UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO
GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO MECÂNICA

JOSÉ AURIR GONÇALVES DE ALMEIDA JÚNIOR

DESENVOLVIMENTO DE UM *SOFTWARE* EDUCACIONAL PARA APOIO AO ENSINO DE LOCALIZAÇÃO E ROTEIRIZAÇÃO EM DISCIPLINAS DE LOGÍSTICA.

FORTALEZA
2017

JOSÉ AURIR GONÇALVES DE ALMEIDA JÚNIOR

DESENVOLVIMENTO DE UM *SOFTWARE* EDUCACIONAL PARA APOIO AO ENSINO DE LOCALIZAÇÃO E ROTEIRIZAÇÃO EM DISCIPLINAS DE LOGÍSTICA.

Monografia submetida à Coordenação do curso de Engenharia de Produção Mecânica da Universidade Federal do Ceará como requisito parcial para obtenção do título de Engenheiro de Produção Mecânica.

Orientador: Professor Dr. Heráclito Lopes Jaguaribe Pontes.

FORTALEZA

2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A448d Almeida Junior, José Aurir Gonçalves de.

Desenvolvimento de um software educacional para apoio ao ensino de localização e roteirização em disciplinas de logística / José Aurir Gonçalves de Almeida Junior. – 2017.
83 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia de Produção Mecânica, Fortaleza, 2017.
Orientador: Professor Dr. Heráclito Lopes Jaguaribe Pontes.

1. Ensino. 2. Logística. 3. Roteirização. 4. Localização. 5. Softwares Educacionais. I. Título.

CDD 658.5

JOSÉ AURIR GONÇALVES DE ALMEIDA JÚNIOR

DESENVOLVIMENTO DE UM *SOFTWARE* EDUCACIONAL PARA APOIO AO ENSINO DE LOCALIZAÇÃO E ROTEIRIZAÇÃO EM DISCIPLINAS DE LOGÍSTICA.

Monografia submetida à Coordenação do curso de Engenharia de Produção Mecânica da Universidade Federal do Ceará como requisito parcial para obtenção do título de Engenheiro de Produção Mecânica.

Aprovada em: _____/_____/_____

BANCA EXAMINADORA

_____

PROF. DR. HERÁCLITO LOPES JAGUARIBE PONTES (ORIENTADOR)
UNIVERSIDADE FEDERAL DO CEARÁ - UFC

_____

PROF. DR. ANSELMO RAMALHO PITOMBEIRA NETO
UNIVERSIDADE FEDERAL DO CEARÁ - UFC

_____

PROF. MARCOS RONALDO ALBERTIN
UNIVERSIDADE FEDERAL DO CEARÁ - UFC

AGRADECIMENTOS

Ao meu orientador, Prof. Heráclito Jaguaribe, por compartilhar sua experiência ao longo deste projeto.

A todos os amigos que, de alguma forma, ajudaram em sua conclusão, em especial Lucas Sampaio e Marília Nunes.

Aos meus pais, Aurir e Vilca, eternas fontes de gratidão.

RESUMO

Os processos educacionais passam por constantes alterações para se adaptarem às novas realidades digitais, oriundas da existência de um desenvolvimento tecnológico incessante, causador de profundas transformações sociais. Neste contexto, *softwares* apresentam-se como ferramentas importantes na educação há décadas. Considerando esta importância, a motivação deste trabalho parte da identificação de uma oportunidade de contribuir para o ensino de métodos de localização e roteirização em disciplinas de logística, através de um *software* desenvolvido com foco educacional. Para tanto, este trabalho inicia pela definição de tecnologias adequadas ao seu desenvolvimento, utilizando-as para a elaboração de algoritmos e de uma interface gráfica capazes de proporcionar ao usuário a aplicação de métodos de localização e roteirização, complementando o ensino teórico destes temas. O sistema desenvolvido é então verificado em comparação com outros sistemas, como o LogWare® e o Excel®, permitindo atestar sua competência como *software* educacional, mostrando suas vantagens, limitações e oportunidades de melhoria.

Palavra-chave: Ensino, Logística, Roteirização, Localização, Softwares Educacionais

ABSTRACT

Educational approaches deal with constant changes in themselves to adapt in the new digital realities that emerges from the existence of an unstoppable technologic development, prompter of deep social transformations. In this context, software's show themselves as important tools in education for decades. Considering this importance, the motivation for this works arises from the identification of an opportunity to contribute to the education of location and routing problems in logistics courses, through the development of a software with educational focus. For that, this work starts with the definition of some technologies, adequate to its development, using them to elaborate algorithms and a graphical user interface, able of offer to the end user the application of location and routing methods, complementing the theoretical teaching of these subjects. The developed system is, then, verified through the comparison with some other reference systems, like LogWare® and Excel®, allowing the certification of its competency as an educational software, showing its advantages, limitations and opportunities for improvement.

Keywords: Education, Logistics, Routing Methods, Location Methods, Educational Software's

LISTA DE SIGLAS

API – *Application Programming Interface*

COG – *Center of Gravity*

CSCMP – *Council of Supply Chain Management Professionals*

CSS – *Cascading Style Sheets*

HTML – *HyperText Markup Language*

JSON – *JavaScript Object Notation*

MULTICOG – *Multiple Center of Gravity*

MVC – *Model View Controller*

PMED – P-mediana

SPA – *Single Page Application*

SVG – *Scalable Vector Graphics*

VRP – *Vehicle Routing Problem*

WPF – *Windows Presentation Foundation*

XAML – *Extensible Application Markup Language*

XML – *Extensible Markup Language*

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização

O desenvolvimento tecnológico incessante das últimas décadas vem sendo impulsionador de profundas transformações culturais, econômicas e sociais. Os processos educacionais também sofrem constante alteração para adaptar-se às novas realidades e possibilidades oferecidas pela tecnologia. Segundo Cruz e Neri (2014, p. 9), "[...] docentes de hoje têm em suas aulas alunos com uma profunda percepção de tecnologias, de forma que precisam estar preparados para utilizar tais tecnologias e para se beneficiar do uso das mesmas".

Neste contexto de evolução tecnológica, percebe-se o surgimento de diversos recursos audiovisuais (e.g. projetores, dispositivos móveis, realidade virtual, etc.) que oferecem novas possibilidades de explorar os sentidos do aluno em busca de maior efetividade do aprendizado. Além de uma imersão passiva, estes recursos também possibilitam interação.

> "É hoje consensual que um aluno que presta atenção retém aproximadamente 10% do que lê, 20% do que ouve, 30% do que vê, 50% do que vê e ouve ao mesmo tempo, 80% do que diz e 90% do que diz fazendo qualquer coisa a propósito da qual reflete e na qual se implica pessoalmente" (ROCHA, 1988 p. 176).

Apesar das porcentagens de precisão questionável, a colocação acima ilustra a importância de recursos capazes de explorar os sentidos do aluno, ressaltando a elevada influência que a participação ativa do aluno tem no processo de aprendizagem. Assim, as tecnologias digitais se apresentam, há décadas, como ferramentas capazes de instigar tal participação ativa, dinamizando o processo educacional.

Neste ambiente tecnológico, os computadores são importantes engrenagens do mundo digital. Valente (1999) afirma que, no Brasil, o uso do computador na educação teve seu início já em 1970. Conforme expõe Giacomazzo (2014), "Na década de 80, o Ministério de Educação e Cultura (MEC) assume a liderança do processo de informatização brasileira [...] até chegar à década de 90 com a criação do Programa Nacional de Informática na Educação". Estas transformações continuam, em um contexto moderno onde internet e *tablets* passam a ter papel crescente na dinâmica do ensino. Em 2014, o Governo Federal efetuou a compra de 600 mil *tablets* para uso em escolas públicas.

Além dos computadores, os *softwares* são elementos fundamentais das tecnologias digitais, que passam a ter significava relevância na dinâmica educacional.

> "Pedagogicamente falando, a utilização de ambientes informatizados, empregando-se *softwares* educativos avaliados previamente pelo professor, acompanhados de uma didática construtiva e evolutiva, pode ser uma solução interessante para os diversos problemas de aprendizagem em diferentes níveis" (MAGEDANZ, 2004. p.6)

Dentre os muitos temas de ensino passíveis de apoio educacional por *softwares*, o presente trabalho explora a Logística, que é um tema amplo do conhecimento humano, onde o termo pode ser conceituado de diversas formas. Como bem ressalta Moura (2006, p.16), "Apesar de sua inegável importância, mesmo alguns dos mais conceituados dicionários de língua portuguesa são pouco esclarecedores sobre o significado do vocábulo Logística".

Conforme define o *Council of Supply Chain Management Professionals* (CSCMP, 2013), Logística é o processo de planejar, implementar e controlar procedimentos para transporte e armazenagem efetivos de bens, serviços e informações relacionadas, de um ponto de origem a outro de consumo, de acordo com as requisições do cliente.

Para atingir seus objetivos de transporte e armazenagem, a Logística procura resolver, dentre outras, problemáticas de roteirização e localização. Enquanto roteirizar é o processo de determinar como uma carga irá se mover entre origem e destino, Ballou (2009) afirma que as decisões sobre localização envolvem a determinação do número, local e proporções das instalações a serem usadas.

No âmbito educacional, Georges e Seydell (2008) ressaltam que a Logística se trata de uma disciplina altamente dinâmica, de forte cunho tecnológico, baseada em sofisticados métodos matemáticos, que vem se modificando e ganhando importância ao longo dos anos.

A respeito do ensino dessa disciplina, em um contexto onde *softwares* apresentam-se como ferramentas importantes na educação, identificou-se uma deficiência na oferta de *softwares*, com foco educacional e gratuitos, que tratem de estratégias de roteirização e localização.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem por objetivo o desenvolvimento de um *software* educacional para apoiar o ensino de roteirização e localização em disciplinas de logística, disponibilizando um recurso didático para o aprendizado.

1.2.2 Objetivos específicos

1. Determinar quais práticas e tecnologias de desenvolvimento de *software* são as mais adequadas para o desenvolvimento deste trabalho, considerando o objetivo educacional.
2. Implementar algoritmos de roteirização e localização capazes de resolver problemáticas comumente tratadas no ensino destas temáticas.
3. Desenvolver uma interface gráfica para o *software*, utilizando as tecnologias e diretrizes definidas.
4. Verificar os algoritmos implementados através da aplicação dos métodos de localização e roteirização na resolução de problemas com soluções previamente conhecidas.

1.3 Justificativa

Segundo Bowersox (2013), o recente crescimento do comércio global expandiu o tamanho e a complexidade das operações logísticas, onde as empresas que têm uma competência logística de alto nível conquistam vantagens competitivas. Essa crescente importância do tema no mercado, acaba se refletindo no universo acadêmico.

> "Aliada ao fato de ter atingido projeção nos últimos anos, a Logística tornou-se estratégica para as empresas, além de promissora e fortemente requisitada pelo mercado, o que gerou maior necessidade do oferecimento deste curso nas universidades. Conciliar teoria e prática no ensino desta disciplina, de modo a transmitir e assimilar seu estado da arte, tornou-se uma tarefa desafiadora para professores e alunos" (GEORGES e SEYDELL, 2008, p. 02).

Mesmo considerando a importância do tema, a oferta de *softwares* gratuitos e com focos educacionais no ambiente acadêmico brasileiro é limitada. Como destaca Georges e Seydell (2008), *softwares* como *Microsoft Excel* ® e outros mais específicos, como roteirizadores, são capazes de resolver uma gama relativamente importante de problemas de natureza Logística. Contudo, além de serem dispendiosos, muitas vezes são oferecidos em versões acadêmicas limitadas. Em relação a estas versões, "infelizmente, ainda não é possível resolver problemas mais complexos com tais ferramentas" (GEORGES e SEYDELL, 2008, p. 06). O que se encontra são ferramentas que acompanham livros-texto, como o LogWare®.

> "O *software* LogWare é uma coletânea de algoritmos da pesquisa operacional para resolver problemas logísticos, mais especializado que o Excel, sem restrições de uso como as versões de avaliação/acadêmicas dos sistemas comerciais. Este *software*, integrante da 4ª edição de Ballou (2001), é capaz de resolver problemas de distribuição, roteirização, localização de centros de distribuição, layouts, estoque, entre outros. Por não ser integrado com um SIG, as coordenadas de localidades como pontos de parada, centros de distribuição etc., têm que ser inseridas manualmente. Por outro lado, pode ser proveitoso ao aluno trabalhar com plantas e mapas" (GEORGES e SEYDELL, 2008, p. 07).

Apesar de se apresentar como uma boa alternativa ao ensino, este *software*, que teve sua primeira versão lançada em 1992, está ficando defasado, não acompanhando o ritmo acelerado de evolução das tecnologias digitais. Interface antiga e incompatibilidade com sistemas operacionais modernos são exemplos de problemas que levam o *software* a ter baixa qualidade. Como destacado por Giraffa (2009), a qualidade do *software* influencia seus objetivos educacionais, onde "muitos programas possuem conteúdo mal formulado, problemas na execução do sistema, interfaces (telas) confusas e assim por diante".

No contexto desta carência de oferta, o presente trabalho se justifica pela oportunidade de desenvolver um *software* para auxiliar o ensino de logística, utilizando-se de tecnologias e padrões modernos de desenvolvimento de software que levem à criação de um *software* de qualidade.

1.4 Metodologia do trabalho

Para que o produto final deste trabalho seja elaborado, pesquisas prévias precisam ser realizadas para embasar as tomadas de decisão. Conforme define Cassarro (1995, p. 45), "uma decisão nada mais é do que uma escolha entre alternativas, obedecendo a critérios previamente estabelecidos".

Quando se trata do desenvolvimento de um *software*, várias decisões precisam ser tomadas. Em relação a interface com o usuário, uma mesma informação pode ser inserida ou apresentada de diversas formas. Assim, se faz necessário a definição de diretrizes que norteiem este desenvolvimento, considerando as boas e modernas práticas que maximizem a ergonomia da interface gráfica e o potencial educativo do sistema. Para essa escolha, será realizada uma pesquisa bibliográfica sobre qualidade de *software*. Segundo Gil (2007, p.45), a principal vantagem da pesquisa bibliográfica reside no fato de permitir ao investigador a cobertura de uma gama de fenômenos muito mais ampla do que aquela que poderia pesquisar diretamente.

Além disso, é preciso definir que tecnologias serão utilizadas, já que todo *software* é desenvolvido utilizando-se uma gama de tecnologias pré-existentes, e estas precisam ser escolhidas em coerência com os objetivos do projeto. Existe uma profusão de opções. A biblioteca virtual Wikipédia mantém uma lista de referência das principais linguagens básicas de programação, que já passa de 250 itens. Cada linguagem oferece outra profusão de opções de *frameworks* e bibliotecas que, combinadas, podem produzir soluções semelhantes para um mesmo problema. Assim, é fácil perceber que existem milhares de tecnologias que podem ser utilizadas para desenvolver um *software*.

Optou-se por realizar uma pesquisa das tecnologias mais utilizadas atualmente, comparando-as com os critérios que mais atendem aos objetivos deste trabalho, como o critério de atualidade. Não se pretende desenvolver um sistema em uma tecnologia ou plataforma obsoleta, ou que dê sinais de utilização decrescente, de forma a tornar o *software* pouco utilizável em um breve intervalo de tempo.

Segundo Gil (2002) e Collis e Hussey (2005), pesquisas exploratórias envolvem, dentre outros, levantamento documental e entrevistas com especialistas, visando proporcionar uma visão geral de um determinado fato, que geralmente tem pouco ou nenhum estudo anterior a seu respeito. Essa pesquisa se faz necessária, pois tecnologias de desenvolvimento de *software* surgem e ficam obsoletas muito rapidamente, exigindo uma análise ampla da utilização destas tecnologias por empresas e especialistas, além de uma análise das tendências de mercado. Busca-se, dessa forma, escolher, entre as tecnologias modernas disponíveis, aquela de maior adequação a este trabalho.

Tomadas as decisões, as diretrizes e tecnologias definidas anteriormente serão utilizadas no desenvolvimento do sistema, que terá os pontos mais relevantes detalhados neste trabalho. Serão implementadas funcionalidades capazes de resolver problemáticas comumente abordadas no ensino de roteirização e localização.

A verificação do sistema será feita através da resolução de alguns problemas selecionados de livros-texto, comparando os resultados obtidos pelo sistema desenvolvido com outras fontes. Para verificar a interface gráfica desenvolvida, será feita uma comparação da forma de inserção do problema, e apresentação dos resultados, entre o *software* desenvolvido LogWare®, tomado como referência. Essa comparação será feita segundo os critérios de qualidade de *software* adotados no desenvolvimento.

1.5 Limitações do trabalho

Como exposto anteriormente, o escopo deste trabalho se restringe ao desenvolvimento de um *software* que aborda problemáticas comumente estudadas no ensino de roteirização e localização em disciplinas de logística. Dentro deste escopo, existem diversos métodos, estratégias e problemáticas abordadas por estas temáticas. Roteirização e localização são temas amplos, e o presente trabalho não tem a ousadia de esgotar todos esses assuntos. Se faz necessário escolher, dentre várias alternativas, quais conteúdos teóricos serão abordados pelo programa, em detrimento dos outros.

A respeito das tecnologias escolhidas, o presente trabalho corre o risco de realizar escolhas inadequadas. Mesmo buscando-se aumentar a acessibilidade do sistema, através de tecnologias modernas e difundidas, ainda se corre o risco de criar um sistema que irá, em poucos anos, se tornar obsoleto ou de difícil manutenção.

> "A velocidade das mudanças tecnológicas, no ambiente informático, reduz o tempo para a necessária reação. O ciclo de renovação é curto – de três a cinco anos –, ao contrário de décadas ou mesmo séculos associados à preservação de objetos físicos. A obsolescência tecnológica é geralmente vista como a principal ameaça técnica para garantir o acesso continuado ao objeto digital" (THOMAZ, 2008, p. 39).

Por fim, o sistema desenvolvido sempre oferecerá várias oportunidades de melhoria, já que permite a criação de novas funcionalidades e melhoria daquelas já implementadas. Esse desenvolvimento, que aumenta a utilidade e confiabilidade do sistema, exige investimentos que, neste trabalho, são limitados aos recursos do autor.

1.6 Estrutura do trabalho

Este trabalho é dividido em 4 capítulos, detalhados a seguir.

O primeiro capítulo contextualiza a problemática do trabalho e justifica os objetivos a serem alcançados, explorando uma visão geral. Além destes objetivos, gerais e específicos, este capítulo apresenta a metodologia a ser utilizada no decorrer do trabalho.

O segundo capítulo concentra o referencial teórico das técnicas utilizadas no capítulo seguinte. São apresentados conceitos relacionados à qualidade e ergonomia de *software*, bem como tecnologias modernas de desenvolvimento de aplicações. Os temas relacionados ao ensino de logística, que serão abordados por este projeto, também são revisados. Os algoritmos necessários ao funcionamento do sistema são apresentados. Também são apresentadas ideias que norteiam o teor educacional do projeto.

O terceiro capítulo aborda todo o desenvolvimento do sistema e os resultados obtidos, em alinhamento com os objetivos iniciais. São realizadas verificações para constatar as funcionalidades do sistema, através da resolução de problemas com solução conhecida, obtidos de livros e outros *softwares*, e a comparação dos resultados. Além disso, é feita uma comparação com um *software* de referência, para avaliar a interface com o usuário.

O quarto e último capítulo apresenta considerações finais sobre o projeto e discute o potencial de expansão do sistema, através de recomendações para trabalhos futuros.

2. REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma revisão dos principais conceitos utilizados na elaboração deste trabalho. Inicialmente, são apresentados os conceitos logísticos de roteirização e localização, bem como os métodos e algoritmos que serão utilizados. Por fim, explora-se conceitos relativos às tecnologias de desenvolvimento de *software*, sua origem, definição e importância, seguido de ideias sobre qualidade de *software*, com foco no teor educacional deste trabalho.

2.1 Logística

2.1.1 Definição e objetivos

Para Ballou (2006), a logística envolve todas as atividades de movimentação e armazenagem, que facilitam o fluxo de produtos, desde o ponto de aquisição da matéria-prima até o ponto de consumo final, assim como dos fluxos de informações que colocam os produtos em movimento, com o propósito de providenciar níveis de serviço adequados aos clientes a um custo razoável. Segundo Cavanha Filho (2001), em diversas outras definições e significados, a logística leva a um conjunto de terminologias para designar as áreas onde se desenvolve, tais como: transportes, distribuição física, suprimentos, administração de materiais e operações.

De acordo com Rodríguez *et al.* (2008), a logística não é mais percebida somente como transportes de mercadorias, que possui uma visão puramente operacional, mas tornou-se uma área estrategicamente essencial para o sucesso das empresas. A logística passou a ser importante na criação de valor de tempo e lugar, como afirma Ballou (2006, p. 33) “produtos e serviços não têm valor a menos que estejam em poder dos clientes quando (tempo) e onde (lugar) eles pretenderem consumi-los”. Gasnier (2002) acrescenta que logística é:

> “[...] o processo de planejar, executar e controlar o fluxo e armazenagem de forma eficaz e eficiente em termos de tempo, qualidade e custos, de matérias primas, materiais em elaboração, produtos acabados e serviços, bem como as informações correlatas, desde o ponto de origem até o ponto de consumo (cadeia de suprimentos), com o propósito de assegurar o atendimento das exigências de todos os envolvidos, isto é, clientes, fornecedores, acionistas, governo, sociedade e meio ambiente” (GASNIER 2002, p. 17).

Conforme Gasnier (2002), a gestão da logística deve considerar as dimensões tempo, qualidade e custos, e define os clientes finais como todas as partes interessadas nos

resultados. De acordo com Bowersox e Closs (2001), o objetivo da logística é fornecer produtos ou serviços no local e momento esperados pelos clientes, e ressaltam que a implementação das melhores práticas logísticas é um dos grandes desafios das organizações na concorrência global. Já em termos de projeto e gerenciamento de sistemas logísticos, cada empresa deve atingir simultaneamente pelo menos seis objetivos diferentes que incluem:

- **Resposta rápida:** atendimento breve e cumprimento de prazos pré-estabelecidos;
- **Variância mínima:** cultura do produto/serviço padronizado ou sem variações;
- **Estoque mínimo:** uso de estoques apenas em situações emergenciais;
- **Consolidação da movimentação:** aperfeiçoar os processos e torná-los sólidos e competitivos;
- **Qualidade:** preocupação se o produto/serviço atende os parâmetros exigidos e encomendados pelo cliente;
- **Apoio ao ciclo de vida:** estender o ciclo de vida do produto/serviço

2.1.2 Roteirização

Para Ballou (2009), a roteirização é o processo logístico que tem como objetivo a melhoria nos trajetos que um veículo deve percorrer, geralmente a fim de minimizar o tempo ou a distância. Afirma ser um dos meios fundamentais para se reduzir os custos e proporcionar melhorias na prestação dos serviços, de forma a reduzir o tempo de transporte e cumprir as metas impostas no processo. Assim, a melhor utilização da frota reflete na necessidade de um menor número de veículos e em menores custos operacionais.

Para Novaes (2007), a problemática de roteirização é mais complexa, e pode ser definida em três fatores fundamentais: decisões, que dizem respeito à programação e ao planejamento de clientes a serem visitados, motoristas e veículos a serem utilizados e sequência das entregas a cada cliente; objetivos, que procuram elevar o nível do serviço, mas com custos baixos; e restrições, que devem ser estabelecidas, como os horários, a jornada de trabalho, o tamanho dos veículos nas vias públicas, dentre outros.

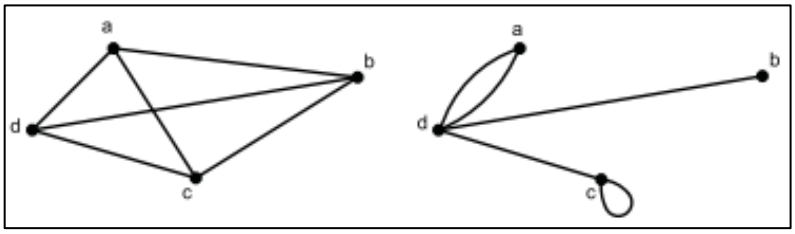
Segundo Laporte *et al.* (2002), a importância da roteirização para a redução de custos e melhoria do nível de serviço na operação logística é muito relevante, pois, através do planejamento, evitam-se os desperdícios, reduzem-se os custos, aperfeiçoa-se o desempenho operacional e, consequentemente, melhora-se o nível de serviço.

2.1.2.1 Grafos

Segundo Tenenbaum *et al.* (1995), um grafo consiste de num conjunto de nós (ou vértices) e de um conjunto de arcos (ou arestas). Cada arco em um grafo é determinado por um par de nós. Um valor, como um custo por exemplo, pode ser associado a um arco. Neste caso, é chamado de grafo ponderado, ou rede, e o valor associado de peso. A Figura 1 ilustra um grafo, seus vértices e arestas.

Segundo Rocha (2005, p.1) "grafo é uma estrutura matemática usada para representar as relações entre as coisas". O autor cita, como exemplo, a representação de cidades que se interligam por estradas – as cidades são nós ou vértices, enquanto as estradas são as arestas ou arcos. Os grafos podem ser utilizados em diversas representações, como distribuição de água pluvial, relacionamentos em redes sociais, estrutura do DNA, desenho da hierarquia de uma empresa, etc.

Figura 1 - Exemplos de grafos



Fonte: Costa (2011)

Para Boaventura (2006), dentre as subestruturas de grafo que oferecem soluções para problemas aplicados, os caminhos se destacam especialmente pelo potencial associado aos problemas de trânsito, transporte e localização em sistemas discretos. Dado um certo grafo, um caminho deste grafo consiste de uma sequência finita alternada de vértices e arestas, começando e terminando por vértices, tal que cada aresta é incidente ao vértice que a precede e ao que a sucede e não há repetição de vértices.

2.1.2.2 Métodos de roteirização

De acordo com Novaes (2007), os métodos de roteirização são utilizados para efetuar o planejamento das rotas de maneira eficiente, para que sua execução possa ser realizada da melhor forma possível. De acordo com Ballou (2006), utilizando os métodos de roteirização para o planejamento das rotas, é possível fazer melhor uso dos recursos

existentes, fazer entregas inteligentes, ter maior controle das rotas ao reduzir a sobreposição de entrega e possibilitando a criação de territórios e rotas rentáveis.

Segundo Cunha (1997), sistemas de roteirização e programação de veículos ou, simplesmente, roteirizadores, são sistemas computacionais que, através de algoritmos, muitas vezes heurísticos, e uma apropriada base de dados, são capazes de obter soluções para problemas de roteirização e programação de veículos com resultados relativamente satisfatórios, consumindo tempo e esforço de processamento relativamente pequenos quando comparados aos gastos nos tradicionais métodos manuais.

> "Embora sejam muitas as variações dos problemas de roteirização, é possível reduzi-los a alguns modelos básicos. Existe o problema de encontrar uma rota ao longo de uma rede em que o ponto de origem seja diferente do ponto de destino. Há um problema similar sempre que se apresentam múltiplos pontos de origem e de destino. Mais complexo ainda é o problema de fazer itinerários quando os pontos de origem e destino são os mesmos" (BALLOU, 2006, p. 191).

De acordo com Von Atzingen (2012), o problema de caminho mínimo (PCM) consiste em determinar um caminho entre dois vértices tal que a somatória dos custos unitários dos arcos (ou alguma outra medida de impedância) que compõem este caminho seja o mínimo. Von Atzingen (2012) ressalta que o PCM vem sendo estudado há mais de 40 anos em diversas áreas de conhecimento como ciência da computação, matemática e engenharia de transportes e que, devido à dificuldade computacional em tratar este problema, a maioria das pesquisas nesta área se concentra no desenvolvimento de algoritmos eficientes para resolver o PCM.

De acordo com Cormen *et al.* (2002), o algoritmo de Dijkstra resolve o problema de caminho mínimo de uma única origem em um grafo orientado. Sedgewick (2002) relata que o algoritmo de Dijkstra foi criado pelo cientista holandês Edsger Dijkstra Wybe, em 1956, para solucionar o problema do caminho mínimo entre dois vértices de um grafo. Para Arenales *et al.* (2007), o algoritmo de Dijkstra:

> "[...] encontra o menor caminho entre quaisquer dois nós da rede, quando todos os arcos têm comprimento não-negativo. Nele é utilizado um procedimento iterativo, determinando, na iteração 1, o nó mais próximo do nó 1, na segunda iteração, o segundo nó mais próximo do nó 1, e assim sucessivamente, até que em alguma iteração o nó destino seja atingido" (ARENALES et al. 2007, p. 6).

O algoritmo pode ser utilizado em grafos orientados ou não orientados. Também é possível viabilizar sua execução com pesos em suas arestas, determinando "graus de dificuldade" em se transpor uma delas. Sedgewick (2002) ressalta que os valores das arestas devem ser não negativos, ou seja, os custos podem ser somente nulos ou positivos. O

resultado proporcionado pelo algoritmo com valores negativos pode não ser exato. A Figura 2 apresenta as etapas de execução deste algoritmo segundo Taha (2008).

Figura 2 - Algoritmo de Dijkstra

Seja u_i a distância mais curta do nó de origem 1 ao nó i, e defina-se d_{ij} ($>=0$) como o comprimento do arco (i,j). Então, o algoritmo define o rótulo para um nó imediatamente posterior, j, como:

$$[u_j, i]=[u_i+d_{ij}, i], d_{ij}>=0$$

O rótulo para o nó inicial é $[0,-]$, o que indica que o nó não tem nenhum predecessor.

Os rótulos dos nós no Algoritmo de Dijkstra são de dois tipos: *temporários e permanentes*. Um rótulo temporário é modificado se for possível encontrar uma rota mais curta até um nó. Se não for possível encontrar uma rota mais curta até um nó. Se não for possível encontrar nenhuma rota melhor, o status do rótulo temporário muda para permanente.

Etapa 0: Rotule o nó de origem com o rótulo permanentes $[0, -]$. Determine $i=1$.

Etapa i:

(a) Calculo os rótulos *temporários* $[u_i+d_{ij}, i]$ para cada nó j que puder ser alcançado partido do nó i, contando que j não seja permanentemente rotulado. Se o nó j já estiver rotulado com $[u_j, k]$ passando por outro nó k, e se $u_i+d_{ij}<u_j$, substitua $[u_j, k]$ por $[u_i+d_{ij}, i]$.

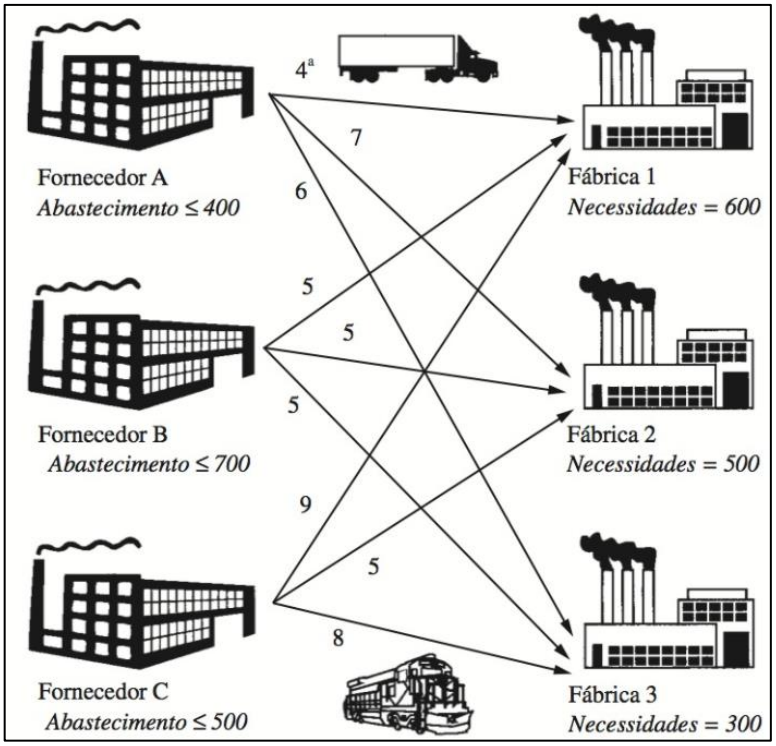
(b) Se todos os nós estiverem rótulos permanentes, pare. Caso contrário, selecione o rótulo $[u_r, s]$, cuja distância $(=u_r)$ é a mais curta entre todos os rótulos temporários (empates são resolvidos arbitrariamente). Determine $i=r$ e repita a etapa i.

Fonte: Taha (2008)

Além de problemas de roteirização com um ponto de origem e outro de destino, Ballou (2006) destaca também os problemas com pontos de origem e destino múltiplos, que normalmente ocorrem quando há mais de um vendedor, fabrica ou armazém para servir a mais de um cliente com o mesmo produto. Ressalta ainda que um tipo especial do algoritmo de programação linear, o método do transporte, é frequentemente aplicado a este tipo de problema.

Diferentemente do problema de caminho mínimo, Taha (2008) explica que o problema de transporte é uma classe especial de problemas de programação linear que trata do envio de uma mercadoria de origens para destinos, onde o objetivo é a determinação de uma programação de expedição que minimize o custo total e, ao mesmo tempo, satisfaça os limites de fornecimento e demanda. A Figura 3 ilustra um problema de origem e destino múltiplos.

Figura 3 – Ilustração do problema de transporte



Fonte: Ballou (2006, p. 196)

De acordo com Caixeta Filho (1995), o modelo tradicional de transporte para um único produto, apresentado através das Equação 1 pode ser resolvido através de técnicas de programação linear, tornando-se normalmente a forma da minimização da função-objetivo Z (representando os custos totais envolvidos), sujeita às comportamentais de oferta e demanda.

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} * T_{ij} \quad (1)$$

$$sujeito\ a: \sum_{j=1}^{n} T_{ij} \leq O_i$$

$$\sum_{i=1}^{m} T_{ij} \geq D_{ij}$$

$$\sum_{i=1}^{m} S_{ij} = \sum_{j=1}^{n} D_i$$

Onde:

T_{ij} = quantidade de determinado produto transportado de *i* para *j*;
T_{ij} = custo de transporte do produto referente a movimentação entre *i* para *j*;
O_{ij} = oferta pelo produto na região *i*;
D_{ij} = demanda pelo produto na região *j*.

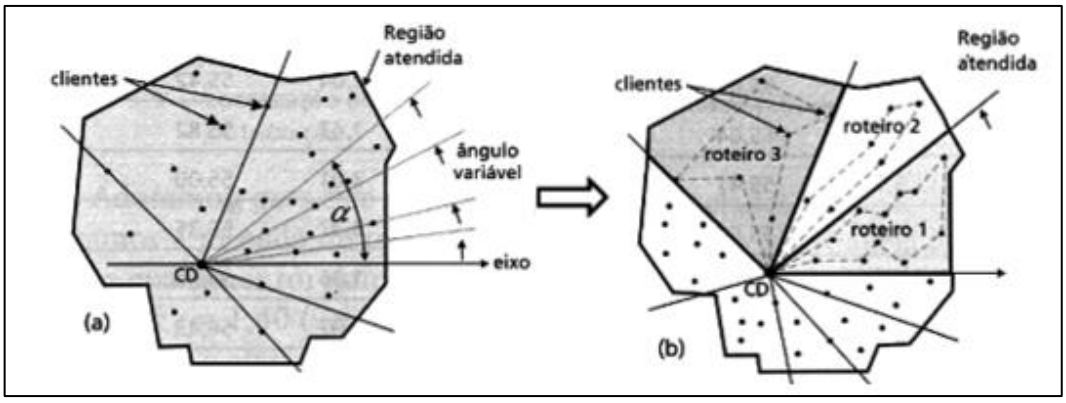
Ballou (2016, p.196) destaca que “o operador logístico frequentemente depara-se com situações de roteirização em que o ponto de origem e o de destino são os mesmos”, e que o problema é conhecido como "problema do caixeiro viajante", sendo um problema de difícil

solução que exige muito processamento computacional quando há muitos pontos envolvidos. Procedimentos de solução cognitivos, heurísticos ou combinações de heurísticos-otimizadores, tem representado boas alternativas. Para Gomes (2008), o Problema do Caixeiro Viajante (PCV) pode ser entendido como o problema de um vendedor que deseja visitar um conjunto de cidades, passando exatamente uma vez por cada uma, voltando ao ponto de partida no final do seu percurso.

Para Cunha (2000), problemas de roteirização de veículos são muitas vezes definidos como problemas de um ou mais caixeiros viajantes que incluem restrições adicionais de capacidade, além de outras que dependem de cada aplicação. Ballou (2006) destaca que a roteirização e programação de veículos (RPV) é uma extensão do problema do caixeiro viajante, onde restrições realistas são incluídas ao problema, como limitações de volume dos veículos, distâncias máximas, janelas de tempo, dentre outras. Examina ainda dois métodos dentre as inúmeras abordagens já́ sugeridas para enfrentar problemas dessa complexidade. Um deles é simples (o método da "varredura"), e o outro, mais complexo, enfrentando elementos mais práticos e produzindo soluções de maior qualidade sob uma gama mais ampla de circunstancias (o método das "economias").

Souza (2016) expõe que o método de varredura, também conhecido como *sweep algorithm*, é considerado uma técnica heurística originada nos estudos de Wren e Holliday (1972) e Gillet e Miller (1974) para resolução de rotas e programação de veículos. A Figura 4 ilustra o método.

Figura 4 – Método de varredura



Fonte: Novaes (2007)

O método da varredura é resumido, por Ballou (2006), em alguns passos. Primeiro, localiza-se todas as paradas num mapa ou grade. Em seguida, traça-se uma linha reta a partir da origem em qualquer direção, girando-a, em sentido horário ou anti-horário, até́ ela fazer a intersecção com uma parada. Nesse momento, se a parada for incluída no roteiro e

a capacidade do veículo não for ultrapassada, continua-se com a rotação da linha até a interseção da próxima parada. Se o volume cumulativo ultrapassar a capacidade do veículo, exclui-se o último ponto e define-se o roteiro (veículo volta à origem). Continuando com a varredura da linha, começa-se um novo roteiro com o último ponto excluído da rota anterior. A varredura continua até se atribuir todos os pontos a roteiros.

O segundo método abordado por Ballou (2006, p. 205) para RPV é o método de Clark-Wright, baseado na abordagem das economias, que "tem atravessado os anos como algo dotado da flexibilidade suficiente para resolver uma ampla coleção de restrições práticas, capaz de gerar soluções que são quase ótimas". Heinen e Osório (2006) afirmam que este foi um dos primeiros algoritmos a serem propostos para solucionar o problema de roteamento de veículos, servindo de base para outros algoritmos mais sofisticados que surgiram mais tarde. Destacam ainda o funcionamento do algoritmo, que ocorre em alguns passos, iniciando-se com a representação em grafo dos pontos que representam os clientes e o depósito de origem.

Em seguida, são criadas $(\mathrm{n}-1)$ rotas iniciais, cada uma delas ligando um dos clientes ao depósito, onde (n) é o número de vértices do grafo. Após isto, são calculadas as economias para todas as possíveis combinações 2 a 2 de vértices, através da fórmula: $S_{ij} = C_{i0} + C_{0j} - C_{ij}$, onde S_{ij} representa a economia obtida ao se unir as rotas que possuem os vértices i e j, e C_{ij} representa o custo de se deslocar do vértice i para o vértice j. As economias são colocadas em uma tabela, que é ordenada de forma decrescente. Para cada linha da tabela de economias, começando pela maior, é verificado se é possível unir as rotas que contém os vértices i e j. As rotas podem ser combinadas se ambos os vértices estiverem em extremos opostos de rotas diferentes, e se a união não violar nenhuma das restrições do modelo (capacidade de carga dos veículos, limite de combustível, janela de tempo, etc). O processo continua com o próximo par de vértices da tabela de economias, até que se chegue ao último item da tabela ou até que não existam mais rotas a serem combinadas.

2.1.3 Localização de instalações

De acordo com Sellito *et al.* (2015) e Pinedo e Abreu (2011), os primeiros estudos sobre problemas de localização surgiram no século XX. O primeiro modelo de localização industrial foi proposto por Alfred Weber, que considerou planos cartesianos na modelagem do problema e este modelo dominou por muito***s*** anos na literatura.

De acordo com Ballou (2006), as decisões sobre localização envolvem a determinação do número, local e proporções das instalações a serem usadas. Essas instalações incluem pontos nodais da rede, como fábricas, portos, armazéns, pontos de varejo e pontos centrais de serviços na rede da cadeia de suprimentos em que os produtos param temporariamente a caminho dos consumidores finais.

Para Ferreira *et al*. (2013), a escolha por uma localização para as instalações pode envolver variados fatores, como custos de transporte; impostos e incentivos; potencial de mercado; qualidade de vida; acesso à infraestrutura de transporte; infraestrutura local de serviços; custo de espaço e disponibilidade para expansão.

2.1.3.1 Métodos de localização de instalações

Existem vários modelos que teorizam a problemática da localização industrial. De acordo com Fitzsimmons (2004), apesar de a seleção do local ser influenciada por fatores qualitativos e oportunos, um estudo baseado em métodos quantitativos, tais como custos e distância, pode dar uma direção mais assertiva e útil que diminua a probabilidade de erros.

De acordo com Ballou (2006), os problemas de localização podem ser classificados segundo alguns critérios, como o de força direcionadora, segundo o qual a localização de instalações é determinada por um fator fundamental, geralmente de forte teor econômico. Na indústria, pode ser a proximidade aos recursos de produção, enquanto no varejo, pode ser a proximidade aos clientes, por exemplo. Para o cálculo do custo total de transporte (CT), utiliza-se a Equação 2:

$$CT = \sum Vi\ Ri\ di \tag{2}$$

Onde:

Vi = volume de carga transportada;
Ri = custo unitário de transporte;
di = distância percorrida.

Para calcular a distância entre os dois pontos, como pontos de demanda e uma origem, utiliza-se a Equação 6:

$$di = \sqrt{(Xi - \bar{X})^2 + (Yi - \bar{Y})^2} \tag{3}$$

Onde:

Xi, Yi = coordenadas dos pontos de fonte ou demanda;
$\bar{X}, \bar{Y}$ = coordenadas de origem;

De acordo com Slack *et al.* (2002), o método mais utilizado para localização de uma planta única, terminal, armazém ou prestador de serviço, é o chamado centro de gravidade exato, método de grade ou método centroide. O modelo é classificado matematicamente como um modelo estático de localização continua. Para Bowersox e Closs (2001), o método centro de gravidade é uma alternativa para buscar a melhor localização geográfica. Este centro pode ser relacionado a vários tipos de taxas, como pesos, volumes e distâncias, para selecionar a alternativa de menor custo.

De acordo com Ballou (2006, p. 247), o processo para encontrar a localização ideal caracteriza-se por sete etapas. A primeira consiste em determinar as coordenadas (X_i, Y_i) para cada ponto de fonte e de demanda, bem como os volumes e os custos unitários de transporte. Na segunda etapa, deve-se calcular a localização do baricentro local $(\bar{X}, \bar{Y})$ através das Equações 4 e 5.

$$\bar{X} = \frac{\sum_i Vi\ Ri\ Xi}{\sum_i Vi\ Ri} \tag{4}$$

$$\bar{Y} = \frac{\sum_i Vi\ Ri\ Yi}{\sum_i Vi\ Ri} \tag{5}$$

Para a terceira etapa, deve-se calcular a distância d_i através da Equação 3, a partir do $\bar{X}$ e $\bar{Y}$ encontrados nas equações 4 e 5. A quarta etapa inicia-se ao substituir a distância d_i nas equações 6 e 7, e recalculá-las para as coordenadas $\bar{X}$ e $\bar{Y}$ revisadas.

$$\bar{X} = \frac{\sum_i Vi\ Ri\ Xi\ /\ d_i}{\sum_i Vi\ Ri\ /\ d_i} \tag{6}$$

$$\bar{Y} = \frac{\sum_i Vi\ Ri\ Yi\ /\ d_i}{\sum_i Vi\ Ri\ /\ d_i} \tag{7}$$

Para a quinta etapa, deve-se recalcular d_i baseado nas coordenadas $(\bar{X}, \bar{Y})$ revisadas. A sexta etapa consiste em repetir a quarta e quinta etapas até que as coordenadas $(\bar{X}, \bar{Y})$ não mudem por sucessivas interações, considerando um certo nível de precisão pretendido. O método é finalizado no sétimo passo, ao se calcular o custo total para a melhor localização, utilizando a Equação 2.

De acordo com Ballou (2006), o número de instalações é um critério para classificação dos métodos de localização, caracterizando-se em decidir a localização de somente um ponto ótimo ou de selecionar múltiplos locais dependendo de certas variáveis. Segundo Wanke (2003), localizar apenas uma ou várias instalações são problemas

consideravelmente diferentes. Na localização única, evita-se a necessidade de considerar fatores como os custos fixos de operações.

Dentre possíveis métodos exatos de localização múltipla, Ballou (2006) destaca o método do múltiplo centro de gravidade e o método *p*-mediana. O método do múltiplo centro de gravidade utiliza o método centroide, que calcula o centro e gravidade exato, em conglomerados de vértices, definidos segundo algum critério.

> "Essas atribuições das instalações podem ser feitas de várias formas, especialmente quando se trata de um problema abrangendo instalações múltiplas e um alto número de pontos de origem e de destino. Uma maneira de abordar tal problema é configurar os conglomerados mediante a concentração dos pontos mais próximos entre si. Depois de se encontrar as localizações de centro de gravidade, os pontos são reatribuídos à estas localizações. Novas localizações de centro de gravidade são encontradas para os conglomerados revisados. O processo continua até que não se encontre mais mudança alguma. Isso completa as computações para um número específico de instalações a ser localizado. E pode ser repetido com diferentes números de instalações" (BALLOU, 2006, p. 442)

Já em relação ao método *p*-mediana, Lorena *et al*. (2001) afirma ser um problema clássico de localização e consiste em localizar *p* centros (medianas) em uma rede de modo a minimizar a soma das distâncias de cada vértice ao centro mais próximo. Para Clemente (1998), o método consiste em escolher um subconjunto de *p* localizações dentre os nós da rede, de forma a atender a todas as demandas de todos os clientes ao custo total mínimo. O modelo p-mediana foi inicialmente formulado por Christofides (1975), conforme Equação 8.

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} * w_j * c * k_{ij} \qquad (8)$$

$$sujeito\ a: \ \sum_{j=1}^{n} k_{ij} = 1 \qquad \forall\ i = 1 \ldots m$$

$$\sum_{i=1}^{m} k_{ii} = p$$

$$k_{ij} \leq k_{ii} \qquad \forall\ i = 1 \ldots m \qquad \forall\ j = 1 \ldots n$$

$$k_{ij} \in \{0,1\}$$

Onde:

n = número de nós;
m = quantidade de possíveis localizações dentro do sistema logístico;
p = número de fábricas ou centros de distribuição a serem instalados;
d_{ij} = distância entre os nós i e j;
w_j = demanda do nó j;
c = custo unitário de transporte por quilômetro;
k_{ij}= a variável de alocação, que será 1 se o nó j é alocado no nó i e 0 caso contrário. Em termos práticos, os nós que forem alocados com o valor 1 para essa variável serão os nós onde serão instaladas as unidades fabris.

2.2 Desenvolvimento de s*oftware*

2.2.1 Tecnologias de desenvolvimento

Os *softwares* são, de acordo com a Lei 9.609/98 (Lei do *Software*), no seu art. 1º:

> "[...] a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados" (BRASIL, LEI Nº 9.609, 1998, art 1º).

Para Pressman e Maxin (2016), *software* abrange programas executáveis em computador, conteúdos (apresentados à medida que os programas são executados), informações descritivas, tanto na forma impressa quanto na virtual, abrangendo praticamente qualquer mídia eletrônica. Segundo Fernandes (2002), é uma sentença escrita em uma linguagem computável, para a qual existe uma máquina (computável) capaz de interpretá-la. Ao interpretar o *software*, a máquina computável é direcionada à realização de tarefas específicas.

De acordo com Sebesta (2009), os primeiros computadores digitais, surgidos na década de 40, tiveram aplicações científicas, onde a linguagem *Assembly* era utilizada, e o Fortran surgia como linguagem de alto nível para maior performance. Afirma ainda que linguagem de programação de alto nível é uma linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana. A respeito da semelhança entre linguagens naturais (e.g inglês, espanhol, chinês) e linguagens de programação, Pressman e Maxin (2016) ressaltam que, em ambos os casos, a linguagem possui sintaxe e semântica, usadas para transmitir ideias ou instruções procedurais de forma eficiente.

Além das linguagens, conforme ressalta Paula Filho (2003), vários outros elementos são importantes no desenvolvimento e na utilização de um *software*.

> "O *software* é a parte programável de um sistema de informática. Ele é um elemento central: realiza estruturas complexas e flexíveis que trazem funções, utilidade e valor ao sistema. Mas outros componentes são indispensáveis: as plataformas de hardware, os recursos de comunicação de informação, os documentos de diversas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados" (DE PÁDUA PAULA FILHO, 2003, p. 11).

Lobo (2009) afirma que a qualidade e a rapidez na construção de um *software* dependem, dentre outros quesitos, do método adotado no ambiente de desenvolvimento, do

controle de qualidade e da utilização de uma arquitetura de *software* adequada. Bass *et al*. (2003) afirmam que a arquitetura de *software* constitui um modelo relativamente pequeno e compreensível, de como o sistema é estruturado e como seus elementos trabalham juntos. Já Clements *et al*. (2010) definem arquitetura como a divisão prudente do "todo" em partes, com relações específicas entre as partes. Na definição do SEI (*Software Engineering Institute*), arquitetura de *software* é a estrutura de componentes de um programa/sistema, os relacionamentos entre esses componentes, os princípios e diretrizes que governam os projetos e a evolução dos *softwares*.

> "A escolha de um padrão de arquitetura deve ser guiada pelas propriedades gerais da aplicação a ser desenvolvida. Antes de escolher um padrão específico, deve-se explorar várias alternativas, pois diferentes padrões de arquitetura implicam em diferentes estruturas e, consequentemente, diferentes sistemas" (LEITE, 2007, p. 91)

A seguir, são explorados diversos elementos presentes na arquitetura deste projeto.

2.2.1.1 HTML

HTML é a sigla em inglês para *HyperText Markup Language*, que, em português, significa linguagem para marcação de hipertexto. Brooks (2007) ressalta que é uma linguagem de marcação usada para especificar a estrutura de um documento. Um navegador de internet (web browser) é um *software* que interpreta estas marcações de estrutura e, então, constrói uma página web com recursos de hipermídia com os quais o usuário pode interagir.

Conforme expõe Bax (2001), nas linguagens de marcação, marcas (*tags*) descritivas definem o início e o fim do texto marcado como unidade ou elemento de informação, que podem ser embutidos uns dentro dos outros. A Figura 5 ilustra como a linguagem de marcação cria a estrutura de uma página simples.

Figura 5 – Exemplo básico de página HTML

Fonte: Mozilla (2017)

Korpela (2005) adverte que HTML, assim como as demais linguagens de marcação, diferem das linguagens de programação, pois estas contêm instruções que transformam a informação, enquanto aquelas somente estruturam a informação.

Dentre as diversas marcações possíveis nesta linguagem, este trabalho faz uso do SVG, abreviatura de *Scalable Vector Graphics*. Conforme definição da Mozilla (2017), gráficos vetoriais escaláveis podem ser embutidos diretamente no HTML para descrever de forma vetorial desenhos e gráficos bidimensionais, quer de forma estática, dinâmica ou animada. A Figura 6 ilustra como o SVG pode gerar gráficos, apresentando na esquerda o código utilizado e na direita o resultado.

Figura 6 – Exemplo de gráfico gerado pelo SVG

Fonte: Mozilla (2017)

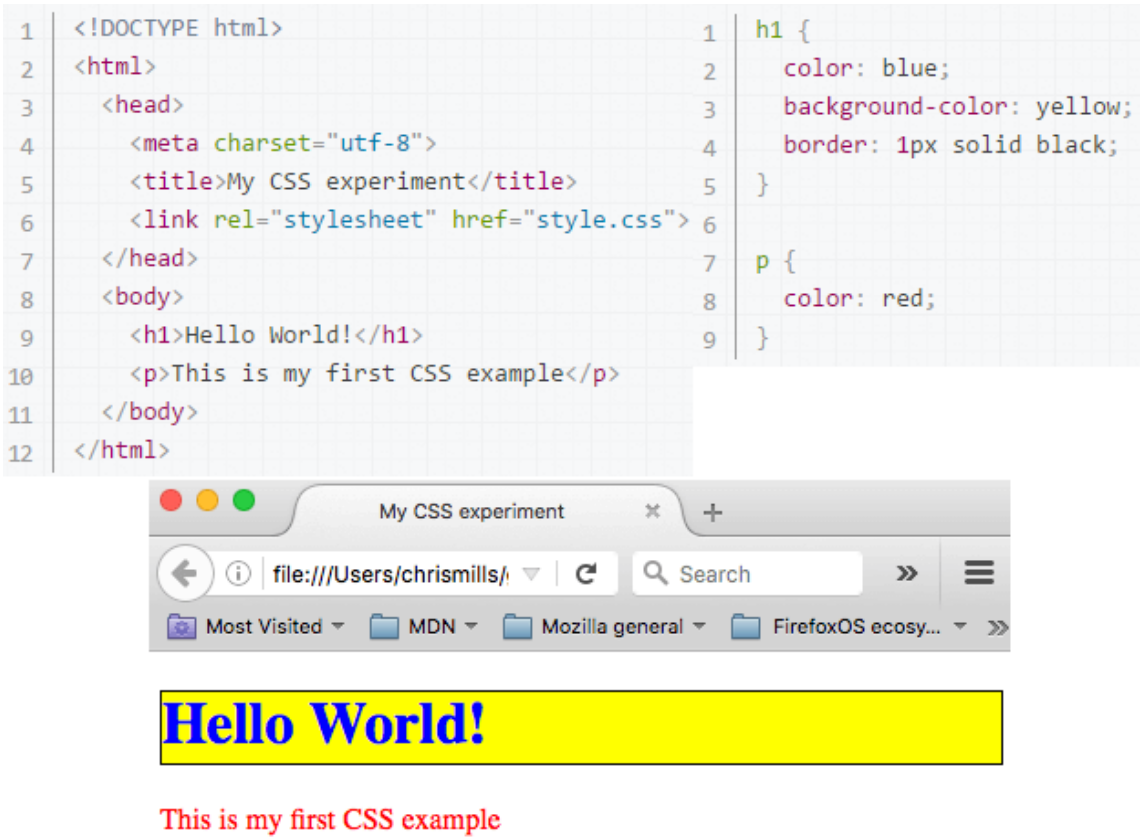
Kaufman (1993) e Chapman (2002) definem gráficos vetoriais como o uso de polígonos para representar imagens em computadores. São baseados em vetores, determinados por vértices, pela sua posição cartesiana e direção. Esses vetores formam caminhos que podem ter vários atributos, como espessura, cor, forma e preenchimento.

2.2.1.2 CSS

Outra linguagem de marcação relevante no desenvolvimento Web é o CSS (*Cascading Style Sheets*). A Mozilla Foundation (2017) define como uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML, descrevendo como elementos são mostrados na tela, no papel, no discurso ou em outras mídias.

Grannel (2007) afirma ser uma linguagem de estilo, usada para especificar a aparência dos vários elementos de um documento que foi definido por uma linguagem de marcação, de forma separada. A Figura 7 ilustra como o CSS, à direita, é capaz de determinar a aparência de uma página HTML, à esquerda. O resultado encontra-se abaixo.

Figura 7 - Exemplo de estilo determinado por CSS



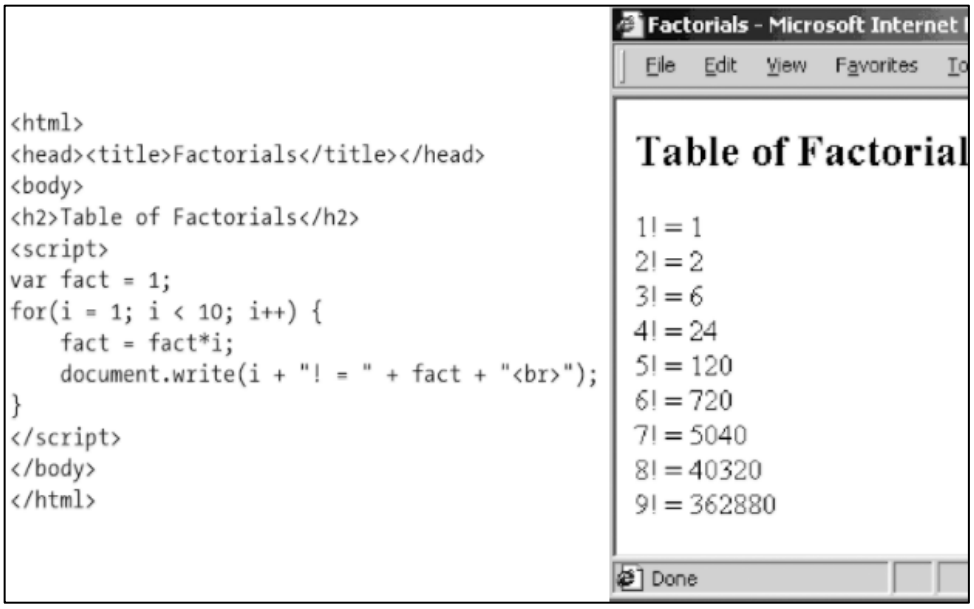
Fonte: Mozilla (2017)

2.2.1.3 JavaScript

Flanagan (2006) define *JavaScript* como uma linguagem de programação interpretada, utilizada em *browser*, que permite aos scripts interagirem com o usuário, controlarem o browser, ou alterarem o documento sendo apresentado pelo browser.

Bortolossi (2012) apresenta *JavaScript* como uma linguagem que permite modificar e integrar, de forma dinâmica e interativa, o conteúdo e a aparência dos vários elementos que compõem uma aplicação elaborada com HTML e CSS. A Figura 8 demonstra como um *script* pode gerar, de forma dinâmica, a estrutura de uma página apresentada em browser.

Figura 8 - Exemplo de script em JavaScript



Fonte: Flanagan (2006)

2.2.1.4 Angular

Conforme expõe Korva (2016), o *Angular* é um *framework* para construção de aplicações web utilizando HTML, CSS e TypeScript. Segundo Mattson (2002), um *framework* é uma abstração e implementação para uma aplicação em um dado domínio do problema. Uma parte que pode ser utilizada em diversos projetos e que, em conjunto com outros componentes, completam uma aplicação.

Srinivasan (2014) afirma que *frameworks* existem para simplificar o desenvolvimento de uma aplicação, implementando recursos comuns à vários sistemas, como autenticação, segurança, e componentes de interface. Afirma ainda que a utilização correta de um *framework* pode reduzir o tempo de desenvolvimento.

Korva (2016) comenta que *frameworks* são utilizados extensivamente na atualidade, sendo essenciais na criação de websites modernos e aplicações de página única. Mikowski (2013) define as aplicações de página única (*SPA- Single Page Application*) como

websites carregados somente uma vez (não precisam de navegação com carregamento recorrente de novas páginas), fornecendo uma experiência similar de imediatismo dos *softwares* tradicionais, enquanto mantem a portabilidade da web.

Segundo seus criadores, *Angular* é um *framework* com uma arquitetura MVC e baseada em componentes. O padrão de arquitetura *Model-View-Controller* (MVC), segundo Bubeck (2003), divide a aplicação interativa em três componentes: *model, view e controller*. O *model* (modelo) encapsula o núcleo de dados e de funcionalidade da aplicação, o view (visualização) apresenta informações oriundas do modelo para o usuário e o *controller* (controlador) manuseia as informações segundo eventos acionados pelo usuário. Grossman (2016) afirma que a arquitetura baseada em componentes consiste em separar uma aplicação em pequenas partes isoladas, os componentes, que podem ser constantemente reutilizadas. Ressalta ainda que esta arquitetura está em ascensão no ambiente de desenvolvimento web.

A Figura 9 demonstra como o *Angular* utiliza as marcações HTML, juntamente com uma sintaxe própria, para garantir funcionamentos específicos à interface.

Figura 9 - Ilustração de marcações em HTML no *Angular Framework*

```
<div class="form-group">
    <label for="all-variables">Muuttuja:</label>
    <select class="form-control" name="all-variables" [ngModel]="ActiveElement.Variable"
    (change)="OnElementVariableChange($event.target.value)">
        <option *ngFor="let variable of Scenario.variables | map"
        [value]="variable.key">{{variable.value.Name}}</option>
    </select>
</div>
```

Fonte: Korva (2016)

A principal linguagem de programação utilizada em *Angular* é o TypeScript. Segundo a *TypeScript Language Specification* (2016), é uma linguagem opcionalmente tipada, sendo um superconjunto do JavaScript, desenvolvido pela Microsoft para facilitar o desenvolvimento de grandes projetos em JavaScript. Um código desenvolvido em TypeScript é compilado para JavaScript, que é a linguagem efetivamente utilizada pelo browser.

Na Figura 10, um código em TypeScript utiliza os recursos do *Angular*, sendo possível perceber similaridades com o JavaScript, bem como sintaxe própria.

Figura 10 - Ilustração de um código Typescript no *Angular Framework*

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({name: 'keys'})
export class KeysPipe implements PipeTransform {
transform(value) {
  let keys:any = [];
  for (let key in value) {
     keys.push( {key: key, value: value[key]} );
   }
    return keys;
  }
}
```

Fonte: Korva (2016)

2.2.1.5 Google Maps

De acordo com a Google®, o *Maps* é um serviço online de pesquisa e visualização de mapas e imagens de satélite da Terra, gratuito até certos níveis de utilização. Svennerbeg (2010) afirma que o serviço foi iniciado em 2005, revolucionando a apresentação de mapas em páginas w*eb*, ao permitir os usuários, de forma gratuita, navegar por mapas de elevada integridade. A Google® permite a utilização destes mapas em outras aplicações, através de uma API (*Application Programming Interface*).

> "Uma Interface de Programação de Aplicação (tradução livre de API), é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços" (FOLDOC, 1995, p. 1).

Svennerbeg (2010) explica que esta API consiste em HTML, CSS e JavaScript funcionando juntos, realizando requisições de informações em servidores à medida que se navega no mapa. A API fornece classes, com métodos e propriedades, que podem ser usadas para controlar o comportamento dos mapas.

2.2.1.6 Armazenamento

Como pode ser constatado através de Carzaniga (1998), um *software*, como informação, precisa ser distribuído para o usuário final. Em relação à distribuição, De Castro (1997) afirma que em um ambiente distribuído, como a *Web*, onde uma comunidade de usuários (clientes) deseja obter dados multimídia simultaneamente ou não, é necessário um sistema que administre o fornecimento do material requisitado de forma eficiente.

> "Um componente desse sistema é um servidor de dados multimídia. Esse servidor realiza o armazenamento de dados como áudio, vídeo e gráficos (entre outros) em algum tipo de dispositivo físico (disco, fita, etc.). Além do armazenamento, um servidor de dados multimídia deve receber pedidos, simultâneos ou não, de acesso aos dados a partir de um grande número de usuários dispostos em locais geograficamente diferentes. Após receber os pedidos o sistema deve honrá-los, transmitindo (sob demanda) o material solicitado ao usuário" (DE CASTRO, 1998, p. 8)

Segundo Dias (2016), o GitHub é um serviço de hospedagem para projetos de *software* livre, permitindo a publicação do código-fonte do sistema e o desenvolvimento colaborativo. Além disso, o GitHub permite a distribuição de *softwares web* para o usuário.

Além da informação do *software*, existem informações de entrada ou saída do sistema, utilizadas ou produzidas pelo usuário. Dentre as alternativas de estruturação dessa informação, Bray (2014) destaca o *JavaScript Object Notation* (JSON) como um formato, independente de outras linguagens, para transmissão e armazenamento de informações. Sendo leve e baseado em texto, define um pequeno conjunto de regras de formatação para realizar uma representação portátil de informação estruturada.

De acordo com o Crunchbase (2017), o Firebase está entre os muitos serviços online que provém uma API para desenvolvedores armazenarem e sincronizarem informação através de múltiplos clientes. O Firebase utiliza o JSON como estrutura de dados.

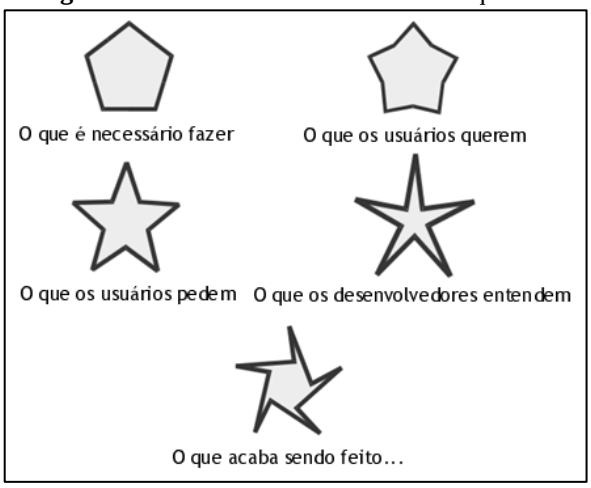
2.2.2 Qualidade de software

Para Deming (1982), a qualidade seria representada pela melhoria contínua de produtos e processos, visando à satisfação dos clientes. Crosby (1992) define qualidade como a conformidade aos requisitos do cliente. Juran e Gryna (1991) conceituarem qualidade como *fitness for use* (adequação ao uso). Garvin (1992) adota diversas dimensões da qualidade: desempenho, características, confiabilidade, conformidade, durabilidade, atendimento, estética e qualidade percebida. Ressalta ainda um produto ou serviço pode ser bem cotado em uma dimensão, mas não em outra, estando em muitos casos inter-relacionadas. Oakland (1994) adverte o aspecto subjetivo da qualidade, onde as necessidades diferem entre as pessoas, e a noção de qualidade depende fundamentalmente da percepção do indivíduo.

Sommerville (2003) destaca que diferentes *stakeholders* têm em mente diferentes requisitos e podem expressá-los de maneiras distintas. Os engenheiros de requisitos precisam descobrir todas as possíveis fontes de requisitos e encontrar pontos comuns e os conflitos. Para Paula Filho (2003), um dos problemas básicos da engenharia de *software* é o

levantamento e documentação dos requisitos dos programas, que, apesar de custar tempo e dinheiro, são fundamentais para evitar o risco de resolver o problema errado. A Figura 11 ilustra as incoerências entre os requisitos percebidos por clientes e desenvolvedores.

Figura 11 - Riscos na incoerência dos requisitos



Fonte: Paula Filho (2013)

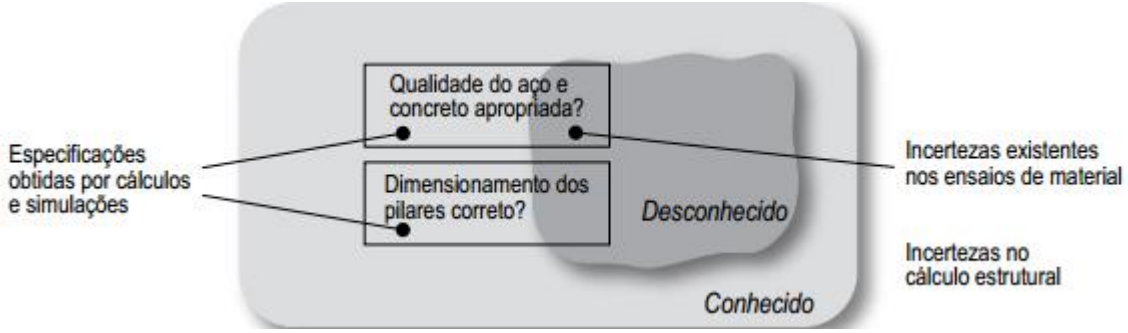
No contexto de *softwares*, qualidade pode ser definida, conforme exposto por Pressman e Maxin (2016), como uma gestão efetiva, aplicada de forma a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. Koscianski e Soares (2006) relatam que as primeiras discussões sobre qualidade de *software* se iniciaram em 1968, e que as preocupações daquela época, incluindo a década de 1970, continuam bastante atuais:

- Cronogramas não observados;
- Projetos com tantas dificuldades que são abandonados;
- Módulos que não operam corretamente quando combinados;
- Programas que não fazem exatamente o que era esperado;
- Programas tão difíceis de usar que são descartados;
- Programas que, simplesmente, param de funcionar.

No que diz respeito a riscos e incertezas, Koscianski e Soares (2006) comparam as diferenças entre o escopo da qualidade de projetos tradicionais e o escopo específico de qualidade de *software*, conforme apresentado pela Figura 12 e pela Figura 13.

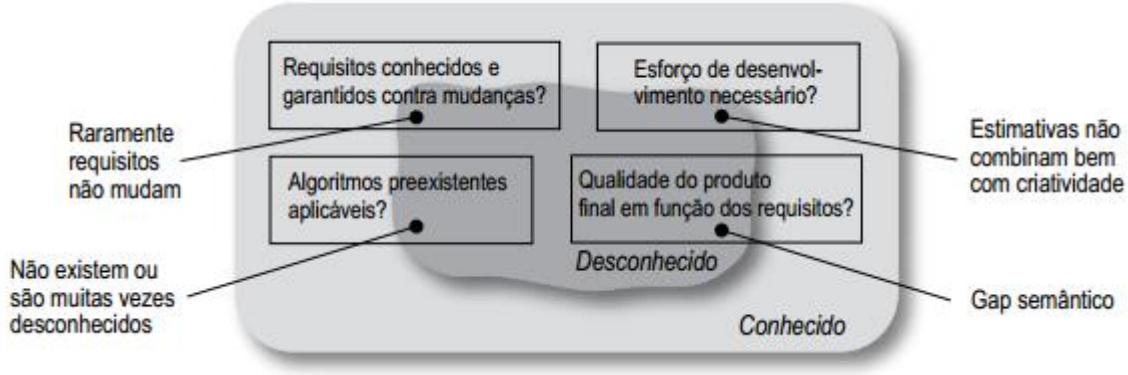
> "É durante o projeto dessa ponte que há mais questões em aberto e soluções de engenharia devem ser desenvolvidas. [...] uma vez terminado o projeto, contudo, existem respostas para a maior parte das questões que dizem respeito à realização. Os carros que irão trafegar não mudarão de largura nem de comprimento repentinamente; a quantidade de veículos e, portanto, os pesos máximos a serem suportados são calculados utilizando-se *software* de simulação. Dificilmente um pilar deverá ser deslocado cinco centímetros para a direita depois que a construção já foi iniciada! [...] comparativamente, quando estamos tratando de *software*, essa zona de sombra que envolve fatores desconhecidos é bem mais abrangente. Quem já trabalhou no desenvolvimento de *software* sabe como é comum resolver problemas semelhantes a encurtar ou alongar uma ponte, "apenas alguns centímetros" alguns dias antes de sua inauguração. Uma mudança nas necessidades declaradas por um usuário pode repercutir em vários elementos da estrutura do programa" (KOSCIANSKI e SOARES, 2006, p. 22-23).

Figura 12 - Zonas de sombra em um projeto de realização de uma ponte



Fonte: Koscianski e Soares (2006)

Figura 13 - Zonas de sombra em projeto de implementação de *Software*



Fonte: Koscianski e Soares (2006)

Segundo Bourque *et al*. (2014), uma comissão internacional de especialistas criou em 2004, no âmbito da IEEE, o SWEBOK (*Software Engineering Body Of Knowledge*, Corpo de Conhecimento de Engenharia de *Software* em tradução livre), que, dentre várias outras áreas do conhecimento, também aborda qualidade de *software*, assim como a ISO 25000:2005, também chamada de SQuaRE (*Software product Quality Requirements and Evaluation*).

Dentre os muitos temas abordados pelo SWEBOK, este trabalho ressalta a Ergonomia de *Software*. Por Koscianski e Soares (2006), percebe-se que requisitos de

ergonomia podem causar impactos até sobre a segurança de operação de um produto, indo além de uma questão estética, sendo um exemplo disso os *softwares* de controle de tráfego aéreo.

> "Não há dúvida de que cada um de nós tem uma visão diferente e muito subjetiva do que é estética. Mesmo assim, a maioria de nós concordaria que uma entidade estética tem certa elegância, um fluir único e uma "presença" que são difíceis de quantificar, mas que, não obstante, são evidentes. Um *software* estético possui essas características" (PRESSMAN e MAXIN, 2016, p. 416).

No âmbito educacional, Alves (2007) cita alguns aspectos pedagógicos para avaliação de *software*: clareza dos conteúdos, assimilação e acomodação, recursos motivacionais, avaliação do aprendizado, carga educacional e tratamento do erro. Batista *et al*. (2004), em conformidade com Campos e Campos (2001), define aspectos que devem ser levados em consideração na avaliação de um *software* educacional:

- **Características pedagógicas:** atributos que evidenciam a conveniência e a viabilidade de uso do *software* em situações educacionais.
- **Facilidade de uso:** atributos que evidenciam a facilidade de uso do *software*.
- **Características da interface:** atributos que evidenciam a presença de recursos e meios que facilitam a interação do usuário com o *software*.
- **Adaptabilidade:** atributos que evidenciam a capacidade do *software* de adaptar-se às necessidades e preferências do usuário e ao ambiente educacional selecionado.
- **Documentação:** atributos que evidenciam que a documentação para instalação e utilização do *software* está completa, é consistente, legível e organizada.
- **Portabilidade:** atributos que evidenciam a adequação do *software* aos equipamentos onde serão instalados.
- **Retorno do investimento:** atributos que evidenciam a adequação do investimento na aquisição do *software*.

3. DESENVOLVIMENTO DO *SOFTWARE*

O presente capítulo descreve cada etapa do desenvolvimento do *software*, utilizando como base os conceitos de roteirização, localização e desenvolvimento de software descritos no capítulo 2.

3.1 Etapas de desenvolvimento

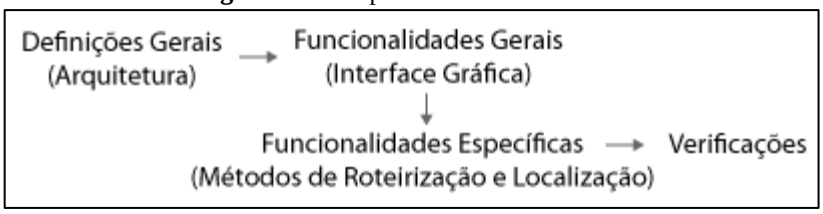
O desenvolvimento se inicia com a definição da arquitetura do *software*, através da escolha das tecnologias envolvidas e resumo de suas interações, que formam as bases de funcionamento do sistema.

Em seguida, é apresentada uma elaboração detalhada da interface gráfica e das funcionalidades gerais do sistema, como manipulação de mapa e grafo, seleção de elementos, abrir e salvar problemas, entre outras.

Após as funcionalidades gerais, segue-se o desenvolvimento de funcionalidades específicas, relacionadas aos métodos de localização e roteirização utilizados e seus algoritmos respectivos.

Ao final, algumas verificações são apresentadas para atestar a eficiência do *software* na aplicação destes métodos, comentando-se os resultados obtidos no contexto dos objetivos e limitações do trabalho. A Figura 14 resume essas etapas.

Figura 14 – Etapas do desenvolvimento



Fonte: Autor (2017)

Eventualmente, o *software* em desenvolvimento pode ser referenciado como **kLog**, para facilitar a leitura do texto.

3.2 Definição da arquitetura

A definição das tecnologias a serem utilizadas é uma das primeiras decisões de desenvolvimento. Na escolha de uma linguagem para implementação dos algoritmos, considerou-se que os utilizados neste trabalho não são muito complexos, sendo possível sua implementação em muitas linguagens. Os problemas a serem resolvidos são de pequena

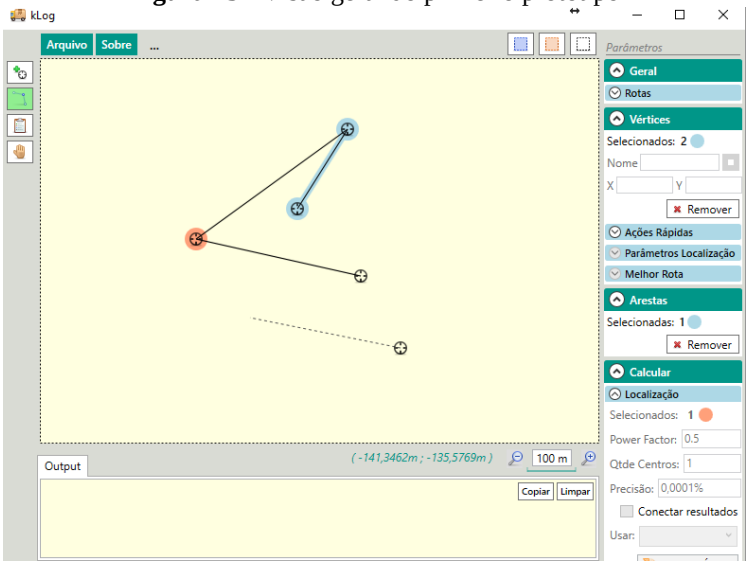
escala, com foco no aprendizado, e não problemas reais que exigem muito processamento e são melhor implementados em tecnologias específicas de alta performance ou que fornecem, em seu ecossistema, uma grande quantidade de *frameworks* e bibliotecas úteis para a resolução de problemas específicos.

Portanto, a capacidade da linguagem de programação a ser escolhida, para implementação dos algoritmos, não é um critério muito relevante. Por outro lado, considerando as características pedagógicas e de interface como requisitos de qualidade, seria necessária uma tecnologia que fornecesse liberdade no desenvolvimento da interface gráfica, e, ao mesmo tempo, robustez, para ser possível explorar ao máximo recursos visuais que colaborem para os objetivos educacionais.

Não são todas as tecnologias de *software* que fornecem esta liberdade, muitas delas fornecem um conjunto limitado de elementos visuais, com baixo grau de customização. Assim, certos comportamentos desejados não poderiam ser implementados neste trabalho, ou em novas funcionalidades futuras. Por exemplo, comparando-se o *Windows Forms*, antiga tecnologia do ambiente Windows ainda muito utilizada, com o WPF (*Windows Presentation Foundation*), uma tecnologia mais recente, pode-se perceber que esta possui “capacidades gráficas bem melhores” e “aplicações com interfaces ricas” (LIMA, 2016, p.1).

Outro critério utilizado na escolha das tecnologias é o custo de desenvolvimento, que, neste trabalho, consiste do tempo e recursos dispendidos pelo autor. Após definir uma tecnologia, existe um dispêndio de tempo ao percorrer sua curva de aprendizado. Assim, considerando este critério e o requisito de uma interface gráfica rica, a tecnologia WPF foi escolhida como primeira opção de desenvolvimento, por permitir uma interface gráfica robusta e ter baixo custo por parte do autor, que já tem proficiência na tecnologia e no ecossistema Windows, com conhecimentos em C# e XAML, linguagens utilizadas no WPF. A Figura 15 mostra uma primeira versão desenvolvida.

Figura 15 - Visão geral do primeiro protótipo



Fonte: Autor (2017)

Esta versão, mesmo funcional, teve seu desenvolvimento abandonado. Durante o desenvolvimento, percebeu-se que alguns requisitos de qualidade não foram considerados nas decisões iniciais. A tecnologia WPF eleva a qualidade do *software* por fornecer rica interface gráfica, mas funciona somente no ambiente Windows, limitando a distribuição do *software* a usuários que utilizam esse sistema operacional. Usuários de Linux, iOS ou Android, por exemplo, ficam impossibilitados de utilizar a aplicação. O requisito de portabilidade, que não foi ponderado, afeta a qualidade final do *software* educacional.

> "Uma tendência que se observa é disponibilizar ambientes que não necessitam de qualquer instalação para serem acessados, como aqueles que executam sobre a plataforma Flash. Desenvolver *softwares* cada vez mais accessíveis via um browser na Web, sem necessidade de o usuário preocupar-se com a instalação, deve fazer parte do conjunto de requisitos a ser incorporado ao projeto dos *softwares* educacionais" (GIRAFFA, 2009, p.27)

A consideração anterior, feita por Giraffa (2009), revela uma tendência ainda atual de se desenvolver aplicações *Web* de alta portabilidade, em um mundo digital que possui vários sistemas operacionais com número expressivo de usuários. A tecnologia *Web* permite distribuir um *software* para um número maior de usuários, de uma forma mais simples, independente de executáveis e instalações, bastando o acesso a um endereço através de um browser.

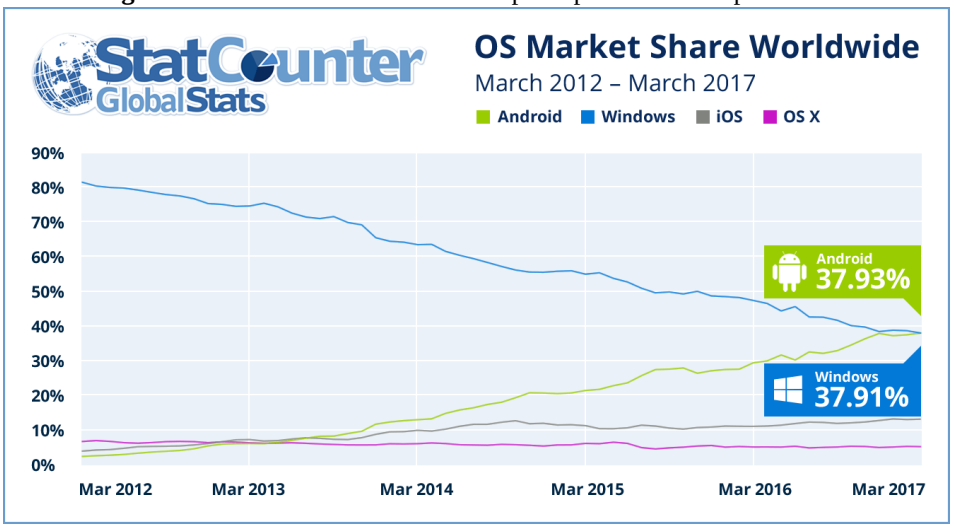
As tecnologias *Web* já foram mais limitadas. Há alguns anos, seria difícil fornecer as mesmas funcionalidades de uma aplicação desenvolvida em WPF, por exemplo, em uma aplicação Web. Este cenário mudou bastante e, atualmente, através do HTML5 e de

frameworks de aplicativos de página única (SPA), já é possível o desenvolvimento de aplicações tão complexas quanto uma aplicação *Desktop* nativa.

Percebe-se, então, um risco que existe na escolha das tecnologias: Uma tecnologia limitada a uma plataforma pode ser uma péssima escolha se esta plataforma tender ao desuso. O *software* pode se tornar inutilizável em alguns anos. O LogWare®, por exemplo, desenvolvido em uma tecnologia antiga, não se adaptou às novas versões de sistema operacional, tendo, atualmente, baixíssima qualidade no requisito de portabilidade.

Percebendo esta mudança de cenário, onde, conforme a Figura 16, não existe mais um sistema operacional que domina o *market share*, e as tecnologias *Web* se tornam cada vez mais robustas, optou-se por reiniciar o desenvolvimento com essas novas tecnologias. O custo de desenvolvimento continua existindo, exigindo um período de treinamento e adaptação com tecnologia Web, mas este fator foi desconsiderado nesta nova decisão.

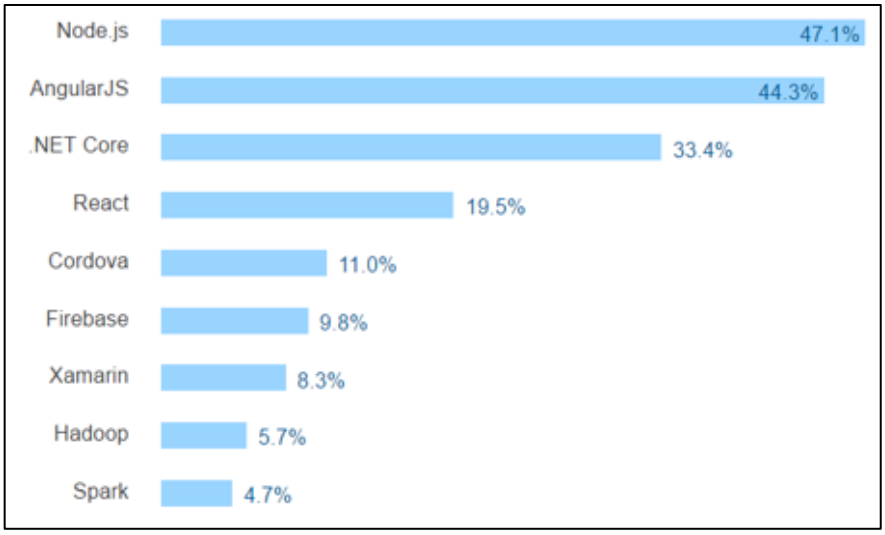
Figura 16 - Divisão do mercado entre principais sistemas operacionais



Fonte: MacMagazine (2017)

Uma vez escolhido o ecossistema *Web*, fica definida a importância da utilização de HTML, CSS e *JavaScript*, mas surge a necessidade de escolher quais *frameworks* ou bibliotecas serão utilizados, frente a uma profusão de opções. Um mesmo sistema pode ser desenvolvido, com resultado semelhante, em diversos *frameworks* diferentes. O StackOverflow, uma comunidade de desenvolvedores, com reconhecimento mundial e mais de 7 milhões de usuários, publicou em janeiro de 2017 uma pesquisa com 64 mil desenvolvedores de 213 países, comparando a utilização e popularidade dos principais *frameworks*, conforme mostra a Figura 17.

Figura 17 - Popularidade mundial de algumas tecnologias de desenvolvimento



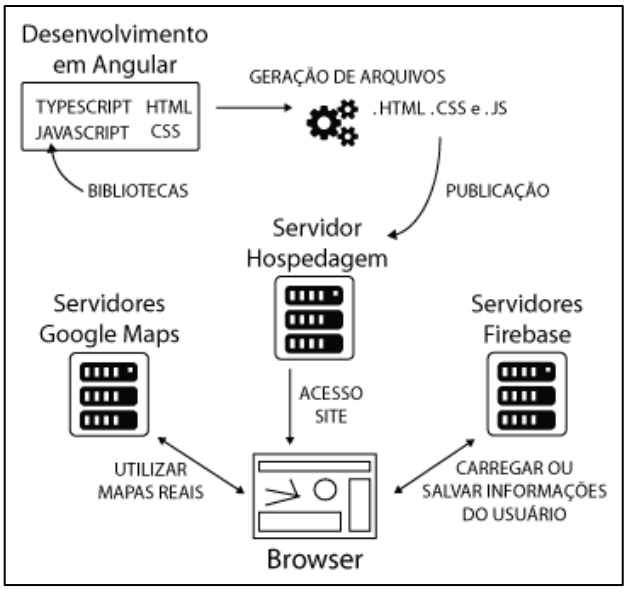
Fonte: StackOverflow (2017)

Esta pesquisa pode servir de referência para a escolha do *framework*. Ela destaca, por exemplo, que utilizar o *Angular* talvez seja mais interessante que utilizar a tecnologia *React*, já que aquela possui uma maior utilização e popularidade, indicativos de qualidade e documentação farta. Contudo, realizando uma pesquisa exploratória nas comunidades de desenvolvedores da *Web*, encontra-se dezenas de artigos de opinião destacando as vantagens de um *framework* sobre os demais, observando grandes divergências de opinião e pouco consenso. Assim, os riscos na escolha continuam existindo. Um *framework* pode cair em desuso em alguns anos, dificultando a compatibilidade com os *browsers* e desenvolvimentos futuros do sistema. O *Angular* é desenvolvido pela Google®, onde a Microsoft® tem coparticipação ao ser a criadora do *TypeScript*, transmitindo a confiabilidade que possuem essas empresas. Por ser um *framework* bastante utilizado e maduro, foi escolhido para este projeto, acreditando-se ser uma escolha que mantenha o sistema compatível e atualizável por muitos anos.

De forma semelhante, o *Firebase* foi o serviço escolhido para realizar o armazenamento de informações do usuário, como problemas modelados. É um serviço gratuito, baseado em tecnologia *Web*, que permite armazenar informações no formato JSON. O *GitHub* é utilizado como repositório gratuito de código aberto, o que permite futuros trabalhos colaborativos, além de permitir também a publicação da aplicação direto para o usuário final, sem necessitar de outro serviço de hospedagem de páginas ou contratação de domínios. Outro serviço utilizado é o Google Maps, que acrescenta funcionalidades interessantes no sistema, ao permitir utilizar mapas reais. É um serviço gratuito para pequenas

e médias demandas, que fornece uma API em *JavaScript* para sua utilização. A Figura 18 apresenta os elementos envolvidos na arquitetura e as tecnologias definidas.

Figura 18 - Arquitetura geral do sistema



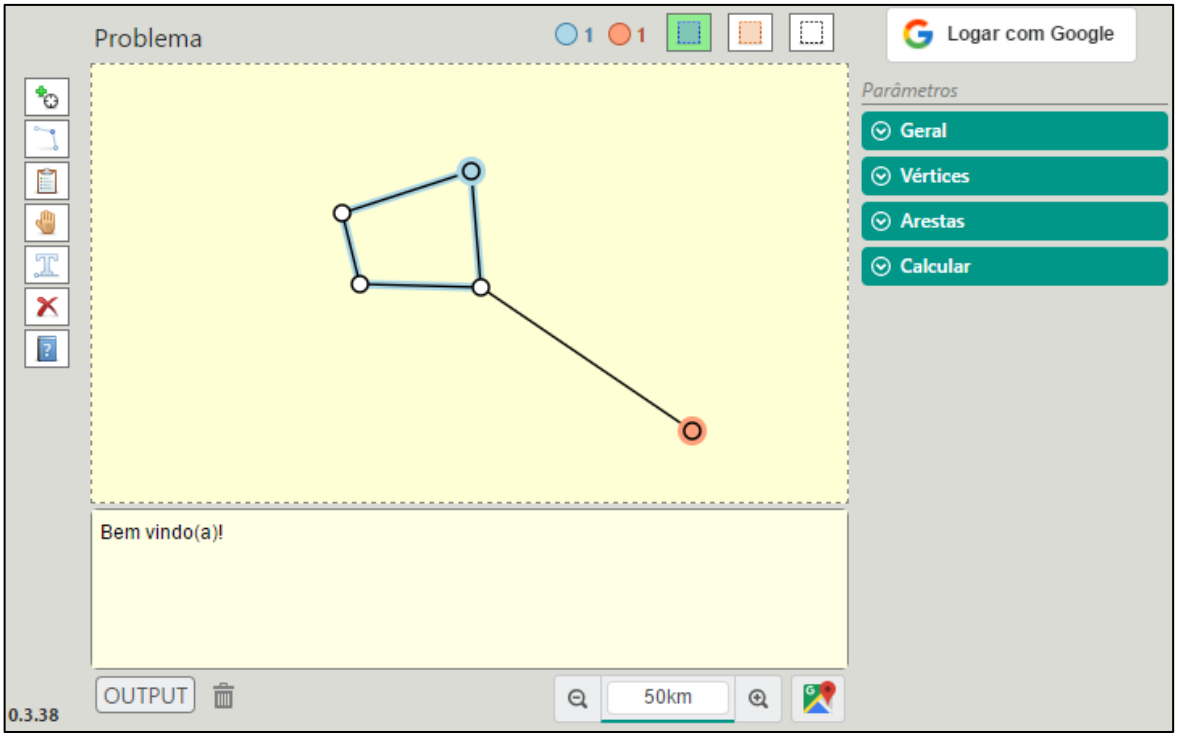
Fonte: Autor (2017)

3.3 Interface gráfica e funcionalidades gerais

3.3.1 Visão geral

Os problemas de localização e roteirização envolvem diversas variáveis de entrada, como custos, taxas, volumes, coordenadas, etc. Dentre essas variáveis, uma estrutura se destaca como elemento central desses problemas, que são as coordenadas das localidades e as relações que estas guardam entre si. Estas informações podem ser representadas visualmente através de grafos, onde os vértices são posicionados de acordo com suas coordenadas geográficas em um mapa imaginário. Assim, este mapa foi definido como elemento central da interface, permitindo ao usuário modelar o problema e visualizar os seus resultados tendo este mapa como referência. Na Figura 19, ele possui fundo amarelo, ao centro, apresentando um grafo de exemplo em seu interior.

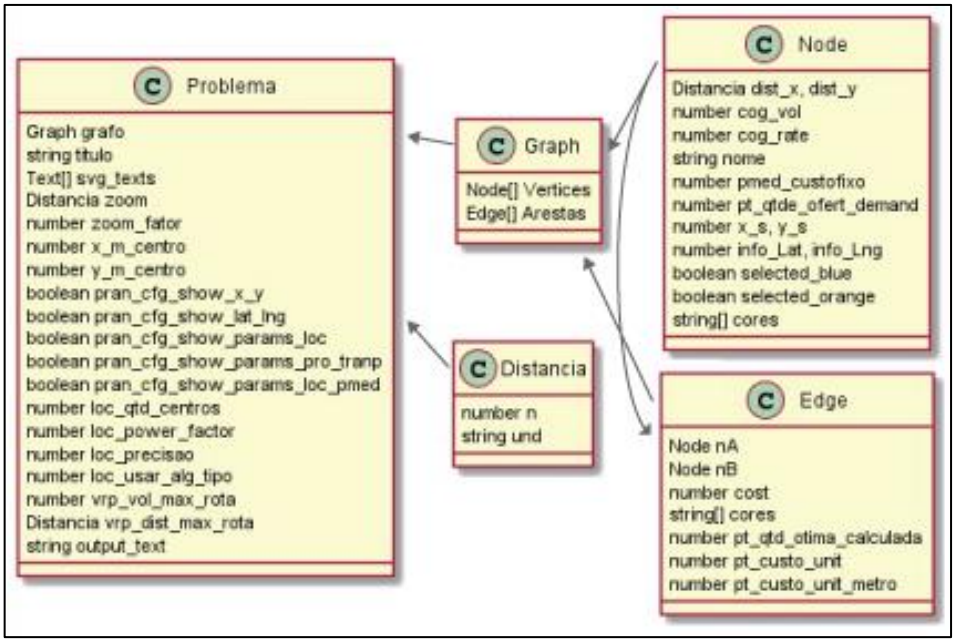
Figura 19 - Visão geral da interface gráfica



Fonte: Autor (2017)

Para o desenvolvimento deste elemento da interface, foram utilizadas marcações SVG, que estão sincronizadas com as informações do problema. Assim, se o problema possui um vértice com certas coordenadas, marcações SVG propriamente configuradas irão desenhar um círculo no local correto. Essa sincronização entre as informações do problema e sua representação visual é possível graças às funcionalidades de *Data Binding* do *Angular*. Por exemplo, dentro da programação do sistema, um objeto central, denominado Problema, contém informações relevantes para a utilização do *software*. A Figura 20 apresenta um esboço do esquemático geral desta entidade. Várias informações foram omitidas por simplificação, já que todas as entidades envolvidas possuem uma grande quantidade de variáveis e métodos utilizados nas funcionalidades criadas. É nesse objeto que são mantidas várias informações do problema modelado, como nome do problema, grafo, vértices, arestas, parâmetros, etc. Várias dessas informações são mantidas em sincronização com a interface gráfica, através de marcações especiais do *Angular*, realizadas junto ao HTML da página.

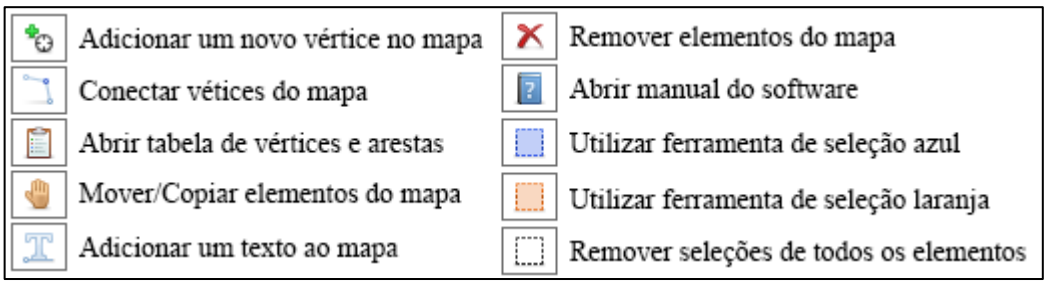
Figura 20 – Esquemático geral da entidade Problema



Fonte: Autor (2017)

Para modelar o problema através do mapa e seus elementos, a interface foi desenhada para fornecer, ao usuário, acesso à diversas ferramentas. Utilizando um padrão consagrado de disposição dos elementos visuais (utilizado por *softwares* como Photoshop®), manteve-se o objeto em edição (o mapa) ao centro, enquanto as possíveis ferramentas de edição encontram-se na periferia da interface. A Figura 21 lista os botões de algumas ferramentas e um resumo de suas funcionalidades.

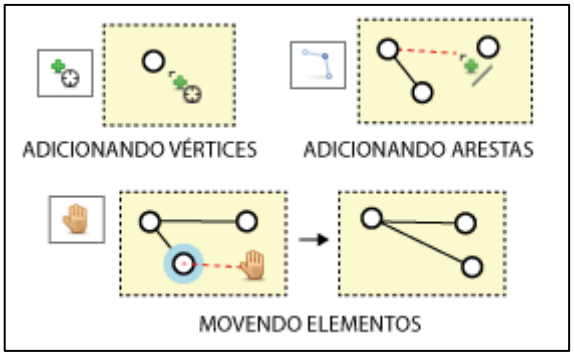
Figura 21 - Algumas ferramentas e suas funcionalidades



Fonte: Autor (2017)

A Figura 22 demonstra algumas dessas ferramentas sendo utilizadas, através da adição de um novo vértice, conexão de dois vértices e movimentação de vértice.

Figura 22 – Exemplos de utilização de algumas ferramentas

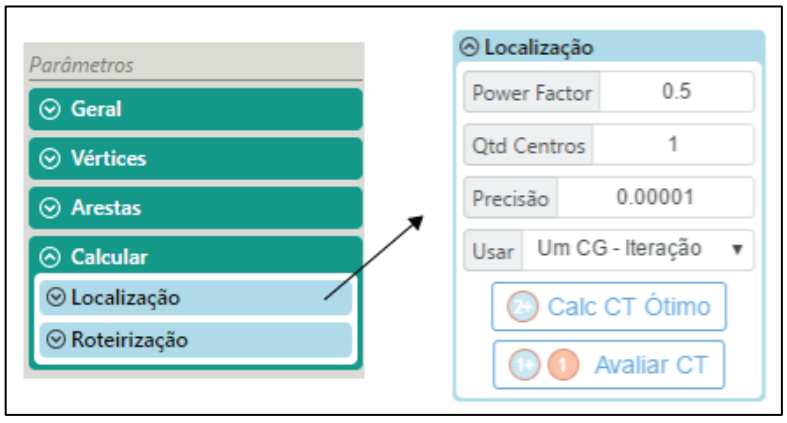


Fonte: Autor (2017)

O sistema ainda permite outras ações úteis, especificadas no manual, como movimentação de vários elementos, duplicação de elementos, seleções invertidas, etc. Essas ações extras podem ser ativadas com certos comandos, por exemplo, ao se pressionar as teclas *Ctrl* ou *Sfhit* enquanto utiliza a ferramenta.

Além das ações executadas por essas ferramentas, existem vários parâmetros configuráveis para se resolver um determinado problema, por exemplo, a precisão a ser utilizada em um algoritmo, a quantidade de centros de distribuição a serem localizados, o custo unitário de transporte em uma rota, etc. Determinou-se duas estruturas na interface com esse propósito. Na primeira, através dos elementos na direita da interface, o usuário pode configurar diferentes parâmetros do problema e executar diferentes ações, como demonstra a Figura 23.

Figura 23 - Elementos visuais para configuração de parâmetros



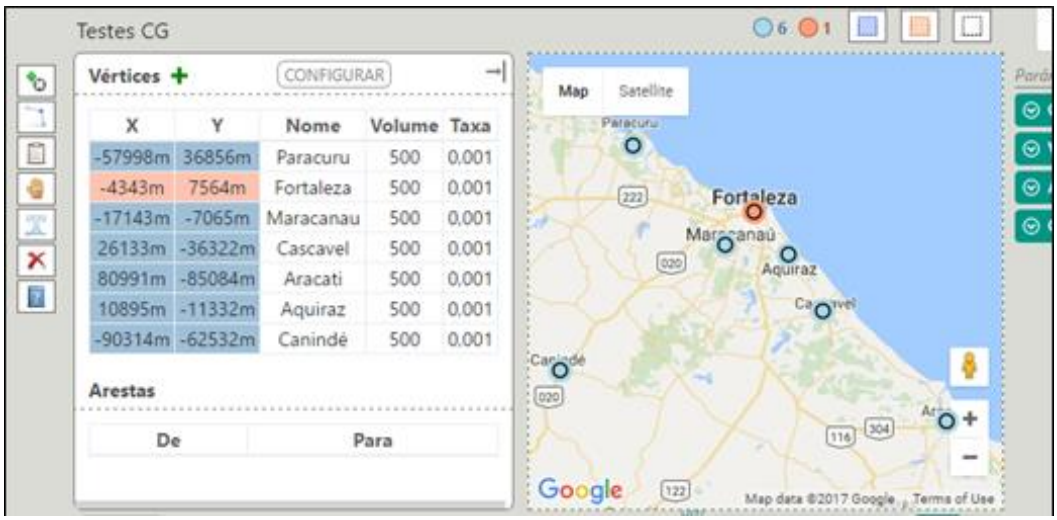
Fonte: Autor (2017)

Todo o sistema utiliza uma abordagem minimalista, em conformidade com requisitos ergonômicos, disponibilizando na interface somente os elementos necessários em certo instante da navegação. Assim, como demonstra a Figura 23, se o usuário navega pela

aplicação resolvendo um problema de localização, quaisquer elementos desnecessários, como aqueles relacionados à roteirização, podem ficar ocultos.

A segunda estrutura elaborada para permitir a configuração de parâmetros de um problema é uma estrutura de tabelas, que listam todos os vértices e arestas do problema. Seguindo a abordagem minimalista, foi configurada para aparecer somente quando necessária, dividindo a tela com o mapa, conforme demonstra a Figura 24.

Figura 24 - Tabela de vértices e arestas

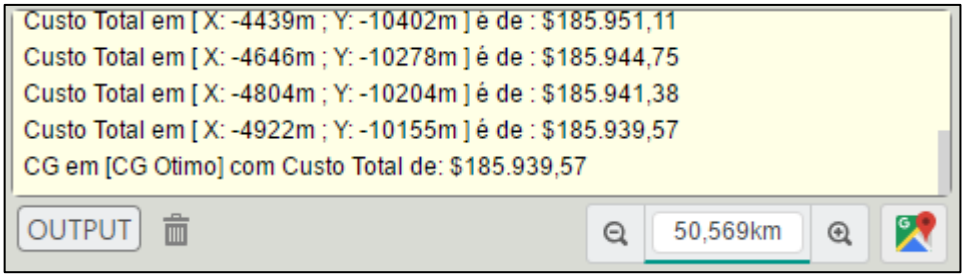


Fonte: Autor (2017)

Seu tamanho horizontal é ajustável para não prejudicar a visualização das informações em problemas de maior escala, assim como as colunas que cada tabela utiliza também são configuráveis de acordo com a aplicação do sistema. Por exemplo, se o usuário desejar resolver um problema de localização por centroide, as colunas da tabela referentes à problema de transporte podem ficar ocultas. Percebe-se que essa abordagem minimalista reaproveita toda a estrutura da interface gráfica para as diversas funcionalidades do sistema, permitindo que uma mesma modelagem do mapa possa ser usada para executar diferentes algoritmos e resolver múltiplos problemas.

Na parte inferior da interface, conforme a Figura 25, foi colocado um campo de saída (*Output*), onde o sistema pode imprimir informações variadas, como informações relevantes durante a execução dos algoritmos e seus resultados finais. Seguindo a abordagem minimalista, este elemento pode ser ocultado da aplicação se não for utilizado, liberando espaço em tela para melhor visualização de outros elementos.

Figura 25 - Informações de saída

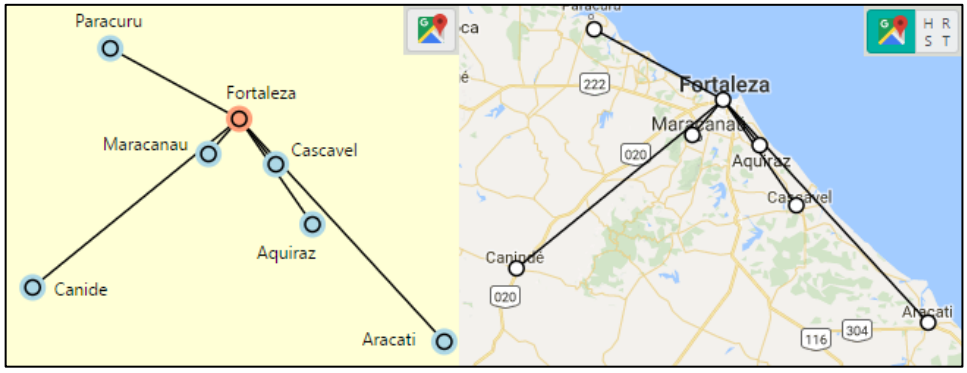


Fonte: Autor (2017)

3.3.2 Manipulação do mapa

Algumas funcionalidades envolvendo o mapa central foram adicionadas por questões ergonômicas e educacionais. Dentre elas, está a disponibilização de mapas reais, através da implementação da API do Google® Maps. O usuário pode, a qualquer momento, sobrepor um mapa real ao seu mapa modelado, que compartilha das mesmas coordenadas do grafo do problema. A Figura 26 demonstra essa funcionalidade.

Figura 26 - Modelando o problema com mapas reais



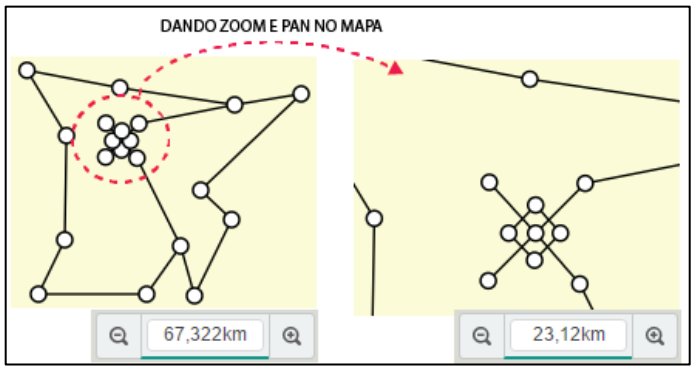
Fonte: Autor (2017)

Dessa forma, o usuário pode comparar rotas e localidades, definidas na modelagem do problema, com rotas e localidades reais, ou, ainda, modelar um problema real. Contudo, o sistema ainda possui algumas limitações ao se trabalhar com localidades de elevada latitude. Enquanto o mapa do modelo, sem a utilização do Google Maps, consiste de um simples plano cartesiano com coordenadas (X, Y), o posicionamento em mapas reais é realizado através de latitudes e longitudes. A representação em tela utilizada pelo Google Maps é uma projeção cilíndrica da superfície do planeta, denominada *Mercator*, apresentada por Gerardus Mercator em 1569. Como deseja-se, por motivos educacionais, utilizar um mapa teórico de coordenadas lineares que permite, a qualquer instante, comparações com mapas

reais, torna-se necessário uma correlação precisa entre as localidades nos dois mapas, bem como uma representação visual consistente. O kLog ainda possui a limitação de ser impreciso para elevadas latitudes, quando se usa a funcionalidade do Google Maps.

Dependendo das dimensões do problema sendo modelado, o espaço disponível em tela pode não ser suficiente para visualização adequada dos elementos do mapa. Por exemplo, certo problema de localização pode ter duas etapas, onde na primeira etapa localiza-se um ponto em uma escala onde várias cidades, representadas por vértices, são visíveis, enquanto em uma segunda etapa a localização passa a ter precisão a nível de bairro ou rua. Assim, torna-se interessante, para a utilização do sistema, a funcionalidade de enfoque ajustável (*zoom*), que altera a escala. Além do *zoom*, elaborou-se um mecanismo de translação (*pan*) da porção visível do mapa. Através dessas duas funcionalidades, o usuário pode ter livre navegação e visualização do mapa. O *zoom* e o *pan* podem ser alterados utilizando diferentes ferramentas, como o botão de rolar do mouse ou alteração direta da escala. A Figura 27 demonstra esta alteração.

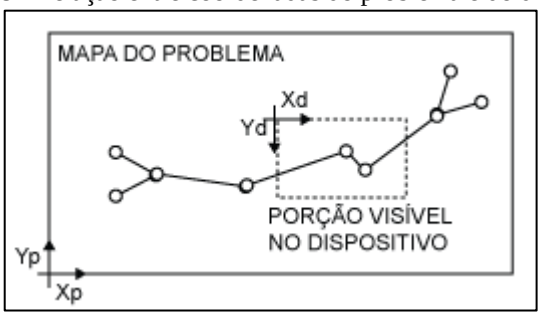
Figura 27 - Demonstração de zoom



Fonte: Autor (2017)

Para implementar essas funcionalidades, é preciso criar um relacionamento entre as coordenadas do modelo, em metros, e as coordenadas dos elementos na interface, em pixels. A visualização em tela do mapa depende do tamanho deste elemento na interface, que é determinado pelo tamanho do dispositivo, da janela do *browser*, e de como o usuário ajustou os elementos da interface. Essa visualização é apenas uma porção do mapa modelado, que tem dimensões infinitas já que se trata de um plano cartesiano livre, onde um vértice pode ser adicionado em qualquer coordenada. A Figura 28 demonstra essa relação, onde se pode perceber que o eixo das ordenadas do dispositivo tem sentido oposto ao do mapa. Isso se deve a um comportamento comum em dispositivos digitais, que considera os valores verticais das coordenadas da tela, em pixels, como crescentes para baixo.

Figura 28 - Relação entre coordenadas do problema e do dispositivo



Fonte: Autor (2017)

Já as Equações 9 e 10 representam as relações entre essas coordenadas, utilizadas na programação do sistema. Assim, sempre que um usuário adiciona um novo vértice em alguma posição (Xd, Yd) do dispositivo, é possível determinar qual a posição (Xp, Yp) correspondente no mapa do problema. Estas equações dependem de variáveis que controlam o *zoom* e o *pan*, já que são com elas que o usuário manipula a porção visível, no dispositivo, do mapa do problema.

$$Xp = (Xd - Xdc) * FatorZoom + Xpan \quad (9)$$

$$Yp = (Ydc - Yd) * FatorZoom + Ypan \quad (10)$$

Onde:

Xp *e* Yp são coordenadas do problema, em metros;
Xd *e* Yd são coordenadas do dispositivo, em pixels;
Xdc *e* Ydc são coordenadas, em pixels, do centro da porção visível do mapa, no dispositivo;
$FatorZoom$ é a variável que controla o *zoom*;
$Xpan$ *e* $Ypan$ são coordenadas de referência para controlar a translação da porção visível do mapa, no dispositivo;

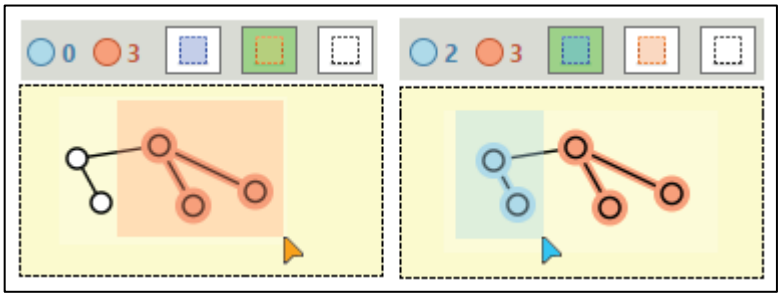
3.3.2.1 Seleção de elementos

A grande maioria das ações executadas no *software*, seja através de ferramentas ou execução de algoritmos, envolverá um ou mais elementos do grafo. Assim, percebeu-se a necessidade de especificar conjuntos de elementos sobre os quais essas ações serão aplicadas. Por exemplo, ao se determinar a localidade de um centro de distribuição para as lojas de uma empresa, através do método de centro de gravidade, é preciso definir o conjunto de vértices que representam as localidades dessas lojas. O sistema pode conter, no grafo, vários outros vértices que não são de interesse dessa ação específica. Ainda exemplificando, pode-se

imaginar a necessidade de mover alguns vértices específicos do grafo, enquanto os outros não se movem, ou alterar algumas informações em certo conjunto de arestas.

Percebendo-se a necessidade de especificar conjuntos de elementos, criou-se as funcionalidades de seleção. Assim, é possível selecionar elementos específicos, formando conjuntos a serem considerados nas ações e algoritmos. Essas seleções podem ser feitas através de cliques simples, ou ferramentas próprias para seleção. Conforme demonstra a Figura 29, foram associadas cores à essas seleções que, de forma ergonômica, indicam quais elementos pertencem ao conjunto de elementos selecionados. A cor azul e a cor vermelha determinam conjuntos diferentes.

Figura 29 – Demonstração das funcionalidades de seleção

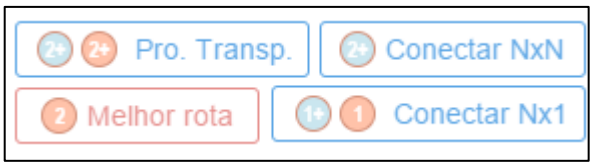


Fonte: Autor (2017)

Como pode ser observado na Figura 29, existe um elemento na tela, próximo às ferramentas de seleção, que indica as quantidades de elementos selecionados em cada conjunto. De forma ergonômica, uma indicação semelhante aparece em alguns botões, indicando como a ação daquele botão se atuará nas seleções. Em algumas ações é útil a utilização de mais de um conjunto de seleção.

Para exemplificar, a Figura 30 apresenta alguns botões. No primeiro botão, de problema de transporte, é preciso especificar, através das seleções, dois ou mais vértices, selecionados de azul, que determinam as origens, e dois ou mais vértices, selecionados de vermelho, que determinam os destinos. Através do segundo botão, dois ou mais vértices, selecionados de azul, serão interconectados. O terceiro botão calcula a melhor rota entre exatamente dois vértices, selecionados de vermelho. Por fim, o quarto botão da Figura 30 conecta todos os vértices selecionados de azul a exatamente um vértice selecionado de vermelho.

Figura 30 – Exemplo de ações e suas dependências dos conjuntos de seleção



Fonte: Autor (2017)

Percebe-se, então, como essa sistemática de seleção é útil para, de forma ergonômica, utilizar uma mesma estrutura para executar ações variadas em elementos específicos.

3.3.3 Salvar e carregar problemas

Contribuindo para o objetivo educacional do *software*, optou-se por viabilizar o armazenamento das informações de um problema modelado para posterior utilização. Essa funcionalidade pode ser útil em diversas situações, como problemas complexos que demoram a serem construídos, ou na reutilização do problema, por um aluno, para eventual apresentação, ou, ainda, a criação, por parte de um professor, de problemas públicos para serem utilizados por alunos.

Salvar informações em arquivos é algo trivial no universo digital, mas a utilização da *Web* como repositório é uma tendência que foi adotada neste sistema. Assim, ao salvar ou carregar um problema salvo, o sistema faz uso de um serviço online, o *Firebase*, dispensando os usuários da necessidade de utilização de arquivos e mídias digitais de gravação. A interface utilizada está demonstrada na Figura 31.

Figura 31 – Interface utilizada para armazenamento de informações

Fonte: Autor (2017)

Por esses elementos da interface, o usuário realiza uma autenticação no sistema, que permite o acesso aos problemas salvos previamente. Estes problemas podem ser excluídos a qualquer momento. Também é possível carregar problemas públicos, configurados através do *Firebase*, que só podem ser alterados por quem administra o *software*, recurso com utilidade voltada aos professores.

3.4 Métodos de localização

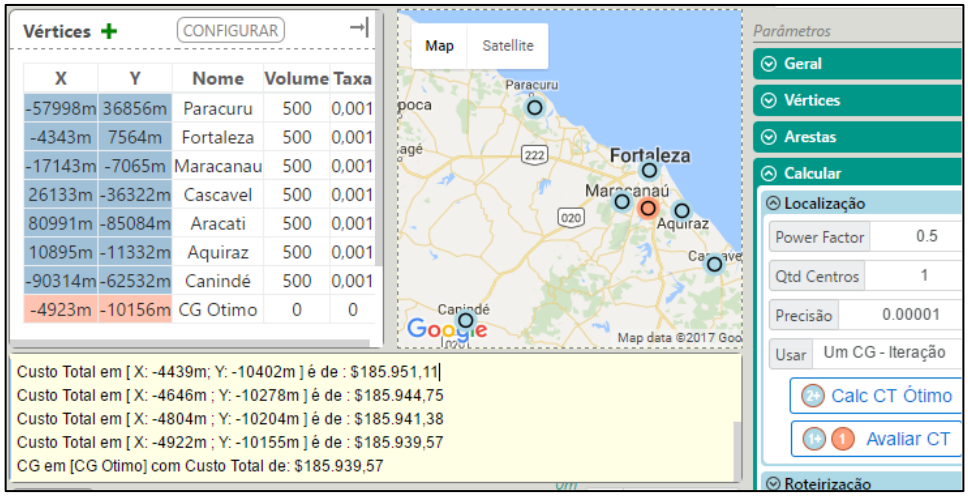
Os métodos de localização disponibilizados através deste desenvolvimento foram: Centro de Gravidade, Múltiplo Centro de Gravidade e *p*-Mediana.

3.4.1 Localização única

Utilizando a estrutura genérica de mapa, grafos e ferramentas, é possível associar diferentes algoritmos, objetivando a resolução de problemas de localização ou roteirização. O primeiro a ser implementado foi de localização de instalação única, utilizando o Método do Centro de Gravidade, conforme Ballou (2006, p.247), previamente apresentado.

Foi realizada a implementação completa do algoritmo utilizando *TypeScript*, onde parte do código é responsável pela interação com a interface. Como parâmetros de entrada do problema, têm-se um conjunto de vértices selecionados de azul, valores de volume e taxa configurados na tabela de vértices, e parâmetros gerais do problema, definidos na direita da interface, conforme demonstra a Figura 32.

Figura 32 - Exemplo de problema de localização



Fonte: Autor (2017)

O algoritmo é iterativo e tem como critério de parada a precisão definida nos parâmetros. O resultado final do problema consiste de um novo vértice, selecionado de vermelho, adicionado automaticamente ao problema, na localização ótima encontrada pelo método centroide, além de informações, no *Output*, referentes ao passo a passo da resolução do problema e o valor do custo final. Além de calcular o custo ótimo, o *software* permite avaliar um custo total para certo vértice específico, utilizando as mesmas taxas e volumes definidos na tabela de vértices, que pode ser comparado com o custo ótimo encontrado. Ainda utilizando o exemplo da Figura 32, pode-se procurar por alguma localidade viável, como terrenos vazios, e avaliar qual seria o custo total nessa localidade, conforme Figura 33.

Figura 33 – Exemplo de avaliação de custo total em local específico

Fonte: Autor (2017)

Esta funcionalidade é útil em situações que o ponto ótimo encontrado se encontra inviável por alguma restrição real. Por exemplo, pode não ser desejável construir um centro de distribuição em um eventual lago, no mar, em florestas, ou onde já existem construções. Assim, deve-se fazer uma busca manual de um local viável, que sempre terá custo total maior que o da localidade ótima, algo que pode ser constatado pela funcionalidade de avaliação de custo total.

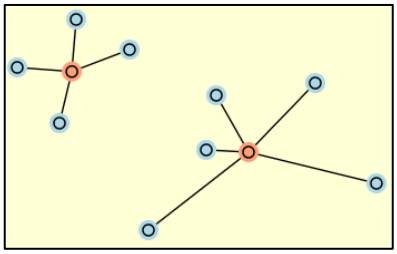
3.4.2 Localização múltipla

Dois métodos foram implementados para encontrar mais de uma localização. O primeiro deles, do múltiplo centro de gravidade, calcula uma quantidade *p* de centros de gravidade, utilizando o método centroide anterior, para *p* agrupamentos de vértices. As informações de entrada são os vértices selecionados de azul, a precisão de parada do método centroide, e a quantidade *p* desejada.

Estes agrupamentos são realizados de acordo com as proximidades entre os vértices, que são calculadas e ordenadas de forma decrescente. A cada iteração do algoritmo, implementado em *TypeScript*, os vértices mais próximos são agrupados, se já não tiverem sido agrupados anteriormente, sendo substituídos por um centro de gravidade. Novas distâncias são calculadas e ordenadas, repetindo o processo de *clustering* até que restem, somente, *p* vértices. Estes *p* vértices finais, que guardam em *Array* a informação de quais vértices originais lhe deram origem, acabam por determinar quais agrupamentos devem ser feitos.

Finalmente, todos os vértices de cada agrupamento são submetidos, pelo algoritmo, ao método centroide, que calcula cada uma das *p* localidades desejadas. Apresenta-se os vértices encontrados ao usuário, em vermelho, conforme ilustra a Figura 34.

Figura 34 – Exemplo visual de resultado de múltiplas localizações



Fonte: Autor (2017)

Outro método disponibilizado foi o da *p*-mediana que analisa, dentre todos os vértices selecionados de azul, quais *p* vértices devem ser escolhidos como instalações de origem. A resolução deste método passa pela modelagem e resolução de um problema de programação linear. Diferentemente dos métodos anteriores, este método não foi completamente implementado em *TypeScript*, necessitando de alguma solução externa para resolver problemas de otimização. Optou-se por utilizar uma biblioteca de código aberto, a jsLPSolver, em *JavaScript*, que possui esse propósito. Portanto, resta ao código desenvolvido em *TypeScript* a modelagem do problema de PL de acordo com os requisitos da biblioteca, além da interação com a interface para definir os parâmetros de entrada e apresentar os resultados.

A formulação do problema de programação linear para o método da *p*-mediana é aquela apresentada no capítulo 2. A Figura 35 apresenta como essa formulação, gerada dinamicamente pelos códigos do sistema, é passada para a biblioteca jsLPSolver. Como parâmetros de entrada, a biblioteca permite o uso de uma sintaxe própria, em forma de texto, que muito se assemelha a uma representação comum de um problema de PL.

Figura 35 – Exemplo de utilização da jsLPSolver

min: 241,20 **C1N2** 121,88 **C1N3** 380,37 **C1N4** 1400,00 **C2N1** [...] 2900,00 **C3N3** 539,23 **C4N1** 3096,42 **C4N2** 5368,6 **C4N3** 1700,00 **C4N4**	1 **C1N1** 1 **C2N1** 1 **C3N1** 1 **C4N1** = 1 [...] 1 **C1N4** 1 **C2N4** 1 **C3N4** 1 **C4N4** = 1 1 **C1N2** -1 **C1N1** <= 0 1 **C1N3** -1 **C1N1** <= 0 [...] 1 **C4N2** -1 **C4N4** <= 0 1 **C4N3** -1 **C4N4** <= 0 1 **C1N1** 1 **C2N2** 1 **C3N3** 1 **C4N4** = 2

Fonte: Autor (2017)

O problema é modelado definindo-se variáveis binárias de alocação, que vão determinar se um vértice de destino será ou não conectado a um vértice de origem. Os vértices candidatos à origem são aqueles que possuem, na tabela de vértices, um custo fixo definido pelo usuário. Esse custo fixo de uma instalação também entra na modelagem da função objetivo. Cada variável binária, representada na forma C_iN_j, é criada usando índices aleatoriamente gerados para cada vértice, e representam a alocação do vértice $\boldsymbol{i}$ como origem do vértice $\boldsymbol{j}$. Ao final da execução do algoritmo, a solução ótima é apresentada em tela, de forma similar à solução da Figura 34. O custo total é apresentado no *Output*.

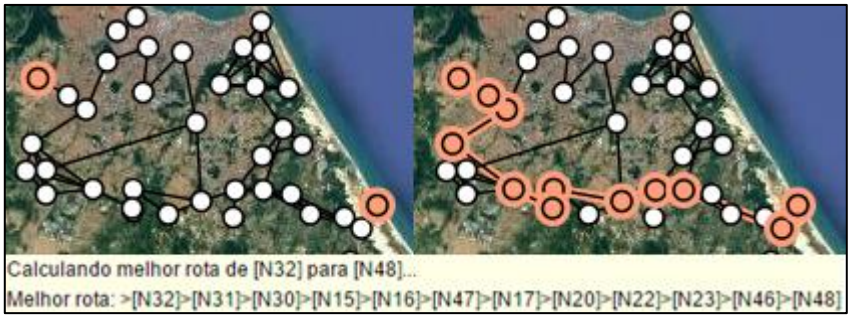
3.5 Métodos de roteirização

Os métodos de roteirização disponibilizados através deste desenvolvimento foram: Melhor Rota, Problema do Transporte, Método de Varredura e Método das Economia.

3.5.1 Melhor rota

O primeiro problema de roteirização implementado consiste em encontrar a melhor rota entre dois vértices do grafo. A solução deste problema só é possível se existir pelo menos uma rota possível entre estes dois vértices. O algoritmo implementado em *TypeScript* é o de Dijkistra, conforme demonstrado no capítulo 2.

Figura 36 – Exemplo genérico de cálculo da melhor rota



Fonte: Autor (2017)

A Figura 36 apresenta um exemplo dessa funcionalidade, onde são definidos, como parâmetros de entrada, os dois vértices de origem e destino selecionados de vermelho. Como resultado, a melhor rota é adicionada ao conjunto de seleção vermelho. Além disso, o Output apresenta algumas informações úteis relacionadas à solução, como a sequência de vértices (por nomes dos vértices, definidos na tabela) da rota encontrada. Neste exemplo, o custo considerado em cada aresta é somente a distância entre os vértices conectados. Foi adicionada uma funcionalidade de cálculo do total de distância de quaisquer arestas selecionadas, permitindo ao usuário calcular a distância total da solução encontrada.

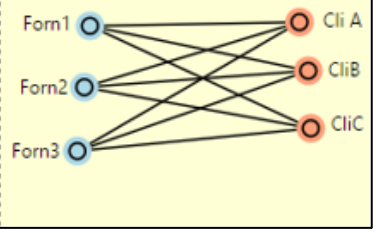
Em outros problemas, pode-se definir um custo diferenciado para arestas específicas. O algoritmo de Dijkistra permite considerar qualquer variável como custo para percorrer uma aresta. Assim, é possível modelar situações mais realistas, considerando estradas desgastadas ou pedágios, e não somente as distâncias cartesianas, como no exemplo da Figura 36. A definição desses custos pode ser feita na tabela de arestas do sistema.

3.5.2 Problema do transporte

Neste problema, é preciso definir quantidades transportadas entre origens, com ofertas, e destinos, com demandas, buscando minimizar o custo total. Entre cada origem e destino, existe uma aresta. Assim, vários vértices e arestas, e suas propriedades, são os parâmetros de entrada do problema, conforme Figura 37.

Figura 37 – Vértices e seus parâmetros em problema exemplo

X	Y	Nome	(PT) Qtd
-191085m	17892m	Forn 1	1200
-196569m	-41207m	Forn 2	1000
-203271m	-102743m	Forn 3	800
11191m	21547m	Cli A	900
19720m	-25366m	Cli B	700
22157m	-80809m	Cli C	1100



Fonte: Autor (2017)

Na tabela de vértices encontrada na Figura 37, pode-se perceber que a informação mais relevante para o problema do transporte encontra-se na última coluna. Essa é a quantidade, de demanda ou de oferta, que será considerada na resolução do problema, para cada vértice. A natureza da quantidade, se oferta ou demanda, não é definida na tabela, mas através dos conjuntos de seleção. No algoritmo do problema, os vértices selecionados de azul serão tratados como origens e os vermelhos como destinos. Na tabela de arestas, conforme Figura 38, são definidas as informações relevantes entre cada origem e destino.

Figura 38 – Arestas e seus parâmetros em problema de exemplo

De	Para	$/und/dis	$/und	Qtd Otima
Forn 1	Cli A	0	1	900
Forn 1	Cli B	0	2	0
Forn 1	Cli C	0	3	300
Forn 2	Cli A	0	2	0
Forn 2	Cli B	0	1	700
Forn 2	Cli C	0	74	0
Forn 3	Cli A	0	3	0
Forn 3	Cli B	0	2	0
Forn 3	Cli C	0	1	800

Fonte: Autor (2017)

A identificação da aresta desejada pode ser feita através dos nomes dos vértices envolvidos, ou através da seleção de uma ou mais arestas, que serão destacadas na tabela conforme sua cor de seleção. A última coluna (Quantidade Ótima) representa o resultado final do problema do transporte, sendo a quantidade ótima a ser transportada de cada origem para cada destino. O custo final é apresentado no *Output*.

Para resolver o problema do transporte, é preciso modelar um problema de programação linear, conforme os parâmetros definidos pelo usuário na Figura 37 e na Figura 38. A problema de PL utilizado encontra-se no capitulo 2. Assim como no método da *p*-mediana, foi utilizada a biblioteca jsLPSolver.

Na Figura 38, o usuário pode definir qual o custo unitário de transporte em uma rota origem/destino específica. Este é o valor definido na coluna *$/und*. Percebe-se, porém, que pouco importa, para o problema nesta configuração, a posição geográfica de cada vértice, pois a quantidade transportada e o custo unitário já definem o custo total da aresta. Esse é o motivo de ser possível dispor, conforme a Figura 37, os vértices em qualquer posição didática, pouco importando sua localização para a modelagem e solução do problema.

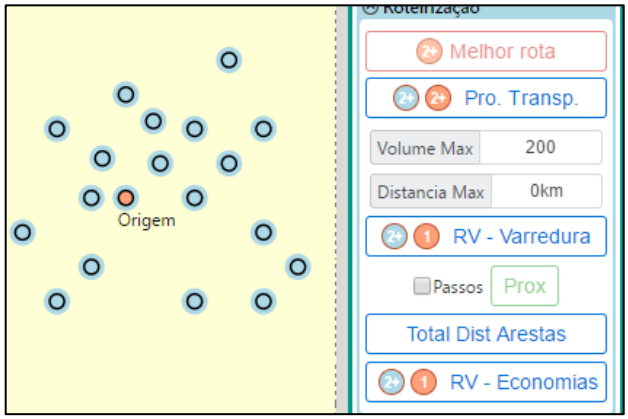
Contudo, em situações mais realistas, as distâncias entre os vértices podem ser consideradas. Basta recalcular os coeficientes das variáveis da função objetivo do modelo de PL adequadamente, antes de resolvê-lo. Além de um custo unitário fixo em cada rota, pode-se considerar um custo unitário por metro, proporcional à distância entre os vértices. É dessa forma que o valor *$/und/dis*, da Figura 38, entra na modelagem do problema.

Após resolver o problema de PL acima, o jsLPSolver fornece o valor de Z da função objetivo, e de cada X_{ij} ótimo, que representam as quantidades transportadas. Todas essas informações são apropriadamente apresentadas na interface.

3.5.3 Método da varredura

Como parâmetros de entrada deste problema têm-se vértices de seleção azul, determinando os destinos, e um único vértice vermelho, como origem. Além disso, o usuário pode opcionalmente definir restrições de volume máximo e distância máxima percorrida que não podem ser ultrapassados em cada roteiro. Uma demonstração desses elementos da interface encontra-se na Figura 39.

Figura 39 – Interface de configuração do método de varredura

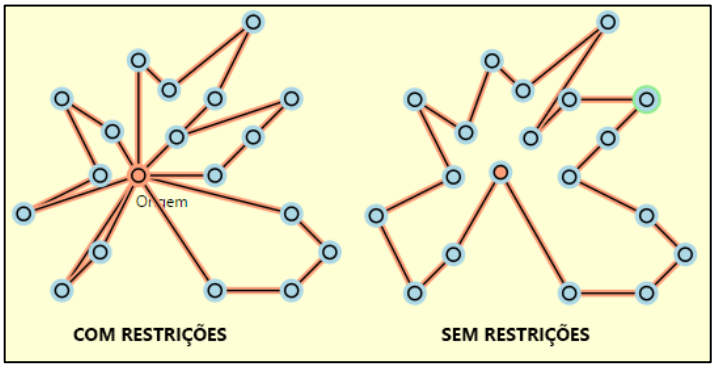


Fonte: Autor (2017)

O algoritmo, implementado em *TypeScript*, busca implementar o comportamento do método, apresentado no capitulo 2, de “varrer”, em sentido horário ou anti-horário, todos os vértices, conectando-os, tomando como centro de rotação a origem. Para tanto, o algoritmo calcula, para cada vértice de destino, o valor da função $Math.atan2(X_d - X_o\ ,\ Y_d - Y_o)$, onde $(X_o\ ,\ Y_o)\ e\ (X_d\ ,\ Y_d)$ representam as coordenadas da origem e do destino, respectivamente. Esse método permite definir a inclinação da reta que passa entre cada combinação origem/destino. Ordenando os valores encontrados em ordem crescente ou decrescente, permite ao algoritmo simular o comportamento proposto de “varredura”.

Assim, segue-se a ordem de vértices a serem “varridos”, conectando-os à medida que as restrições de volume e distância são respeitadas. Quando uma restrição for ultrapassada, o algoritmo conecta o vértice à origem, iniciando uma nova rota. Se não existem restrições, o método da varredura representa uma heurística para o método do caixeiro viajante, calculando uma rota única, saindo e voltando para a origem, conforme Figura 40.

Figura 40 – Resultados exemplos do método de varredura

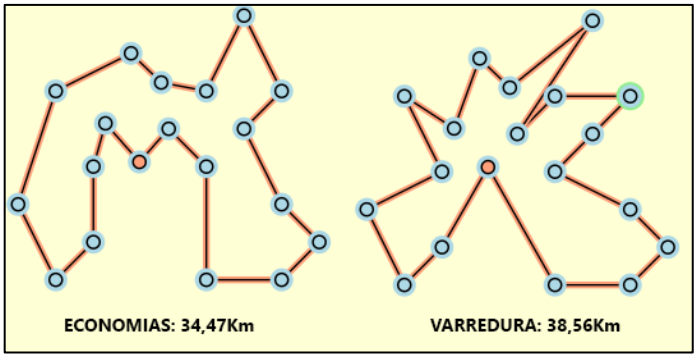


Fonte: Autor (2017)

3.5.4 Método das economias

No sistema, existem semelhanças na utilização do método das economias e do método das varreduras. Os parâmetros de entrada são configurados da mesma forma, e os resultados, também, apresentados de forma similar, dependendo das restrições de volume máximo e distância máxima para cada rota. Contudo, o método de Clark e Wright é bem mais eficiente, sendo uma das heurísticas mais utilizadas para resolução do problema do caixeiro viajante. A Figura 41 ilustra essa eficiência, comparando o resultado de roteamentos sem restrições utilizando o método das economias, com distância total quase 10,61% menor que a solução da varredura.

Figura 41 – Comparativo de distância total em exemplos de roteirização

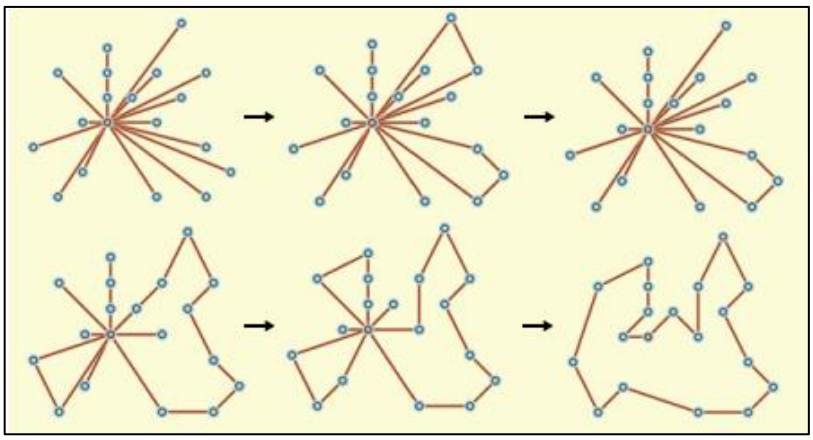


Fonte: Autor (2017)

O algoritmo de resolução foi implementado em *TypeScript*, conforme considerações do capitulo 2. Em resumo, o algoritmo cria uma rota inicial para cada vértice de destino (azul), contendo somente o respectivo vértice e a origem (vermelho). Em seguida, calcula-se, para todas as combinações 2 a 2 de vértices de destino, a suposta economia que se teria ao agrega-los em uma mesma rota. Essas economias formam uma lista, ordenada em forma decrescente. O algoritmo então percorre essa lista, agrupando os vértices, quando possível, segundo algumas restrições do método e as restrições de rota configuradas pelo usuário.

Durante o desenvolvimento, percebeu-se ser possível demonstrar visualmente o funcionamento de cada passo do algoritmo, para que o usuário constate os mecanismos do método durante a formação das novas rotas. Essa funcionalidade está alinhada com os requisitos educacionais do sistema. A Figura 42 ilustra seu funcionamento.

Figura 42 – Demonstrativo passo a passo do método das economias



Fonte: Autor (2017)

Pela Figura 42, que seleciona alguns passos de exemplo, pode-se perceber que, inicialmente, todos os vértices formam uma mesma rota, em linha reta, com a origem. Aos poucos, essas rotas vão sendo removidas, dando lugar à rotas com mais vértices, até que o método encontre a solução final. O usuário pode executar esse passo a passo utilizando o menu lateral, ou pode executar o algoritmo de forma direta, apresentando diretamente o resultado.

3.6 Verificações

As funcionalidades logísticas apresentadas anteriormente passaram por testes de verificação para atestar seu funcionamento com certo grau de confiabilidade. Eventualmente, alguns erros foram encontrados, levando a correções na programação dos algoritmos ou da lógica da interface. A metodologia utilizada foi de comparação com sistemas estáveis, que produzem resultados confiáveis, além da aplicação de problemas de solução conhecida. A partir deste ponto, o *software* em desenvolvimento será referenciado como **kLog**, para facilitar a leitura do texto. Ao final das verificações, são discutidos os resultados e considerações finais.

3.6.1 Métodos de localização

Para os métodos de localização, problemas solucionados do livro-texto do Ballou (2006) foram utilizados, assim como a comparação com o LogWare®, por ser um *software* reconhecido na área educacional do ensino de logística. O programa, na sua versão 6.0, é distribuído com problemas de exemplo que utilizam o método do centro de gravidade. A Figura 43 apresenta a interface de configuração dos parâmetros de entrada de um problema.

Figura 43 – Parâmetros de entrada de um problema exemplo do LogWare®

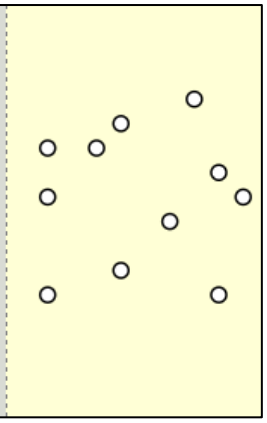
Point no.	Point label	X coor-dinate	Y coor-dinate	Volume	Transport rate
1	M1	2	1	3000000	0,002
2	M2	5	2	5000000	0,0015
3	M3	9	1	17000000	0,002
4	M4	7	4	12000000	0,0013
5	M5	2	5	9000000	0,0015
6	M6	10	5	10000000	0,0012
7	M7	2	7	24000000	0,002
8	M8	4	7	14000000	0,0014
9	M9	5	8	23000000	0,0024
10	M10	8	9	30000000	0,0011
11	P11	9	6	147000000	0,0005

Fonte: *Software* LogWare® (2017)

Além dos parâmetros da Figura 43, o LogWare® utiliza o *Map Scaling Factor*, um multiplicador utilizado para ajustar as coordenadas de entrada à escala desejada, e um *Power Factor*, que ajusta a linearidade das distâncias. O primeiro parâmetro não é considerado pelo kLog, pois suas coordenadas de entrada utilizam unidades que podem ser especificadas livremente pelo o usuário (como km, m ou *yard*), tornando o multiplicador desnecessário. Já o *Power Factor* é um parâmetro configurável no kLog também. O valor de 0,5 considera uma linha reta entre as distâncias. Valores maiores são usados em algumas modelagens, como meio de consideração das irregularidades das vias do problema, como estradas. Neste exemplo, o P*ower Factor* considerado é de 0,5. A Figura 44 demonstra a modelagem equivalente no kLog.

Figura 44 – Parâmetros de entrada de um problema exemplo no kLog

X	Y	Nome	Volume	Taxa
2km	1km	N1	3000000	0,002
5km	2km	N2	5000000	0,0015
9km	1km	N3	17000000	0,002
7km	4km	N4	12000000	0,0013
2km	5km	N5	9000000	0,0015
10km	5km	N6	10000000	0,0012
2km	7km	N7	24000000	0,002
4km	7km	N8	14000000	0,0014
5km	8km	N9	23000000	0,0024
8km	9km	N10	30000000	0,0011
9km	6km	N11	147000000	0,0005



Fonte: Autor (2017)

Durante a resolução do método do centroide, os dois *softwares* também apresentam algumas diferenças. O LogWare® pergunta ao usuário, a cada iteração do algoritmo, se este deseja continuar sua execução, até que esteja satisfeito com a precisão do resultado. Já o kLog recebe esta precisão como um parâmetro de entrada, sendo este o critério

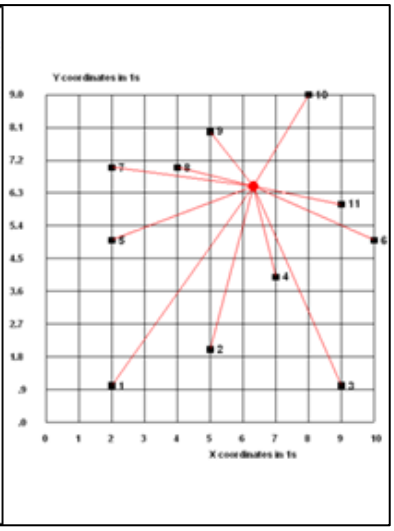
de parada para a execução do algoritmo. Para o problema exemplo aqui tratado, a precisão foi definida como 0,00000001.

Ao se comparar a Figura 45 com a Figura 46, pode-se perceber como os dois *softwares* apresentam os resultados de forma diferente, mas com valores satisfatoriamente próximos, mesmo neste problema com volumes elevados. As coordenadas do centro de gravidade convergem para um ponto comum, ao passo que o custo total, que se encontra em valores bilionários, difere em poucas unidades já após alguns ciclos de iteração.

Figura 45 – Resultados finais de um problema exemplo - cog - LogWare®

Title: Example

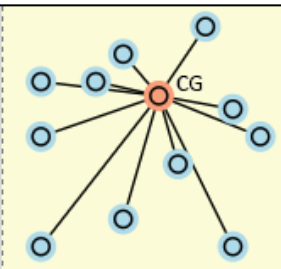
Iteration	X coordinate	Y coordinate	Cost
0	6.252	5.969	1,109,391,857.82
1	6.360	6.306	1,101,223,729.66
2	6.370	6.409	1,100,482,096.92
3	6.358	6.444	1,100,374,978.18
4	6.344	6.459	1,100,339,059.40
5	6.332	6.467	1,100,321,362.17
6	6.323	6.472	1,100,311,985.23
7	6.317	6.475	1,100,306,963.78
8	6.312	6.477	1,100,304,270.13
9	6.308	6.479	1,100,302,824.53
10	6.306	6.480	1,100,302,048.55
11	6.304	6.481	1,100,301,631.94
12	6.302	6.482	1,100,301,408.25
13	6.301	6.483	1,100,301,288.14
14	6.300	6.483	1,100,301,223.63
15	6.300	6.483	1,100,301,188.99
16	6.300	6.483	1,100,301,170.39
17	6.299	6.484	1,100,301,160.40



Fonte: *Software* LogWare® (2017)

Figura 46 – Resultados finais de um problema exemplo - cog - kLog

4km	7km	N8	14000000	0,0014
5km	8km	N9	23000000	0,0024
8km	9km	N10	30000000	0,0011
9km	6km	N11	147000000	0,0005
6298m	6484m	CG Otimo	0	0



OUTPUT

Custo Total em [X: 6298.42m ; Y: 6484.04m] é de : 1.100.301.126,49
Custo Total em [X: 6298m ; Y: 6484m] é de : 1.100.301.123,99
CG em [CG Otimo] com Custo Total de: 1.100.301.123,99

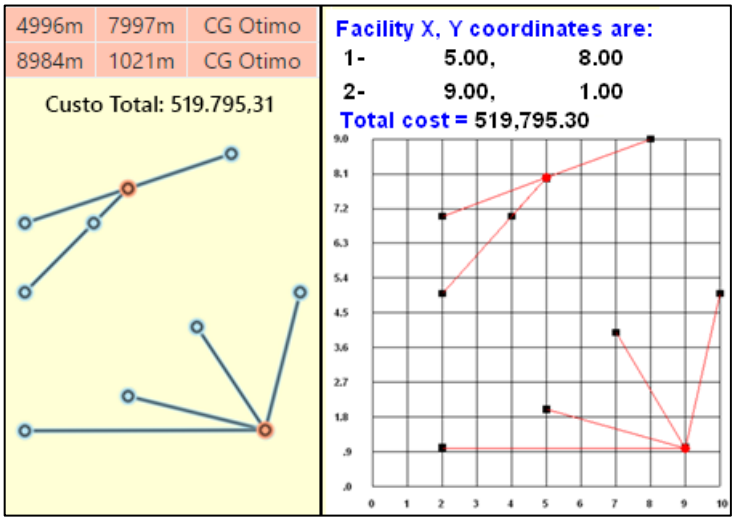
Fonte: Autor (2017)

Outra diferença entre os dois sistemas está nas possibilidades de ação do usuário após a resolução do problema. No LogWare®, o usuário fica limitado à visualização dos valores e coordenadas finais. No kLog, pode-se reaproveitar estas coordenadas em ações subsequentes, como busca de um centro de gravidade viável. Ainda exemplificando, pode-se imaginar uma modelagem, onde várias localidades dentro de diferentes cidades são

consideradas, calculando-se um centro de gravidade para cada cidade. Em seguida, pode-se utilizar estes centros para modelagens em maior escala, a nível de estado ou região, utilizando os centros de gravidade representativos de cada cidade. Como último exemplo de ação possível, pode-se transladar os vértices do resultado para alguma localidade real e realizar análises em mapas reais.

De forma semelhante, problemas exemplo foram comparados entre o kLog e LogWare®, para verificação do método do múltiplo centro de gravidade. A configuração dos parâmetros de entrada é semelhante ao exemplo anterior. A Figura 47 apresenta resultados de cada sistema, para um mesmo problema, selecionados para comparação.

Figura 47 – Resultados finais de um problema exemplo multicog - kLog



Fonte: Autor (2017)

É possível perceber que existem diferenças nos resultados encontrados pelos sistemas, apesar de mínimas. O custo total diferiu em torno de 0,01%, enquanto as coordenadas variaram entre 0,08% (4996m e 5km) e 2,06% (1021m e 1km), aproximadamente. As causas dessas diferenças não foram determinadas com confiança. Pode existir alguma falha na implementação do algoritmo do kLog, não identificada, ou as precisões dos valores são consideradas de forma diferente pelos sistemas. O kLog tem precisão de 1 metro para todas suas distâncias, e tem a precisão critério de parada do método centroide configurada pelo usuário. Os detalhes do algoritmo interno do LogWare® são desconhecidos, e ele pode estar efetuando arredondamentos. Contudo, como as diferenças encontradas em repetidos testes se mantiveram mínimas, a funcionalidade foi mantida no kLog, passível de melhorias futuras.

O método da *p*-mediana também possui parâmetros de entrada semelhantes aos anteriores, adicionando-se apenas a coluna de “Custo Fixo” à tabela de vértices do kLog. A verificação também foi realizada por comparação de resultados. Uma diferença a ser observada entre os dois sistemas, quando o LogWare® resolve problemas utilizando latitudes e longitudes, diz respeito a consideração da taxa. Enquanto o kLog considera uma taxa unitária por metro, o LogWare® considera uma taxa unitária por milha. Assim, a comparação neste caso só se torna possível ao considerar uma conversão de 1609,34 metros por milha. Isso posto, a Figura 48 e Figura 49 apresentam um problema exemplo configurado no de forma equivalente nos dois sistemas.

Figura 48 – Configuração de um problema exemplo de p-mediana - kLog

Nome	Volume	Taxa	CustoFixo
Los Angeles	40000	0.000005405943	2500000
New York	50000	0.000005405943	3700000
Atlanta	170000	0.000005405943	1400000
Minneapolis	140000	0.000005405943	0
Boston	30000	0.000005405943	3100000
Seattle	20000	0.000005405943	1250000
Phoenix	230000	0.000005405943	1100000
Denver	30000	0.000005405943	1500000
Memphis	90000	0.000005405943	0
Baltimore	120000	0.000005405943	0
Chicago	240000	0.000005405943	2900000
Cincinnati	100000	0.000005405943	1700000

Fonte: Autor (2017)

Figura 49 – Configuração de um problema exemplo de p-mediana – LogWare®

Number of facilities to locate: 2 — Latitude/Longitude coordinates

LOCATION DATA

Point no.	Point label	Lati-tude	Longi-tude	Volume	Transport rate	Fixed cost	Candidate sites
1	Boston MA	42.36	71.06	30000	0.0087	3100000	X
2	New York NY	40.72	74.00	50000	0.0087	3700000	X
3	Atlanta GA	33.81	84.63	170000	0.0087	1400000	X
4	Baltimore MD	39.23	76.53	120000	0.0087	0	
5	Cincinnati OH	39.14	84.51	100000	0.0087	1700000	X
6	Memphis TN	35.11	89.96	90000	0.0087	0	
7	Chicago IL	41.84	87.64	240000	0.0087	2900000	X
8	Minneapolis MN	44.93	93.20	140000	0.0087	0	
9	Phoenix AZS	33.50	112.07	230000	0.0087	1100000	X
10	Denver CO	39.77	105.00	300000	0.0087	1500000	X
11	Los Angeles CA	34.08	118.37	40000	0.0087	2500000	X
12	Seattle WA	47.53	122.32	20000	0.0087	1250000	X

Fonte: Autor (2017)

Após a resolução do problema, os resultados em cada sistema são demonstrados pelas Figura 50 e Figura 51.

Figura 50 – Resultados de um problema exemplo de p-mediana – kLog

Fonte: Autor (2017)

Figura 51 – Resultados de um problema exemplo de p-mediana – LogWare®

Fonte: Autor (2017)

Em ambos os resultados, Phoenix e Cincinnati foram corretamente escolhidos como instalações de fornecimento para as demais. O agrupamento realizado também foi idêntico. Contudo, o custo final calculado diferiu em quase 17%. Essa diferença elevada não se deve à resolução do problema de PL, mas a outros dois fatores. O primeiro, menos relevante, foi o método de localização das cidades para formulação do problema equivalente no kLog. Enquanto o LogWare® definiu latitudes e longitudes específicas, a localização no kLog foi feita de forma visual pelo nome das cidades. A segunda fonte de erro é a limitação já comentada, no kLog, quando se utiliza o Google Maps em elevadas latitudes, como neste problema exemplo. Dado essas duas fontes de erro, as coordenadas utilizadas pelo algoritmo ficaram bem distorcidas em relação às coordenadas do LogWare®, mas não distorcidas o suficiente para alterar a alocação dos vértices, neste caso.

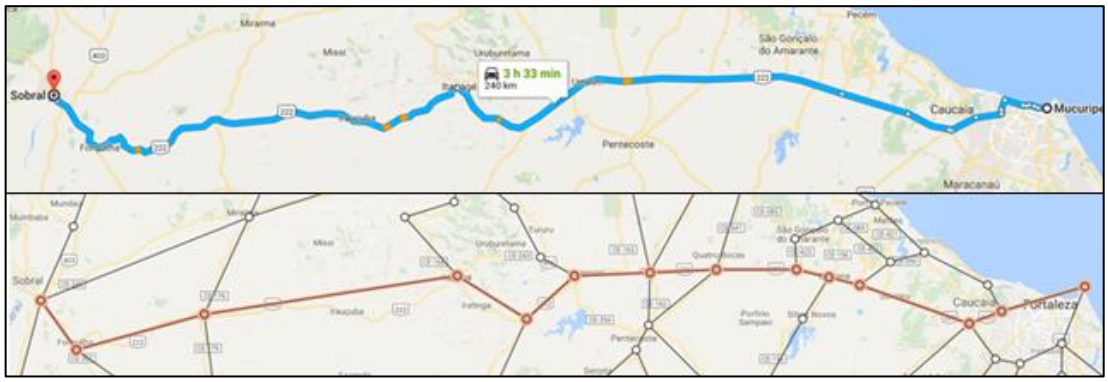
Contudo, outros testes comparativos realizados com menores latitudes e uma quantidade menor de vértices diminuíram consideravelmente as diferenças. Além disso, esse problema não é encontrado quando se trabalha, em ambos os sistemas, com coordenadas

lineares, não necessitando do Google Maps para localização visual. De qualquer forma, espera-se que, em versões futuras, o sistema possa remover essa limitação.

3.6.2 Métodos de roteirização

Afim de atestar a funcionalidade do método do caminho mínimo, desenvolvido com o algoritmo de Djikistra, foram feitas comparações manuais entre os resultados do kLog e de outros sistemas, como o LogWare® e o Google Maps, retornando resultados satisfatórios. A Figura 52 exemplifica um teste de verificação, onde as principais cidades da região Norte do Ceará foram modeladas no kLog, juntamente com as principais vias que as interligam. Pode-se perceber, pela Figura 52, a semelhança de roteiro no cálculo da melhor rota entre Sobral-CE e o porto do Mucuripe, em Fortaleza-CE.

Figura 52 – Comparação de rotas utilizando o kLog e o Google® Maps



Fonte: Autor (2017)

Vale ressaltar as imprecisões e riscos do método de verificação da Figura 52. Os mapas do Google Maps apresentam elevada continuidade e precisão. Não são somente as vias principais (modeladas no kLog) que podem ser utilizadas, mas quaisquer vias mapeadas pela Google. As vias possuem sinuosidades e elevações reais, consideradas nos algoritmos internos do Google Maps. Por outro lado, a modelagem do kLog é simplificada, limitando-se a vértices que representam as principais cidades, interligadas por arestas totalmente retas. Somente as principais vias estaduais e federais foram modeladas. Mesmo com estas limitações, a rota calculada pelo kLog é bem semelhante à do Google® Maps, contendo praticamente as mesmas cidades e vias utilizadas, entre várias possíveis no modelo.

A confiabilidade do kLog na resolução do problema do transporte está diretamente ligada à confiabilidade da biblioteca jsLPSolver, a solução escolhida para resolver problemas de programação linear utilizando *JavaScript*. Assim, resta ao kLog atestar

a qualidade do método através da correta modelagem do problema e do funcionamento adequado da interface gráfica, além da correta utilização da biblioteca.

A metodologia utilizada foi semelhante às verificações anteriores, através da inserção manual de vários problemas previamente resolvidos e comparação de resultados. Eventuais erros são corrigidos até se atingir estabilidade das funcionalidades.

O *software* utilizado como referência foi o Excel®, que é capaz de resolver, através do suplemento *Solver*, problemas de programação linear, como o problema do transporte. Problemas de referência, conforme o exemplo da Figura 53, foram elaborados, comparando-os com problemas idênticos no kLog.

Figura 53 – Resolução do problema do transporte utilizando Excel®

Custos Unitários

	Cli1	Cli2	Cli3
Fab1	5	4	7
Fab2	2	1	3

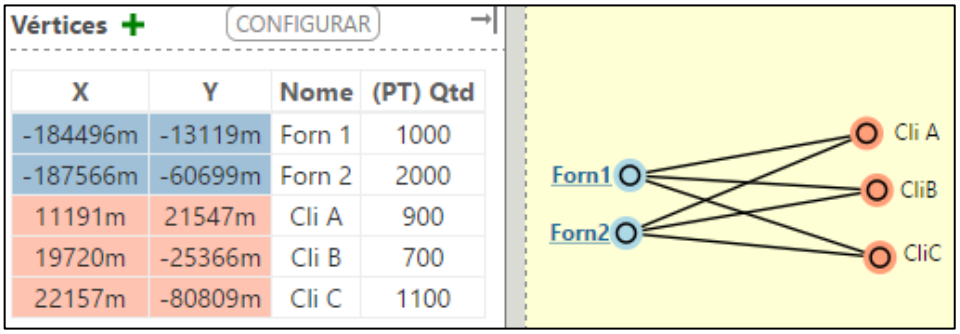
Quantidades transportadas

	Cli1	Cli2	Cli3	*Total*		**Oferta**
Fab1	700	0	0	700	<=	1000
Fab2	200	700	1100	2000	<=	2000
Total	900	700	1100			
	=	=	=			Custo Total
Demanda	900	700	1100			**7900**

Fonte: Autor (2017)

No problema da Figura 53, os custos unitários de cada rota entre fornecedores e clientes são inicialmente definidos. O *Solver* é configurado de forma a alterar os valores das quantidades transportadas até atingir o custo total mínimo de 7900. As restrições de oferta e demanda precisam ser respeitadas. Já no kLog, as configurações destes parâmetros, bem como os valores ótimos encontrados, podem ser constatadas pela Figura 54 e pela Figura 55.

Figura 54 – Vértices do problema de exemplo no kLog



Fonte: Autor (2017)

Na interface, o valor mínimo encontrado para o custo total, de 7900, é apresentando no *Output*.

Figura 55 – Arestas do problema de exemplo no kLog

Arestas

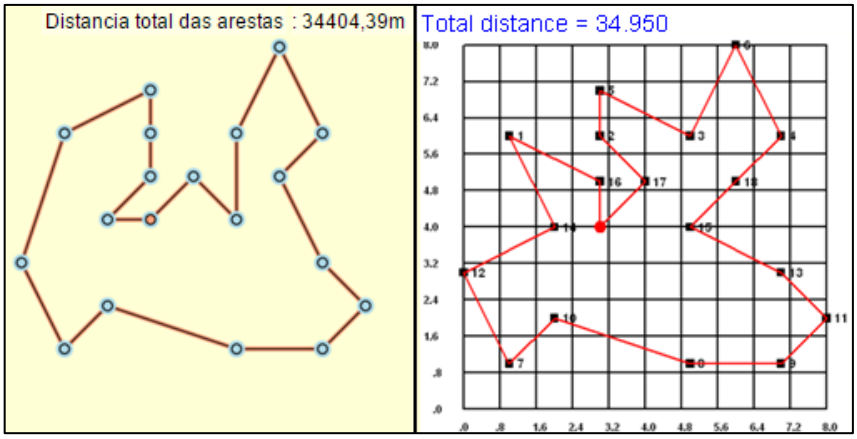
De	Para	$/und/dis	$/und	Qtd Otima
Forn 1	Cli A	0	5	700
Forn 1	Cli B	0	4	0
Forn 1	Cli C	0	7	0
Forn 2	Cli A	0	2	200
Forn 2	Cli B	0	1	700
Forn 2	Cli C	0	3	1100

Fonte: Autor (2017)

Parte da verificação do método de varredura se deu de forma manual e visual. Em qualquer problema modelado, pode-se determinar, visualmente, qual a ordem de conexões que serão feitas entre os vértices, dada a natureza simples do método. A outra parte da verificação diz respeito as restrições de volume total e distância total de cada roteiro, que foram respeitadas nos testes modelados.

De forma semelhante, a verificação das restrições nos testes com o método das economias foi feita através da análise dos resultados e constatação de rotas dentro dos limites configurados. Contudo, o roteiro definido por este método não pode ser analisado de forma manual, dado sua complexidade, precisando ser comparado com fontes confiáveis. O LogWare® utiliza o mesmo algoritmo em seu módulo ROUTESEQ. Porém, neste módulo, o LogWare® não aceita parâmetros de restrição, limitando-se a gerar um caminho que é solução heurística do problema do caixeiro viajante. Dado que as restrições já foram verificadas através de métodos manuais, limitou-se a comparar o resultado das rotas geradas pelo kLog, sem restrições, com aquelas geradas pelo LogWare®. A Figura 56 apresenta um exemplo dessa comparação.

Figura 56 – Problemas de teste do método das economias no kLog e no LogWare®



Fonte: Autor (2017)

Analisando este teste, pode-se perceber que existem algumas diferenças no roteiro, e o resultado no kLog é levemente melhor (rota aproximadamente 1,56% menor) que o resultado do LogWare® para o mesmo problema. Este é um resultado dentro da normalidade, assim como outros que foram encontrados. O método de Clark-Wright é uma heurística, não necessariamente encontrando o resultado ótimo, e possuindo algumas fontes de aleatoriedade. No exemplo da Figura 56, algumas combinações dos vértices podem gerar um mesmo valor de economia. A etapa do algoritmo responsável pelo ordenamento decrescente das economias pode, de forma aleatória, criar ordenamento diferentes quando encontra combinações de vértices com a mesma economia. Essa aleatoriedade pode acabar gerando um roteiro final diferente, mas com resultados semelhantes, como foi constatado neste teste. Esta aleatoriedade pode ser observada no kLog, ao se tentar resolver um mesmo problema diversas vezes. Eventualmente, outros roteiros serão gerados, com pequenas diferenças e distâncias totais semelhantes.

3.6.3 Discussão das verificações

Um método mais confiável de verificação dos algoritmos implementados no kLog seria a configuração de um ambiente automatizado utilizando centenas ou milhares de grafos como parâmetros de entrada, comparando-se as saídas do algoritmo desenvolvido com resultados previamente conhecidos e confiáveis. Contudo, este método não foi adotado pela dificuldade de sua operacionalização, não havendo, portanto, um maior controle preventivo de qualidade de *software*. A metodologia de verificação utilizada não evita o risco de eventuais erros ocorrerem durante a utilização dos usuários, exigindo ações corretivas em versões futuras.

Os vários testes de verificação realizados estão dentro das limitações de recursos deste trabalho. Mesmo assim, eles foram importantes para apresentar problemas que foram corrigidos durante o desenvolvimento. Algumas imperfeições encontradas ainda precisam ser corrigidas em versões futuras, mas as funcionalidades foram mantidas pois os resultados foram considerados estáveis e satisfatórios o suficiente para se manterem na versão inicial do sistema.

Dado o foco de utilização educacional do sistema, algumas imprecisões encontradas não são de grande impacto em sua qualidade, já que não há a pretensão do seu uso em situações reais críticas e profissionais, que envolvem centenas ou milhares de variáveis. Encontra-se maior relevância para a qualidade, nas verificações, ao se comparar a existência de recursos visuais e funcionalidades com foco didático, em relação a outros *softwares* que também resolvem os problemas modelados.

Outras ações interessantes de serem realizadas em futuras versões, indo além das precisões matemáticas dos métodos, são pesquisas de validação com usuários do sistema, capazes de investigar a percepção de alunos e professores quanto à utilidade lúdica do *software*, possibilitando a correção de problemas, melhoraria de limitações e desenvolvimento de novas funcionalidades.

4. CONCLUSÃO

Este trabalho teve por objetivo o desenvolvimento de um *software* educacional para apoiar o ensino de roteirização e localização em disciplinas de logística. A justificativa deste objetivo foi apresentada frente a relevância das tecnologias digitais nos processos de ensino.

Disponibilizou-se um embasamento teórico relacionado aos assuntos abordados, iniciando-se com a apresentação dos métodos de roteirização e localização que foram utilizados, seguindo-se da conceituação das tecnologias necessárias ao desenvolvimento do sistema. Alguns tópicos de engenharia de software foram explorados, já que alguns requisitos de qualidade, como interface ergonômica e usabilidade, norteiam as decisões de tecnologia e interface com o usuário.

Todos os quatro objetivos específicos deste trabalho foram alcançados durante o seu desenvolvimento.

Atingir o primeiro objetivo, de determinar quais práticas e tecnologias de desenvolvimento de software são as mais adequadas para o desenvolvimento deste trabalho, foi possível através de uma ponderação das opções de tecnologias disponíveis e de uma tomada de decisão alinhada com requisitos de qualidade que considerem o objetivo educacional do sistema. Buscou-se elevar a longevidade do mesmo, frente aos riscos de obsolescência que foram apresentados.

O segundo objetivo específico, de implementar algoritmos de roteirização e localização capazes de resolver problemáticas comumente tratadas no ensino destas temáticas, foi alcançado através da elaboração de algoritmos em *TypeScript*, com eventual utilização da biblioteca jsLPSolver, baseando-se em formulações detalhadas destes métodos fornecidas pelos autores referenciados no embasamento teórico.

O terceiro objetivo foi alcançado ao se desenvolver uma interface gráfica para o software, utilizando as tecnologias e diretrizes definidas no primeiro objetivo específico. O desenvolvimento dessa interface foi constantemente orientado pela utilização educacional do sistema, apresentando como elemento principal um mapa central no qual um grafo pode ser modificado utilizando-se diversas ferramentas desenvolvidas de forma ergonômica e lúdica. Utilizando-se estes elementos visuais, pode-se modelar problemas de roteirização e localização, resolvidos através dos algoritmos implementos no segundo objetivo, sendo os resultados devidamente apresentados na interface.

O quarto objetivo específico foi alcançado, que consistia em verificar os algoritmos implementados através da aplicação dos métodos de localização e roteirização na resolução de problemas com soluções previamente conhecidas. Para tanto, utilizou-se de variadas fontes de problemas, como aqueles que acompanham o LogWare®. As soluções obtidas com o sistema desenvolvido foram comparadas com estas fontes, e as vantagens e limitações do sistema foram ressaltadas. Por fim, foi ressaltado a importância da realização de métodos mais confiáveis de verificação e validação em futuros desenvolvimentos do sistema.

4.1 Recomendações para futuros trabalhos

A principal oportunidade de melhoria deste trabalho encontra-se no aproveitamento da interface e suas funcionalidades para a implementação de novos métodos, tornando o sistema capaz de resolver uma quantidade maior de problemas logísticos. A estrutura de grafo, as ferramentas de seleção, a tabela de vértices e de arestas são exemplos de elementos reutilizáveis em novos métodos e algoritmos.

A crescente utilização de dispositivos móveis no universo digital foi considerada na escolha das tecnologias, já que o *Angular* permite uma adaptação rápida do sistema à dispositivos móveis, utilizando tecnologias como o *Ionic*. Contudo, essa adequação deve ser desenvolvida, verificada e validada. Atualmente, o foco da versão inicial do sistema encontra-se em navegadores *Desktop*, em especial o Google® Chrome, onde foi desenvolvido.

Certamente existem problemas de compatibilidade com outros navegadores, dado o tamanho da programação do sistema e as peculiaridades existentes em cada navegador. Algumas funções e abordagens na programação devem ser incompatíveis com alguns navegadores, necessitando de investigação para encontrar um método que garanta a mesma funcionalidade nos principais *browsers*. Dado o custo de tempo para garantir alta compatibilidade, ficou fora do escopo deste trabalho.

O controle de qualidade do sistema e a verificação dos algoritmos dos métodos também estão limitados. Existem práticas bem mais interessantes que poderiam ser implementadas, como *Unit Testing*. Além disso, pesquisas de opinião poderiam ser realizadas com alunos e professores para melhor definição dos requisitos de qualidade e identificação de oportunidades de melhoria.

4.2 Considerações finais

Todo o sistema, incluindo código-fonte, encontra-se disponível na *Web*, na data de publicação deste trabalho. Para utilização do sistema, pode-se acessar o endereço https://aurirjr.github.io/kLog/ (o L precisa ser maiúsculo). Através do endereço https://github.com/aurirjr/kLog é possível a participação colaborativa, permitindo a identificação de erros, sugestão de melhorias, ou implementação de novas funcionalidades.

REFERÊNCIAS

ALVES, L. M. F. **Análise de *Softwares* Educacionais**. Disponível em < http://www.uel.br/seed/nte/analisedesoftwares.html > Acesso em: Junho 2017.

ANGULAR. ***Architecture overview***. 2016. Disponível em: < https://angular.io/docs/ts/latest/guide/architecture.html > Acesso em: Maio de 2016.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa Operacional para cursos de Engenharia.** Rio de Janeiro: Campus, 2007.

BALLOU, R. H. **Gerenciamento da Cadeia de Suprimentos/Logística Empresarial**. 5 ed. Porto Alegre: Bookman, 2006.

BASS, L; CLEMENTS, P.; KAZMAN, R. ***Software Architecture in Practice*** (2 ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

BATISTA, S. C. F; BARCELOS, G. T.; ELENA, C. **Avaliar é Preciso: o caso de softwares educacionais para Matemática no Ensino Médio**. Trabalho apresentado no I WorkComp Sul, Florianópolis, 2004.

BAX, M. P. **Introdução às linguagens de marcas.** Ciência da Informação, v. 30, n. 1, p. 32-38, 2001.

BOAVENTURA NETTO, P. O. **Grafos: Teoria, Modelos, Algoritmos**. 4. ed. São Paulo: Edgar Blucher, 2006.

BORTOLOSSI, H. J. **Criando conteúdos educacionais digitais interativos em matemática e estatística com o uso integrado de tecnologias:** GeoGebra, JavaView, HTML, CSS, MathML e JavaScript. Revista do Instituto GeoGebra Internacional de São Paulo. ISSN 2237-9657, v. 1, n. 1, 2012.

BOURQUE, P. et al. ***Guide to the software engineering body of knowledge (SWEBOK (R)):*** Version 3.0. IEEE Computer Society Press, 2014.

BOWERSOX, D J. et al. **Gestão logística da cadeia de suprimentos**. AMGH Editora, 2013.

BOWERSOX, D.J.; CLOSS, D.J. **Logística empresarial: o processo de integração da cadeia de suprimentos**. São Paulo: Atlas, 2001.

BRAY, T. ***The javascript object notation (json) data interchange format***. 2014.

BROOKS, D. R. ***An Introduction to HTML and JavaScript for Scientists and Engineers***. London: Springer-Verlag., 2007.

BUBECK, S. ***Applications Programming in Smaltalk-80*: *How to use Model-View-Controller***. Smalltalk-80 v2 5 (1992).

CARZANIGA, A. et al. ***A characterization framework for software deployment technologies***. Colorado State Univ Fort Collins Dept Of Computer Science, 1998.
CASSARO, A. **Sistemas de informações para tomada de decisões**. São Paulo: Pioneira, 1995.

CAIXETA FILHO, João Vicente. **A Modelagem de Perdas em Problemas de Transporte**. Revista Teoria e Evidência Econômica, v. 3, n. 06, 1995.

CAVANHA FILHO, A.O. **Logística: novos modelos**. Rio de Janeiro: Qualitymark, 2001.

CHAPMAN, N; CHAPMAN, J. **Digital Multimedia.** Wiley. p. 86. ISBN 0-471-98386-1, 1992.

CHRISTOFIDES, N. **Graph theory: an algorithmic approach**. Londres: Academic Press, 1975.

CLEMENTE, A.; FERNANDES, E. **Planejamento e projetos**. In: CLEMENTE A. (Org.) Projetos empresariais e públicos. São Paulo: Atlas, 1998.

COLLIS, J; HUSSEY, R; **Pesquisa em Administração: Um guia prático para alunos de graduação e pós-graduação**. 2. ed. Porto Alegre: Bookman, 2005.

CORMEM, T.H., et al. (2002) **Algoritmos: Teoria e Prática**. Editora Campus.

COSTA, Polyanna Possani da. **Teoria dos grafos e suas aplicações**. 2011.

COUNCIL OF SUPPLY CHAIN MANAGEMENT PROFESSIONALS. **Suply Chain Management Terms and Gloassary** Disponivel em: < http://cscmp.org/CSCMP/Educate/SCM_Definitions_and_Glossary_of_Terms/CSCMP/Educate/SCM_Definitions_and_Glossary_of_Terms.aspx >. Acesso em: Junho 2017.

CROSBY, P.B. **Qualidade sem lágrimas**. Rio de Janeiro: Livraria José Olimpo, 1992.

CRUZ, A. G. da; NERI, D. F. de M. **A inserção de tablets em escolas da rede pública estadual na cidade de Petrolina-PE: uma percepção dos educadores educandos**. Revista de Educação do Vale do São Francisco - Revasf, Petrolina, v. 4, n. 6, p.06-26, dez. 2014.

CUNHA, C.B. **Uma contribuição para o problema de roteirização de veículos com restrições operacionais**. São Paulo. Tese (Doutorado). Escola Politécnica Universidade São Paulo. Departamento de Engenharia de Transportes, Universidade de São Paulo, 1997.

CUNHA, C. B. (2000) **Aspectos práticos da aplicação de modelos de roteirização de veículos a problemas reais**. Transportes, v.8 , n.2, p.51-74.

DE CASTRO, M. A. S. et al. **Infra-estrutura de suporte à editoração de material didático utilizando multimídia.** Revista Brasileira de Informática na Educação, v. 1, n. 1, p. 61-70, 1997.

DE PÁDUA PAULA FILHO, W. **Engenharia de software**. LTC, 2003.

DEMING, W. E. ***Quality, productivity and competitive position*.** Boston: MIT Press, 1982.
DIAS, L. F. et al. **Uma Análise Preliminar de Projetos de Software Livre que Migraram para o GitHub.** Sociedade Brasileira de Computação–SBC, p. 65, 2016.

FERNANDES, J. **O que é um programa (Software)**, 2002. Disponível em: <http://www.cic.unb.br/~jhcf/MyBooks/iess/Software/oqueehsoftware.html>. Acesso em: junho 2017.

FITZSIMMONS, J.; FITZSIMMONS, M. **Administração de serviços: operações, estratégias e tecnologia da informação**. 4. ed. Porto Alegre –RS: Bookman, 2004.

FLANAGAN, D. **JavaScript: the definitive guide**. O'Reilly Media, Inc., 2006.

GARVIN, D. A. **Gerenciando a qualidade**: **a versão estratégica e competitiva**. Rio de Janeiro : Qualitymark, 1992

GASNIER, D.G. **A dinâmica dos estoques:** guia prático para planejamento, gestão de materiais e logística. São Paulo: IMAM, 2002. 316p.

GEORGES, M. R. R; SEYDELL, M. R. R. **Dificuldades no ensino da logística**. In: V CONVIBRA–Congresso Virtual Brasileiro de Administração. 2008.

GIACOMAZZO, G. F; FIUZA, P. J. **A implantação do *tablet* educacional na perspectiva dos professores**. Revista Tecnologias na Educação–Ano, 2014.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 4ed. São Paulo: Atlas, 2007.

GIL, A.C. **Métodos e técnicas de pesquisa social**. 4 ed. São Paulo: Atlas, 1994. 207p.

GIRAFFA, L. M. M. **Uma odisséia no ciberespaço**: O software educacional dos tutoriais aos mundos virtuais. Brazilian Journal of Computers in Education, v. 17, n. 01, p. 20, 2009.

GOMES, Carla Sofia de Assunção. **Problema do caixeiro viajante**. 2008. Dissertação de Mestrado. Universidade de Aveiro.

GRANNELL, C. ***The Essential Guide to CSS and HTML Web Design***. New York: APress, 2007.

GROSSMAN, D. **Angular 1 component based architecture**. Disponível em: <http://blog.grossman.io/angular-1-component-based-architecture-2/> Acesso em: Julho de 2016.

HEINEN, Milton Roberto; OSÓRIO, Fernando Santos. **Algoritmos genéticos aplicados ao problema de roteamento de veículos**. HÍFEN, v. 30, n. 58, 2006.

JURAN, J. M.; GRYNA, F. M. **Controle da qualidade handbook:** conceitos, políticas e filosofia da qualidade. São Paulo : Makron Books, 1991. v.1.

KAUFMAN, A. ***Rendering, Visualization and Rasterization Hardware*.** Springer Science & Business Media. pp. 86–87. *ISBN 978-3-540-56787-5, 1993.*

KORPELA, J. ***Programs vs. markup.*** IT and communication. Tampere University of Technology, 2005.

KORVA, J. et al. ***Developing a web application with Angular 2*****:** Graphical editor for Happywise's Cove Trainer. 2016.

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de software**. São Paulo: Novatec, v. 3, 2006.

LAPORTE, G.; M, GENDREAU; J.Y. P. F. SEMET. ***Classical and modern heuristics for the vehicle routing problem, International Transactions in Operational Research***, v.7, n4/5, pp. 285-300, 2002.

LEITE, R. C. G. **Um *framework* para automação/integração do processo de desenvolvimento de projetos de estruturas reticuladas tridimensionais**. 2007.

LIMA, Andre Alves. **Windows Forms ou WPF, qual utilizar**. 2016. < http://www.andrealveslima.com.br/blog/index.php/2016/07/27/windows-forms-ou-wpf-qual-utilizar/ > Acesso em: Junho de 2017.

LOBO, E. JR. **Guia prático de engenharia de software.** Universo dos Livros Editora, 2009.

LORENA, L.A.N.; SENNE, E.L.F.; PAIVA, A.C. e M.A. PEREIRA - **Integração de modelos de localização a sistemas de informações geográficas**. Gestão & Produção, vol.8, n.2, São Carlos, 2001.

MAGEDANZ, A. **Computador: Ferramenta de trabalho no Ensino (de Matemática)**. 2004. 14f. Curso de Pós-Graduação Lato Sensu Univates. Disponível em: < http://ensino.univates.br/~magedanza/pos/artigo_final_adriana_magedanz.pdf > *Acesso em: Junho 2017.*

MATIAS-PEREIRA, J. **Manual de Metodologia da Pesquisa Científica (Livro Digital)**. 3a Ed. São Paulo, SP: Editora Atlas, 2012.

MATTSSON, M. BOSCH, J. ***Framework Composition*****:** Problems, Causes and Solutions. University of Carlson Ronneby, Department of Computer Science and Business Administration. 2002.

MIKOWSKI, M. S.; POWELL, J. C. ***Single page web applications***. B and W, 2013.

MOURA, B. **Logística**: **conceitos e tendências**. Centro Atlantico, 2006.

MOZILLA. ***Developer Network***. 2017. Disponível em: < https://developer.mozilla.org > . Acesso em Maio 2017.

NAKAGAWA, E. Y; RODRÍGUEZ, L. M. **Arquitetura de Software**.2002.

NONATO, L. G. **Algoritmo de Dijkstra**. 2004. Disponível em: < http://www.lcad.icmc.usp.br/~nonato/ED/Dijkstra/node84.html> Acesso em: Junho de 2017.

NOVAES, A. G. **Logística e gerenciamento da cadeia de distribuição : estratégia, operação e avaliação**. 3. ed. Rio de Janeiro: Elsevier, 2007.

OAKLAND, J. S. **Gerenciamento da qualidade total.** São Paulo : Nobel, 1994.

PIOVESAN, A; TEMPORINI, R. apud THEODORSON, G. A. & THEODORSON, A. G. **Pesquisa exploratória: procedimento metodológico para o estudo de fatores humanos no campo da saúde pública**. Scielo Public Health, 1995. Disponível em: < www.scielosp.org/scielo.php?pid=S0034-89101995000400010 >. Acesso em: Junho 2017.

PRESSMAN, R; MAXIM, B. **Engenharia de Software**. 8ª Edição. McGraw Hill Brasil, 2016.

ROCHA. F. (1988) **Correntes Pedagógicas Contemporâneas**. Aveiro. Livraria Estante Editora.

ROCHA, M. N. **Recursos visuais e audiovisuais no ensino**: construção e exploração de materiais pedagógicos. Disponível em: < www.cf-francisco-holanda.rcts.pt/accoes/2005/accao_7.htm > Acesso em: Junho 2017.

ROCHA, Daniel Amaral de Medeiros. **Introdução a Teoria dos Grafos**. Curso preparatório para a Maratona de Programação 2005, Sociedade Brasileira de Computação (SBC). Disponível em: <http://danielamaral.wikidot.com/introducao-a-teoria-dos-grafos>. Acesso em: Junho 2017

RODRÍGUEZ, C. M. T.; COELHO, L. C.; FOLMANN, N. **Como a logística pode ajudar a aumentar a percepção de valor de seu produto?** Revista Mundo Logística, n. 07, ano II, nov/dez, 2008.

SEBESTA, R.W. **Conceitos de linguagens de programação**. Bookman Editora, 2009.

SEI. ***Software Engineering Institute*** Disponível em: < www.iso.org/iso/ catalogue_detail .htm?csnumber=35733 >. Acesso em: Junho 2017

SILVA, M. S. **HTML5: A linguagem de marcação que revolucionou a web**. 2ª edição. Novatec Editora, 2014.

SOUZA, Anderson Willian et al. **Aplicação do método de varredura na roteirização de frota em uma empresa de transporte e distribuição de cargas fracionadas**. Exacta, v. 14, n. 1, 2016.

SRINIVASAN, A. ***Cloud Computing. Internal reference.*** 2016. Disponível em: <http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/operating-systems-and-serveradministration/virtualization/9789332537439/chapter-15-understanding-services-andapplications-by-type/ch15sec5_html?uicode=ouluuas.>. Acesso em: Junho de 2017.

SVENNERBERG, G. ***Beginning Google Maps API 3***. Apress, 2010.

TAHA, H. A. **Sistemas de Filas**. In: TAHA, H. A. Pesquisa Operacional. Ed. 8. São Paulo: Person Prentice Hall, 2008. Cap. 15, p. 247-270

TENENBAUM, G. ***Introduction to analytic and probabilistic number theory***. Vol. 46. Cambridge: Cambridge university press, 1995.

THOMAZ, K. P. **Documentos eletrônicos de caráter arquivístico:** fatores condicionantes da preservação. Perspectivas em Ciência da Informação, v. 10, n. 1, 2008.

VALENTE, J. A. **O computador na sociedade do conhecimento**. Campinas: UNICAMP/NIED, 1999.

VON ATZINGEN, Jorge et al. **Análise comparativa de algoritmos eficientes para o problema de caminho mínimo**. 2012.

WANKE, Peter – **Logística e Gerenciamento da Cadeia de Suprimentos**. São Paulo.Editora Atlas. 2003

WIKIPEDIA CONTRIBUTORS. ***List of programming languages***. Disponível em: < https://en.wikipedia.org/w/index.php?title=List_of_programming_languages&oldid=780938727>. Acesso em: Maio 2017.