimindo cada resultado para um elemento diferente da matriz de demandas inicialmente criada. Caso seja detectado que passou-se para outro *cluster*, então o somatório recomeça do zero.

```
aux=1;
          for a=1:linhas
                 if  a==linhas || BC_nova(a,1)~=BC_nova(a+1,1)
                     demanda_total=demanda(a,1);
                     if demanda_total>BC_utilizada(a,5)
                         text='nao coube';
                         aux=0;
                     end
                 else
                     posicao_cluster=1;
                     for b=1:linhas
                         if BC_nova(b,1)==BC_nova(a,1) && b~=a
                             diferenca_janelas(posicao_cluster,1)=abs(BC_nova(b,6)-
BC_nova(a,6));
                             if
min(diferenca_janelas)<diferenca_maxima_de_janelas_de_tempo
                                 aux_janela=1

                             else
                                 aux=0;
                             end
                         end
                     end
                 end
```

Figura 41. Trecho 28 do código implementado.

Com a matriz de capacidade demandada do barco ao longo das unidades definida, cabe agora descobrir se esta demanda rompe a capacidade disponível no barco. Para descobrir a capacidade demandada ao fim de cada cluster, percorre-se toda a matriz procura-se pelas posições em que detecta-se o fim de um cluster. Nessas posições, procura-se o valor na matriz do somatório das demandas e compara-se com a capacidade do barco que atende ao *cluster*. Se a capacidade do barco for menor do que a demanda, então atribuímos a uma variável ***aux*** o valor zero, impedindo que esta configuração seja usada.

Caso esta configuração seja aprovada pelo critério da capacidade do barco, avalia-se esta configuração sobre o critério da diferença entre o início das janelas de tempo dentro de cada *cluster*. Isso é importante pois uma configuração com inícios de janelas de tempo muito diferentes pode se tornar inviável durante as etapas posteriores da pesquisa que envolvem a definição da programação do porto e a capacidade de atender às janelas de tempo.

Para analisar as janelas, elege-se um elemento por vez de dentro do cluster e compara-se sua janela com as outras janelas do mesmo *cluster*. Repete-se o mesmo procedimento para todas as unidades. O objetivo é que encontre-se pelo menos uma outra janela neste *cluster* que ocorra em uma diferença de tempo menor do que o

valor do parâmetro ***diferenca_maxima_de_janelas_de_tempo*** inicialmente definido no início do código. Caso isso não exista, também atribui-se à variável ***aux*** o valor zero, descartando-se esta configuração.

Caso não haja nenhuma atribuição de zero à variável aux, esta configuração não está descartada e pode ser utilizada.

```
if aux==1 && tentativa_seed~=1

[linhas_ultimas,colunas_ultimas]=size(ultimas_configuracoes);

                        if colunas_ultimas>=501
                            inicio_ultimas=colunas_ultimas-500;
                        else
                            inicio_ultimas=1;
                        end


                        for percorre_ultima=inicio_ultimas:colunas_ultimas
                            analisado1=BC_nova(:,6);
                            analisado2=ultimas_configuracoes(:,percorre_ultima);
                            [linha1,coluna1]=size(analisado1);
                            [linha2,coluna2]=size(analisado1);
                            if linha1~=linha2
                                if
BC_nova(:,6)==ultimas_configuracoes(:,percorre_ultima)
                                    existe_config_igual=1;
                                    posicao_igual=percorre_ultima
                                else
                                    existe_config_igual=0;
                                end
                            end
                        end
                    end
```

Figura 42. Trecho 29 do código implementado.

Com o que foi feito até agora, existiria chances de a falta de um horizonte de soluções gerar loopings com a mesma solução ou com um conjunto de soluções se repetindo infinitamente. Isso poderia acontecer, por exemplo, caso nenhuma troca fosse mais possível e os seeds fossem redefinidos como os mesmos. Para evitar isso, implementa-se, então, neste ponto, uma rotina que confira se houve repetição desta mesma configuração alguma outra vez. Caso isso seja detectado, em outro momento futuro do código é implementada uma aleatoriedade, que faz com que novas soluções sejam encontradas.

Deve-se, então, implementar um registro com as configurações antigas já utilizadas. Como já há uma coluna na matriz analisada composta por tags que representam o nome da unidade, isso foi aproveitado para compor este registro. Para evitar que seja

demandado um tempo computacional demasiadamente grande, não são registrados todas as configurações utilizadas desde o início do programa. Optou-se por guardar apenas as 500 últimas configurações.

Assim, a cada nova solução encontrada, percorre-se todas as últimas 500 soluções (ou menos, caso ainda não existam 500 soluções). Caso não exista uma solução igual, atribui-se à variável ***existe_config_igual*** o valor 0, fazendo-se com que esta solução possa ser posteriormente aceita. Caso seja encontrada alguma solução igual, é dado valor 1 à variável binária ***existe_config_igual***, além de ser registrado em que posição isso ocorre através da variável ***posicao_igual***.

```
if aux==1 && aux_janela==1
        if tentativa_seed==1 || existe_config_igual==0

                BC_utilizada=BC_nova;
                custo_escolhida = custo;
                aviso='Nova clusterizacao escolhida';
                achou_nova_solucao=1;
        else
                aviso='voltou a uma solucao antiga';
                existe_config_igual=1;
        end
end
```

Figura 43. Trecho 30 do código implementado.

Se a troca de unidades não exceder a capacidade do barco e também nenhuma outra configuração igual for encontrada, então esta solução é aceita e atribui-se à variável ***achou_nova_solucao*** o valor 1. Considera-se esta nova configuração como a ***BC_utilizada*** e atribui-se à variável ***custo_escolhida*** o seu custo. A partir de agora, portanto, tenta-se achar soluções com custos menores do que esse.

Caso alguma dessas condições não ocorra, a variável recebe valor zero e a solução não será considerada.

```
[l,final_rodada]=size(ultimas_configuracoes);
if tentativa_seed~=1
     percorre_1=coluna_config;
        for percorre_2=1:final_rodada
            if percorre_1~=percorre_2
                if
ultimas_configuracoes(:,percorre_1)==ultimas_configuracoes(:,percorre_2)

total_sol_iguais_todos(percorre_1,1)=total_sol_iguais_todos(percorre_1,1)+1;
                    total_sol_iguais=max(total_sol_iguais_todos);
                    if total_sol_iguais>=3
                        achou_nova_solucao=0;
                        text='existe solucoes iguais';
                        registrar=1;
                    end
                end
            end
        end
    end
```

Figura 44. Trecho 31 do código implementado.

Ao final do processo iterativo de trocas, verifica-se se foi encontrada uma solução igual a outra já registrada, para evitar que entre-se em um *looping* de soluções iguais. Para isso, percorre-se a matriz com as configurações anteriormente escolhidas e compara-se todas as colunas entre si. Registra-se o número total de repetições para cada coluna. Caso o valor máximo deste valor dentre todas as configurações exceda um valor predeterminado, admite-se que o programa entrou em *looping* e novas soluções não serão encontradas. Neste caso, atribui-se à variável ***achou_nova_solucao*** o valor 0.

```
if achou_nova_solucao==0
        coluna_config=coluna_config+1;
        if coluna_config>500
            coluna_config=1;
        end
    end
```

Figura 45. Trecho 32 do código implementado.

Caso não tenha-se chegado a nenhuma solução idêntica no passado, nesta parte do código define-se em que posição será guardada a atual configuração na matriz que registra as configurações por que já se passou, a matriz ***ultimas_configuracoes.*** Caso chegue-se a linha 500, volta-se ao início da matriz, evitando-se que a matriz ***ultimas_configuracoes*** cresça infinitamente. Para os outros casos, registramos a atual configuração na posição exatamente posterior ao do último registro.

No caso de uma solução igual a outra anterior ser encontrada, não muda-se de coluna. Assim, na próxima iteração, após ser possível achar novas soluções com uma aleatoriedade que será gerada em breve, o elemento problemático será eliminado e não se entrará em um looping que sempre acusará a existência de muitas soluções iguais.

```
if custo_antiga==custo_escolhida
        passo=passo+1;
       variacao_custo(passo,1)=custo_escolhida-custo_primeira_rodada;
        historico_clusterizacoes(:,:,passo)=BC_utilizada(:,:);
        tentativa=100;
```

Figura 46. Trecho 33 do código implementado.

Como já citado anteriormente, será realizado trocas de posições de unidades até se chegar à troca de número 100 ou até nenhuma nova troca resultar em novas soluções. Este trecho do código serve exatamente para identificar que não houve novas soluções. Essa situação é identificada em caso de o custo ser idêntico ao custo da última rodada. Caso isso ocorra, atribui-se à variável ***tentativa*** o valor 100 e o looping termina.

Como não garante-se que após a reclusterização haverá soluções melhores, a saída do nosso programa virá daqui. Guarda-se, então, os dados referentes a esta solução: a variação do custo em relação a configuração inicial é registrada no vetor ***variacao_custo*** e a configuração referente a cada um desses custos é registrada na matriz ***historico_clusterizacoes***.

```
if tentativa_seed==1 ||  variacao_custo(passo,1)<variacao_minimo_custo
            variacao_minimo_custo=variacao_custo(passo,1);
            BC_custo_minimo=BC_utilizada;
```

Figura 47. Trecho 34 do código implementado.

Para análise posterior no final do programa, cataloga-se neste trecho do programa qual foi a melhor solução até agora. O menor custo encontrado será atribuído à variável ***variacao_minimo_custo*** e a configuração referente a este custo estará na matriz ***BC_custo_minimo***.

```
                if tentativa_seed==1
                 aumento_da_capacidade_nos_clusters_escolhida=0;
                 capacidade_restante_em_cada_barco_escolhida=0;
             else

aumento_da_capacidade_nos_clusters_escolhida=aumento_da_capacidade_nos_clusters
                pos_capacidade=0;
                for percorre_l=1:linhas
                        if percorre_l==1 ||
BC_custo_minimo(percorre_l,1)~=BC_custo_minimo(percorre_l-1,1)
                            pos_capacidade=pos_capacidade+1;

capacidade_demandada_de_cada_barco(pos_capacidade,1)=BC_custo_minimo(percorre_l,2);
                        else

capacidade_demandada_de_cada_barco(pos_capacidade,1)=capacidade_demandada_de_cada_b
arco(pos_capacidade,1)+BC_custo_minimo(percorre_l,2);
                        end
                end


                for percorre_linha=1:linhas_seed_antiga
                    if percorre_linha==1
                         seeds(percorre_linha,1)=seeds(percorre_linha,1);
                    end
                    if percorre_linha>linhas_seed
                        seeds(percorre_linha,5)=600;
                    end
                end

                capacidade_restante_em_cada_barco_escolhida=seeds(:,5)-
capacidade_demandada_de_cada_barco(:,1);
            end
         end
    end
```

Figura 48. Trecho 35 do código implementado.

Em um ponto posterior do código, na fase de reclusterização, é permitido alterar a capacidade do barco que atende a cada cluster. Pra se ter uma noção de como a capacidade se altera, o aumento ou diminuição da capacidade nos clusters é guardada na variável ***aumento_da_capacidade_nos_clusters***. Regista-se, também, como isso se organiza para a configuração de custo mínimo, que é a que será escolhida, na variável ***aumento_da_capacidade_nos_clusters_escolhida***. É importante registrar isso pois pode ser que na configuração de custo mínimo tenha-se uma configuração de barcos que seja impossível na prática.

Da mesma maneira, compara-se o somatório das demandas de cada cluster com a capacidade do barco que serve a este cluster. Assim, é possível definir a capacidade que está sobrando a cada barco para a caso escolhido. Isso será guardado para ser analisado ao fim do programa.

```
        BC_antiga=BC_utilizada;
         BC=BC_utilizada;
         BC_nova=BC_utilizada;
         tentativa=tentativa+1;
```

Figura 49. Trecho 36 do código implementado.

Ao fim do *looping* de trocas da posição de unidades, zeram-se as variáveis que mudam ao longo da iteração.

Chega-se, com isso, ao fim do processo iterativo de trocas de unidades. Agora, parte-se para a segunda parte do código, em que serão definidos *seeds* e, a partir deles, novos clusters serão montados.

```
inicio_percorre=1;
num_seeds=1;
soma_lat=0;
soma_long=0;
numero_de_unidades=0;
linhas;
percorre_linhas=1;
seeds=0;
```

Figura 50. Trecho 37 do código implementado.

Primeiramente, inicializa-se algumas variáveis que serão utilizadas no processo de construção dos *seeds*. A partir de agora, pretende-se criar novos clusters a partir unidades que serão definidas como *seeds*, que são aquelas unidades que estarão mais próximas do centro geométrico de cada *cluster*.

```
if achou_nova_solucao==0 && tentativa_seed~=1
    numero_aleatorio=rand(guarda_numero_inicial_seeds,1)
    unidade_intermediario=linhas*numero_aleatorio
    unidade_escolhida=ceil(unidade_intermediario);
    total_sol_iguais_todos=zeros(500,1);
    for percorre_seed=1:guarda_numero_inicial_seeds
        unidade_escolhida_como_seed=unidade_escolhida(percorre_seed,1);
        seeds(percorre_seed,1)=BC_utilizada(unidade_escolhida_como_seed,1);
        seeds(percorre_seed,2)=BC_utilizada(unidade_escolhida_como_seed,2);
        seeds(percorre_seed,3)=BC_utilizada(unidade_escolhida_como_seed,3);
        seeds(percorre_seed,4)=BC_utilizada(unidade_escolhida_como_seed,4);
        seeds(percorre_seed,5)=BC_utilizada(unidade_escolhida_como_seed,5);
    end
```

Figura 51. Trecho 38 do código implementado.

Para os casos de *looping* com soluções constantemente iguais como já discutidos anteriormente é neste trecho do código que implementa-se uma solução.

Anteriormente, foi definido que, em caso de as soluções encontradas serem iguais a soluções antigas, a variável ***achou_nova_solucao*** seria zero, marcando a ocorrência deste problema. Assim, caso isto ocorra, esta seção do código é ativada. A única exceção em que esta situação não é verificada ocorre quando ***tentativa_seed*** é igual a 1, porque neste período não necessariamente a variável ***achou_nova_solucao*** já está definida.

Para proporcionar uma rotina que elimina a chance de que continue-se chegando sempre a soluções iguais, uma perturbação deve ser realizada no problema com o objetivo de que ele chegue a novas soluções. Optou-se por gerar seeds aleatórios neste caso. Para isso, inicialmente, gera-se uma quantidade de números aleatórios iguais ao número de clusters que ocorre na configuração vinda da planilha de entrada. Este número aleatório poderá ser qualquer número entre 0 e 1. Em seguida, multiplica-se este valor pelo número de unidades. Depois, arredonda-se o valor para o inteiro exatamente superior, evitando-se assim que escolha-se a unidade zero, que não existe.

```
else

while percorre_linhas<=linhas

    if
(percorre_linhas==linhas)||(BC_utilizada(percorre_linhas,1)~=BC_utilizada(percorre_
linhas+1,1))

        numero_de_unidades=numero_de_unidades+1;
        soma_lat=soma_lat+BC_utilizada(percorre_linhas,3);
        soma_long=soma_long+BC_utilizada(percorre_linhas,4);
        media_lat=(soma_lat/numero_de_unidades);
        media_long=(soma_long/numero_de_unidades);
        soma_lat=0;
        soma_long=0;
elseif BC_utilizada(percorre_linhas,1)==BC_utilizada(percorre_linhas+1,1)
        soma_lat=soma_lat+BC_utilizada(percorre_linhas,3);
        soma_long=soma_long+BC_utilizada(percorre_linhas,4);
        numero_de_unidades=numero_de_unidades+1;
    end
```

Figura 52. Trecho 39 do código implementado.

Para os casos usuais, em que não há a repetição de soluções, prossegue-se normalmente como descrito no artigo de Koskosidis&Powell[20]. Segundo esta metodologia, o primeiro passo consiste em determinar qual é o centro geométrico de

cada *cluster*. Para isso, percorre-se cada cluster somando as latitudes. Ao fim, divide-se o resultado pelo número de unidades, encontrando-se assim uma média para as latitudes. Isso é feito da mesma maneira também para a longitude. Ao início de cada cluster o somatório é zerado e a recomeça novamente.

```
if
(percorre_linhas==linhas)||(BC_utilizada(percorre_linhas,1)~=BC_utilizada(percorre_
linhas+1,1))


    soma_minimizada=1000;

    percorre_linhas2=inicio_percorre;




    while (percorre_linhas2)<=(inicio_percorre+numero_de_unidades-1)


        if numero_rodada==1 || num_seeds<=guarda_numero_inicial_seeds

        diferenca_lat=BC_utilizada(percorre_linhas2,3)-media_lat;
        diferenca_long=BC_utilizada(percorre_linhas2,4)-media_long;
        soma_das_diferencas=(diferenca_lat^2+diferenca_long^2)^0.5;



        if (soma_das_diferencas<soma_minimizada)
            soma_minimizada=soma_das_diferencas;
            seeds(num_seeds,1)=BC_utilizada(percorre_linhas2,1);
            seeds(num_seeds,2)=BC_utilizada(percorre_linhas2,2);
            seeds(num_seeds,3)=BC_utilizada(percorre_linhas2,3);
            seeds(num_seeds,4)=BC_utilizada(percorre_linhas2,4);
            seeds(num_seeds,5)=BC_utilizada(percorre_linhas2,5);
        end
        percorre_linhas2=percorre_linhas2+1;
        else
            percorre_linhas2=percorre_linhas2+1;
    end
    end
            num_seeds=num_seeds+1;
            soma_minizada=1000;
            inicio_percorre=inicio_percorre+numero_de_unidades;
            numero_de_unidades=0;
    end



    percorre_linhas=percorre_linhas+1;
end
    end
```

Figura 53. Trecho 40 do código implementado.

Com o centro geométrico do *cluster* definido, resta agora descobrir qual é a unidade que encontra-se mais perto deste e que, portanto, será o seed deste cluster. Para isso, a cada nova definição de centro geométrico, verifica-se qual a distância de cada unidade até este centro. Compara-se sucessivamente as distâncias encontradas e

define-se aquela que esteja mais próxima. Quando isso ocorre, a matriz seeds é preenchida com os dados dessa unidade.

Essa matriz caracteriza-se por ter as mesmas características das matrizes com que se trabalhou até agora, com a diferença de que agora trabalha-se apenas com as unidades que são *seeds.*

Em outras fases do código, é possível que a clusterização atualmente analisada tenha que ter tido um incremento no número de clusters para que a configuração se tornasse viável. Como o programa é iterativo, a tendência seria que este número maior de clusters fosse mantido e cada vez incrementado ainda mais. Deve-se, então, criar um ponto no código que evite que novos clusters sejam criados infinitamente. Por isso, aqui, limita-se o número de *seeds* ao número de clusters que existiam originalmente. Inclusive, isso evita que sejam formados *seeds* para *clusters* criados com sobras de clusterizações anteriores e que, por isso, não apresentam necessariamente clusters perto um dos outros.

Por isso, limita-se o número de *seeds* ao número de *clusters* inicial, presente na tabela com os dados de entrada, impedindo-se que seja criado mais *seeds* do que a variável ***guarda_numero_inicial_seeds.***

```
if numero_rodada==1
        guarda_numero_inicial_seeds=num_seeds-1;
    end
    numero rodada=numero rodada+1;
```

Figura 54. Trecho 41 do código implementado.

A variável ***guarda_numero_inicial_seeds*** é criada também neste ponto do código. Ela representa o número de *seeds* que foram criados na primeira das tentativas, ou seja, um número diretamente baseado nos dados de entrada.

```
[linhas_seed,colunas_seed]= size(seeds);
numero_proximo_seed=linhas_seed+1;
```

Figura 55. Trecho 42 do código implementado.

Em caso de ser impossível adicionar uma unidade a qualquer *cluster*, futuramente será necessária a criação de novos *clusters* para alocar esta unidade. Deve-se, entretanto, saber qual é o número do próximo cluster que deva ser criado. Para isso, aproveita-se este ponto do programa, inicializando-se uma variável cujo objetivo é registrar qual é o número de *seeds* que existem até agora. Ela será denominada ***numero_proximo_seed*** e a partir dela que futuramente será decidido qual o identificador numérico que será atribuído a qualquer novo *cluster* que venha a ser criado.

```
distancias_ate_seeds_original=0;

for contador_linha_BC=1:linhas
    numero_do_cluster=BC_utilizada(contador_linha_BC,1);
    for numero_do_seed=1:linhas_seed
distancias_ate_seeds(numero_do_seed,contador_linha_BC)=((BC_utilizada(contador_linh
a_BC,3)-seeds(numero_do_seed,3))^2+(BC_utilizada(contador_linha_BC,4)-
seeds(numero_do_seed,4))^2)^0.5;

distancias_ate_seeds_original(numero_do_seed,contador_linha_BC)=distancias_ate_seed
s(numero_do_seed,contador_linha_BC);
    end
        distancia_perto=10000;
        distancia_longe=10000;
        for percorre_distancias=1:linhas_seed
            if
distancias_ate_seeds(percorre_distancias,contador_linha_BC)<distancia_perto

distancia_perto=distancias_ate_seeds(percorre_distancias,contador_linha_BC);
                unidade_mais_perto(contador_linha_BC)=percorre_distancias;
                pos1=percorre_distancias;
                pos2=contador_linha_BC;
            end
        end
                distancias_ate_seeds(pos1,pos2)=9999;


        for percorre_distancias=1:linhas_seed
            if
distancias_ate_seeds(percorre_distancias,contador_linha_BC)<distancia_longe

distancia_longe=distancias_ate_seeds(percorre_distancias,contador_linha_BC);
                unidade_mais_perto(contador_linha_BC)=percorre_distancias;
            end
        end
        regret(contador_linha_BC,1)=distancia_longe-distancia_perto;
end
```

Figura 56. Trecho 43 do código implementado.

Ao chegar nesse ponto, com os *seeds* já definidos, pode-se finalmente atribuir as unidades aos *clusters*. Segundo o artigo de Koskosidis&Powell[20], deve-se clusterizar inicialmente as unidades com o maior valor da função Regret.

A função Regret exige que saiba-se qual é o primeiro e o segundo *seeds* mais próximos para cada unidade. Para isso, é feito então dois *loopings*, um percorrendo

todas as unidades e outro percorrendo todos os *seeds*. O objetivo é que, para cada unidade, monte-se uma matriz denominada ***distancias_ate_seeds***, que mostre a distância desta unidade até cada um dos *seeds*. Copia-se esta matriz também na ***distancias_ate_seeds_original*** para que seja possível guardar os dados iniciais em uma e manipular a outra sem a perda de dados.

Com a distância até os *seeds* definida, o próximo passo consiste em descobrir a ordem de proximidade destes. Para isso, cria-se um outro *looping*, cujo objetivo é determinar qual é o menor dos valores de distância¡, guardando-o na variável ***distancia_perto*** e eliminando-se este valor de um segundo levantamento, substituindo-o por um número com valor muito grande. Repete-se, então o procedimento, agora sem a distância anteriormente selecionada, retornando-se, assim, a segunda distância mais perto, que atribuímos à variável ***distancia_longe***.

Com esses dois elementos definidos, pode-se calcular a função Regret para essa unidade. Repete-se o procedimento para todas as outras unidades.

```
soma_demanda=zeros(linhas_seed,1);

regret_ja_escolhido=1000;

    num_unidade_mexida=zeros(linhas,1);
    parada=100;
```

Figura 57. Trecho 44 do código implementado.

O próximo objetivo é começar a alocar as unidades nos *clusters* de acordo com a função regret. Como preparação para esta fase, algumas variáveis que serão utilizadas ao longo do procedimento são inicializadas.

```
while parada>-1
         maior_valor_regret=0;
        for percorre_regret=1:linhas
            if regret(percorre_regret,1)>=maior_valor_regret

                maior_valor_regret=regret(percorre_regret,1);
                pos=percorre_regret;
            end
        end

           regret(pos,1)=-1;
           parada=max(regret);
```

Figura 58. Trecho 45 do código implementado.

Através do looping ***percorre_regret***, procura-se dentre as unidades qual é aquela que possui o maior valor de Regret e, por causa disso, deve ser reclusterizada primeiramente. Quando encontra-se esta unidade, atribui-se sua posição na matriz à variável ***pos***, que levará para as partes posteriores do código a informação de qual unidade deve ser clusterizada.

Após isso, atribui-se à variável Regret afetada o valor -1, eliminando-se este valor de futuras análises. A ideia é repetir este procedimento quantas vezes forem necessárias até que o máximo da função Regret seja -1, ou seja, até que todas as unidades presentes sejam reclusterizadas.

```
distancia_ate_seeds_escolhida=9999;
matriz_ordem_clusterizacao=zeros(1,1);

posicao_ordem_clusterizacao=1;
define=1;
matriz_avaliada=0;
matriz_avaliada=distancias_ate_seeds_original(:,pos);
cont=0;

while define>0
    for percorre_prioridade_seed=1:linhas_seed
    if matriz_avaliada(percorre_prioridade_seed,1)<distancia_ate_seeds_escolhida;
        posicao_em_que_ocorre=percorre_prioridade_seed;

distancia_ate_seeds_escolhida=distancias_ate_seeds_original(percorre_prioridade_see
d,pos);
    end
    end
    matriz_avaliada(posicao_em_que_ocorre,1)=9999;

matriz_ordem_clusterizacao(posicao_ordem_clusterizacao,1)=posicao_em_que_ocorre;
    posicao_ordem_clusterizacao=posicao_ordem_clusterizacao+1;
    distancia_ate_seeds_escolhida=9999;

if min(matriz_avaliada)==9999
    define=0;
end
cont=cont+1;
end
```

Figura 59. Trecho 46 do código implementado.

Deve-se, agora, alocar a unidade de posição ***pos***, que foi definida anteriormente. Entretanto, não necessariamente garante-se que será possível alocar esta unidade no *cluster* mais próximo. Pode ser que, devido à falta de espaço no *cluster*, seja necessário alocar a unidade no segundo *cluster* mais próximo ou até mais longe. Por isso, é essencial que seja montada uma matriz que mostre a ordem de preferência de clusters. Esta ordem de preferência estará relacionada a distância da unidade aos seeds de cada *cluster*.

Para fazer isso, então a matriz ***distancias_ate_seeds_original*** é replicada na matriz ***matriz_avaliada*** para que esta segunda possa ser modificada. A seguir, percorre-se essa matriz a procura da menor distância. Ao se chegar na menor distância, atribuiremos a este elemento o valor 9999 e repete-se o procedimento até todos os elementos se transformarem em 9999, só que agora, sem considerar as distâncias já escolhidas. Assim, é possível catalogar todas as distâncias até seeds em ordem.

O objetivo é obter a matriz ***posicao_ordem_clusterizacao*** ordenanda, respondendo-se qual a prioridade de *clusters* a que a unidade considerada deve ser alocada.

```
posicao=1;
        foi_alocado=0;

        passou_por_aqui=0;

        posicao_cluster_afetado2=1;

        while foi_alocado<1

        cluster_afetado=matriz_ordem_clusterizacao(1,1);

        tenta_aumentar_cluster=0;

        while tenta_aumentar_cluster<2
         if num_unidade_mexida(cluster_afetado,1)==0
             var_aux1=0;
         else
            var_aux1=sum(demanda_novo(cluster_afetado,:),2);
         end

         linha_janela=linhas_seed;
```

Figura 60. Trecho 47 do código implementado.

A seguir, definimos os *looping* que serão responsáveis pela alocação de todas as unidades. O primeiro *looping* irá ocorrer até a unidade for alocada. Já o segundo looping será responsável por atuar em casos de inviabilidade da configuração, tentando-se aumentar o tamanho do barco que serve a cada um dos clusters, de acordo com uma ordem de prioridades.

```
if cluster_afetado<linha_janela
         percorre_janela=1;
         [linhas_janela_nao_usa,coluna_janela]=size(inicio_janela_novo)
        while percorre_janela<=coluna_janela && percorre_janela~=0
            mexido=num_unidade_mexida(cluster_afetado,1);
            if  mexido+1~=1 && abs(BC_utilizada(pos,6)-
inicio_janela_novo(cluster_afetado,percorre_janela))>diferenca_maxima_de_janelas_de
_tempo
                janelas_muito_diferentes=1;
            else
                janelas_muito_diferentes=0;
                percorre_janela=coluna_janela;
            end
            percorre_janela=percorre_janela+1;
        end
         end
```

Figura 61. Trecho 48 do código implementado.

Assim como antes, ao se trocar os *clusters*, as janelas de tempo podem ser rearranjadas de maneira que se impossibilite a formação dos clusters na prática. Caso já haja uma unidade alocada em um *cluster*, verifica-se aqui se existe qualquer outra unidade no *cluster* que apresente uma janela de tempo cujo início esteja mais perto do que o máximo pré-estabelecido. Caso não haja uma janela compatível, então atribui-se à variável ***janelas_muito_diferentes*** o valor 1 e posteriormente outras medidas serão adotadas.

```
if seeds(cluster_afetado,5)>=var_aux1+BC_utilizada(pos,2) &&
soma_unidades(cluster_afetado,1)+1<=numero_maximo_de_unidades_por_clusters &&
janelas_muito_diferentes==0

            soma_unidades(cluster_afetado,1)=soma_unidades(cluster_afetado,1)+1;
```

Figura 62. Trecho 49 do código implementado.

Devido a limitações de robustez do código e exigência de tempo computacional inviável, a segunda parte do trabalho, o modelo de roteamento considerando janelas de tempo, só pode trabalhar com um número limitado de unidades por vez. Como é impossível analisar todas as unidades de uma vez só no modelo de roteamento já que o número de unidades é bastante grande, próximo de 60, então definiu-se que serão analisados 3 clusters saídos do algoritmo de clusterização por vez.

Assim, como existe uma limitação quanto ao número de unidades que podem ser analisadas por vez no roteamento, então é limitado o número de unidades presentes em cada *cluster*. Sabe-se que a limitação do algoritmo de roteamento é de

aproximadamente 12 unidades por vez. Assim, limitou-se o número de unidades por cluster a 4. Nesta parte do código, é realizado este somatório de unidades.

A estrutura condicional no início deste trecho de código é essencial no código. Ela impede que aloque-se unidades a *clusters* rompendo-se a capacidade do barco. Também é ela que impede que aloque-se mais de 4 unidades por cluster e se inviabilize o roteamento feito posteriormente. Por fim, ela também impede que haja unidades com janelas de tempo muito diferentes dentro de um mesmo cluster.

```
        num_unidade_mexida(cluster_afetado,1)=num_unidade_mexida(cluster_afetado,
1)+1;

         mexido=num_unidade_mexida(cluster_afetado,1);

         num_cluster_novo(cluster_afetado,mexido)=cluster_afetado;
         demanda_novo(cluster_afetado,mexido)=BC_utilizada(pos,2);
         latitude_novo(cluster_afetado,mexido)=BC_utilizada(pos,3);
         longitude_novo(cluster_afetado,mexido)=BC_utilizada(pos,4);
         capacidade_novo(cluster_afetado,mexido)=seeds(cluster_afetado,5);
         nome_unidade_novo(cluster_afetado,mexido)=BC_utilizada(pos,6);
         inicio_janela_novo(cluster_afetado,mexido)=BC_utilizada(pos,7);
         fim_janela_novo(cluster_afetado,mexido)=BC_utilizada(pos,8);
         tempo_servico_novo(cluster_afetado,mexido)=BC_utilizada(pos,9);

           foi_alocado=1;

           tenta_aumentar_cluster=2;
```

Figura 63. Trecho 50 do código implementado.

Chegamos, então na parte do código em que efetivamente começa-se a montar os novos clusters. Para facilitar os procedimentos, trabalha-se nessa parte com matrizes um pouco diferentes das utilizadas até aqui. Haverá uma matriz diferente para cada atributo presente nas colunas das matrizes originais. E, em cada matriz de atributo, linhas diferentes representarão *clusters* diferentes e colunas diferentes representarão unidades diferentes. Assim, há a possibilidade de adicionar unidades a *clusters* que não sejam o último apenas adicionando mais colunas a matriz. Caso fosse utilizado o mesmo modelo de matrizes antigo, seria necessário adicionar linhas no meio da matriz e alterar as linhas de todas as unidades posteriores, o que seria bem mais complexo.

```
else
            posicao=posicao+1;
            if posicao<=linhas_ordem_clusterizacao
                cluster_afetado=matriz_ordem_clusterizacao(posicao,1);
                txt=linhas_ordem_clusterizacao;
            else
                tenta_aumentar_cluster=2;
            end
        end

[linhas_ordem_clusterizacao,colunas_ordem_clusterizacao]=size(matriz_ordem_clusteri
zacao);
```

Figura 64. Trecho 51 do código implementado.

Caso seja impossível alocar a unidade no *cluster*, seja porque rompeu-se a capacidade do barco, seja porque alocou-se mais do que um número pré-determinado a cada *cluster*, seja porque as janelas de temo são incompatíveis, então entra-se nesta fase do código.

O primeiro passo consiste em tentar alocar a unidade em *clusters* mais distantes. Para isso, aumenta-se a variável ***posição*** e avança-se uma posição na ***matriz_ordem_clusterizacao***, repetindo-se o todo o processo para *seeds* mais longínquos. Caso percorra-se todas os *clusters* e ainda assim não consiga-se alocar a unidade, então tenta-se aumentar a capacidade do barco. Para isso, atribui-se à variável ***tenta_aumentar_cluster*** o valor 2, saindo-se deste looping.

```
var_aux2=sum(demanda_novo(cluster_afetado,:),2);
        if seeds(cluster_afetado,5)<=var_aux2+BC_utilizada(pos,2) ||
soma_14_unidades(cluster_afetado,1)+1>numero_maximo_de_unidades_a_cada_3_clusters
|| janelas_muito_diferentes==1

                if foi_alocado~=1
                if passou_por_aqui<linhas_seed
                    seeds(cluster_afetado2,5)=600;

cluster_afetado2=matriz_ordem_clusterizacao(posicao_cluster_afetado2,1);
                    passou_por_aqui=passou_por_aqui+1;
                    posicao_cluster_afetado2=posicao_cluster_afetado2+1;
```

Figura 65. Trecho 52 do código implementado.

Caso tente-se alocar a unidade a todos os *clusters* sem sucesso, prossegue-se agora tentando aumentar a capacidade dos *clusters*. Aumenta-se a capacidade do barco que serve ao *cluster* mais próximo inicialmente e, em caso de insucesso, o procedimento é repetido para *clusters* cada vez mais longes, seguindo a ordem de distância aos *seeds* especificada anteriormente na matriz ***matriz_ordem_clusterizacao***. A variável

passou_por_aqui serve para marcar o número de vezes que esse procedimento é refeito, encerrando o processo em caso de todos os *seeds* já terem tido a capacidade do barco aumentada.

```
else

soma_demanda_proximo_cluster=soma_demanda_proximo_cluster+BC_utilizada(pos,2)
                    if soma_demanda_proximo_cluster<=600

num_unidade_mexida(numero_proximo_seed,1)=num_unidade_mexida(numero_proximo_seed,
1)+1;
                        mexido=num_unidade_mexida(numero_proximo_seed,1);
                        num_cluster_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=numero_proximo_seed;
                        demanda_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,2);
                        latitude_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,3);
                        longitude_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,4);
                        capacidade_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=600;
                        nome_unidade_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,6);
                        inicio_janela_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,7);
                        fim_janela_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,8);
                        tempo_servico_novo_sobra(numero_proximo_seed-
linhas_seed,mexido)=BC_utilizada(pos,9);
                        foi_alocado=1;
                else
                        numero_proximo_seed=numero_proximo_seed+1;
                        soma_demanda_proximo_cluster=0;

                end
                end
```

Figura 66. Trecho 53 do código implementado.

Em caso de insucesso nas tentativas de se aumentar a capacidade dos barcos que servem aos *clusters*, então resta apenas a opção de se alocar a unidade em um *cluster* extra a ser criado. Cria-se um novo *cluster* em uma matriz separada, realizando-se posteriormente a alocação da unidade. A definição do identificador numérico referente ao cluster é definida pela variável numero_proximo_seed e a posição desta unidade dentro do *cluster* é definida pela variável num_unidade_mexida.

```
capacidade_demandada_de_cada_barco=sum(demanda_novo,2);

capacidade_restante_em_cada_barco=seeds(cluster_afetado,5)-
capacidade_demandada_de_cada_barco;
```

Figura 67. Trecho 54 do código implementado.

Guarda-se aqui a capacidade demandada de cada barco e a capacidade restante em cada um. Essas variáveis serão guardadas para que possa-se analisar se na prática essa nova configuração de fato é. Como explicado anteriormente, guarda-se também em uma outra variável as configurações de barcos para o caso com o menor custo.

```
for cluster_afetado=1:linhas_seed
        if capacidade_restante_em_cada_barco(cluster_afetado,1)>200
            seeds(cluster_afetado,5)=400;
        end
        end
```

Figura 68. Trecho 55 do código implementado.

Deve-se atentar, entretanto, para o fato de que até agora não há nenhuma trava para limitar o tamanho dos barcos. A tendência seria que ao fim de várias rodadas todos os clusters acabassem com barcos com a capacidade máxima, já que não há nenhuma forma de diminuir tamanhos de barcos.

Por isso, adicionou-se esse trecho no código. O objetivo é que caso seja possível diminuir o tamanho do barco, isto seja feito. Como considera-se tamanhos de barcos uniformes iguais a 400m² ou 600m² de deck, então analisa-se se há a disponibilidade de 200m² de deck sobrando em algum barco. Caso haja, o tamanho do barco é diminuído.

```
if tentativa_seed==1
            capacidade_guarda(1:linhas_seed,1)=capacidade_novo(1:linhas_seed,1);
        end
```

Figura 69. Trecho 56 do código implementado.

Guarda-se também a capacidade de cada barco na primeira das rodadas para que possa-se analisar a capacidade dos barcos variou.

```
        [linha_final,coluna_final]=size(seeds);
        for percorre_linha=1:linha_final
            if percorre_linha==1
                capacidade_guarda_usa=capacidade_guarda;
            end
            if percorre_linha>linhas_seed
                capacidade_guarda_usa(percorre_linha,1)=0;
            end
        end
        aumento_da_capacidade_nos_clusters=seeds(:,5)-capacidade_guarda_usa(:,1);
        [linhas_capac, colunas_capac]=size(capacidade_novo);
```

Figura 70. Trecho 57 do código implementado.

Por fim, pode-se calcular qual foi o aumento da capacidade dos barcos. Para isso, compara-se as capacidades do barco que atende a cada seeds atual com as capacidades dos barcos iniciais.

```
for cont1=1:linhas_capac
            for cont2=1:colunas_capac
                if capacidade_novo(cont1,cont2)~=0
                   capacidade_novo(cont1,cont2)=seeds(cont1,5);
                end
            end
        end
```

Figura 71. Trecho 58 do código implementado.

Agora que sabe-se quais são as unidades que efetivamente tiveram suas capacidades de barco alteradas, altera-se esses valores na matriz de capacidades, que é aquela que montará a matriz que realmente utilizamos ao longo do programa. Com isso, tornamos as mudanças realizadas definitivas.

```
while colunas1~=colunas2
            if colunas1<colunas2
                num_cluster_novo(1,colunas1+1)=0;
                demanda_novo(1,colunas1+1)=0;
                latitude_novo(1,colunas1+1)=0;
                longitude_novo(1,colunas1+1)=0;
                capacidade_novo(1,colunas1+1)=0;
                nome_unidade_novo(1,colunas1+1)=0;
                inicio_janela_novo(1,colunas1+1)=0;
                fim_janela_novo(1,colunas1+1)=0;
                tempo_servico_novo(1,colunas1+1)=0;
            else
                num_cluster_novo_sobra(1,colunas2+1)=0;
                demanda_novo_sobra(1,colunas2+1)=0;
                latitude_novo_sobra(1,colunas2+1)=0;
                longitude_novo_sobra(1,colunas2+1)=0;
                capacidade_novo_sobra(1,colunas2+1)=0;
                nome_unidade_novo_sobra(1,colunas2+1)=0;
                inicio_janela_novo_sobra(1,colunas2+1)=0;
                fim_janela_novo_sobra(1,colunas2+1)=0;
                tempo_servico_novo_sobra(1,colunas2+1)=0;
             end
                [linhas1,colunas1]=size(num_cluster_novo);
                [linhas2,colunas2]=size(num_cluster_novo_sobra);
        end
```

Figura 72. Trecho 59 do código implementado.

Até agora, trabalhou-se com duas matrizes separadas: uma para os clusters regulares e outra para clusters extras criados. O próximo passo consiste, então, em juntar essas matrizes em uma só. Entretanto não necessariamente as duas matrizes apresentam o mesmo número de colunas, o que impossibilitaria sua concatenação. Para resolver

este problema, adiciona-se colunas de zero em uma das duas matrizes com o objetivo de fazer com que as duas acabem com o mesmo número de colunas.

```
num_cluster_novo=[num_cluster_novo;num_cluster_novo_sobra];
        demanda_novo=[demanda_novo;demanda_novo_sobra];
        latitude_novo=[latitude_novo;latitude_novo_sobra];
        longitude_novo=[longitude_novo;longitude_novo_sobra];
        capacidade_novo=[capacidade_novo;capacidade_novo_sobra];
        nome_unidade_novo=[nome_unidade_novo;nome_unidade_novo_sobra];
        inicio_janela_novo=[inicio_janela_novo;inicio_janela_novo_sobra];
        fim_janela_novo=[fim_janela_novo;fim_janela_novo_sobra];
        tempo_servico_novo=[tempo_servico_novo;tempo_servico_novo_sobra];
```

Figura 73. Trecho 60 do código implementado.

Com as matrizes com o mesmo tamanho de colunas, podemos agora concatená-las. Com isso, são obtidas matrizes únicas para todos os *clusters*. Entretanto, ainda trabalha-se com uma matriz diferente para cada propriedade.

```
%montando novamente a matriz BC
[linhas_da_matriz,colunas_da_matriz]=size(num_cluster_novo);
elemento=1;
for itera_linhas=1:linhas_da_matriz
        for itera_colunas=1:colunas_da_matriz
            if num_cluster_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,1)=num_cluster_novo(itera_linhas,itera_colunas);
            end

             if demanda_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,2)=demanda_novo(itera_linhas,itera_colunas);
             end

              if latitude_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,3)=latitude_novo(itera_linhas,itera_colunas);
              end

               if longitude_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,4)=longitude_novo(itera_linhas,itera_colunas);
               end

                if capacidade_novo(itera_linhas,itera_colunas)~=0
                BC_reclusterizada(elemento,5)=capacidade_novo(itera_linhas,1);

                if nome_unidade_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,6)=nome_unidade_novo(itera_linhas,itera_colunas);
                end

                if inicio_janela_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,7)=inicio_janela_novo(itera_linhas,itera_colunas);
                end

                if fim_janela_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,8)=fim_janela_novo(itera_linhas,itera_colunas);
                end

                if tempo_servico_novo(itera_linhas,itera_colunas)~=0
BC_reclusterizada(elemento,9)=tempo_servico_novo(itera_linhas,itera_colunas);
                elemento=elemento+1;
                end
        end
```

Figura 74. Trecho 61 do código implementado.

Por fim, monta-se novamente a matriz como ela era no início do programa: a cada linha uma unidade diferente e a cada coluna um atributo diferente. A diferenciação de qual *cluster* a que pertence a unidade ocorre através de identificadores numéricos.

```
verifica_variacao_custo=variacao_custo;

while min(verifica_variacao_custo)~=1000
    menor_variacao_custo=1000;
for percorre_variacao_custo=1:linha_variacao_custo
    if verifica_variacao_custo(percorre_variacao_custo,1)<=menor_variacao_custo;
        menor_variacao_custo=verifica_variacao_custo(percorre_variacao_custo,1);
        posicao_menor=percorre_variacao_custo;
    end
end
    verifica_variacao_custo(posicao_menor,1)=1000;
    ordem_custos(passo,1)=posicao_menor;
    passo=passo+1;
end
```

Figura 75. Trecho 62 do código implementado.

Ao fim de todas as iterações obtém-se um vetor com os custos, expresso na variável ***variacao_custo***. Também há um histórico de clusterizações expresso na matriz ***historico_clusterizacoes***. Nessa parte do código, os custos serão ordenados, determinando em que clusterizações foram obtidos as menores distâncias totais percorridas, que são aquelas mais interessantes para a nossa análise.

```
%%%%%%Clusterizacao com o melhor custo

posicao_clusterizacao=ordem_custos(1,1)

 numero_cluster_output=historico_clusterizacoes(:,1,posicao_clusterizacao);
 demanda_output=historico_clusterizacoes(:,2,posicao_clusterizacao);
 latitude_output=historico_clusterizacoes(:,3,posicao_clusterizacao);
 longitude_output=historico_clusterizacoes(:,4,posicao_clusterizacao);
 capacidade_barco_output=historico_clusterizacoes(:,5,posicao_clusterizacao);
 tag_nome_unidade_output=historico_clusterizacoes(:,6,posicao_clusterizacao);
 inicio_janela_output=historico_clusterizacoes(:,7,posicao_clusterizacao);
 fim_janela_output=historico_clusterizacoes(:,8,posicao_clusterizacao);
 tempo_servico_output=historico_clusterizacoes(:,9,posicao_clusterizacao);
```

Figura 76. Trecho 63 do código implementado.

Para que fosse dado uma maior quantidade de opções de soluções para as seções posteriores da pesquisa, foram disponibilizadas em uma tabela de Excel as 10 melhores soluções encontradas pelo algoritmo de clusterização. Para isso, foi realizado o procedimento presente nesta seção do código repetidas vezes, para que

fossem obtidos o melhor resultado, segundo melhor resultado e assim por diante. A posição destas soluções no histórico de clusterizações é definido pela variável ***ordem_custos***, cujos elementos representam a posição na matriz com históricos da melhor solução, segunda melhor solução e assim por diante.

```
[linhas_latitude, colunas_latitude]=size(latitude_output);

 soma_lat=0;

 soma_long=0;

 inicio_cluster=1;

 contador=1;

 for percorre_latitude=1:linhas_latitude

     if  percorre_latitude~=linhas_latitude &&
numero_cluster_output(percorre_latitude,1)==numero_cluster_output(percorre_latitude
+1,1)

          soma_lat=soma_lat+latitude_output(percorre_latitude,1);

          soma_long=soma_long+longitude_output(percorre_latitude,1);

     elseif percorre_latitude~=linhas_latitude &&
numero_cluster_output(percorre_latitude,1)~=numero_cluster_output(percorre_latitude
-1,1)

          media_lat=latitude_output(percorre_latitude,1);

          media_long=longitude_output(percorre_latitude,1);

          inicio_cluster=percorre_latitude+1;

          contador=contador+1;

     else

          soma_lat=latitude_output(percorre_latitude,1);

          soma_long=longitude_output(percorre_latitude,1);

          media_lat(contador,1)=soma_lat/(percorre_latitude-inicio_cluster+1);

          media_long(contador,1)=soma_long/(percorre_latitude-inicio_cluster+1);

          inicio_cluster=percorre_latitude+1;

          contador=contador+1;

     end

 end
```

Figura 77. Trecho 64 do código implementado.

Pretende-se que a fase posterior da pesquisa, o roteamento considerando janelas de tempo, processe 3 *clusters* por vez. Até aqui, garantiu-se que dentro de cada cluster, as rotas formadas fossem as menores possíveis. Entretanto, não garante-se que o mesmo ocorreria caso esses trios de *clusters* tivessem suas rotas rearranjadas, já que

não necessariamente estes *clusters* seriam próximos uns dos outros. Para contornar este problema, rearranja-se a solução, ordenando *clusters* por proximidade. Assim, ao se trabalhar de 3 em 3 *clusters*, acabar-se-ia selecionando *clusters* próximos.

Na prática, o primeiro passo para este procedimento, está compreendido neste trecho do código: realiza-se somatórios de latitudes e longitudes dentro de cada cluster, obtendo-se como resultado as médias destes.

```
analisa_media_lat=media_lat;
analisa_media_long=media_long;
analisa_media_lat(1,1)=100000;
analisa_media_long(1,1)=100000;
ordem_cluster=0;
ordem_cluster(1,1)=1;
cluster_com_que_estou_comparando=1;
posicao_ordem_cluster=2;

 while min(analisa_media_lat)<100000

     distancia_menor=100000000;

  for percorre_latitude=1:linhas_latitude

      numero_do_cluster=numero_cluster_output(percorre_latitude,1);

         if analisa_media_lat(numero_do_cluster,1)~=100000

            distancia=((media_lat(numero_do_cluster,1)-
media_lat(cluster_com_que_estou_comparando,1))^2+(media_long(numero_do_cluster,1)-
media_long(cluster_com_que_estou_comparando,1))^2)^0.5;



         if distancia<distancia_menor
             ordem_cluster(posicao_ordem_cluster,1)=
numero_cluster_output(percorre_latitude,1);
             distancia_menor=distancia;
         end
         end
  end

  for percorre_latitude=1:linhas_latitude
          numero_do_cluster=ordem_cluster(posicao_ordem_cluster,1);
          cluster_com_que_estou_comparando=numero_do_cluster;
          analisa_media_lat(numero_do_cluster,1)=100000;
          analisa_media_long(numero_do_cluster,1)=100000;
  end

  posicao_ordem_cluster=posicao_ordem_cluster+1;

 end
```

Figura 78. Trecho 65 do código implementado.

Em seguida, com as médias de latitude e longitude de cada cluster calculadas, fixa-se o primeiro *cluster* assim como antes e encontra-se qual é o *cluster* mais próximo deste. Repete-se o procedimento com este novo *cluster*. Assim, consegue-se ordenar os *clusters* em ordem de proximidade.

```
[linha_ordem, coluna_ordem]=size(ordem_cluster)

 elemento_mexido=1;

 for percorre_ordem=1:linha_ordem
     buscar_cluster=ordem_cluster(percorre_ordem,1);
     for percorre_matriz=1:linhas
if numero_cluster_output(percorre_matriz,1)==buscar_cluster
numero_cluster_output1(elemento_mexido,1)=numero_cluster_output(percorre_matriz,1);
nome_unidade_output1(elemento_mexido,1)=nome_unidade_output(percorre_matriz,1);
latitude_output1(elemento_mexido,1)=latitude_output(percorre_matriz,1);
longitude_output1(elemento_mexido,1)=longitude_output(percorre_matriz,1);
demanda_output1(elemento_mexido,1)=demanda_output(percorre_matriz,1);
inicio_janela_output1(elemento_mexido,1)=inicio_janela_output(percorre_matriz,1);
fim_janela_output1(elemento_mexido,1)=fim_janela_output(percorre_matriz,1);
tempo_servico_output1(elemento_mexido,1)=tempo_servico_output(percorre_matriz,1);el
emento_mexido=elemento_mexido+1;
         end
     end
 end
```

Figura 79. Trecho 66 do código implementado.

Partindo-se da nova ordenação proposta, monta-se novamente a matriz com os *clusters*, só que agora com *clusters* ordenados por distância. Com isso, chega-se a configurações onde *clusters* mais próximos estejam ordenados próximos.

```
for percorre_matriz1=1:linhas
     elemento_analisado=1;
     for percorre_matriz2=1:linhas
         if
numero_cluster_output1(percorre_matriz1,1)==numero_cluster_output1(percorre_matriz2
,1) && percorre_matriz1~=percorre_matriz2

diferenca_janelas(elemento_analisado,1)=abs(inicio_janela_output1(percorre_matriz1,
1)-inicio_janela_output1(percorre_matriz2,1));
             elemento_analisado=elemento_analisado+1;
         end

         if percorre_matriz2==linhas
         if min(diferenca_janelas)<=diferenca_maxima_de_janelas_permitida
            janela_permitida=janela_permitida+1/linhas;
         end
         end
     end
     diferenca_janelas=0;
 end

 if janela_permitida<1
     solucao_escolhida=solucao_escolhida+1;
     posicao_clusterizacao=ordem_custos(solucao_escolhida,1);
     janela_permitida=0;
 end
end
```

Figura 80. Trecho 67 do código implementado.

Entretanto, não necessariamente garante-se que as soluções encontradas possam ser viáveis em um roteamento considerando a presença de janelas de tempo de atendimento de unidades que devam ser cumpridas. Pode ser que dentro de um mesmo *cluster* existam janelas de tempo muito diferente e que, portanto, exigiriam tempos de espera do barco irrealistas ou tornariam seu atendimento inviável. Por isso, além das dez melhores configurações, também procurou-se ter como saída do programa as dez melhores configurações em que as janelas de tempo dentro de um mesmo *cluster* não sejam tão diferentes. Compara-se todas os tempos em que ocorrem início de janelas de tempo dentro de cada *cluster* e define-se se dentro deste existe pelo menos alguma janela que seja menor do que o parâmetro ***diferenca_maxima_de_janelas_permitida*** definido no início do código pelo usuário. Caso isso não ocorra, soluções cada vez mais piores são selecionadas, até que alguma atenda a esta restrição.

```
 tabela_para_excel_1=[numero_cluster_output];

 tabela_para_excel_2=nome_unidade_output;

 tabela_para_excel_3=[latitude_output, longitude_output, demanda_output,
inicio_janela_output, fim_janela_output,tempo_servico_output];

 acao2 = xlswrite('C:\diretório de saída \dados_output.xlsm', tabela_para_excel_1,
'Plan1', 'A4')

 acao3 = xlswrite(xlswrite('C:\diretório de saída \dados_output.xlsm',
tabela_para_excel_1, 'Plan1', 'B4')

 acao4 = xlswrite(xlswrite('C:\diretório de saída \dados_output.xlsm',
tabela_para_excel_1, 'Plan1','C4')
```

Figura 81. Trecho 68 do código implementado.

Por fim, tanto para as melhores soluções quanto as melhores soluções considerando-se a restrição das janelas, registra-se a solução em uma tabela de Excel. As 10 primeiras planilhas serão preenchidas com as dez melhores solução. Já a 11ª até a 20ª planilha são preenchidas com as 10 melhores soluções, considerando-se a restrição de janelas de tempo.