

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Fellipe Augusto Alves Gurgel

**Aplicação web para ensino de resolução gráfica
em disciplinas de Pesquisa Operacional**

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Fellipe Augusto Alves Gurgel

**Aplicação web para ensino de resolução gráfica em
disciplinas de Pesquisa Operacional**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

Resumo

O propósito deste trabalho é o desenvolvimento de uma aplicação *web* que auxilie instrutores e alunos de Pesquisa Operacional no estudo do método gráfico de resolução de problemas de programação linear com duas variáveis de decisão. Este documento contém todo o projeto utilizado na engenharia da aplicação, como requisitos, prototipagem, fluxograma, tecnologias utilizadas no desenvolvimento e testes. O projeto resulta em uma aplicação funcional que resolve problemas de maximização ou minimização da programação linear com duas variáveis de decisão, provendo uma solução visual em um gráfico de duas dimensões onde é possível experimentar retas, regiões com soluções factíveis e vértices concorrentes como possíveis soluções ótimas. A ferramenta está disponível online e abertamente para uso, cópia do código-fonte ou colaboração.

Palavras-chave: Pesquisa Operacional, programação linear, resolução gráfica, aplicação web, JavaScript.

Lista de ilustrações

Figura 1 – Região factível delimitada pela restrição $x_1 = 60$. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)	17
Figura 2 – Adição da reta $x_2 = 50$ à solução. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)	18
Figura 3 – Região factível depois da adição da reta $x_1 + 2x_2 = 120$ ao gráfico. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)	18
Figura 4 – Gráfico da solução após a adição da última restrição, $x_1 + x_2 = 20$. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)	19
Figura 5 – Solução final com valor ótimo C destacado. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)	19
Figura 6 – Protótipo de tela da aplicação com um exemplo resolvido	24
Figura 7 – Fluxograma da aplicação construído com a ferramenta <i>draw.io</i>	25
Figura 8 – Disposição dos arquivos e pastas da aplicação	27
Figura 9 – Aplicação em estado inicial	28
Figura 10 – Exemplo 2.1 resolvido na aplicação desenvolvida neste trabalho	29
Figura 11 – Gráfico da solução redefinido após excluir a última restrição $x_1 \leq 60$	30
Figura 12 – Exemplo 4.2.1: restrição $16x_1 + 12x_2 \leq 96$	32
Figura 13 – Exemplo 4.2.1: região factível depois da adição da restrição $20x_1 + 20x_2 \geq 140$	32
Figura 14 – Exemplo 4.2.1: solução ótima	32
Figura 15 – Exemplo 4.2.1: ponto (7,0) não é a solução ótima	33
Figura 16 – Exemplo 4.2.1: ponto (0,8) não é a solução ótima	33
Figura 17 – Exemplo 4.2.2: restrição $3x_1 + x_2 \leq 50$	35
Figura 18 – Exemplo 4.2.2: adição da restrição $2x_1 + 3x_2 \leq 40$	35
Figura 19 – Exemplo 4.2.2: adição da restrição $x_1 \leq 15$	35
Figura 20 – Exemplo 4.2.2: adição da restrição $x_2 \leq 10$	36
Figura 21 – Exemplo 4.2.2: ponto (15,0) não é a solução ótima	36
Figura 22 – Exemplo 4.2.2: ponto (5,10) não é a solução ótima	36
Figura 23 – Exemplo 4.2.2: ponto (0,10) não é a solução ótima	37

Lista de tabelas

Tabela 1	– RF001: Definir função objetivo	21
Tabela 2	– RF002: Minimizar ou maximizar uma função objetivo	21
Tabela 3	– RF003: Preencher restrição(ões)	22
Tabela 4	– RF004: Adicionar restrições	22
Tabela 5	– RF005: Remover restrições	22
Tabela 6	– RF006: Desenhar um gráfico de duas dimensões respeitando as inequa- ções das restrições	22
Tabela 7	– RF007: Desenhar as retas que representam as equações das restrições no gráfico	22
Tabela 8	– RF008: Desenhar os pontos de intersecção entre as retas que represen- tam as equações das restrições	22
Tabela 9	– RF009: Mostrar região factível	22
Tabela 10	– RF010: Mostrar o valor final da função objetivo no vértice destacado .	22
Tabela 11	– RF011: Botão para limpar todos os campos	23
Tabela 12	– RNF001 - Código aberto	23
Tabela 13	– RNF002 - Processamento feito apenas no <i>front-end</i>	23

Lista de abreviaturas e siglas

UFU	Universidade Federal de Uberlândia
TCC	Trabalho de conclusão do curso
FACOM	Faculdade de Computação
PO	Pesquisa Operacional
PL	Programação Linear
PPL	Problema de Programação Linear
PPLs	Problemas de Programação Linear
FO	Função objetivo
IDE	<i>Integrated Development Environment</i>
JS	<i>JavaScript</i>
D3	<i>Data Driver Documents</i>
HTML	<i>Hypertext Markup Language</i>
SVG	<i>Scalable Vector Graphics</i>
CSS	<i>Cascading Style Sheets</i>
RF	Requisito Funcional
RNF	Requisito Não-Funcional
SWOT	<i>Strengths, Weaknesses, Opportunities e Threats</i>
PNG	<i>Portable Network Graphics</i>
XML	<i>eXtensible Markup Language</i>
MVC	<i>Model, View and Controller</i>
DOM	<i>Document Object Model</i>

Sumário

1	INTRODUÇÃO	8
1.1	Objetivo	10
1.2	Métodos	10
1.2.1	Tecnologias	10
1.3	Organização do Trabalho	11
1.3.1	Introdução	11
1.3.2	Revisão Bibliográfica	11
1.3.3	Desenvolvimento	11
1.3.4	Resultados	12
1.3.5	Conclusão	12
2	REVISÃO BIBLIOGRÁFICA	13
2.1	Pesquisa Operacional	13
2.1.1	Método de Resolução Gráfica	16
2.2	Trabalhos Correlatos	19
3	DESENVOLVIMENTO	21
3.1	Requisitos	21
3.1.1	Requisitos Funcionais	21
3.1.2	Requisitos Não-funcionais	23
3.2	Protótipo	23
3.3	Fluxograma	24
3.4	Tecnologias empregadas	24
3.4.1	Balsamiq Mockups	25
3.4.2	Draw.io	25
3.4.3	IntelliJ IDEA	25
3.4.4	Bootstrap	26
3.4.5	D3.js	26
3.4.6	GitHub e GitHub Pages	26
3.4.7	Padrão estrutural do código	27
4	RESULTADOS	28
4.1	Validação dos requisitos	28
4.1.1	Limitações funcionais	30
4.2	Avaliação	31
4.2.1	Exemplo de minimização	31

4.2.2	Exemplo Maximização	33
5	CONCLUSÃO	38
5.1	Trabalhos Futuros	38
5.1.1	Associar reta no gráfico à respectiva restrição	38
5.1.2	Desenhar e mostrar o vetor gradiente da função objetivo	39
5.1.3	Destacar solução ótima de forma diferente	39
5.1.4	Adicionar materiais teóricos na página	39
	REFERÊNCIAS	40

1 Introdução

Eficiência e produtividade são conceitos muito importantes para organizações quando se trata de produção. A fim de atingir esses objetivos, as organizações (comerciais, industriais ou governamentais) tomam decisões quanto à melhor forma de usar os recursos disponíveis, sejam eles humanos, financeiros ou materiais (HILLIER; LIEBERMAN, 2010).

À medida em que as organizações crescem, é comum dividi-las em unidades. Entre elas estão a **gestão** e a **operação**, que impactam diretamente na produção. Aproveitar recursos de forma mais eficiente se torna um desafio para essas áreas. Torna-se necessário investigar as limitações da organização antes de tomar tais decisões. O objetivo dessas áreas sob essas análises é otimizar a produção eficientemente. Em termos gerais, elas procuram produzir o máximo com o disponível ou gastar menos para obter maior retorno; ou ainda, aproveitar tudo que se tem disponível para obter quantidade e qualidade.

Por tomada de decisão entende-se que seja “escolher uma opção entre alternativas de soluções viáveis” (MARINS, 2011). Normalmente, o procedimento que precede a análise dessas alternativas é padrão: identificar o problema, formular o objetivo, analisar limitações, calcular possíveis soluções, avaliar as alternativas e aplicar a melhor escolha. Neste contexto, a Pesquisa Operacional se faz presente para auxiliar os profissionais nas tomadas de decisão sobre a utilização eficiente de recursos, pois propõe métodos coerentes e consistentes para chegar em soluções factíveis (MARINS, 2011).

Originalmente com propósitos militares, a Pesquisa Operacional (também referida como PO) é um ramo interdisciplinar da matemática aplicada que propõe técnicas para modelagem e solução de problemas de otimização de recursos que, por sua vez, influenciam as tomadas de decisão (TANENBAUM *et al.*, 2017). Logo, a PO tornou-se bastante difundida na Engenharia de Produção e em setores como manufatura, transportes, construção, telecomunicações, planejamento financeiro, militar e serviços públicos, por exemplo.

Hillier e Lieberman (2010, p. 4) citam exemplos de aplicações da PO como forma de resolução de problemas de otimização. Alguns são:

- Maximizar as operações de produção nas fábricas químicas para atingir os objetivos de produção (eivar quantidade e qualidade, por exemplo) a um custo mínimo;
- Otimizar o desenho de uma rede de transporte rodoviário e suas rotas, isto é, entregar mais em menos tempo;
- Maximizar o lucro na alocação de tipos de aeronaves em voos domésticos. Em outras

palavras, decidir quantos e quais tipos ou tamanhos de aviões fariam tais voos minimizando o desperdício de recursos (lugares disponíveis, pessoas, combustível, etc);

- Programar a escala de funcionários para fornecer um nível de atendimento ao cliente adequado a um custo mínimo.

Percebe-se, então, que a PO tem grande relevância ao maximizar a produção ou minimizar os gastos.

A PO sugere modelos e algoritmos que consomem conceitos matemáticos e computacionais, tais como a modelagem matemática, a programação linear, a programação dinâmica, grafos, redes, filas, etc. Somados à crescente complexidade dos problemas reais de PO, torna-se inevitável a utilização de softwares para realizar os seus métodos de resolução. Logo, o ensino de PO é relevante durante a formação de profissionais das áreas de exatas e negócios, o que pode ser desafiante, visto a complexidade da natureza dos problemas reais de PO.

Portanto, é interessante abordar formas de resolução manuais para facilitar o entendimento de conceitos importantes da disciplina de PO (ou **Otimização**, como é chamada em alguns cursos). Em outras palavras, é pertinente um método didático para visualizar como chegar em um conjunto de soluções factíveis e na solução ótima (aquela que melhor resolve o problema da organização).

Dentro da Programação Linear (PL), encontra-se o método de resolução gráfica como uma alternativa descomplicada que se restringe à problemas simples e que podem ser representados por um gráfico de duas dimensões. Sua abordagem intuitivamente visual facilita a demonstração da natureza geral e dos conceitos de programação linear (KYDD, 2012). Isto é, torna-se mais fácil “enxergar” a resolução do problema mostrando as limitações físicas do desenho de um gráfico de duas dimensões.

Desenvolver a resolução gráfica exige do instrutor habilidades de desenho gráfico ou o manuseio de instrumentos para projetar o gráfico e mostrar como as interações são feitas. Mesmo com ferramentas tecnológicas, é difícil capturar a natureza dinâmica de um problema de programação linear. Além do mais, essa manipulação pode ser trabalhosa e tediosa, o que acaba por desmotivar ou confundir os alunos ou até mesmo o próprio instrutor da disciplina de PO. Portanto, para que discentes e instrutores consigam aproveitar melhor o tempo em sala de aula, é interessante o uso de uma aplicação acessível e descomplicada para o ensino dessa resolução.

1.1 Objetivo

Em um ambiente de aprendizado ativo, é importante que os alunos estejam engajados em pensar naquilo que estão desenvolvendo, bem como experimentar diferentes possibilidades de forma interativa e dinâmica. Portanto, a fim de prover um ambiente assim, este trabalho de TCC propõe o desenvolvimento de uma ferramenta *web* visual, que auxilia o instrutor e os discentes a representar e a praticar os conceitos da PL usando a resolução gráfica.

Essa aplicação permite ao usuário inserir os dados de um Problema de Programação Linear (PPL) com duas variáveis de decisão e suas restrições e demonstrar de forma gráfica e numérica os passos da resolução até chegar em soluções ótimas, mostrando e flexibilizando interações com as variáveis, restrições, regiões factíveis e não factíveis.

De forma a deixar esta ferramenta acessível, a mesma será disponibilizada como *Open Source* (de graça e de código aberto), em português, e estará disponível online, sem demandar instalações adicionais, apenas um navegador (*web browser*).

1.2 Métodos

Para o desenvolvimento deste software, o modelo iterativo e incremental mostrou ser o mais vantajoso, devido a flexibilidade de alteração nas etapas, o que diminui a taxa de erros (MAGALHÃES, 2012). Porém, dentre as opções mais específicas deste modelo (*SCRUM*, *Extreme Programming*, etc), nenhuma trouxe o conforto e a segurança de que seria a melhor opção. Sendo assim, o desenvolvimento técnico desta aplicação seguirá as seguintes etapas:

1. Levantamento de requisitos funcionais e não funcionais do sistema;
2. Criação de um protótipo do sistema;x
3. Modelagem do sistema (fluxograma);
4. Estudo prático das tecnologias propostas para o desenvolvimento do sistema;
5. Implementação e testes do *front-end* do sistema;
6. Avaliação do sistema com exemplos da literatura;
7. Avaliação do sistema junto a usuários reais (instrutores e discentes).

1.2.1 Tecnologias

Como levantado, esta ferramenta será apenas *client-side*, isto é, não precisa de receber respostas de um servidor para desenhar as respostas da resolução gráfica. As mesmas

são processadas pelo próprio navegador do usuário. Meditante tal requisito, encontra-se o *JavaScript* como a linguagem principal de programação escolhida para o desenvolvimento técnico desta aplicação, a qual funcionará em conjunto com HTML e CSS.

O *JavaScript* (*JS*) é a linguagem de programação interpretada com capacidade de orientação a objetos mais usada atualmente nos navegadores. Ele permite que *scripts* controlem o navegador, altere páginas *web* e interaja com o usuário. Por isso é comumente chamada de linguagem *client-side*, pois os *scripts* rodam no computador do cliente e não em um servidor *web* (FLANAGAN, 2006).

Existem várias bibliotecas *JavaScript* disponíveis, mas a biblioteca *D3.js* (*Data Driver Documents*) é a mais usada para trabalhar com manipulação e visualização de dados. Logo, tendo em vista a necessidade deste software desenhar gráficos e regiões dinâmicas, essa biblioteca se torna uma alternativa plausível para dar vida à aplicação. A *D3.js* trabalha em conjunto com outras tecnologias web básicas: *HTML* (*Hypertext Markup Language*), *SVG* (*Scalable Vector Graphics*), e *CSS* (*Cascading Style Sheets*). Todas são tecnologias *client-side* que rodam em navegadores.

1.3 Organização do Trabalho

Este trabalho de monografia é dividido em cinco capítulos, incluindo a Introdução (capítulo 1). De forma resumida, são eles:

1.3.1 Introdução

No capítulo 1 define-se o que PO, PL e a resolução gráfica, bem como a importância e utilidade dessas no ambiente acadêmico. Também é apresentado o objetivo, a metodologia e uma breve explicação das tecnologias de desenvolvimento escolhidas para a aplicação *web* resultante deste trabalho.

1.3.2 Revisão Bibliográfica

No capítulo 2 é feita uma revisão conceitual das termologias usadas na PO e na PL, combinados com exemplos. Por fim, este capítulo traz breves comparações com outros trabalhos relacionados, os quais trazem uma abordagem semelhante ao propósito deste trabalho de monografia.

1.3.3 Desenvolvimento

No capítulo 3 são feitas descrições mais detalhadas do processo de desenvolvimento da aplicação, desde os requisitos levantados até a implantação da aplicação em ambiente

local, passando pelas ferramentas utilizadas e prototipação. Cada passo é explicado em detalhes para que seja possível sua reprodução e obtenção do resultado final deste trabalho.

1.3.4 Resultados

No capítulo 4 são detalhadas as funcionalidades implementadas na aplicação, mostrando os *outputs* gerados no final do desenvolvimento. Inclui também o comportamento e resultados que a aplicação apresentou conforme os testes realizadas a partir de exemplos da bibliografia. Por fim, são citadas as limitações da aplicação.

1.3.5 Conclusão

No capítulo 5 consta as considerações finais sobre a aplicação resultante e esta monografia. São também sugeridas funcionalidades evolutivas para trabalhos futuros sobre este. Finalmente, são listadas as referências bibliográficas.

2 Revisão Bibliográfica

2.1 Pesquisa Operacional

Segundo [Marins \(2011, p. 13\)](#), a PO “propõe uma abordagem científica na solução de problemas: observação, formulação do problema, e construção de modelo científico (matemático ou de simulação)”. Portanto, faz parte do processo de PO investigar o problema, formular o objetivo, analisar limitações e, depois de chegar em alguma(s) solução(ões), avaliar as alternativas e implantá-la(s).

Assim, a fim de obter melhores resultados, é preciso observar a organização e identificar os sistemas e os problemas passíveis de otimização. Em seguida, começa-se o processo de formulação do problema, o qual visa listar os objetivos, as variáveis que estão sujeitas às decisões quantitativas (tratadas como variáveis de decisão), quais as limitações dessas variáveis (restrições) e quais aspectos estão envolvidos no processo, mas fogem ao controle das decisões ([CARDOSO, 2011](#)). Toda esta etapa tende a ser bastante abstrata, pois exige coleta de dados relevantes, os quais não são muito precisos, às vezes. Logo, exige não apenas conhecimento dos processos da organização e de PO, mas matemático também. Por isso, geralmente a equipe responsável por estudar e formular problemas de otimização é composta por aqueles que entendem da gestão e da operação da organização.

A próxima fase é transformar as conclusões dessas observações em uma forma mais concisa e conveniente para análise, que posteriormente se encaixará em uma das técnicas da PO para obter soluções que poderão melhorar a performance do processo investigado. Isto é, tudo que foi observado deve ser representado de forma quantitativa, expressando a essência do problema como um todo e mostrando a interdependência das variáveis em um cenário de causa-efeito ([HILLIER; LIEBERMAN, 2010](#)). Em vista disso, a PO adota **modelos matemáticos** para esta representação, que simplesmente traduz o objetivo e as limitações em expressões matemáticas (equações, inequações, funções ou lógica simbólica).

Tendo em vista a melhoria do rendimento ou da produtividade de um processo, considerando os recursos disponíveis (capital, matéria-prima, mão-de-obra, equipamentos, tempo, etc.), normalmente são definidos em modelos matemáticos lineares. Ou seja, a parametrização gerada a partir da formulação do problema é representada somente por funções e restrições lineares (de primeiro grau) ([CARDOSO, 2011](#)). Depois disso, é possível usá-lo como entrada em ferramentas da PO (normalmente algorítmicas) a fim de obter uma solução ou um conjunto de soluções suficientemente boas, chamadas **solução(ões) ótima(s) ou factível(eis)**.

A **Programação Linear** (PL) é uma das ferramentas mais utilizadas pela PO para resolver modelos matemáticos lineares. Ela propõe planos de atividades — algoritmos — para obter soluções viáveis que visam maximizar ou minimizar o modelo especificado. Um problema deste tipo é também chamado de Problema de Programação Linear (PPL), o qual possui um objetivo de maximização ou minimização em função de variáveis de decisão lineares e sujeito a restrições também lineares (CARDOSO, 2011).

A abordagem da PL estabelece alguns termos para compor um modelo matemático válido, que forneça uma representação mais precisa do problema real e que, posteriormente, sirva de entrada para os métodos de resolução. São eles:

- **Variáveis de decisão:** alternativas quantificáveis para decisão. Normalmente expressas por $x_1, x_2, \dots, x_n$;
- **Função objetivo (FO):** meta a ser atingida, ou uma medida de desempenho. Normalmente, maximizar ou minimizar uma função matemática em função das variáveis de decisão;
- **Restrições:** limitações nos valores das variáveis de decisão expressas por equações ou inequações.

A fim de ilustrar esses conceitos, apresenta-se a seguir um problema de programação linear.

Exemplo 2.1: Supondo a produção de dois modelos de computadores, A e B, em uma fábrica. Considere que o modelo A garante R\$18,00 de lucro, enquanto B, R\$30,00. Na produção, o modelo A requer 1 gabinete pequeno e 1 unidade de disco; o modelo B requer 1 gabinete grande e 2 unidades de disco. Em estoque existem 60 unidades do gabinete pequeno, 50 unidades do gabinete grande e 120 unidades de disco. Deve-se considerar também que, devido a decisões de mercado, a produção mínima dos dois modelos de computadores (juntos) deve ser de 20 unidades (GABRIEL, 2016).

Maximizar o lucro dessa produção usando o máximo de recursos disponíveis trata-se de um problema de PO. Portanto, um modelo matemático que representa o problema deve ser extraído dos dados fornecidos. Logo, define-se:

Variáveis de decisão: visto que o que queremos saber é “qual a quantidade máxima de computadores A e B é possível produzir usando o que tem no estoque?”, podemos dizer que x_1 representa a quantidade a ser produzida do modelo de computador A e x_2 representa a quantidade a ser produzida do modelo B.

Função objetivo: maximizar o lucro da produção de A e B. Ou seja, considerando que cada computador do modelo A gera R\$18,00 de lucro e cada computador do

B gera R\$30,00, devemos aumentar ao máximo a quantidade de ambos para obter mais lucro. Matematicamente:

$$\mathbf{Max} \ f(x_1, x_2) = 18x_1 + 30x_2 \quad (2.1)$$

Restrições: temos que considerar que cada computador do modelo A gasta 1 gabinete pequeno e 1 unidade de disco, e que o modelo B gasta 1 gabinete grande e 2 unidades de disco. No estoque existem 60 gabinetes pequenos (GP), 50 gabinetes grandes (GG) e 120 unidades de disco (UD). E devem ser produzidos no mínimo 20 computadores (MIN). Portanto, a quantidade de computadores A (x_1) e B (x_2) deve se restringir a essas condições. Matematicamente podemos dizer que o problema está sujeito a:

$$GP \rightarrow 1x_1 + 0x_2 \leq 60 \quad (2.2)$$

$$GG \rightarrow 0x_1 + 1x_2 \leq 50 \quad (2.3)$$

$$UD \rightarrow 1x_1 + 2x_2 \leq 120 \quad (2.4)$$

$$MIN \rightarrow x_1 + x_2 \geq 20 \quad (2.5)$$

Consequentemente, x_1 e x_2 devem ser maiores ou iguais a zero.

Modelos matemáticos deste tipo, formatados segundo as premissas da PL, passam a ser reconhecidos também por modelo de PO ou de PL.

O objetivo da PL é encontrar o valor mais favorável dentre um conjunto de soluções factíveis ou viáveis (aquelas que, ao ser aplicada na função objetivo, respeitam todas as restrições). Este valor é a solução ótima, que será o maior valor para um PPL de maximização ou o menor valor caso o PPL seja de minimização ([MARINS, 2011](#)).

Em situações reais, os PPLs apresentam um número considerável de variáveis e restrições que inviabilizam uma resolução manual. Nestes casos, recomenda-se a utilização de softwares específicos para PL, como softwares que usam o Método Simplex ([CARDOSO, 2011](#)), o qual não entra no propósito deste TCC.

É importante ressaltar que tais soluções são reflexos do modelo matemático, logo, refletem a idealização abstrata do problema real e não uma representação exata deste. Isso se dá porque geralmente existem incertezas e fatores imprecisos incorporados aos problemas práticos. Por esta razão, afirma-se que a PO busca soluções para melhorar o desempenho de algum cenário, mas não necessariamente a solução ideal.

2.1.1 Método de Resolução Gráfica

Tendo em vista que problemas de programação linear com duas variáveis de decisão podem ser representados em um gráfico de duas dimensões, é possível demonstrar como chegar no conjunto de soluções factíveis e na solução ótima através da **resolução gráfica**, proposta pela PL.

Este método segue alguns passos até encontrar a solução ótima. O objetivo é desenhar um gráfico bidimensional de modo que fique visível a região de soluções viáveis. Para isso, primeiramente, é preciso que o modelo matemático esteja descrito no formato de um PPL:

$$\textbf{maximizar ou minimizar } f(x_1, x_2) = ax_1 + bx_2 \quad (2.6)$$

$$\textit{sujeito a : } \left\{ \textit{conjunto de restrições sob as variáveis de decisão } x_1 \textit{ e } x_2 \right.$$

onde a e b são constantes da função objetivo.

Os eixos do gráfico a resolução apresenta representam os valores de x_1 e x_2 do PPL. Em seguida, as inequações das restrições do modelo matemático são transformadas em equações simplesmente por mudar os sinais de desigualdade por de igualdade. Depois, tais equações são desenhadas em forma de retas lineares sob o gráfico, delimitando a região factível a partir delas. Por fim, analisa-se todos os vértices do gráfico gerado para definir a solução ótima.

Exemplo 2.2 (Resolução do Exemplo 2.1): Retomando o problema do exemplo, o modelo de PL correspondente é:

$$\begin{aligned} \textbf{Max } f(x_1, x_2) &= 18x_1 + 30x_2 \\ \textit{sujeito a : } &\left\{ \begin{array}{l} 1x_1 + 0x_2 \leq 60 \\ 0x_1 + 1x_2 \leq 50 \\ 1x_1 + 2x_2 \leq 120 \\ 1x_1 + 1x_2 \geq 20 \\ x_1, x_2 \geq 0 \end{array} \right. \end{aligned} \quad (2.7)$$

Modelado o problema, é possível começar o método de resolução gráfica. Como as variáveis de decisão x_1 e x_2 são sempre maiores que zero, considera-se o gráfico apenas na região positiva dos eixos.

O primeiro passo é traduzir as restrições, passando-as de inequações para equações. Basta mudar os sinais de maior-igual e menor-igual para igual. Sendo assim, tem-se:

$$\left\{ \begin{array}{l} x_1 = 60 \\ x_2 = 50 \\ x_1 + 2x_2 = 120 \\ x_1 + x_2 = 20 \end{array} \right.$$

Desta forma, as equações lineares se tornam passíveis de serem desenhadas como retas no plano gráfico. Assim, desenhando uma a uma, a região factível vai sendo delimitada.

A Figura 1 mostra a região factível delimitada pela reta correspondente a $x_1 = 60$, que passa pelo ponto “A”.

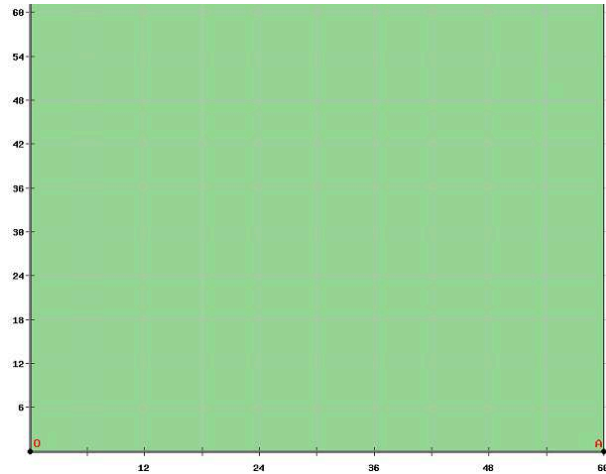


Figura 1 – Região factível delimitada pela restrição $x_1 = 60$. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)

A região colorida contém as soluções factíveis. Isto é, dentro da região colorida, existe um par (x_1, x_2) que, no futuro, poderá ser a solução ótima para, nesse caso, maximizar a função objetivo.

Assim que as demais retas que representam as restrições vão sendo desenhadas, a região factível fica menor e, conseqüentemente, se torna mais intuitivo encontrar os pontos que fazem parte do conjunto de soluções factíveis.

Ao adicionar a reta correspondente a $x_2 = 50$, obtém-se a Figura 2. A região factível é então delimitada também pela reta “B-C”.

Na Figura 3 a reta correspondente à terceira restrição, $x_1 + 2x_2 = 120$, é dada pela reta “F-G”.

A Figura 4 mostra como fica o gráfico após a adição da última restrição, $x_1 + x_2 = 20$, representada pela reta “H-I”.

O método da resolução gráfica demonstra que a solução ótima estará em um dos vértices formados na região factível (nesse caso, A, C, D, E, H ou I). Sendo assim, testando os valores de cada um destes vértices, temos que a solução ótima para o problema do **Exemplo 2.1** é o vértice C(60,30):

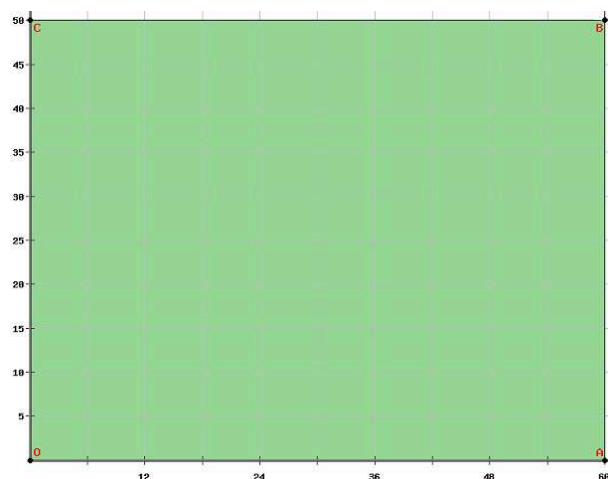


Figura 2 – Adição da reta $x_2 = 50$ à solução. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)

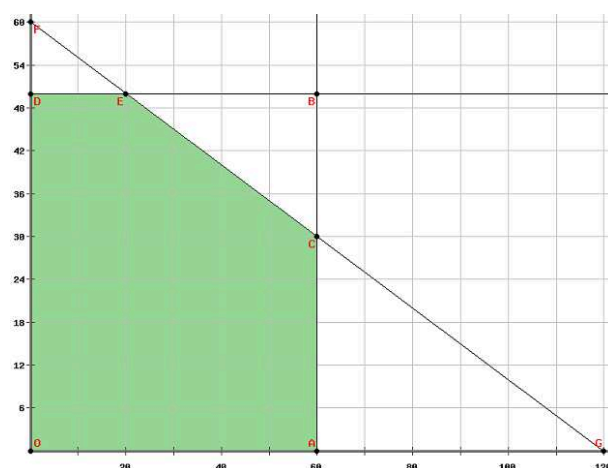


Figura 3 – Região factível depois da adição da reta $x_1 + 2x_2 = 120$ ao gráfico. Fonte: PHPSimplex (GRANJA; RUIZ, 2006)

Por fim, na Figura 5, tem-se uma reta colorida que passa pelo vértice correspondente à solução ótima.

Isto é, ao aplicarmos os valores do vértice C(60,30) na função objetivo “ $f(x_1, x_2) = 18x_1 + 30x_2$ ”, obtém-se o maior valor possível considerando todas as restrições do problema.

Voltando ao problema real e interpretando a solução: como x_1 representa a quantidade a ser produzida de computadores do modelo A e x_2 a quantidade a ser produzida do modelo B, para maximizar o lucro da fábrica de computadores, considerando as limitações do estoque e a estratégia de mercado, deve-se produzir 60 computadores do modelo A e 30 do modelo B.

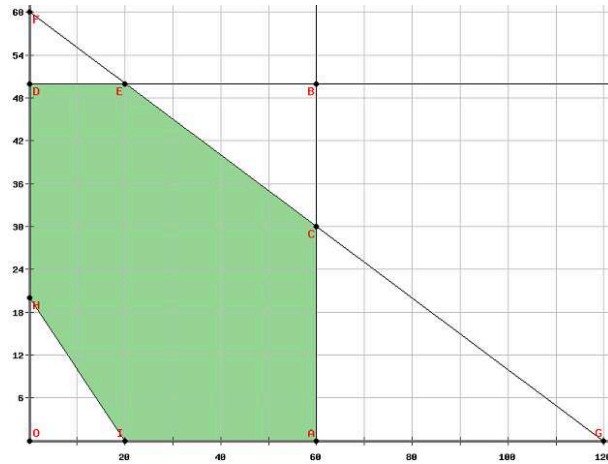


Figura 4 – Gráfico da solução após a adição da última restrição, $x_1 + x_2 = 20$. Fonte: PHPSimplex ([GRANJA; RUIZ, 2006](#))

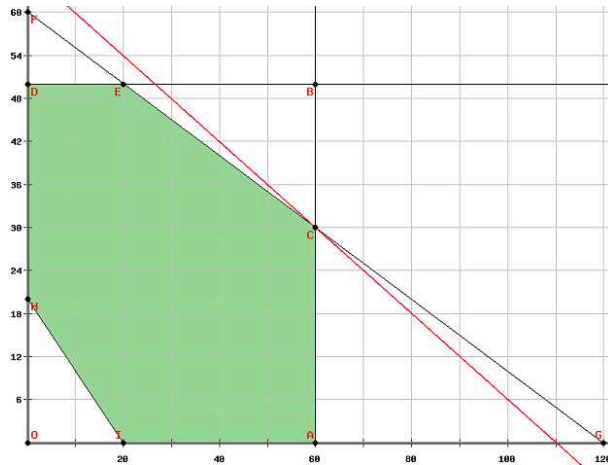


Figura 5 – Solução final com valor ótimo **C** destacado. Fonte: PHPSimplex ([GRANJA; RUIZ, 2006](#))

2.2 Trabalhos Correlatos

Existem alguns *softwares* que mostram a construção da resolução gráfica de forma dinâmica e que permitem experimentar o conjunto de soluções factíveis. Este tipo de instrumento ajuda a engajar os alunos no processo de aprendizagem.

O artigo de [Kydd \(2012\)](#) apresenta métricas da eficiência do uso de uma ferramenta *web* para se ensinar os conceitos da PL. Ela demonstra indicadores de como o uso de uma ferramenta *web* pode beneficiar o aprendizado de alunos sobre conceitos da PL, usando a ferramenta *Linear Programming Applet (LP Java Applet:)*. Em suas demonstrações, ela aparentemente resolve bem o problema demonstrando um passo-a-passo interativo e dinâmico de como foi construída a solução. No entanto, hoje esta mesma ferramenta não está mais disponível.

O *Tutorial IOR* é um material que vem no CD-ROM que acompanha o livro

“Introdução à Pesquisa Operacional” de [Marins \(2011\)](#) e está incluso um procedimento interativo para a aplicação da resolução gráfica para PPLs. No entanto, mesmo com o livro disponível em bibliotecas públicas, nem sempre o CD-ROM o acompanha de fato. Além disso, entende-se que CDs já são uma tecnologia ultrapassada que inviabiliza os alunos de levá-lo para a sala de aula, por exemplo. Muitos *laptops* não vêm com leitor de CD, inclusive.

Em contrapartida, o **GLP-Tool** (*Graphical Linear Programming Tool*), uma aplicação desenvolvida por [Pereira e Fernandes \(2013\)](#) utilizando o *Wolfram Mathematica*, apresenta um ambiente interativo e intuitivo. Porém, este software depende da instalação do *Wolfram CDF (Computable Document Format) Player* e cadastros nas plataformas *Wolfram* e *Wolfram Demonstrations*. Todo este processo está suscetível a muitas dúvidas e erros, se tornando ainda mais trabalhoso, desmotivante e tedioso.

A solução então seria um software online, como o “**PHPSimplex**” ([GRANJA; RUIZ, 2006](#)), o “**Graphical Method Calculator**” ([SHAH; SHAH, 2003](#)) e o “**Linear programming grapher (two variables)**” ([WANER, 2013](#)), os quais oferecem um ambiente bem intuitivo e fácil para definir as funções e restrições do modelo de PL. Porém, além de estarem em inglês, nenhuma deles mostra um passo-a-passo para a solução. Todos efetuam os cálculos e mostram o resultado final em um único passo, ficando a cargo do aluno entender o processo da resolução gráfica. O que os tornam muito úteis para corrigir exercícios de PL, por exemplo. Existe, ainda, versões do “**Linear programming grapher (two variables)**” de [WANER](#) para *Android* e *iOS*; no entanto, ambos são aplicativo pagos.

Em se tratando de aplicativos para *smartphones*, eles são, de fato, uma boa alternativa por serem práticos e móveis, além de que ficaria fácil trazer tais ferramentas para a sala de aula. O aplicativo “**Simplex PL**” de [GLÓRIA; CALDAS; BARBOSA](#) é um dos poucos gratuitos disponíveis. No entanto, apresenta o mesmo problema das aplicações citadas anteriormente: não mostra um passo-a-passo para a solução.

3 Desenvolvimento

Este capítulo detalha todo o processo de desenvolvimento da aplicação, desde os requisitos levantados até a disponibilização online dela, passando pelas ferramentas utilizadas e criação de protótipos. Cada passo é explicado em detalhes para que seja possível sua reprodução e continuidade.

3.1 Requisitos

Na engenharia de software, o levantamento de requisitos é uma das fases iniciais para o desenvolvimento de *software*. Esta etapa consiste em definir as principais funcionalidades e características da aplicação, as quais guiarão as próximas fases: prototipação, implementação e testes (PRESSMAN; MAXIM, 2016). Neste caso, tais requisitos foram colhidos incrementalmente e são baseados nas necessidades do discente ou docente para resolver um problema de programação linear usando a resolução gráfica.

Os requisitos podem ser definidos em funcionais e não-funcionais (PRESSMAN; MAXIM, 2016).

3.1.1 Requisitos Funcionais

Requisitos funcionais são definições dos comportamentos e tarefas indispensáveis da aplicação. Eles especificam quais funcionalidades devem ser implementadas para que a aplicação seja útil.

Nesta seção, são listados nas tabelas de 1 a 11, os onze requisitos funcionais para a aplicação desenvolvida neste trabalho.

Tabela 1 – RF001: Definir função objetivo

RF001	Definir função objetivo
O usuário deve ser capaz de definir e editar os dois coeficientes da função objetivo a qualquer momento.	

Tabela 2 – RF002: Minimizar ou maximizar uma função objetivo

RF002	Minimizar ou maximizar uma função objetivo
A aplicação deve disponibilizar uma opção para o usuário escolher entre minimizar ou maximizar a função objetivo.	

Tabela 3 – RF003: Preencher restrição(ões)

RF003	Preencher restrição(ões)
O usuário deve ser capaz de associar valores aos coeficientes das restrições, bem como as suas “limitações” (valores que vêm depois do sinal de desigualdade nas inequações do problema). Lembrando que o problema se restringe a apenas duas variáveis de decisão (x_1 e x_2).	

Tabela 4 – RF004: Adicionar restrições

RF004	Adicionar restrições
A aplicação deve disponibilizar uma opção para o usuário adicionar novas restrições.	

Tabela 5 – RF005: Remover restrições

RF005	Remover restrições
A aplicação deve disponibilizar uma opção para o usuário remover restrições.	

Tabela 6 – RF006: Desenhar um gráfico de duas dimensões respeitando as inequações das restrições

RF006	Desenhar um gráfico de duas dimensões respeitando as inequações das restrições
A aplicação deve conter um gráfico de duas dimensões o qual respeite os alcances das retas das equações que representam as restrições.	

Tabela 7 – RF007: Desenhar as retas que representam as equações das restrições no gráfico

RF007	Desenhar as retas que representam as equações das restrições no gráfico
Na aplicação, cada restrição deve ser representada por uma reta no gráfico a partir da equação que a representa.	

Tabela 8 – RF008: Desenhar os pontos de intersecção entre as retas que representam as equações das restrições

RF008	Desenhar os pontos de intersecção entre as retas que representam as equações das restrições
A aplicação deve mostrar ao usuário onde e quais são os pontos de intersecção entre as retas que representam as equações das restrições.	

Tabela 9 – RF009: Mostrar região factível

RF009	Mostrar região factível
A aplicação deve destacar a parte do gráfico que representa a região factível.	

Tabela 10 – RF010: Mostrar o valor final da função objetivo no vértice destacado

RF010	Mostrar o valor final da função objetivo no vértice destacado
A aplicação deve mostrar o valor final da função objetivo nos vértices destacadas.	

Tabela 11 – RF011: Botão para limpar todos os campos

RF011	Botão para limpar todos os campos
A aplicação deve fornecer um botão que limpa todos campos (função objetivo e restrições) e o gráfico.	

3.1.2 Requisitos Não-funcionais

Requisitos não-funcionais são as condições que a aplicação deve obedecer em termos de qualidade de uso, gerenciamento e contexto de ambientes, ou seja, eles vão além de descrever apenas como o sistema deve se comportar (papel dos requisitos funcionais). Esses podem ou não ser especificados pelo usuário ou cliente. Geralmente, dizem respeito à performance, usabilidade, portabilidade, gerenciamento de testes e *deploys*, e condições do ambiente de operação da aplicação (HASSAN, 2009).

Neste trabalho, foram especificados dois requisitos não-funcionais, descritos nas tabelas 12 e 13.

Tabela 12 – RNF001 - Código aberto

RNF001	Código Aberto
A aplicação deve ser desenvolvida usando ferramentas de gratuitas (bibliotecas, IDE e repositório) para que ela seja disponibilizada e reproduzida abertamente.	

Tabela 13 – RNF002 - Processamento feito apenas no *front-end*

RNF002	Processamento feito apenas no <i>front-end</i>
O sistema deve ser desenvolvido para processar seus dados apenas no <i>front-end</i> , afim de evitar dependências de rede ou internet para eventuais conexões com um <i>back-end</i> .	

3.2 Protótipo

A prototipagem também é uma importante etapa na engenharia de software, pois permite ao time de desenvolvimento uma ideia tangível daquilo que está prestes a ser implementado. Protótipos são, portanto, uma proposta flexível de solução, os quais permitem testar e validar ideias e processos de uma forma menos custosa. Nele é possível enfatizar, definir, idealizar e explorar soluções e identificar possíveis limitações cedo no desenvolvimento .

A Figura 6 representa o protótipo de alta fidelidade da aplicação, porém não evolucionário, ou seja, o mais próximo da solução final, mesmo que não funcional. Esse foi incrementado conforme sugestões de design e usabilidade, a partir de protótipos de baixa fidelidade, projetados com caneta, lápis e papel. Este protótipo foi construído utilizando a ferramenta *Balsamiq Mockups*.

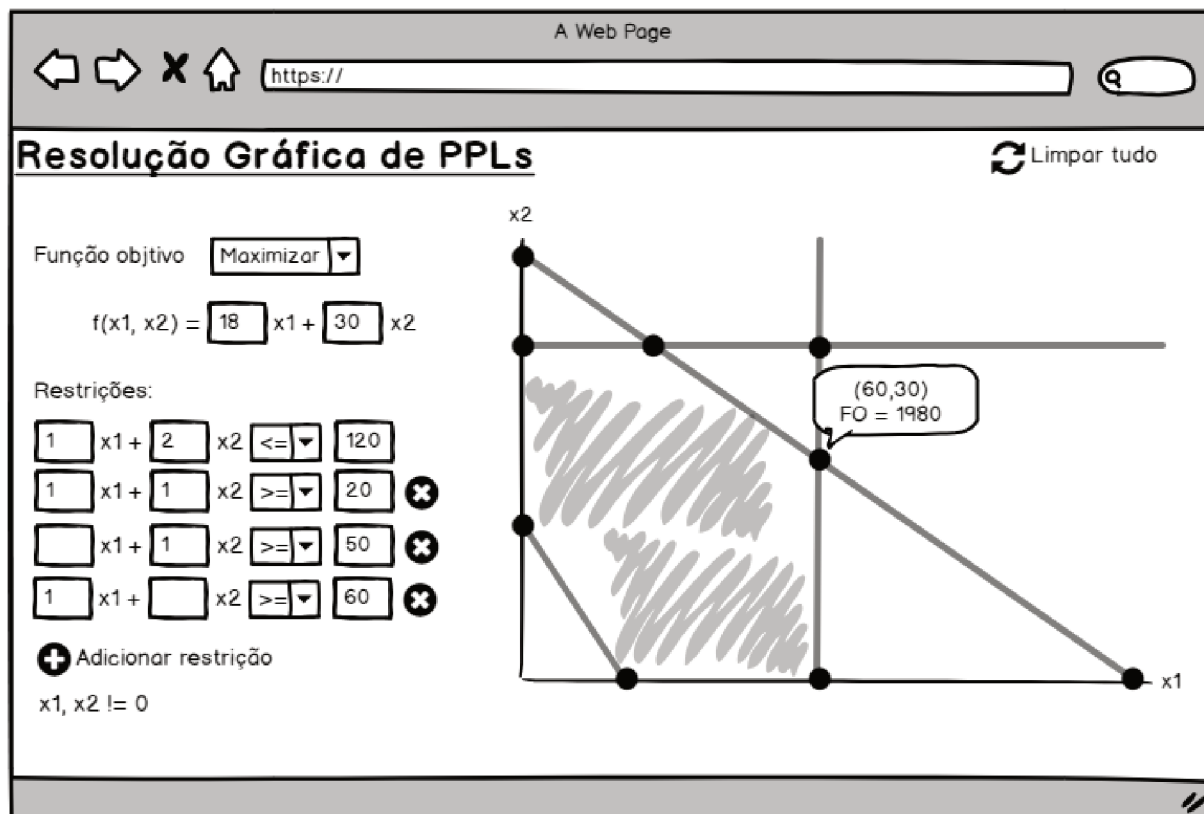


Figura 6 – Protótipo de tela da aplicação com um exemplo resolvido

3.3 Fluxograma

Depois de definir o que a aplicação deve fazer e como ela deve se apresentar, a próxima etapa constitui em projetar um fluxograma que representa o algoritmo por trás da solução.

Um fluxograma é uma forma gráfica/visual de representar os processos de um *software* sob uma visão lógica e algorítmica. Através dele, conseguimos enxergar o passo-a-passo da aplicação desde o início até o final, passando pelo processamento das entradas do usuário. A Figura 7 mostra o fluxograma construído com a ferramenta *draw.io* para a aplicação desenvolvida neste trabalho.

3.4 Tecnologias empregadas

Tanto nas etapas de especificação e projeto, quanto no desenvolvimento e *deploy* da aplicação, utiliza-se de ferramentas que melhor se aplicam na solução do problema, escolhidas pela facilidade de utilização e popularidade. Uma vez estabelecido como a aplicação deve se comportar, em termos de entradas e saídas, como a mesma deve se apresentar e seu algoritmo principal, definiu-se as tecnologias a serem usadas no projeto e desenvolvimento. Nesta seção, tais ferramentas são descritas.

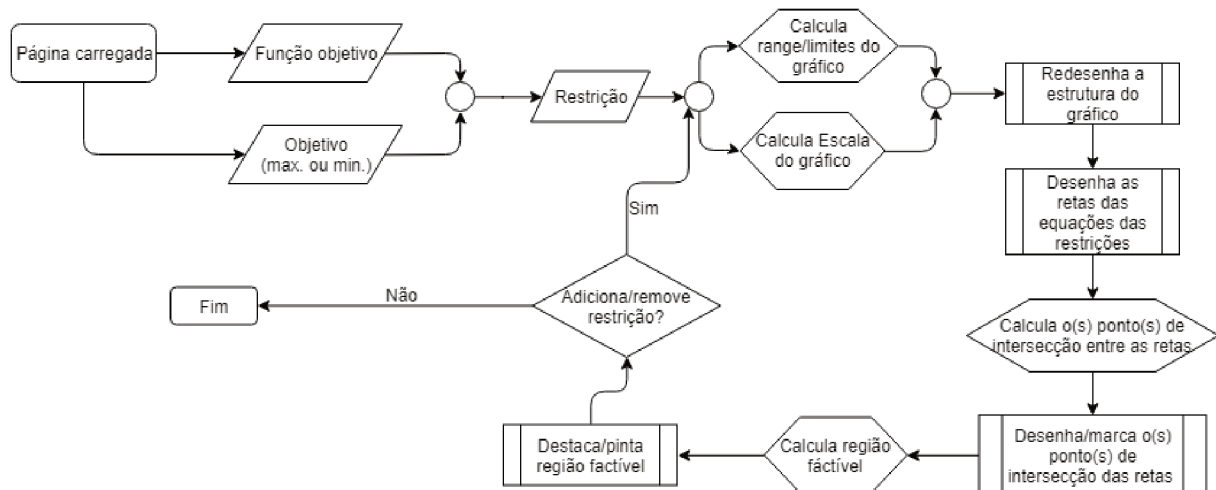


Figura 7 – Fluxograma da aplicação construído com a ferramenta *draw.io*

3.4.1 Balsamiq Mockups

A ferramenta *Balsamiq Mockups* (BALSAMIQ, 2017), desenvolvida pela empresa *Balsamiq Studio*, é utilizada na construção de protótipos. Para o propósito deste trabalho, ela foi aproveitada para criar o protótipo da tela da aplicação a partir de um rascunho manuscrito e depois para exportar para uma imagem PNG.

Essa é uma ferramenta online e completa para esboçar e criar *wireframes* de forma colaborativa, iterativa e versionada, sendo utilizadas não somente para protótipos de *website*, como foi o caso neste trabalho, mas também para interfaces *mobile*.

3.4.2 Draw.io

Assim como o *Balsamiq Mockups*, o *draw.io* (DRAW.IO, 2017) também é uma ferramenta online muito útil no momento da especificação e projeção de uma aplicação. Contudo, *draw.io* é utilizado principalmente na criação de diagramas UML, de árvore, de redes, mapas mentais, de sequência, *SWOT*, entre outros.

Para este trabalho, o *draw.io* foi utilizado para construir o fluxograma que representa o algoritmo macro da aplicação e para exportar esse em uma imagem PNG.

3.4.3 IntelliJ IDEA

IntelliJ IDEA (JETBRAINS, 2018) é uma IDE (do inglês, ambiente de desenvolvimento integrado) para desenvolvimento de software construída pela *JetBrains*. Esta ferramenta suporta diferentes linguagens de programação como Java, Groovy, Python, JavaScript, PHP, SQL, entre outras; conta com assistência no momento de programar e possui um ambiente que possibilita integrações com *plugins*, outros serviços e *frameworks*.

A aplicação deste trabalho foi inteiramente desenvolvida e versionada utilizando esta IDE, pois a mesma está integrada ao serviço de versionamento *GitHub*.

3.4.4 Bootstrap

Bootstrap (BOOTSTRAP, 2018) é um *framework front-end* para desenvolvedores *web*, utilizada na criação de sites com tecnologia responsiva sem esforço CSS. Além disso, o *Bootstrap* possui vários componentes (*plugins*), em JavaScript (jQuery) que auxiliam o desenvolvedor a implementar: *tooltip*, *menu-dropdown*, *modal*, *slideshow*, entre outros, sem muita dificuldade, apenas acrescentando algumas configurações no código, e sem a necessidade de se criar muitos *scripts*.

Neste trabalho, o *Bootstrap* foi utilizado na página que o usuário acessa, a fim de apresentá-la com um *design* mais suave e intuitivo, sem demandar muitas linhas de código ou a importação de diferentes *plugins JavaScript*, por exemplo.

3.4.5 D3.js

D3.js (*Data-Driver Documents*) (BOSTOCK, 2017) é uma biblioteca *JavaScript* usada para trabalhar com manipulação e visualização de dados. A *D3.js* trabalha em conjunto com outras tecnologias web básicas: HTML, SVG, e CSS. Todas essas são tecnologias *client-side* que funcionam em navegadores *web*.

Esta biblioteca foi utilizada para implementar todo o algoritmo que manipula o gráfico da aplicação, dada a necessidade de se desenhar retas, pontos e regiões dinâmicas no gráfico de acordo com as entradas do usuário. Além disso, permite a criação de uma interface com aparência agradável.

3.4.6 GitHub e GitHub Pages

O GitHub (GITHUB, 2018a) é uma plataforma online de controle de versão distribuído, gerenciamento e hospedagem de código-fonte de softwares. Ele provê várias *features* para desenvolvedores gerenciarem seus projetos de forma colaborativa (aberta ou privada). O *GitHub* usa o *Git* para registrar as mudanças no código-fonte (apesar do mesmo registrar mudanças outros tipos de arquivos) de aplicações que são, geralmente, editadas por vários desenvolvedores.

Neste projeto, apesar de contar com apenas um desenvolvedor, o GitHub é usado para gerenciar as iterações da aplicação e hospedar o código de forma aberta e colaborativa. Assim, outros desenvolvedores podem evoluir a aplicação ou usar o código dela em outros projetos.

Também utiliza-se o *GitHub Pages* (GITHUB, 2018b), que é um serviço oferecido pelo *GitHub* para hospedar páginas *web* dos usuários do *GitHub*. Esse serviço também pode ser usado para *blogs*, documentações de projetos e até livros, e geralmente possui o endereço de acordo com o nome do usuário e o subdomínio *github.io*. A página *web* desenvolvida neste trabalho foi feita usando o *GitHub Pages* e está disponível online e abertamente para que qualquer usuário ou desenvolvedor possa acessar.

3.4.7 Padrão estrutural do código

A aplicação desenvolvida neste projeto foi projetada para funcionar utilizando apenas navegadores *web*, ou seja, a aplicação é somente *client-side*, não precisando de processamento ou armazenamentos de dados através da rede.

Geralmente, o desenvolvimento de software é passível de ter um padrão estrutural para a organização dos códigos-fonte e arquivos do projeto, visto que é uma boa prática entre desenvolvedores, principalmente se a aplicação for colaborativa ou com o código aberto. Desta forma, é fácil entender o que e como está sendo desenvolvido o processo, de forma intuitiva e lógica.

Considerando estes detalhes, a aplicação deste trabalho segue as sugestões estruturais de Sanchez (2012), as quais recomendam seguir uma certa estrutura de nomes e disposição para arquivos e pastas. Este padrão é bastante difundido para projetos *web* utilizando HTML, CSS e JavaScript. A Figura 8 mostra tal estrutura.

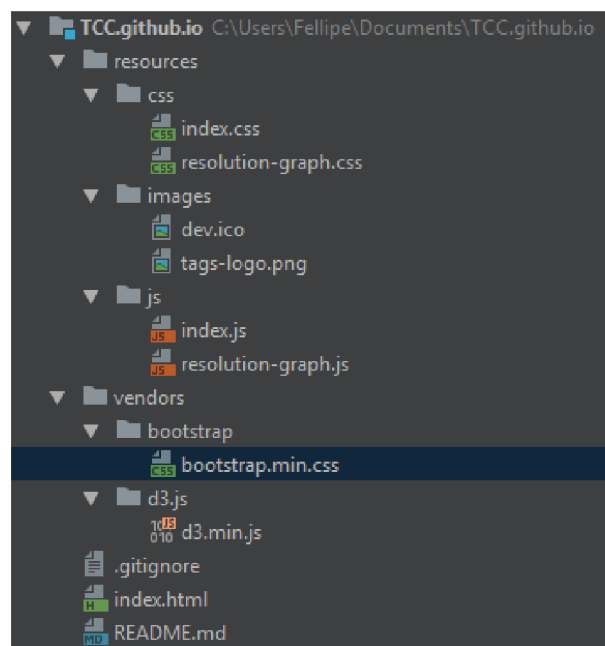


Figura 8 – Disposição dos arquivos e pastas da aplicação

4 Resultados

Neste capítulo, são detalhadas as funcionalidades implementadas a partir do projeto e das tecnologias descritas no Capítulo 3, mostrando os as saídas geradas, o comportamento e as limitações que a aplicação apresentou conforme os testes realizadas a partir de exemplos da bibliografia.

4.1 Validação dos requisitos

Utilizando a IDE *IntelliJ IDEA* e combinando *Bootstrap* e a biblioteca JavaScript *D3.js*, foi possível desenvolver o que foi projetado nos requisitos funcionais e no protótipo da aplicação.

A Figura 9 mostra a aplicação em seu estágio inicial, ou após o usuário clicar no botão “Limpar Campos”. Isto é, a página *web* recém carregada, sem nenhuma entrada por parte do usuário. Por ela, os requisitos funcionais 1, 2, 3 e 11 são validados, ou seja, é possível comprovar que a aplicação permite definir e editar uma função objetivo, escolher entre maximizar ou minimizar a mesma, preencher as duas variáveis de decisão na restrição e voltar ao estado inicial ao clicar no botão “Limpar Campos”.

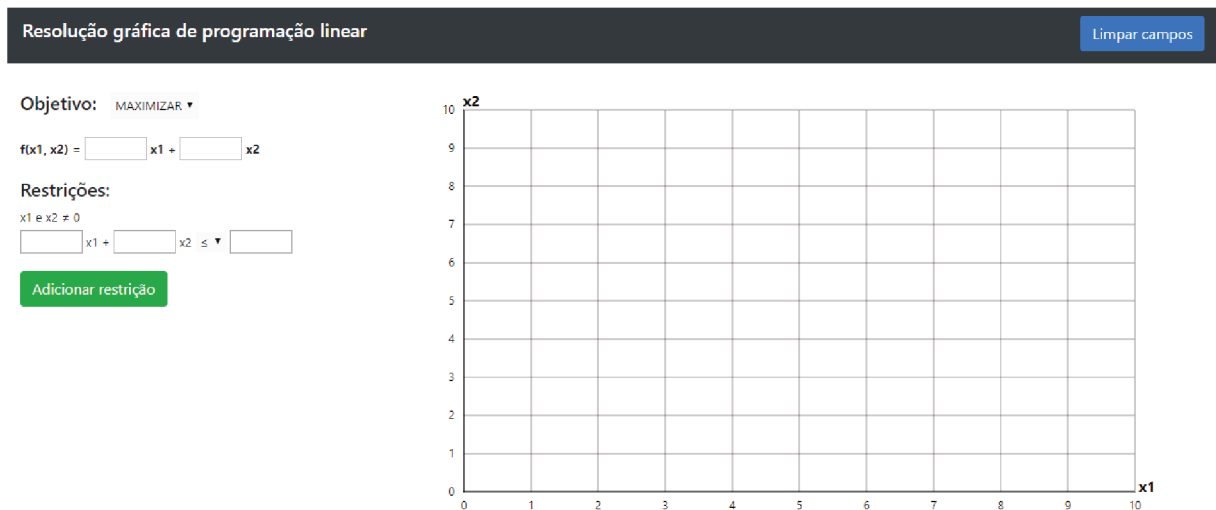


Figura 9 – Aplicação em estado inicial

A Figura 10 foi criada baseada no Exemplo 2.1 (Capítulo 2) através da aplicação desenvolvida neste trabalho. Pela figura, é possível validar os requisitos funcionais 4, 5, 6, 7, 8, 9 e 10, bem como o protótipo mostrado na Figura 6 (Capítulo 1.3.3. Em outras palavras, comprova-se que a aplicação permite:

- Adicionar restrições através do botão “Adicionar restrição”;

- Remover restrições através do botão vermelho ao lado delas. A Figura 11 mostra o estado do gráfico após a exclusão da quarta restrição (ou seja, $x_1 \leq 60$);
- Ajustar automaticamente no gráfico os valores exibidos nos eixos conforme as equações das restrições;
- Representar tais restrições por meio de retas no gráfico;
- Caso as retas tenham pontos de intersecção entre elas ou eixos, destacar estes pontos;
- Mostrar a região factível do PPL;
- Mostrar o valor final da função objetivo nos pontos destacados.

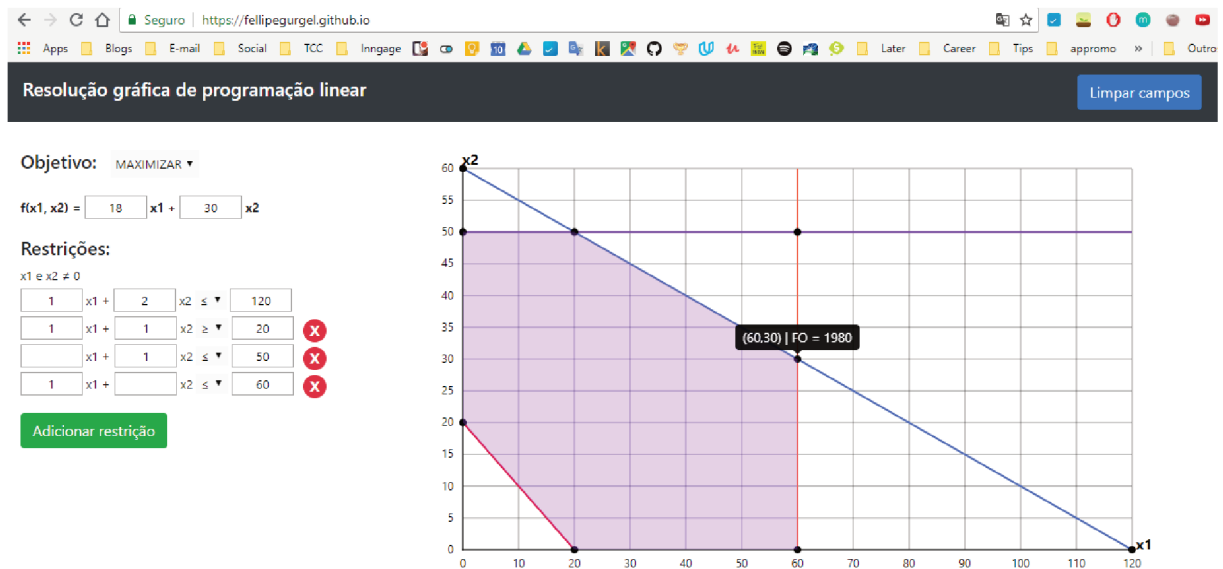


Figura 10 – Exemplo 2.1 resolvido na aplicação desenvolvida neste trabalho

A fim de cumprir o requisito não-funcional da Tabela 12, a aplicação está disponível para *download* ou colaboração de outros desenvolvedores através da plataforma *GitHub*. O código-fonte se encontra no link <https://github.com/fellipegurgel/fellipegurgel.github.io> e a página pode ser acessada online pelo endereço <https://fellipegurgel.github.io/>, conforme está visível na barra de endereço das figuras 10 e 11.

Para colaborar com o projeto, os desenvolvedores podem clonar ou fazer *download* do repositório *Git*, trabalhar localmente e submeter suas mudanças de volta para o repositório remoto no *GitHub*. Tais alterações são automaticamente adicionadas à página *github.io*.

Desde que o(a) desenvolvedor(a) tenha um navegador *web*, não é necessário que ele(a) instale ou configure nenhuma ferramenta adicional para executar a aplicação. Para

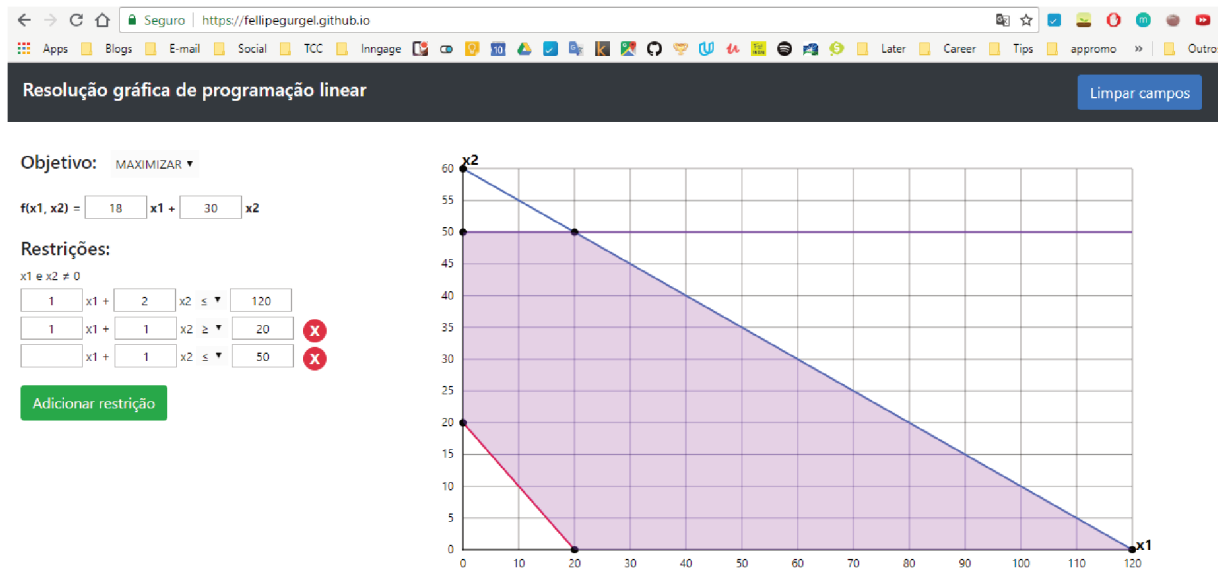


Figura 11 – Gráfico da solução redefinido após excluir a última restrição $x_1 \leq 60$

enviar as mudanças para o repositório remoto, o colaborador deve ter instalado o *Git* ou outra IDE que suporte esta integração, caso não esteja utilizando *IntelliJ IDEA*. Esta, por sua vez, já possui o serviço *Git* integrado.

O último requisito (não-funcional) da aplicação consiste em “funcionar sem necessidade de transferências de dados pela rede”. Em outras palavras, a aplicação deve operar sem depender de internet ou *back-end*, todo processamento deve ser feito no navegador do usuário. Esse requisito foi atendido quando se optou por desenvolver uma aplicação *client-side*.

A Figura 8 mostra que todos os arquivos necessários para o funcionamento da aplicação independem de tráfego de dados pela rede. Uma vez carregada a página, todo processamento das entradas do usuário é feito pelo navegador do usuário por meio do JavaScript.

4.1.1 Limitações funcionais

A aplicação não é capaz mostrar mais de uma região factível. Caso duas ou mais restrições tenham domínios diferentes, ou seja, não há intersecção entre os pontos que respeitem as restrições, formando duas ou mais regiões distintas, a aplicação apresentará somente uma delas. Isso é uma limitação do algoritmo que calcula os pontos que delimitam a região factível. Pretende-se tratar essa limitação em trabalhos futuros.

4.2 Avaliação

Nesta seção, são mostrados dois exemplos para avaliação dos critérios de aceitação (requisitos) usando a aplicação desenvolvida neste trabalho. O objetivo destes exemplos é comprovar o real aproveitamento da ferramenta junto a problemas reais de sala de aula no ensino de Pesquisa Operacional.

4.2.1 Exemplo de minimização

(LATECKI, 2008) Uma empresa fabrica bicicletas de 18 marchas e 21 marchas em duas fábricas diferentes, **A** e **B**. A fábrica **A** cobra R\$1000 por dia para fabricar 16 bicicletas de 18 marchas e 20 de 21 marchas. Enquanto isso, na fábrica **B**, são cobrados por dia R\$800 para produzir 12 bicicletas de 18 marchas e 20 de 21 marchas.

A empresa recebeu uma encomenda de 96 bicicletas de 18 marchas e 140 de 21 marchas. Agora é necessário calcular quantos dias de serviço será requisitado a cada fábrica, a fim de minimizar os custos para produzir a quantidade encomendada.

Tendo, portanto, x_1 como a quantidade de dias operados na fábrica **A** e x_2 os dias operados na **B**, pode-se dizer que a empresa busca minimizar o resultado (custo) de $f(x_1, x_2) = 1000x_1 + 800x_2$.

A partir disso, é possível dizer que as restrições do problema são delimitadas pelo tipo de bicicleta, ou seja:

$$\begin{cases} 16x_1 + 12x_2 \geq 96 \\ 20x_1 + 20x_2 \geq 140 \end{cases}$$

Sendo assim, tem-se um PPL definido como:

$$\begin{aligned} &\textbf{minimizar} \quad f(x_1, x_2) = 1000x_1 + 800x_2 \\ &\textit{sujeito a} : \quad \begin{cases} 16x_1 + 12x_2 \geq 96 \\ 20x_1 + 20x_2 \geq 140 \\ x_1, x_2 \geq 0 \end{cases} \end{aligned} \tag{4.1}$$

Usando a aplicação desenvolvida neste trabalho, foi possível ilustrar a resolução gráfica deste problema. As figuras 12 e 13 demonstram as inclusões das restrições na ferramenta, na ordem que aparecem no problema.

Por fim, é possível ver na figura 14 que o ponto **(3,4)** é a solução ótima para esse exemplo. Em outras palavras, com 3 dias operando na fábrica **A** e 4 dias operando na **B**, a empresa consegue produzir 96 bicicletas de 18 marchas e 140 de 21 marchas, garantindo menor custo (R\$6.200).

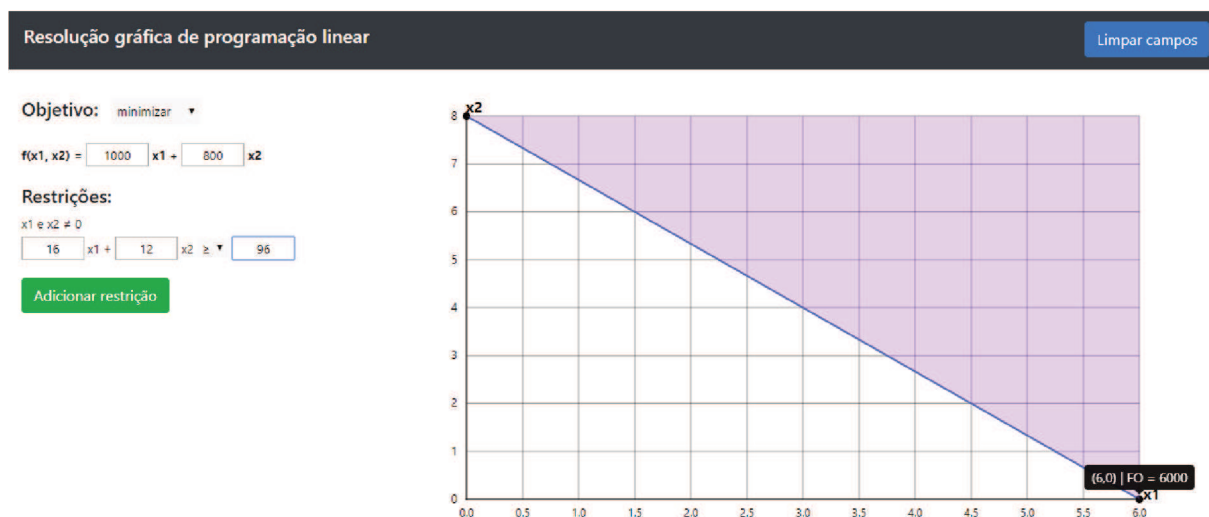
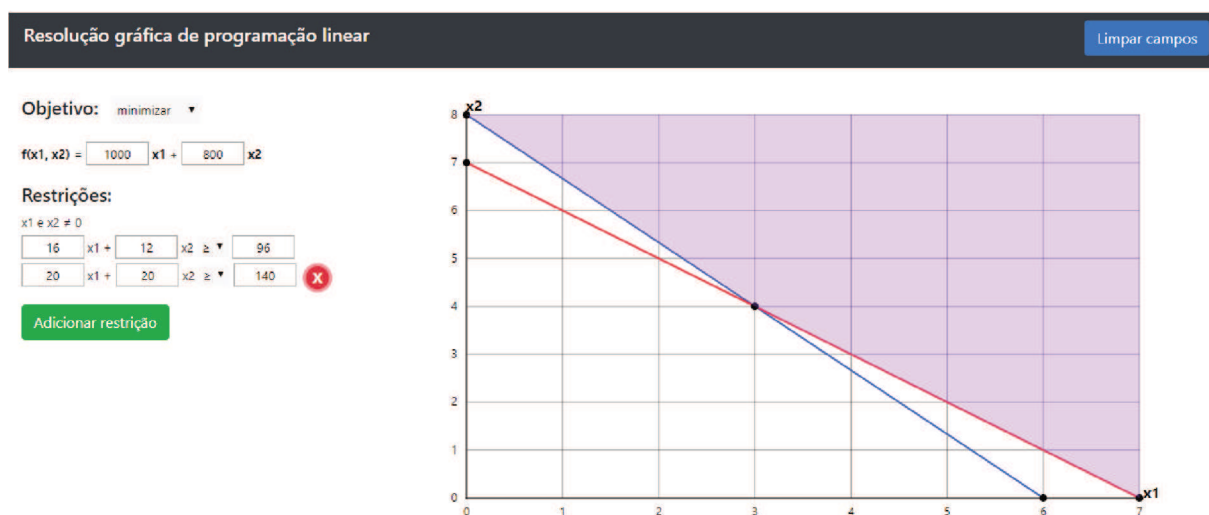
Figura 12 – Exemplo 4.2.1: restrição $16x_1 + 12x_2 \leq 96$ Figura 13 – Exemplo 4.2.1: região factível depois da adição da restrição $20x_1 + 20x_2 \geq 140$ 

Figura 14 – Exemplo 4.2.1: solução ótima

A garantia de que as demais soluções factíveis não seriam ótimas é demonstrada através das figuras 15 e 16, as quais apresentam valores superiores a 6200 para a função objetivo.

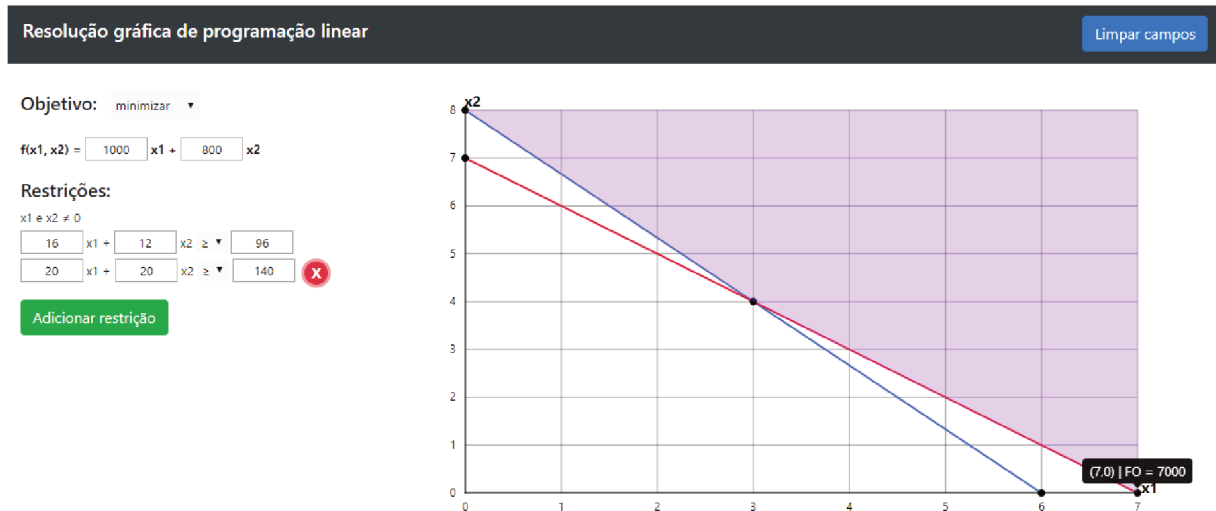


Figura 15 – Exemplo 4.2.1: ponto (7,0) não é a solução ótima



Figura 16 – Exemplo 4.2.1: ponto (0,8) não é a solução ótima

4.2.2 Exemplo Maximização

(GALVINL, 2017) Uma lanchonete vende dois tipos de suco, o “Refrescante” e o “Super-Demais”. O Refrescante é feito com 3 laranjas, 2 maçãs e 1 pedaço de gengibre. O Super-Demais contém 1 pedaço de melancia, 3 maçãs e 1 laranja. O estoque de laranjas é de 50, de maçãs é 40, de gengibre é 15 pedaços e de melancia é 10 pedaços.

Sendo x_1 o número de copos de suco Refrescante que a lanchonete faz e x_2 o número de copos de sucos Super-Demais, é possível representar a situação em um sistema de inequações.

$$\begin{cases} 3x_1 + x_2 \leq 50 \\ 2x_1 + 3x_2 \leq 40 \\ x_1 \leq 15 \\ x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

Agora supondo que um copo do suco Refrescante seja vendido por R\$ 6 e do Super-Demais por R\$ 8, cada, e que o dono da lanchonete quer maximizar a receita das vendas dos sucos considerando as limitações dos ingredientes em estoque, pode-se dizer que é de interesse do dono da lanchonete querer otimizar o número de copos de sucos Refrescante e Super-Demais a fim de conseguir a maior receita.

Este é um PPL, onde $f(x_1, x_2)$ representa a receita das vendas dos dois sucos Refrescante (x_1) e Super-Demais (x_2). O objetivo, portanto, é:

$$\begin{aligned} &\textbf{Maximizar} \quad f(x_1, x_2) = 6x_1 + 8x_2 & (4.2) \\ &\textit{sujeito a :} \quad \begin{cases} 3x_1 + x_2 \leq 50 \\ 2x_1 + 3x_2 \leq 40 \\ x_1 \leq 15 \\ x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases} \end{aligned}$$

Como este é um problema com duas variáveis de decisão, é possível resolvê-lo utilizando o método gráfico da PL. As figuras 17, 18, 19 e 20 ilustram a adição da função objetivo e das restrições na ordem que aparecem na equação 4.2, e o vértice destacado em cada uma delas mostra o maior valor naquele estado.

Após a adição da última restrição (Figura 20), é possível ver que o ponto que maximiza o valor da função objetivo é o ponto **(15,3.34)** que a totaliza 116.67. Em outras palavras, para o dono da lanchonete maximizar a receita das vendas de suco, considerando as quantidades em estoque e a receita dos sucos, ele deve vender 15 copos do Refrescante e 3.34 do Super-Demais e, assim, obter o máximo de receita, que é R\$ 116.67.

Os demais pontos da região factível resultam em valores menores que 116.67. Como demonstrado nas figuras 21, 22 e 23, provando que o ponto **(15,3.34)** é, de fato, o que maximiza a FO.

Figura 17 – Exemplo 4.2.2: restrição $3x_1 + x_2 \leq 50$ Figura 18 – Exemplo 4.2.2: adição da restrição $2x_1 + 3x_2 \leq 40$ Figura 19 – Exemplo 4.2.2: adição da restrição $x_1 \leq 15$

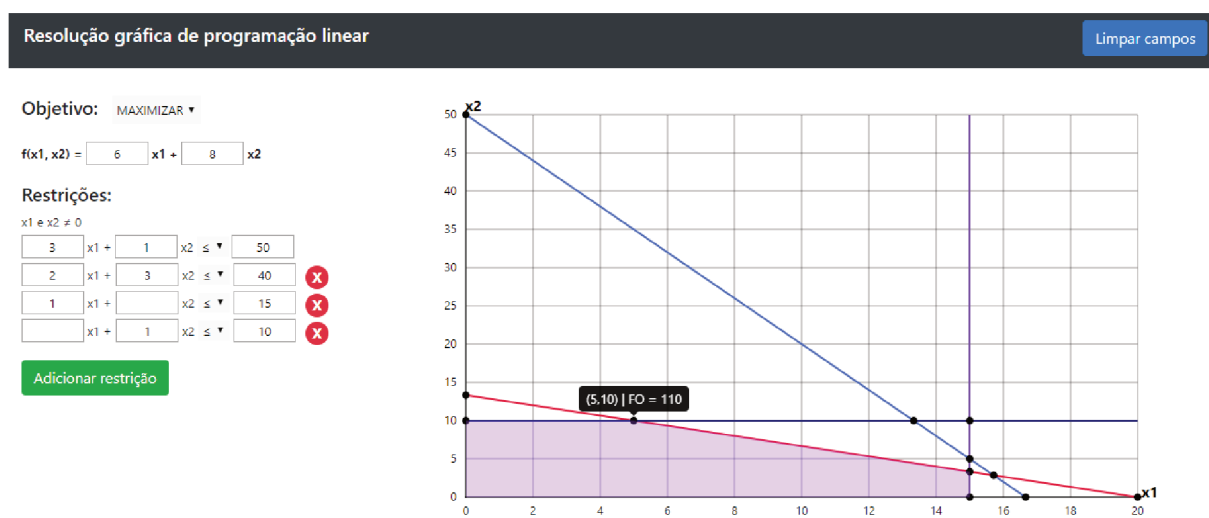
Figura 20 – Exemplo 4.2.2: adição da restrição $x_2 \leq 10$ Figura 21 – Exemplo 4.2.2: ponto $(15, 0)$ não é a solução ótimaFigura 22 – Exemplo 4.2.2: ponto $(5, 10)$ não é a solução ótima

Figura 23 – Exemplo 4.2.2: ponto $(0,10)$ não é a solução ótima

5 Conclusão

Resolver problemas de Programação Linear utilizando a resolução gráfica é muito importante didaticamente, pois introduz e reforça o pensamento conceitual da PL e, consequentemente, da PO. Sendo assim, torna-se muito conveniente aos instrutores dessa disciplina adotarem ferramentas que facilitem o ensino do método.

Mesmo que os problemas resolvidos através do método gráfico não sejam tão complexos em termos computacionais, e instrutores ou discentes disponham de ferramentas tecnológicas, ainda assim pode ser trabalhoso criar um gráfico de duas dimensões dinâmico e preciso. Logo, torna-se um desafio para tais usuários.

Este trabalho, por sua vez, apresenta uma ferramenta exclusiva para resolver PPLs através da resolução gráfica. Tal aplicação permite ao usuário inserir os dados de um PPL com duas variáveis de decisão e suas restrições. Por sua vez, essa aplicação mostra de forma gráfica e numérica os a resolução em termos de retas, região factível e soluções ótimas e permite que o usuário edite o problema em tempo real, sem precisar carregar a aplicação novamente.

A aplicação está acessível para uso no endereço <<https://fellipegurgel.github.io/>> e seu código-fonte também está disponível para *download* ou colaboração na plataforma *GitHub* através do endereço <<https://github.com/fellipegurgel/fellipegurgel.github.io>>.

5.1 Trabalhos Futuros

A aplicação desenvolvida é composta pelos requisitos considerados mais relevantes para resolver e ensinar PPLs através do método gráfico. No entanto, algumas outras funcionalidades e melhorias são sugeridas a fim de incrementar a experiência do usuário e engajar ainda mais o mesmo no estudo ou ensino de PL.

Nesta sessão, detalha-se as sugestões de funcionalidades a serem trabalhados futuramente.

5.1.1 Associar reta no gráfico à respectiva restrição

No momento da concepção deste trabalho, é possível saber qual reta representa qual restrição no momento em que ela é preenchida. No entanto, como a maioria dos problemas têm mais de uma restrição, usuários podem se sentir desorientados na solução, caso não tenham acompanhado o preenchimento das variáveis na ferramenta e/ou não possuem facilidade para identificar qual reta, região ou ponto está relacionada a qual

restrição.

Sendo assim, seria interessante se a aplicação associasse a cor da reta nas restrições; assim sabe-se que, por exemplo, a reta de cor azul está relacionada a restrição com *label* azul.

5.1.2 Desenhar e mostrar o vetor gradiente da função objetivo

Outra forma didática para encontrar a solução ótima é considerar a direção do vetor gradiente (vetor normal) da função objetivo.

Sem entrar em detalhes teóricos desta solução, ela apresenta uma abordagem mais matemática do que gráfica, o que pode ser interessante para instrutores trabalharem também a descoberta da resposta de forma mais lógica e não tanto visual. Portanto, esta funcionalidade seria um bom incremento para a aplicação.

5.1.3 Destacar solução ótima de forma diferente

Da forma que a aplicação foi desenvolvida, para achar a solução ótima, basta procurar pelo ponto com maior/menor valor para função objetivo.

Para problemas com uma ou mais soluções ótimas, ou até mesmo nenhuma, seria interessante destacar isso de forma diferente, deixando mais evidente a existência ou não da solução ótima.

5.1.4 Adicionar materiais teóricos na página

No desenvolvimento deste trabalho, considerou-se que aqueles que chegam até a aplicação sabem exatamente do que se trata e apenas a usam. No entanto, como esta ferramenta também é usada por alunos para estudar a resolução gráfica, seria interessante que a mesma possuísse uma sessão com materiais teóricos sobre, por exemplo:

- Pesquisa Operacional;
- Programação Linear;
- Problemas de PL;
- Exemplos de soluções, identificando cada elemento delas;
- Como é calcular retas, pontos concorrentes e região factível.

Referências

- BALSAMIQ. *Balsamiq Mockups*. 2017. Website Balsamiq Mockups. Disponível em: <<https://balsamiq.com/products>>. Acesso em: 01 jul. 2018. Citado na página 25.
- BOOTSTRAP. *Bootstrap*. 2018. Bootstrap Documentation. Disponível em: <<https://getbootstrap.com/docs/4.0/getting-started/introduction/>>. Acesso em: 03 jul. 2018. Citado na página 26.
- BOSTOCK, M. *Data-Driven Documents*. 2017. D3.js Documentation. Disponível em: <<https://d3js.org/>>. Acesso em: 03 jul. 2018. Citado na página 26.
- CARDOSO, A. *Fundamentos da Pesquisa Operacional*. 2011. UNIFAL-MG. Disponível em: <<http://www.unifal-mg.edu.br/matematica/files/file/po.pdf>>. Acesso em: 20 jul. 2017. Citado 3 vezes nas páginas 13, 14 e 15.
- DAM, R.; SIANG, T. *Design Thinking: Get Started with Prototyping*. 2018. Interaction Design Foundation. Disponível em: <<https://www.interaction-design.org/literature/article/design-thinking-get-started-with-prototyping>>. Acesso em: 30 jul. 2018. Nenhuma citação no texto.
- DRAW.IO. *draw.io Examples*. 2017. Website draw.io. Disponível em: <<https://about.draw.io/features/examples/>>. Acesso em: 01 jul. 2018. Citado na página 25.
- FLANAGAN, D. *JavaScript: the definitive guide*. EUA: O'Reilly Media, Inc., 2006. Citado na página 11.
- GABRIEL, P. H. R. *Primeira Lista de Exercícios*. 2016. FACOM-UFU. Disponível em: <<http://www.facom.ufu.br/~phrg/files/gsi027/otm-lista01.pdf>>. Acesso em: 27 jul. 2017. Citado na página 14.
- GALVINL, D. *Linear programming, graphically*. 2017. University of Notre Dame. Disponível em: <https://www3.nd.edu/~dgalvin1/10120/10120_S17/Topic23_3p3_Galvin_2017_short.pdf>. Acesso em: 05 jul. 2018. Citado na página 33.
- GITHUB. *Documentation alongside your code*. 2018. GitHub Documentation. Disponível em: <<https://github.com/features#documentation>>. Acesso em: 03 jul. 2018. Citado na página 26.
- _____. *GitHub Pages: Websites for you and your projects*. 2018. GitHub Pages Documentation. Disponível em: <<https://pages.github.com/>>. Acesso em: 04 jul. 2018. Citado na página 27.
- GLÓRIA, J. P.; CALDAS, D.; BARBOSA, T. Desenvolvimento de um aplicativo móvel para cálculo de métodos de programação linear na plataforma Android. *Exatas & Engenharia*, v. 5, n. 12, 2015. ISSN 2236-885X. Disponível em: <http://www.seer.perspectivasonline.com.br/index.php/exatas_e_engenharia/article/view/656>. Citado na página 20.

- GRANJA, D. I.; RUIZ, J. J. *PHPSimplex: Graphical method*. 2006. PHPSimplex. Disponível em: <<http://www.phpsimplex.com/simplex/simplex.htm?l=pt>>. Acesso em: 20 jul. 2017. Citado 5 vezes nas páginas 3, 17, 18, 19 e 20.
- HASSAN, A. E. *Non Functional Requirements (NFR) – Quality Attributes*. 2009. Queen's University. Disponível em: <http://research.cs.queensu.ca/~ahmed/home/teaching/CISC322/F09/slides/CISC322_02_Requirements.pdf>. Acesso em: 30 jul. 2018. Citado na página 23.
- HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à Pesquisa Operacional*. Porto Alegre: AMGH, 2010. Citado 2 vezes nas páginas 8 e 13.
- JETBRAINS. *IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains*. 2018. Website IntelliJ IDEA. Disponível em: <<https://www.jetbrains.com/idea/>>. Acesso em: 03 jul. 2018. Citado na página 25.
- KYDD, C. T. The effectiveness of using a web-based applet to teach concepts of linear programming: An experiment in active learning. *Institute for Operations Research and the Management Sciences*, v. 2, n. 2, p. 78–88, 2012. Citado 2 vezes nas páginas 9 e 19.
- LATECKI, L. J. *Solving linear programming problems using the graphical method*. 2008. Temple University. Disponível em: <<https://cis.temple.edu/~latecki/Courses/RobotFall08/Talks/LinearProgramming.pdf>>. Acesso em: 06 jul. 2018. Citado na página 31.
- MAGALHÃES, G. *Os Modelos de Desenvolvimento de Software*. 2012. Protocolo TI. Disponível em: <<http://protocoloti.blogspot.com.br/2012/03/os-modelos-de-desenvolvimento-de.html>>. Acesso em: 28 jul. 2017. Citado na página 10.
- MARINS, F. A. S. *Introdução à Pesquisa Operacional*. São Paulo: Cultura Acadêmica, 2011. Citado 4 vezes nas páginas 8, 13, 15 e 20.
- PEREIRA, J. C.; FERNANDES, S. Two-variable linear programming: a graphical tool with mathematica. In: LOJA, A.; BARBOSA, J. I.; RODRIGUES, J. A. (Ed.). *1st International Conference on Algebraic and Symbolic Computation*. Lisboa: Associação Portuguesa de Mecânica Teórica, Aplicada e Computacional, 2013. Citado na página 20.
- PRESSMAN, R.; MAXIM, B. *Engenharia de Software*. 8. ed. Porto Alegre: McGraw Hill Brasil, 2016. Citado na página 21.
- SANCHEZ, R. *How to organize your HTML, CSS, and Javascript files*. 2012. Appcropolis. Disponível em: <<http://appcropolis.com/blog/web-technology/organize-html-css-javascript-files/>>. Acesso em: 04 jul. 2018. Citado na página 27.
- SHAH, V. P.; SHAH, P. N. *Graphical Method calculator*. 2003. AtoZmath. Disponível em: <<http://cbom.atozmath.com/CBOM/Simplex.aspx?q=gm>>. Acesso em: 20 jul. 2017. Citado na página 20.
- TANENBAUM, M.; HOLSTEIN, W. K.; EILON, S.; ACKOFF, R. L. *Operations research*. 2017. Encyclopedia Britannica. Disponível em: <<https://www.britannica.com/topic/operations-research#ref22348>>. Acesso em: 10 jul. 2017. Citado na página 8.

WANER, S. *Linear programming grapher (two variables)*. 2013. Finite mathematics & Applied calculus Resources for students. Disponível em: <<http://www.zweigmedia.com/utilities/lpg/index.html?lang=en>>. Acesso em: 28 jul. 2017. Citado na página 20.