UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Vieira Dias

Aplicação web para edição e execução de códigos em linguagem GNU MathProg

Uberlândia, Brasil

2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Vieira Dias

Aplicação web para edição e execução de códigos em linguagem GNU MathProg

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 Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2020

Agradecimentos

Agradeço a Deus por ter me sustentado e dado condições de chegar até aqui. Sem dúvidas, sem o auxílio dele não chegaria a esse ponto. Agradeço também aos meus familiares que sempre acreditaram em mim, e foram porto seguro nos momentos mais difíceis, e aos amigos e colegas feitos nessa caminhada, em especial ao Gabriel Botelho da Silva que me ajudou muito no começo do curso, ao Willian Santos Silva que foi o maior ajudador quando o assunto era desenvolvimento e ao Marcelo Henrique Machado que foi quem me introduziu no mundo empresarial. Aos professores, meus sinceros agradecimentos; ao orientador, sou extremamente grato por assumir o desafio juntamente comigo além de ter sido quem abriu meus olhos no segundo período ao lecionar a matéria "Lógica para Computação" e ter me motivado e mostrado que era possível avançar no curso e chegar nesse momento. Aos demais que acreditaram, estiveram juntos, orando e incentivando e caminhando lado a lado, meu muito obrigado.

"Salmos 126

1 Quando o SENHOR trouxe do cativeiro os que voltaram a Sião, estávamos como os que sonham.

2 Então a nossa boca se encheu de riso e a nossa língua de cântico; então se dizia entre os gentios: Grandes coisas fez o Senhor a estes.

3 Grandes coisas fez o Senhor por nós, pelas quais estamos alegres.

4 Traze-nos outra vez, ó Senhor, do cativeiro, como as correntes das águas no sul.

5 Os que semeiam em lágrimas segarão com alegria.

6 Aquele que leva a preciosa semente, andando e chorando, voltará, sem dúvida, com alegria, trazendo consigo os seus molhos."

Resumo

O objetivo deste trabalho de conclusão de curso é desenvolver uma aplicação web para resolver modelos escritos na linguagem GNU MathProg. O usuário dessa aplicação poderá fazer uso dos recursos disponíveis no sistema, como: editar um modelo (ou carregar um previamente), gravar um modelo em seu computador e consultar exemplos predefinidos. Uma vez escrito o modelo, o software fará uso dos seus recursos e bibliotecas para compilar o modelo inserido e, se não houver erros (que serão relatados pelo software), usará a ferramenta GLPSOL para obter os resultados para o problema. Todo o projeto foi desenvolvido no Front-End e sua execução é feita em tempo real sem a persistência de dados e nem a necessidade de montagem de ambiente ou servidores para armazenar a aplicação. Nesta monografia, são apresentados estudos de caso que validam a ferramenta desenvolvida.

Palavras-chave: Pesquisa Operacional, Otimização , aplicação web, Resolução, solver.

Lista de ilustrações

Lista de abreviaturas e siglas

PO	Pesquisa Operacional
TCC	Trabalho de Conclusão de Curso
UFU	Universidade Federal de Uberlândia
FACOM	Faculdade da Computação
HTML	*Hypertext Markup Language*
CSS	*Cascading Style Sheets*
JS	*Java Script*

Sumário

1 Introdução

A Pesquisa Operacional (PO) é uma disciplina presente em diversos cursos. Especificamente, nos cursos de Sistemas de Informação e Ciência da Computação da Universidade Federal de Uberlândia, existem disciplinas obrigatórias relacionadas ao tema. Essa disciplina tem como objetivo a aplicação de métodos, técnicas e ferramentas de PO na modelagem e solução de problemas da área.

A PO tem se tornado cada dia mais importante nas mais diversas áreas. Tanto na parte empresarial quanto na científica, gerir bem os recursos disponíveis para determinadas atividades se tornou essencial, tendo em vista o grande crescimento da demanda por atualizações constantes de tecnologias e necessidade de agilidade na entrega dos projeto, a máxima efetividade juntamente com o mínimo gasto possível se torna imprescindível. É nesse cenário que a pesquisa operacional é introduzida. Dispondo de artifícios para auxiliar nas tomadas de decisão relativas a melhor disposição dos recursos, ela propõe formas consistentes para alcançar soluções viáveis e até mesmo ótimas para a alocação dos mesmos (SOBRAPO, 2014).

A PO tem grande importância no que diz respeito a maximizar os lucros e minimizar custos. A seguir, são litados alguns problemas dessa área, conforme descrito por Hillier e Lieberman (2013):

- Escalas de funcionários para atendimento aos cliente com menor custo possível.
- Ajustar rede de transportadoras para melhor velocidade de entrega minimizando os custos.
- 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.

Tendo em vista os benefícios da PO, é essencial aborda-lá durante a formação dos profissionais que possam atuar nessa área. Nesse contexto, é interessante o desenvolvimento de ferramentas que auxiliem no processo de ensino–aprendizagem dessa disciplina. Kantor (2016) desenvolveu uma aplicação web para resolução de problemas de Programação Linear (PL), tópico coberto pela disciplina de PO. Essa aplicação permite resolver problemas de PL modelados utilizando a linguagem GNU MathProg (MAKHORIN, 2016).

Essa ferramenta web, no entanto, possui algumas limitações. Por exemplo, seu autor empregou diversas bibliotecas exclusivas do navegador Google Chrome, tornando seu uso restrito a navegadores similares. Além disso, muitas dessas bibliotecas, nos últimos

anos, foram descontinuadas, tornaram-se depreciadas. Como consequência, sua execução não é mais possível.

Por esse motivo, optou-se por desenvolver uma nova aplicação, tendo como base o trabalho original (KANTOR, 2016). Assim, o objetivo principal deste trabalho é desenvolver uma aplicação web que permita executar programas desenvolvidos na linguagem GNU MathProg. Essa nova aplicação teve sua iteração com usuário melhorada tanto na sua aparência e quanto na iteração. Para isso, foram utilizados conceitos e ferramentas das áreas de experiência do usuário (UX, do inglês *User Experience*) e Interação Humano Computador (IHC). Finalmente, a interface está toda em língua portuguesa, para permitir sua utilização em sala de aula sem restrições.

Essa ferramenta permite que estudantes desenvolvam e implementem modelos de PL, contendo uma função objetivo e diversas restrições. Ao final, a ferramenta executa o modelo mostrando sua solução (caso seja viável) e apresentando os valores das variáveis de decisão. Além disso, caso o modelo possua erros na sintaxe da linguagem, o usuário é informado pela aplicação, a qual indica a linha onde o erro foi encontrado. Finalmente, a aplicação traz modelos pré-desenvolvidos, os quais podem ser usados como consulta para aprendizado, bem como base para desenvolvimento de novos modelos. A ferramenta desenvolvida está disponível para uso no endereço: <http://gvdmathprog.atwebpages.com/> e seu código-fonte também está disponível para download ou colaboração na plataforma GitHub através do endereço: <https://github.com/GabrielVieiraDias/MathProg-Solver.git>

2 A Linguagem GNU MathProg

A linguagem GNU MathProg faz parte do pacote *GNU Linear Programming Kit* (GLPK). Esse pacote destina-se à solução de problemas de programação linear (LP) em grande escala, programação inteira mista (MIP) e outros problemas relacionados (FSF, 2012). Trata-se de um conjunto de rotinas escritas em linguagem ANSI C e organizadas na forma de uma biblioteca que pode ser invocada por outros programas. O GLPK disponibiliza, também, uma ferramenta (*solver*) que permite implementar e avaliar modelos de PL (denominado `glpsol`).

A GNU MathProg é uma linguagem de alto nível para a criação de modelos de programação matemática. Embora seja específica para o GLPK, essa linguagem se baseia na em um subconjunto da AMPL (FOURER; GAY; KERNIGHAN, 2003), linguagem proprietária com propósitos semelhantes. A MathProg tem sido comumente referenciada como GMPL (*GNU Mathematical Programming Language*).

2.1 Componentes da Linguagem GNU MathProg

A *MathProg* é uma linguagem de modelagem cuja sintaxe é relativamente simples. Um programa escrito nessa linguagem é composto, basicamente, pelos mesmos componentes de um modelo de PL, ou seja (CERON, 2006):

- Variáveis de decisão;
- Função objetivo;
- Restrições;
- Parâmetros (ou dados).

As variáveis de decisão atuam como as incógnitas do problema, ou seja, termos cujos valores serão obtidos pela solução do modelo. A solução do problema é dada por suas características em conjunto: função objetivo e restrições, que define a qualidade da solução em função das variáveis de decisão. Os limites da região de viabilidade são estabelecidos pelas restrições, ou seja, inequações e equações permitem ao modelo levar em conta as limitações físicas do sistema. Finalmente, os parâmetros são os valores imutáveis do problema, ou seja, dos dados de entrada — valores que permanecem fixos até a solução do problema.

2.2 Exemplos

Para ilustrar o funcionamento da linguagem MathProg, nesta seção apresenta-se um exemplo, adaptado de Winston (2002 apud CERON, 2006). Trata-se do problema da Marcenaria de Gepeto, descrito a seguir.

> **Exemplo 2.2:** Marcenaria de Gepeto fabrica dois tipos de brinquedos de madeira: soldados e trens. Um soldado é vendido por \$27,00 e usa \$10,00 em matéria-prima. Cada soldado fabricado aumenta a mão de obra variável e os custos gerais do Gepeto em \$14,00. Um trem é vendido por \$21,00 e usa \$9,00 em matéria-prima. Cada trem construído aumenta os custos gerais de trabalho e despesas do Gepeto em \$10,00.
>
> A fabricação de soldados e trens de madeira requer dois tipos de mão de obra especializada: carpintaria e acabamento. Um soldado requer 2 horas de trabalho de acabamento e 1 hora de trabalho de carpintaria. Um trem requer 1 hora de acabamento e 1 hora de trabalho de carpintaria. A cada semana, a Gepeto consegue obter toda a matéria-prima necessária, mas apenas 100 horas de acabamento e 80 horas de carpintaria.
>
> A demanda por trens é ilimitada, mas no máximo 40 soldados são comprados por semana. Gepeto quer maximizar os lucros semanais (ou seja, a diferença entre receitas e custos).

A partir do Exemplo 2.2, é possível modelar duas variáveis de decisão, x_1 e $x2$, sendo que x_1 é a quantidade (unidades) de soldados produzidos a cada semana e x_2 é a quantidade de trens produzidos a cada semana. Tendo conhecido as variáveis de decisão, pode-se modelar a função objetivo para o problema, que será a maximização da receita menos os custos para cada objeto. Assim:

$$\text{maximizar} \quad z = (27 - 10 - 14)x_1 + (21 - 9 - 10)x_2 \tag{2.1}$$

De maneira simplificada:

$$\text{maximizar } z = 3x_1 + 2x_2 \tag{2.2}$$

Após isso, é necessário analisar as restrições do problema. Observa-se, pelo enunciado, que existem três restrições:

Restrição 1: no máximo 100 horas de acabamento, sendo que cada soldado demanda duas horas dessa tarefa, ou seja:

$$2x_1 + x_2 \leq 100 \tag{2.3}$$

Restrição 2: no máximo 80 horas de carpintaria:

$$x_1 + x_2 \leq 80 \tag{2.4}$$

Restrição 3: no máximo 40 soldados são comprados semanalmente:

$$x_1 \leq 40 \tag{2.5}$$

Além dessas três restrições, é necessário garantir a não-negatividade, ou seja, $x_1 \geq 0$ e $x_2 \geq 0$. Portanto, o Exemplo 2.2 fica modelado da seguinte maneira:

$$\text{maximizar} \quad z = 3x_1 + 2x_2$$

$$sujeito\ a: \begin{cases} 2x_1 + x_2 \leq 100 \\ x_1 + x_2 \leq 80 \\ x_1 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$$

Tendo o modelo, é possível construir um programa na linguagem MathProg, conforme mostrado no código-fonte da Figura 1. Nessa figura, são apresentados diversos elementos da sintaxe da linguagem, listados a seguir:

1. Comentários, os quais são delimitados por `/* */`, no caso de comentários de bloco, ou iniciados por `#` para comentários de linha.
2. Dois pontos `:`, utilizado para definir as funções, separando seu nome de sua definição.
3. Comando `s.t.`, do inglês *subject to* ("sujeito a"), utilizado para definir cada uma das restrições. (Apesar de não serem obrigatórios, é aconselhável ter esses marcadores para melhor legibilidade do código.)
4. Ao final do código, deve-se inserir a palavra reservada `end;`.
5. No caso da aplicação mostrada aqui, a palavra reservada `solve` dá início à resolução do problema.

Como é possível observar, um programa escrito nessa linguagem é uma transcrição bastante próxima ao que é construído em um modelo matemático. Nesse exemplo, nas linhas 8 e 9 são definidas as variáveis de decisão; além disso, define-se, também, o tipo dessas variáveis — no caso do Exemplo 2.2, essas variáveis têm que assumir valores inteiros (`integer`) positivos. A função objetivo é definida e nomeada na linha 12 e as três restrições são criadas nas linhas 15, 16 e 17. Os nomes atribuídos a elas serão utilizados como saída pelo *solver* para exibir os resultados (CERON, 2006; SOTTINEN, 2009).

Figura 1 – Modelo em GNU MathProg para Exemplo 2.2.

```
1  /*
2   * Exemplo Gepeto
3   *
4   * Encontre a solução ótima para maximizar o lucro
5   */
6
7  # Variáveis de decisão
8  var x1 >= 0 integer;     # soldado
9  var x2 >= 0 integer;     # trem
10
11 # Função objetivo
12 maximize profit: 3*x1 + 2*x2;
13
14 # Restrições
15 s.t. acabamento: 2*x1 + x2 <= 100;
16 s.t. carpintaria: x1 + x2 <= 80;
17 s.t. demanda: x1 <= 40;
18
19 solve;
20 end;
```

Fonte: adaptado de Ceron (2006).

Devido à sua relativa simplicidade, a linguagem GNU MathProg tem um importante valor educacional. De fato, essa linguagem tem sido, muitas vezes, utilizada em sala de aula como o intuito de ilustrar a resolução de problemas de uma maneira mais direta (SOTTINEN, 2009). Diante disso, este TCC propõe o desenvolvimento de uma aplicação web que permita ao usuário executar essa linguagem sem a necessidade de instalar o GLPK em suas máquinas; como consequência, sua utilização torna-se menos restrita.

3 Desenvolvimento

Neste capítulo são apresentadas as decisões de projeto que levaram ao desenvolvimento da ferramenta proposta neste trabalho. Com base nessas decisões, estabeleceram-se os requisitos da ferramenta e seus casos de uso. Além disso, foram escolhidas as ferramentas de desenvolvimento a serem utilizadas.

3.1 Decisões de Projeto

O objetivo deste TCC é o desenvolvimento de uma aplicação web que permita ao usuário implementar modelos na linguagem GNU Mathprog. Ao mesmo tempo, essa aplicação tem que ser acessível, demandando pouca ou nenhuma configurações por parte dos usuários. Por esses motivos, focou-se no desenvolvimento de uma ferramenta do tipo *Front End*, ou seja, um programa web que é executado direto no navegador, sem a necessidade de servidores locais ou remotos. Por isso, a ferramenta pode ser hospedada em qualquer servidor web. Caso o usuário prefira, pode também efetuar o download da mesa, executando-a localmente.

Por não possuir um *Back End*, ficou decidido que a aplicação não armazenará dados, ou seja, cabe ao usuário manter cópias locais de seus arquivos. Dessa maneira, evita-se o uso de bancos de dados, simplificando ainda mais o processo de configuração da ferramenta. Conforme já mencionado, servidores web foram descartados, já que não haveria a necessidade de alocação de portas HTTP, nem de carregamento de *containers*.

A ferramenta original, na qual este trabalho se baseia (KANTOR, 2016), era implementada utilizando bibliotecas específicas do navegador Google Chrome. Porém, devido a mudanças nas politicas de segurança de navegadores, foi necessário excluir tais bibliotecas (entre elas, a `chrome.fileSystem`, necessária para carregar modelos e permitir seu download), pois algumas funcionalidades foram desativadas. Para manter o desenvolvimento *Front End*, buscou-se alternativas como jQuery e Bootstrap, permitindo a manipulação de arquivos.

Finalmente, toda a interface da ferramenta foi reescrita. Ainda, todos os seus menus e botões foram traduzidos para a língua portuguesa, tornando-a mais acessível a diferentes usuários.

3.2 Requisitos da Aplicação

Antes de iniciar o desenvolvimento, foi necessário realizar o levantamento de requisitos da ferramenta. Na área de Engenharia de Softwares (PRESSMAN; MAXIM, 2016), o levantamento de requisitos é uma das primeiras e mais importantes etapas para o desenvolvimento de um programa de computador, sendo responsável por identificar as principais funcionalidades de uma aplicação.

Este trabalho visa o desenvolvimento de uma ferramenta para auxiliar discentes que estejam estudando ou tenham interesse em Pesquisa Operacional, auxiliando, assim, na resolução de modelos de otimização linear. Tendo esse objetivo em mente, foram elencados diversos requisitos e, a partir deles, foi dada continuidade no processo de desenvolvimento — sempre considerando os requisitos como a base para implementação e usabilidade da aplicação.

3.2.1 Requisitos funcionais

Requisitos funcionais são definições dos comportamentos e tarefas indispensáveis da aplicação (PRESSMAN; MAXIM, 2016). Esses requisitos especificam quais funcionalidades devem ser implementadas para que a aplicação seja útil. Em outras palavras, são os objetivos a serem cumpridos no desenvolvimento. Neste trabalho, foram elencados doze requisitos funcionais, os quais são apresentados e descritos a seguir:

RF001 Carregar um modelo pronto.

O usuário da aplicação deve ser capaz de abrir um modelo existente no seu diretório a qualquer momento.

RF002 Editar um modelo.

O sistema deve permitir que o usuário edite um modelo na interface disponibilizada para inserção de problemas.

RF003 Salvar um modelo.

O usuário poderá fazer o download do modelo que editou a qualquer momento. O arquivo a ser baixado deverá ter um nome estabelecido pelo usuário seguido da extensão extensão "`.mod`", que é o padrão de arquivo na linguagem MathProg.

RF004 Criar novo modelo.

O sistema deve oferecer ao usuário a opção de criar um novo modelo a qualquer momento mediante as restrições de: o console estar limpo ou, caso contrário, confirmar ação solicitada pelo usuário (e, assim, sobrescrever o conteúdo visível na tela).

RF005 Resolver um programa.

O sistema deve exibir na tela um botão que possibilite a resolução do modelo em

qualquer momento da sua construção. Caso o modelo não esteja pronto, ou possua erros, serão exibidas mensagens de erro.

RF006 Nomear um modelo.

O usuário terá que nomear o seu modelo de modo que possar criar um arquivo para armazená-lo.

RF007 Exibição de exemplos.

É necessário ter uma aba com exemplos de modelos já conhecidos e ao selecionar algum, será preenchido todo o código desse modelo no editor. Isso é importante em uma ferramenta utilizada para fins didáticos.

RF008 Saída de dados.

O programa deverá fornecer um console com saída da resolução do modelo, seja apresentando sua solução, seja exibindo mensagens de erro e alertas.

RF009 Exibir dados.

O programa deverá fornecer no console os dados do problema resolvido.

RF010 Exibir variáveis.

O programa deverá fornecer no console os valores de todas as as variáveis utilizadas na criação do modelo.

RF011 Exibir restrições.

O programa deverá fornecer no console as restrições implementadas no modelo, mostrando se elas foram respeitadas.

RF012 Registros da aplicação.

O programa deverá fornecer no console os registros (*logs*) da aplicação, mostrando todas as etapas de resolução do GLPK.

3.3 Requisitos não-funcionais

Os requisitos não-funcionais não estão relacionados, diretamente, com funcionalidades do sistema (PRESSMAN; MAXIM, 2016). Tratam-se, no entanto, de condições que a aplicação deve obedecer em termos de qualidade de uso, gerenciamento e contexto de ambientes, podendo, ou não ser especificados pelo usuário. No contexto deste trabalho, foram estabelecidos três desses requisitos:

RNF001 Compatibilidade.

O programa deverá funcionar nos principais (senão todos) navegadores disponíveis atualmente, independente do sistema operacional.

RNF002 Código aberto.

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

RNF003 Processamento em *Front End*.

A aplicação deve ser desenvolvida para processar seus dados apenas em *Front End*, a fim de simplificar sua utilização e torná-la mais acessível.

3.4 Casos de uso

Uma vez definidos os requisitos da aplicação, é necessário definir seus casos de uso. Na Engenharia de Software, os casos de uso podem ser vistos como um "roteiro" de uso para o sistema, mostrando como esse vai se comportar (PRESSMAN; MAXIM, 2016).

O primeiro passo para a criação de um caso de uso é identificar os atores que atuarão na história (PRESSMAN; MAXIM, 2016). Atores são os diferentes usuários, tanto pessoas como sistemas, que usarão o sistema no contexto da função e comportamento a serem descritos. No caso deste trabalho, os atores serão dois: o discente, que é o usuário da ferramente, e o próprio sistema. A partir disso, são listados quatro casos de uso para a aplicação:

Caso 1: Usuário insere um novo modelo.

O usuário terá na sua tela principal tudo que precisa para começar a criação de um novo modelo. O usuário pode começar digitando sua fórmula matemática na caixa de inserção e pode resolver, nomear e salvar esse modelo. O sistema deverá exibir alguma mensagem caso a execução do modelo dê errado e permitirá o download do arquivo com o nome escolhido e com as informações digitadas no console.

Caso 2: Usuário abre um novo modelo.

O usuário que já tem um modelo feito em sua máquina, poderá abri-lo no console da aplicação, desde que esteja salvo com a extensão `.mod`. Uma vez aberto o modelo, ele poderá ser resolvido, alterado e salvo novamente. O sistema apresentará alguma mensagem de erro se algo estiver errado; caso contrário, resolverá ou fará o download do arquivo com as informações digitadas no console.

Caso 3: - Usuário quer ver um exemplo.

O usuário entra no sistema e escolhe um exemplo na barra de navegação; o sistema, então, irá inserir automaticamente no console as informações solicitadas e o modelo poderá resolvido sem erros, ou poderá ser editado e salvo como o usuário quiser.

Caso 4: Salvar modelo.

O usuário somente terá a opção de armazenar o modelo salvando-o, ou seja, o sistema

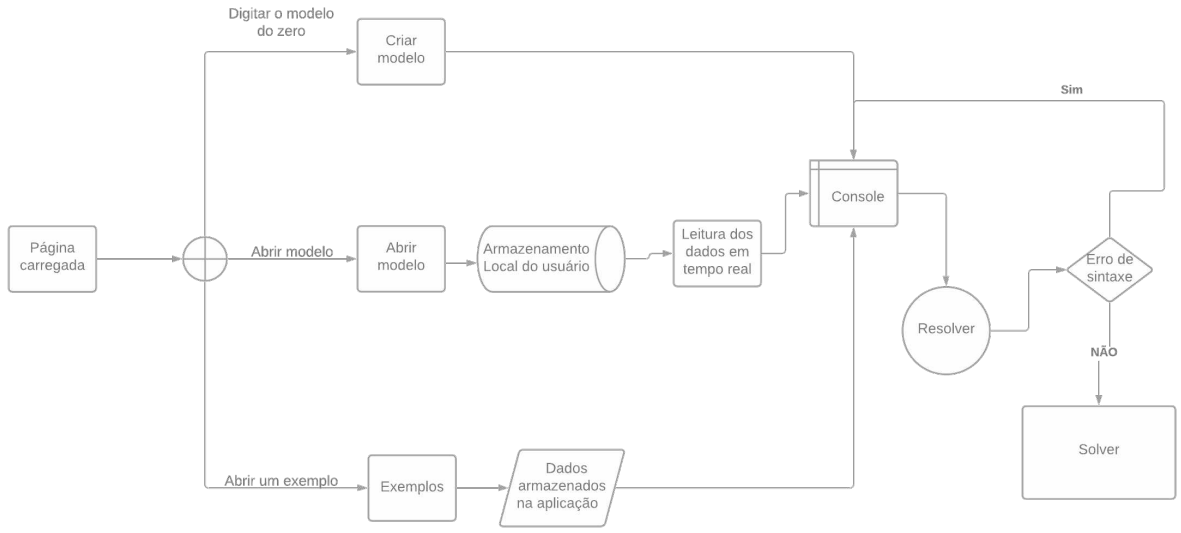
não permite armazenamento de dados no servidor. O usuário solicita ao sistema que o modelo seja salvo e o sistema faz o download do modelo escolhido.

3.5 Modelagem do Sistema

Após a definição dos requisitos e dos casos de uso, foram projetados dois fluxogramas. Esses permitem entender o funcionamento do sistema, apresentando todos o seu funcionamento (ou seja, exibindo fim-a-fim o que se passa no sistema).

A Figura 2 apresenta o fluxograma que ilustra a criação e execução de um modelo. Já na Figura 3, é exibido o fluxo relacionado ao salvamento de um modelo, ou seja, as etapas necessárias para efetuar seu download. Ambos fluxogramas foram construídos utilizando a ferramenta online[1] *lucid.app*.

Figura 2 – Fluxograma de criação e resolução de modelo.



Fonte: autoria própria.

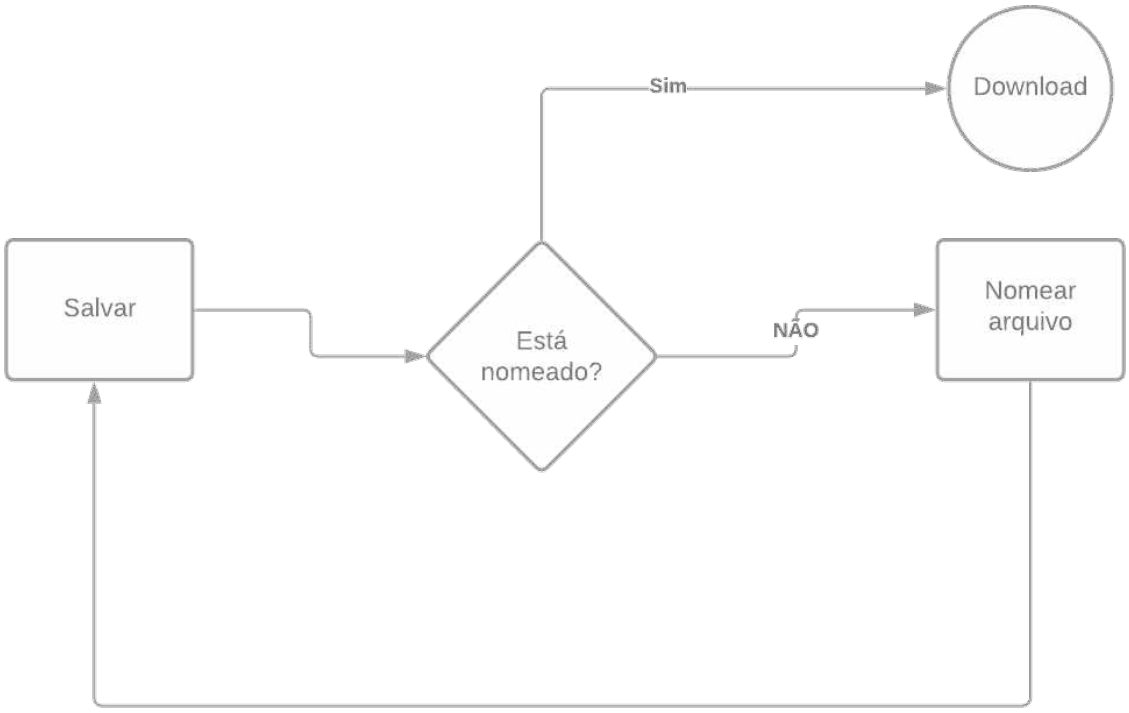
3.6 Ferramentas Utilizadas

Conforme mencionado, para este projeto foram utilizadas apenas linguagens de *Front End*, de modo que o usuário tenha maior facilidade no momento de usufruir de seus serviços. Nesse sentido, optou-se por utilizar as seguintes ferramentas:

- HTML
- CSS

[1] <https://lucid.app/>

Figura 3 – Fluxograma de salvamento do modelo.



Fonte: autoria própria.

- JavaScript
- Bootstrap
- jQuery

Essas ferramentas são descritas com mais detalhes nas próximas seções. Além delas, foi utilizada uma implementação da biblioteca GLPK feita em JavaScript[2], a qual tornou-se a a base para o desenvolvimento da aplicação aqui proposta.

3.6.1 HTML

O HTML (MDN, 2019b) (do inglês, *HyperText Markup Language*, ou linguagem de marcação de hipertexto), é uma linguagem de marcação utilizada para apresentar informações no navegador. Os navegadores traduzem essas marcações em elementos de tela para melhor visualização do usuário. Uma estrutura bem simplificada de um código HTML é mostrada na Figura 4.

Neste projeto, essa linguagem foi utilizada para criar uma página contendo todos os componentes necessários para criar e exibir o modelo. Nessa página, foi inserido o

[2] Disponível em <https://hgourvest.github.io/glpk.js/> (Acesso em 23 nov. 2020).

Figura 4 – Exemplo de HTML.

Fonte: autoria própria.

console onde será apresentada a saída do modelo e, utilizando os recursos da linguagem, foram modelados os botões, caixa de texto, estruturas de divisão de página, alertas e demais elementos da página.

3.6.2 CSS

O CSS (MDN, 2019a) (do inglês, *Cascading Style Sheets*, ou folhas de estilo em cascata), é uma linguagem para estilização de conteúdos web. Neste trabalho, o HTML é utilizado apenas para trazer os elementos da página; todos os elementos de estilo estão contidos no CSS, de modo a simplificar o desenvolvimento da aplicação. Além disso, optou-se por implementar o CSS em um arquivo separado, independente do arquivo HTML; com isso, a manutenção do código se torna mais simples. Nesse caso, o arquivo HTML passa a ter uma referência para o arquivo CSS. Na Figura 5 é apresentado um trecho de código do CSS implementado para a aplicação mostrada neste trabalho.

O CSS é de suma importância para aprimorar a experiência do usuário, visto que temos ajustes de layout, dispondo os elementos de maneira mais amigável ao leitor, cores que ajudam a identificar melhor a função de cada componente, bem como o fundo da página que tem a coloração pensada para não ser agressiva aos olhos do usuário.

Neste trabalho, foram feitas várias tratativas no arquivo CSS para que sua manutenção fique o mais fácil possível. Nesse sentido, alguns elementos de estilo definidos por Kantor (2016), que estavam embutidos no código HTML, foram transcritos para o CSS, sendo removidos do código original.

Figura 5 – Exemplo de CSS.

Fonte: autoria própria.

3.6.3 JavaScript

O JavaScript (MDN, 2020) é uma linguagem multi-paradigma (orientada a objetos, imperativa e funcional) que serve para controlar ações do lado do cliente (ou seja, *Front End*) e permite implementar itens complexos em páginas web. Essas páginas podem, assim, ser dinâmicas e atualizarem informações em tempo real.

O JavaScript controla as ações do sistema de acordo com a ação do usuário. Em conjunto com o jQuery (descrito a seguir), é responsável por gerir todo o comportamento da página, desde a ação de pressionar um botão, até a resolução do modelo. Como o JavaScrip é todo controlado no cliente, ele permite executar funções sem a necessidade de um *Back End*. Com isso, não é preciso ter um servidor dedicado à aplicação desenvolvida, o que facilita seu uso da aplicação e a torna mais acessível.

3.6.4 jQuery

O jQuery (JQUERY, 2020) é uma biblioteca do Javascript que auxilia o desenvolvimento de tarefas complexas, tornando-as mais simples, rápidas e compatíveis com diferentes navegadores. Com ela, é possível utilizar recursos mais concisos e de usabilidade mais simples. O jQuery é uma biblioteca de código aberto e foi desenvolvida por John Resig (JQUERY, 2020) para tornar mais simples a manipulação do HTML, a seleção de elementos DOM (*Document Object Model*), manipulação de CSS, efeitos e animações, métodos de eventos do HTML, AJAX e outras funcionalidades genéricas.

O jQuery não fazia parte da implementação original (KANTOR, 2016), tendo sido a principal adaptação deste trabalho. Sua introdução no sistema tornou mais fácil o desenvolvimento de algumas funcionalidades, como carregamento e salvamento de modelos.

3.6.5 Bootstrap

O Bootstrap (BOOTSTRAP, 2020) é um *framework* de estilização que o desenvolvimento de aplicações web responsivas. Assim, o desenvolvedor não precisa criar diversos códigos em CSS para a implementação do que lhe é proposto. Sem o Bootstrap, seria necessário usar um recurso do CSS chamado *Media Queries*, que permite aplicar um estilo diferente para cada tamanho de tela. Porém, o Bootstrap traz a facilidade de executar essa tarefa sem aumentar demasiadamente a complexidade do CSS.

Além disso, o Bootstrap possui componentes em JavaScript (mais especificamente jQuery) que ajudam o desenvolvedor a criar componentes da aplicação, como menus, molduras e apresentações sem grandes dificuldade, apenas acrescentando algumas configurações no código.

4 Resultados

Neste capítulo, são apresentados exemplos que ilustram o funcionamento da aplicação e, consequentemente, validar os requisitos levantados na Seção 3.2, bem como os casos de uso listados na Seção 3.4.

4.1 Primeiro Exemplo

O primeiro caso de teste mostrado aqui é a resolução do Exemplo 2.2, visto no Capítulo 2, ou seja:

$$\text{maximizar} \quad z = 3x_1 + 2x_2$$

$$sujeito\ a: \begin{cases} 2x_1 + x_2 \leq 100 \\ x_1 + x_2 \leq 80 \\ x_1 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$$

Ao acessar a ferramenta, o usuário irá se deparar com uma interface composta por diferentes módulos, conforme visto na Figura 6. O menu superior contém as funcionalidades gerais, atendendo aos requisitos (esse menu será detalhado na Seção 4.2). Logo abaixo, há uma caixa onde será inserido o nome do arquivo para ser salvo; ainda mais abaixo ainda, a caixa de texto principal, onde será escrito todo problema a ser resolvido. Na parte inferior dessa caixa, tem um botão para criar um novo modelo (essa opção limpa o conteúdo da caixa de texto), outro para abrir um modelo existente do dispositivo do usuário (faz *upload* do arquivo) e um para salvar o modelo escrito na caixa de texto. Finalmente, observa-se o botão mais importante que é o de resolver o problema.

Para criar um novo modelo, o usuário deve preencher a caixa de texto com o programa escrito em GNU MathProg, conforme ilustrado na Figura 7. Com o programa matemático transcrito para o console, é possível resolvê-lo clicando no botão “Resolver”. Em seguida, a ferramenta exibirá o “Painel de Controle”, logo abaixo, mostrado o valor da função objetivo (no caso, $180,00), das variáveis de decisão e das restrições (conforme mostrado na Figura 8). Observa-se que a solução é ótima, ou seja, obteve-se um valor sem violar nenhuma restrição.

No caso do problema do Gepeto, não foi utilizado no modelo nenhum comando de saída, logo, a aba “Saída” não será preenchida. (A linguagem MathProg possui comandos como `printf`, por exemplo, para exibir mensagens aos usuários.) Analogamente, são se utilizou dados adicionais para o modelo, de modo que a aba “Dados” também não é utilizada nesse exemplo.

Figura 6 – Tela inicial vazia

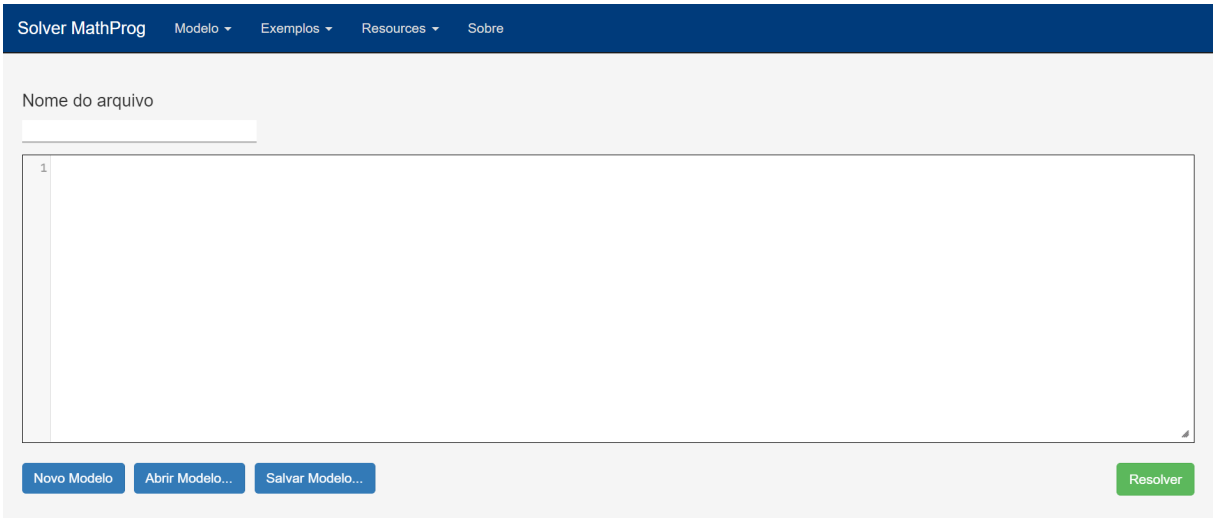
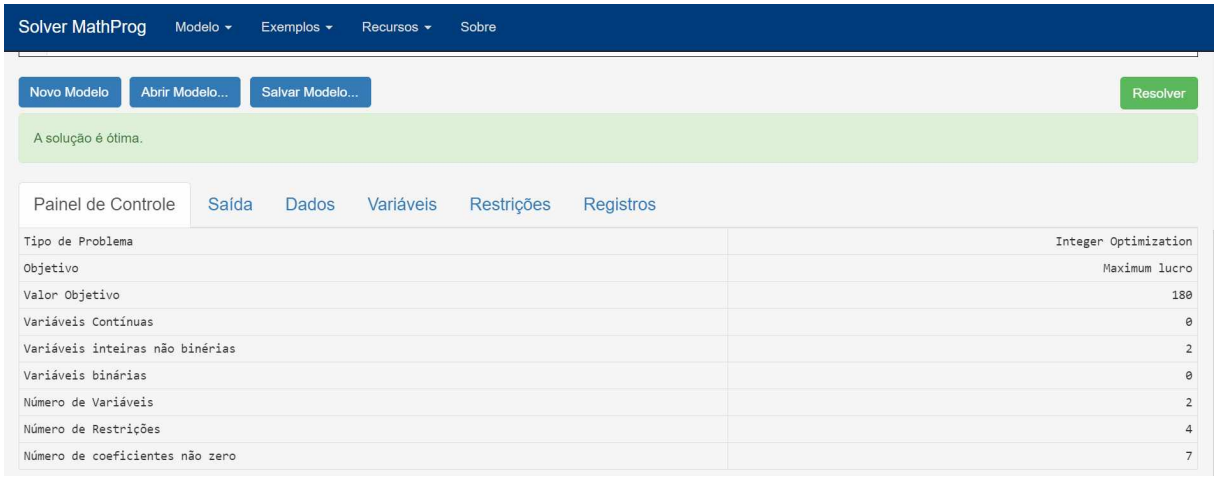


Figura 7 – Implementação do exemplo 2.2 em linguagem GNU MathProg

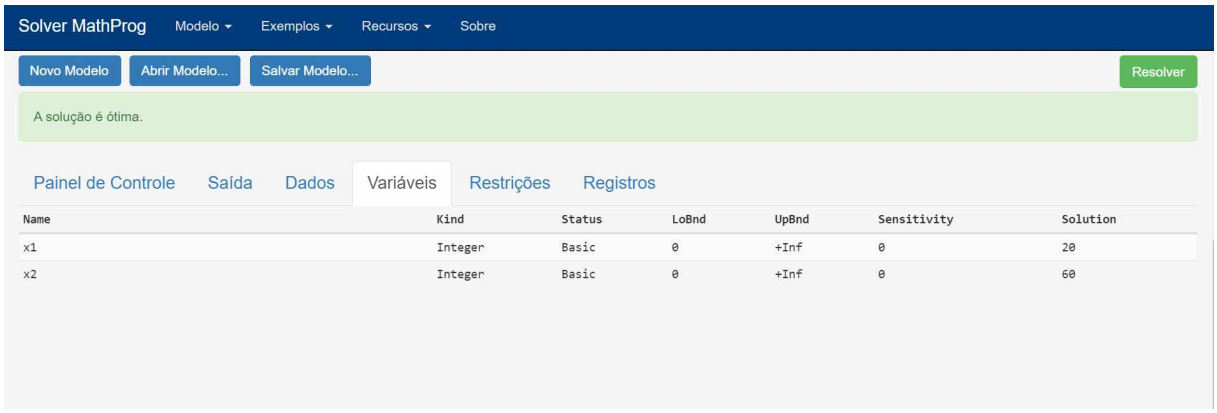
```
/*
# Problema Giapetto's

Encontre a solução ideal para maximizar o lucro do Giapetto
*/

# Variáveis de Decisão
var x1 >=0 integer;  # soldado
var x2 >=0 integer;  # trem

# Função objetivo
maximize profit: 3*x1 + 2*x2;

# Restrições
s.t. acabamento: 2*x1 + x2 <= 100;
s.t. carpintaria: x1 + x2 <= 80;
s.t. demanda: x1 <= 40;

solve;
```

Figura 8 – Resultado da execução do modelo após clicar em "Resolver"



Por outro lado, as abas "Variáveis" e "Restrições" trarão informações mais detalhadas sobre esses componentes do modelo, conforme mostrado nas figuras 9 e 10, respectivamente.

Figura 9 – Aba variáveis populada.

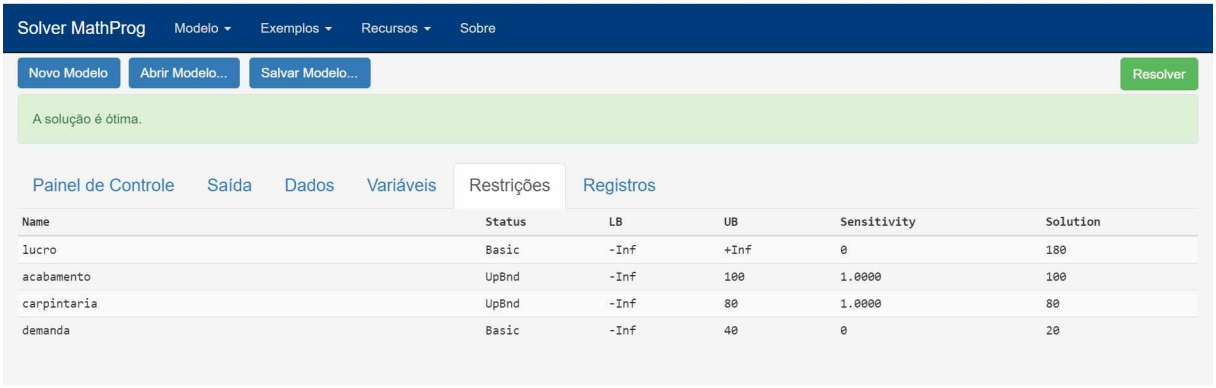


Finalmente, a aba "Registros" traz o *log* da execução do *solver* (Figura 11). Tais informações podem ser utilizadas, por exemplo, para encontrar erros no modelo ou em sua execução.

4.2 Segundo Exemplo

Neste segundo exemplo, é ilustrado o carregamento de um modelo pré-definido, presente na aplicação. Partindo, novamente, da tela inicial (vista na Figura 6), observa-se uma barra de menus superior. Ao passar o cursor do mouse sobre o menu "Exemplos", são exibidos alguns grupos de problemas, separados por tipos (KANTOR, 2016), conforme ilustrado na Figura 12.

Figura 10 – Aba restrições populada.



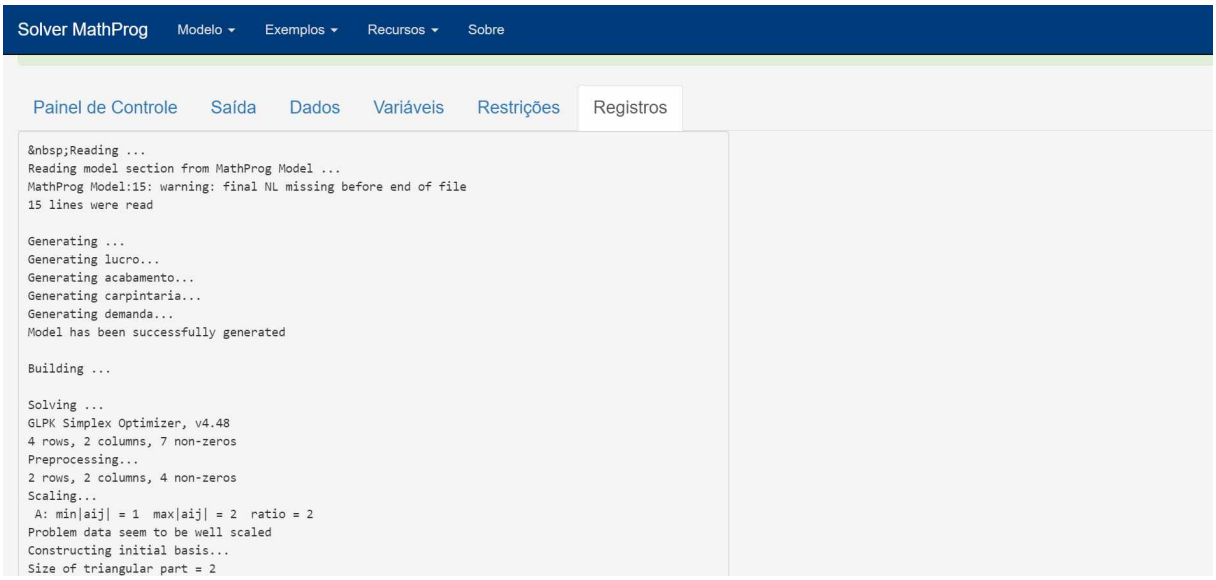


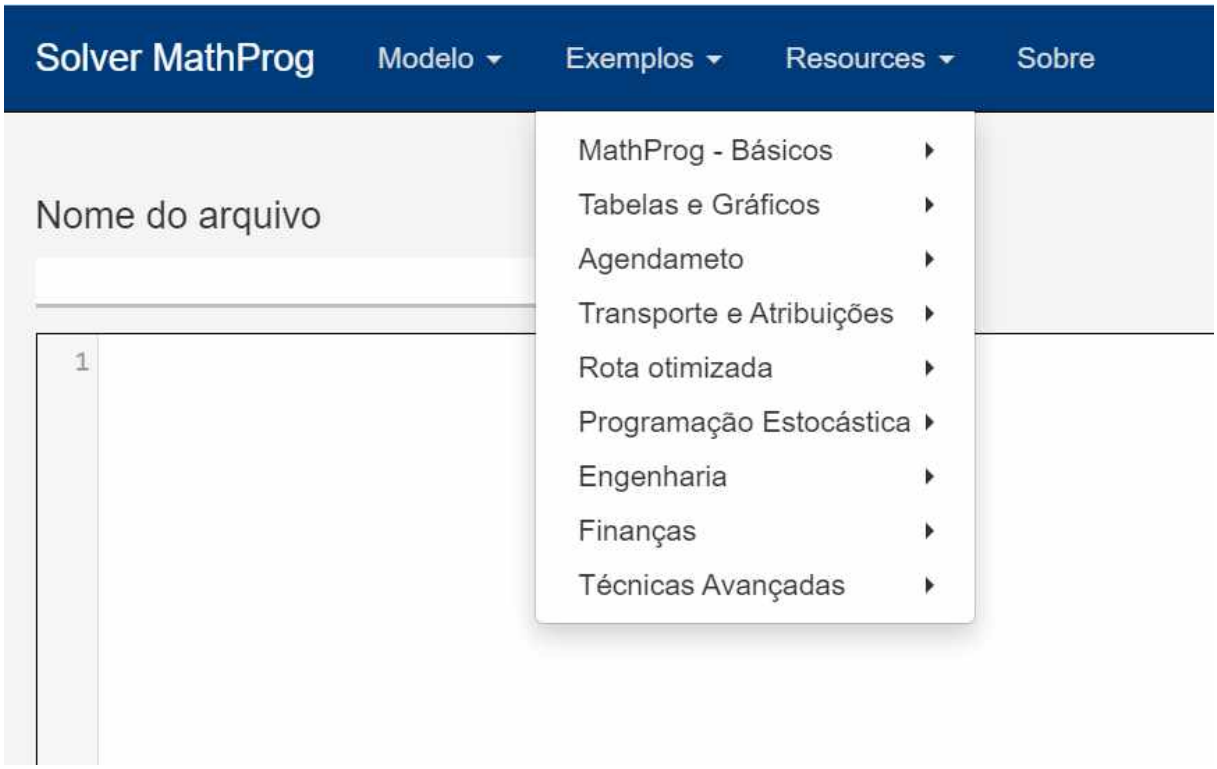
Figura 11 – Aba registros populada.

Agora, com a aba Exemplos selecionada, é possível escolher algum dos grupos apresentados. Passando o cursor sobre o grupo escolhido, o mesmo expandirá e mostrará os problemas que estão contidos nele (Figura 13).

Neste caso de teste, escolheu-se um problema de transporte onde se tem oferta e demanda de produtos, bem como o custo total de se transportar um item do produtor ao consumidor. Ao clicar nesse (Figura 14), automaticamente ele será preenchido na caixa de texto reservada para inserção de problemas (Figura 15).

A partir daqui, todos os passos para resolver o problema são os mesmo. A caixa de texto contém a uma configuração da inicial, porém o modelo pode ser editado. Observa-se que os nomes de variáveis e comentários ainda estão em inglês, pois foram transcritos diretamente do código original (KANTOR, 2016), mas podem ser traduzidos em trabalhos futuros. Após selecionar a opção “Resolver”, a aplicação analisa se a codificação está correta e, sem seguida, resolverá o problema utilizando o *solver* do pacote GLPK (Figura 16).

Figura 12 – Selecionando a aba Exemplos



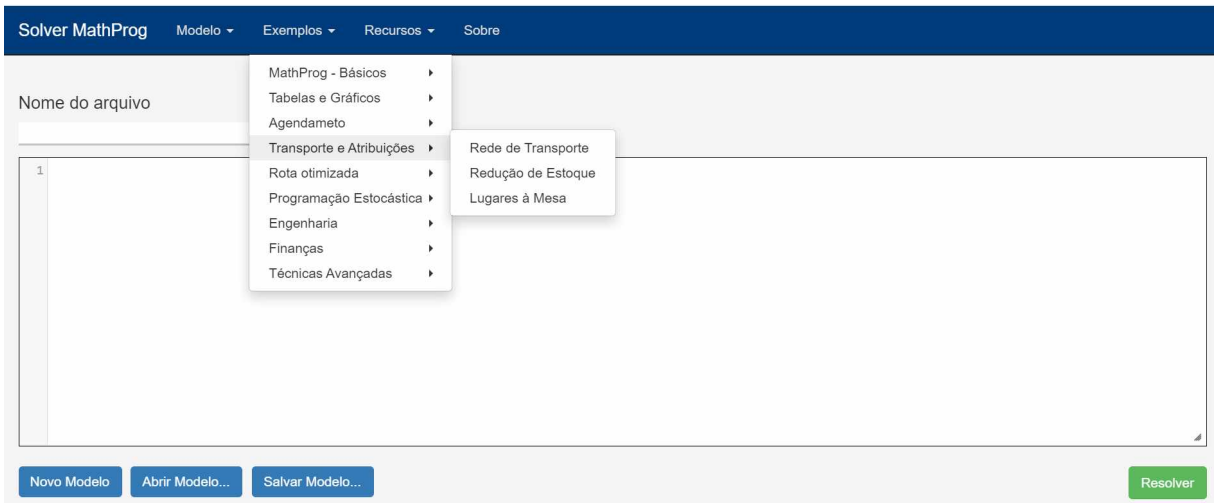


Figura 13 – Escolhemos o grupo 'Transportes e Atribuições'

Figura 14 – Clicando no exemplo escolhido

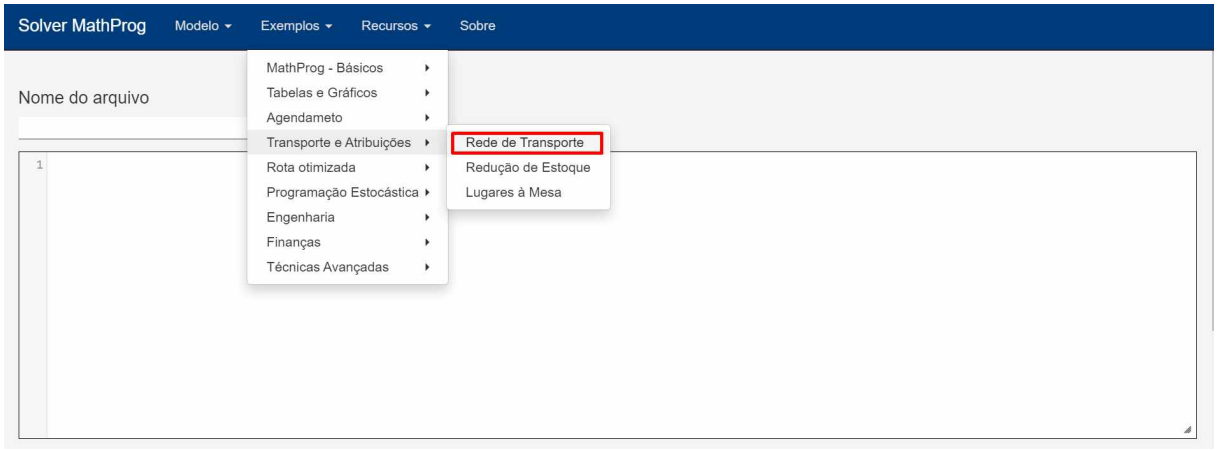
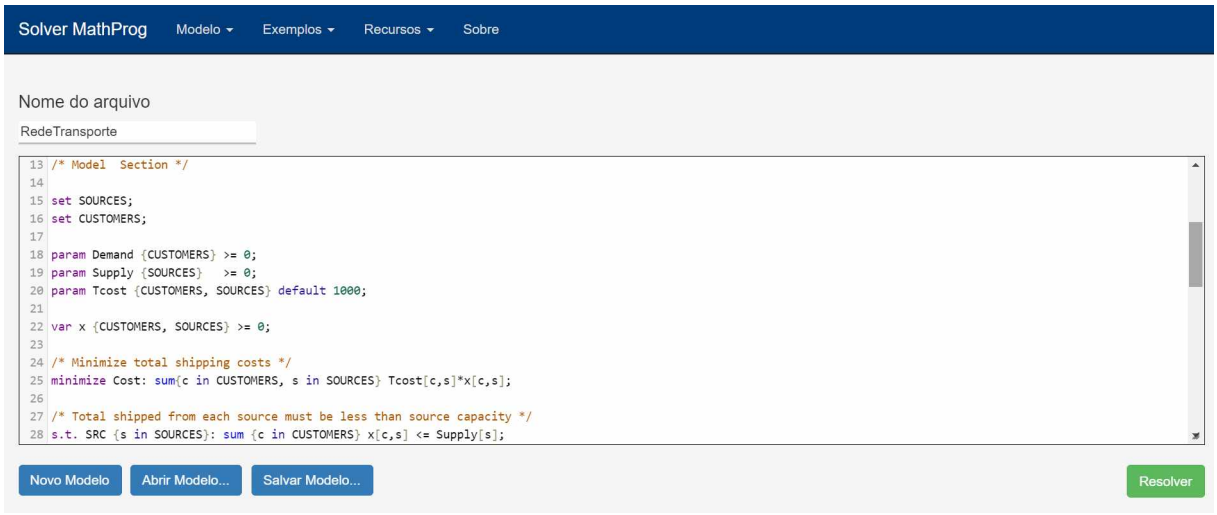


Figura 15 – Console preenchido com o exemplo



4.3 Terceiro Exemplo

O terceiro exemplo ilustra a abertura de um modelo previamente salvo. Para isso, inicialmente, deve-se escolher a opção "Abrir Modelo", disponível abaixo da caixa de texto da aplicação (Figura 17). Caso haja algum caractere na caixa de texto, será aberto um *modal* confirmando a ação do usuário (Figura 18), pois isso fará com que o console seja limpo e todas as alterações perdidas. Essa medida é essencial pois, caso o usuário clique acidentalmente no botão, ele não perderá todo progresso, podendo cancelar a ação.

Nesse momento, tudo que havia sido inserido no console do antigo problema será apagado e será aberta uma janela no navegador para que o usuário escolha o problema que deseja inserir no console (Figura 19). É importante destacar que o arquivo com o

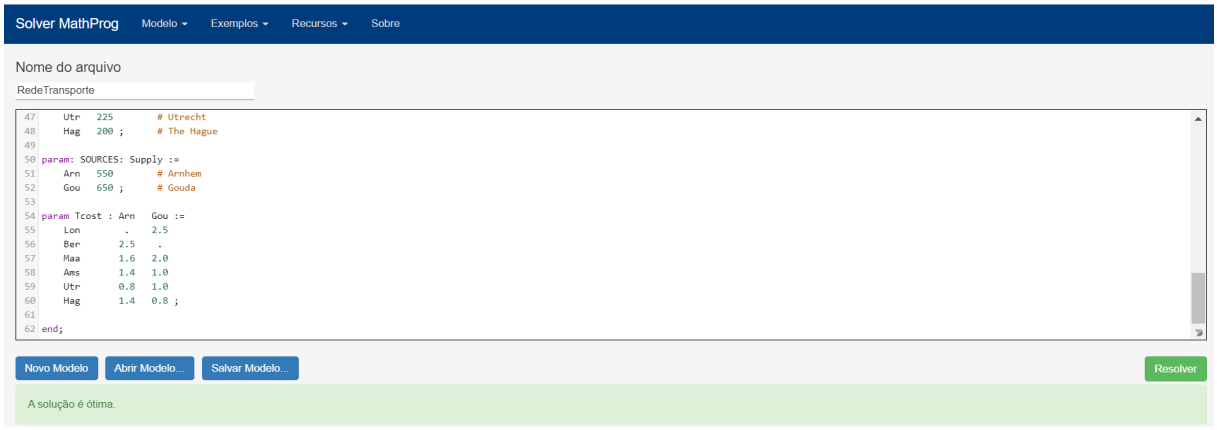
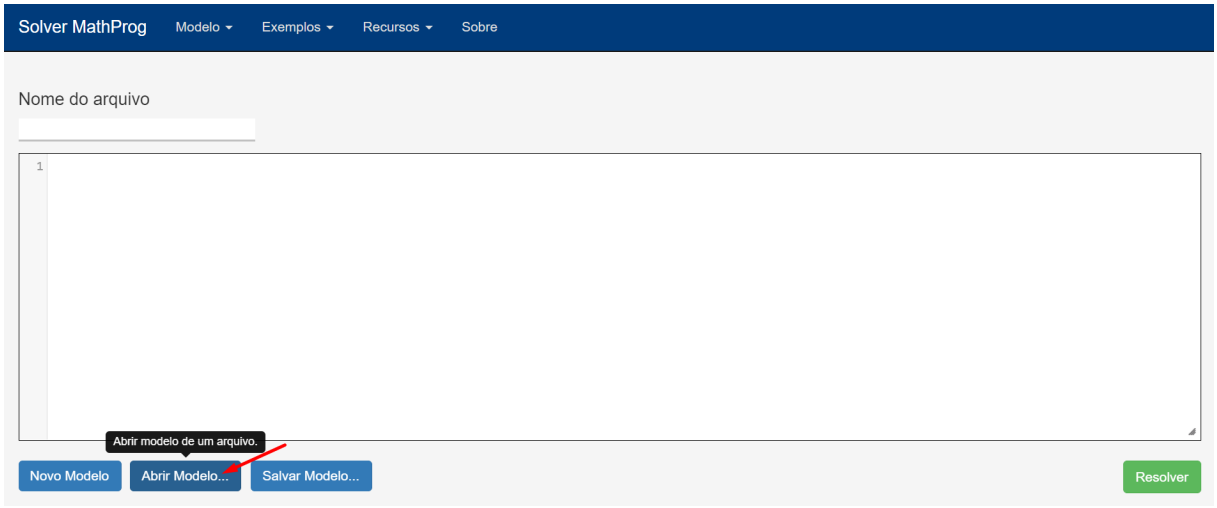


Figura 16 – Exemplo resolvido.

Figura 17 – Clicando em Abrir Modelo.



modelo possui a extensão `.mod`, conforme definido na documentação da linguagem (FSF, 2012). Ao selecionarmos o arquivo documento, é possível clicar em "Abrir" para que ele seja transferido para o console (Figura 20).

Agora, o nosso modelo já está pronto para ser resolvido. Ele será inserido na tela automaticamente, conforme mostrado na Figura 21, e pode ser resolvido seguindo os passos mostrados na Seção 4.1

Também é possível fazer o download do modelo. Inicialmente, é necessário nomear o arquivo; esse requisito visa auxiliar o usuário a não se perder caso faça vários downloads de modelos diferentes. Sem inserir um nome, o arquivo será salvo com um nome genérico o que poderia dificultar o manuseio correto após o mesmo ser salvo. A Figura 22 mostra essa exigência. Já as figuras 23 e 24 ilustram o funcionamento do botão "Salva" e a confirmação de salvamento, respectivamente.

Figura 18 – Modal solicitando confirmação do usuário.

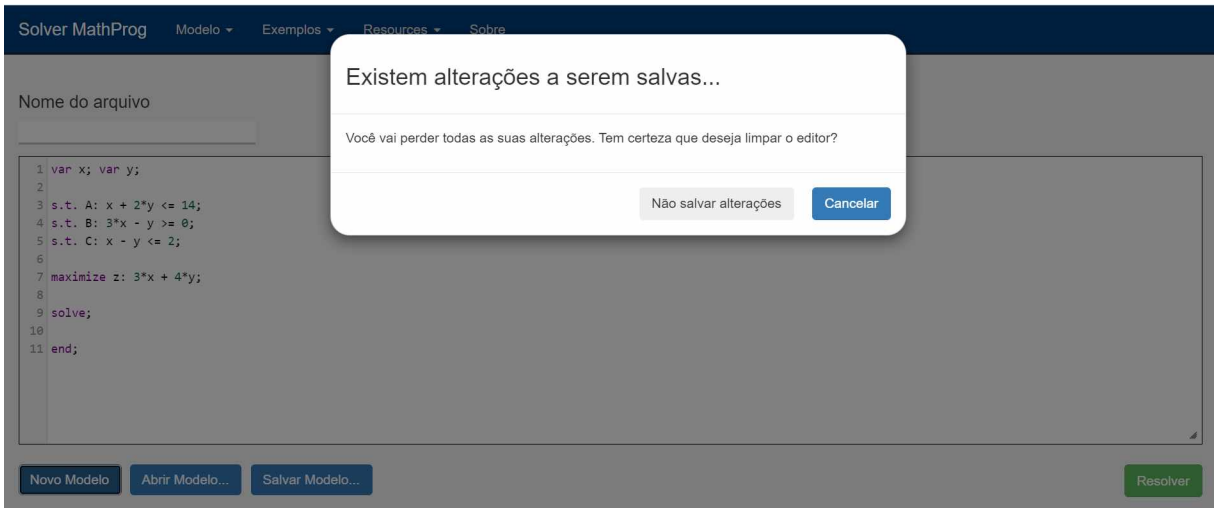
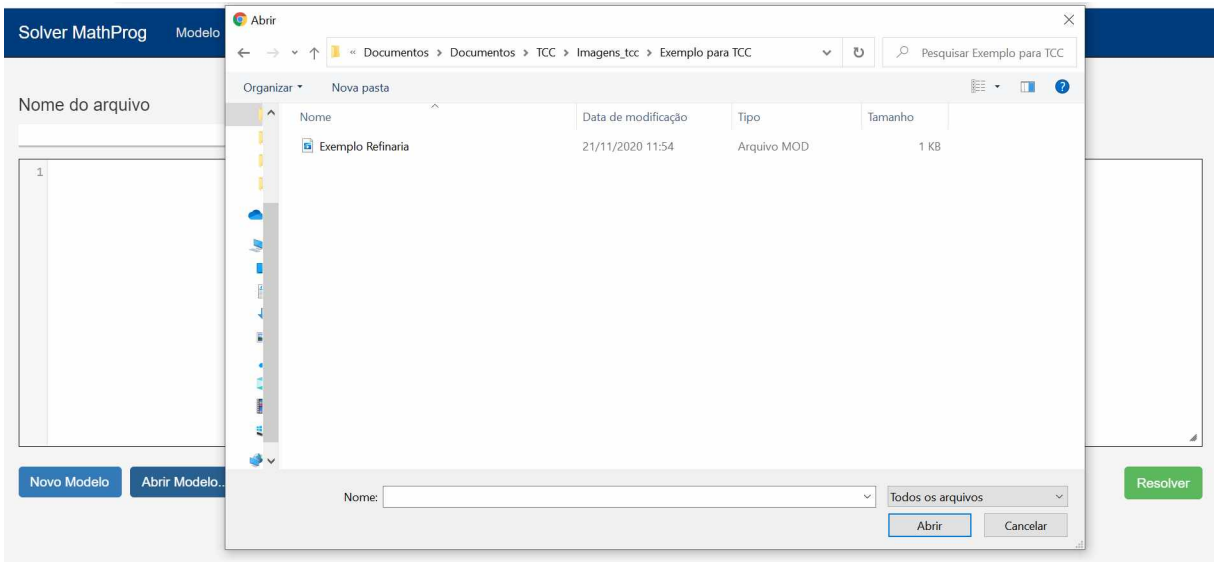


Figura 19 – Janela do Explorer aberta.

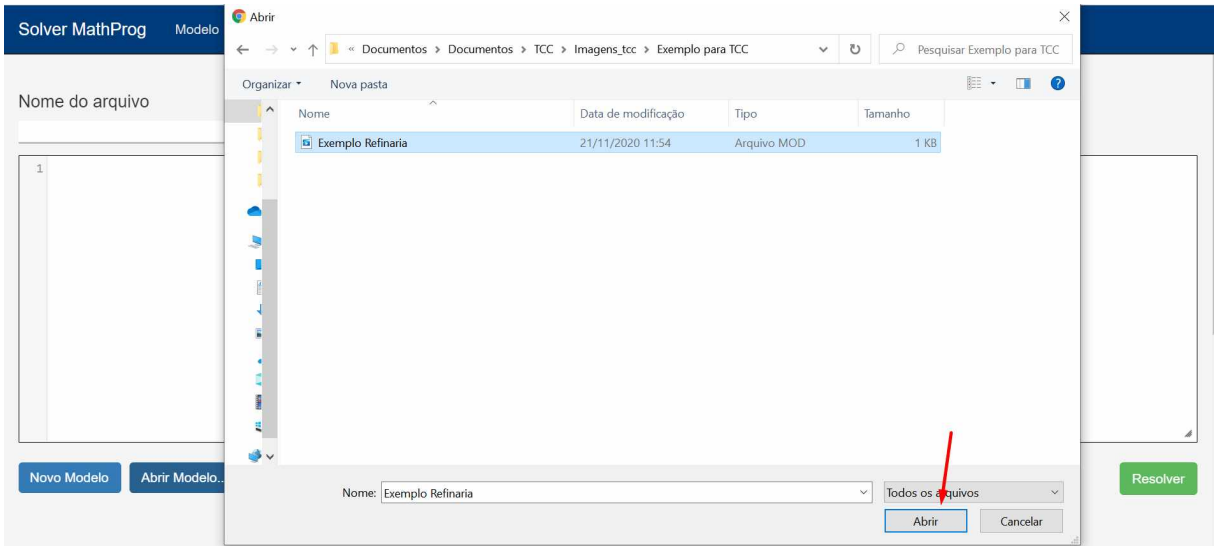


4.4 Quarto Exemplo

Finalmente, apresenta-se um caso para ilustrar que o sistema está preparado para situações onde o modelo inserido não apresenta uma solução ótima. O exemplo a seguir (Figura 25) não tem uma solução viável, pois não existe um valor válido que respeite a restrição. Clicando em "Resolver", aparece uma mensagem notificando essa situação (Figura 26).

Esse modelo ilustra, também, um comando de escrita (função `display`), com o objetivo de visualizar os valores das variáveis. Como esse modelo não tem solução, ambas as variáveis terão valores nulos, conforme pode ser observado na aba "Saída" (Figura 27). Finalmente, na guia "Registro", tem-se toda a execução do *solver*, indicando que o problema

Figura 20 – Achamos o arquivo, vamos abrir.



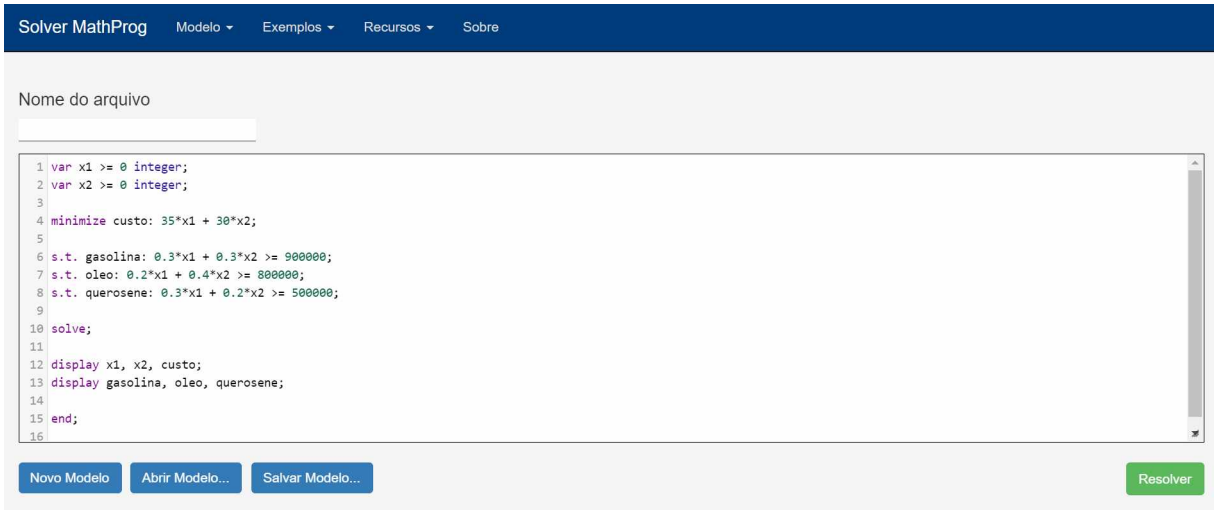


Figura 21 – Arquivo populado no console.

Figura 22 – Deve-se preencher o nome do arquivo.

não possui solução (Figura 28).

4.5 Dificuldades Encontradas

Apesar de ser uma adaptação de um trabalho já existente, foram encontradas diversas dificuldades ao longo do desenvolvimento deste TCC. Além de diversos erros que

Figura 23 – Nome preenchido.

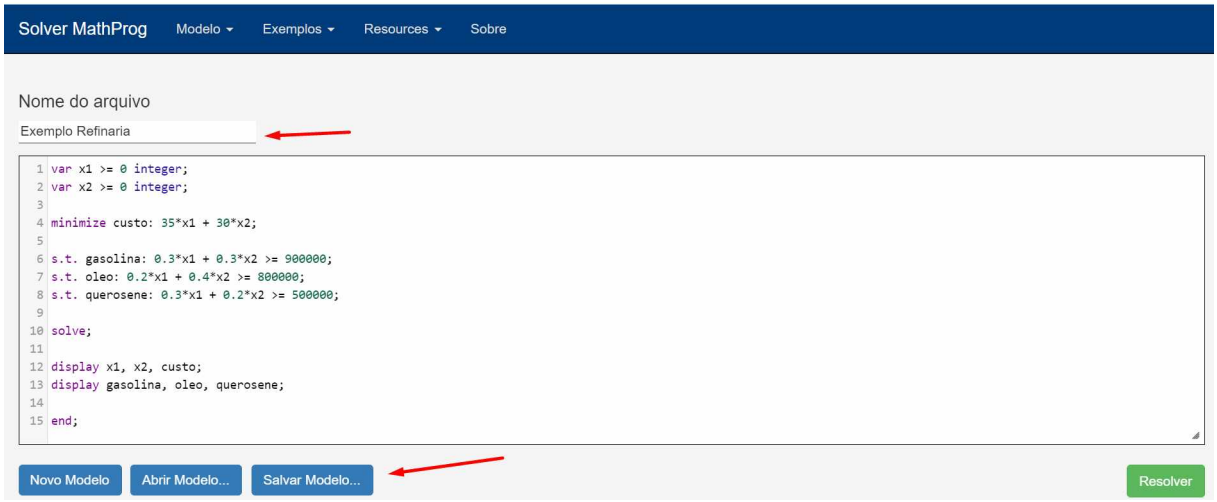
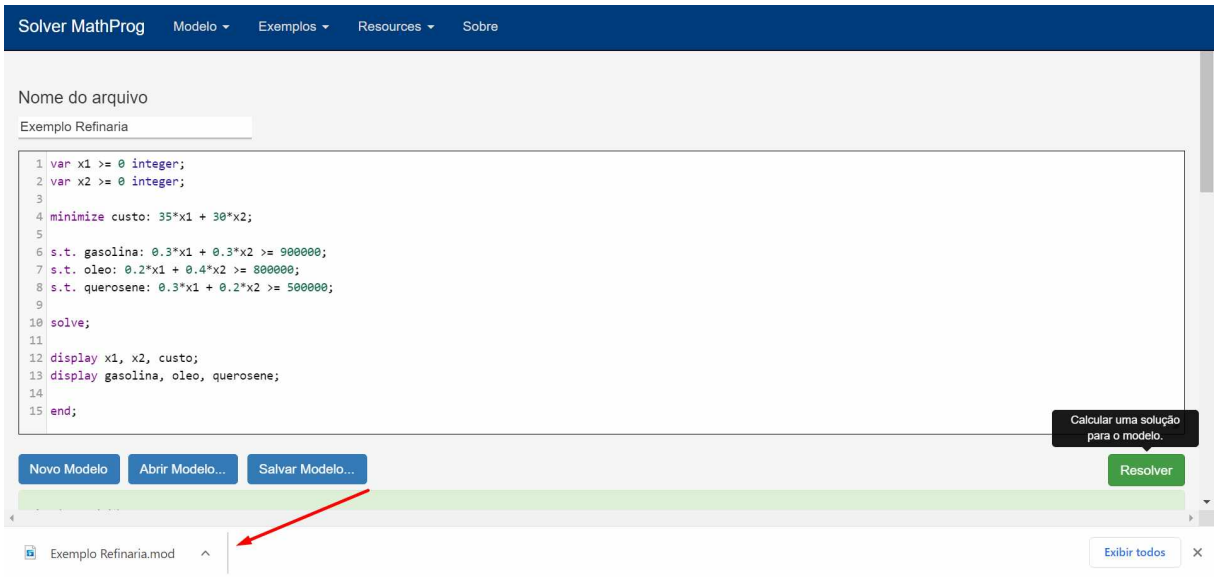


Figura 24 – Dados do problema.



impediam o funcionamento do sistema, foi necessário, também, lidar com o fato de o sistema ter de ser feito todo no *Front End*. Assim, todos os botões, exceto o "Resolver", não executavam nenhuma ação quando acionados, impedindo todas as ações do usuário, como abrir e salvar um modelo, acessar os exemplos e nomear o arquivo. A utilização de uma versão muito antiga do Bootstrap (versão 3.3.4), complicou ainda mais os ajustes. Como é um projeto relativamente grande, tornou-se impraticável substituir o Bootstrap para uma versão mais recente.

Além disso, o código original possuía muitos estilos embutidos no HTML, o que dificultou muito a implantação de boas práticas de programação web. Por esse motivo, retiramos todos estilos embutidos e remanejamos para o arquivo dedicado (CSS) exclusivamente para a finalidade. Foram corrigido, também, todos os erros e pontos de atenção

Figura 25 – Dados do problema.

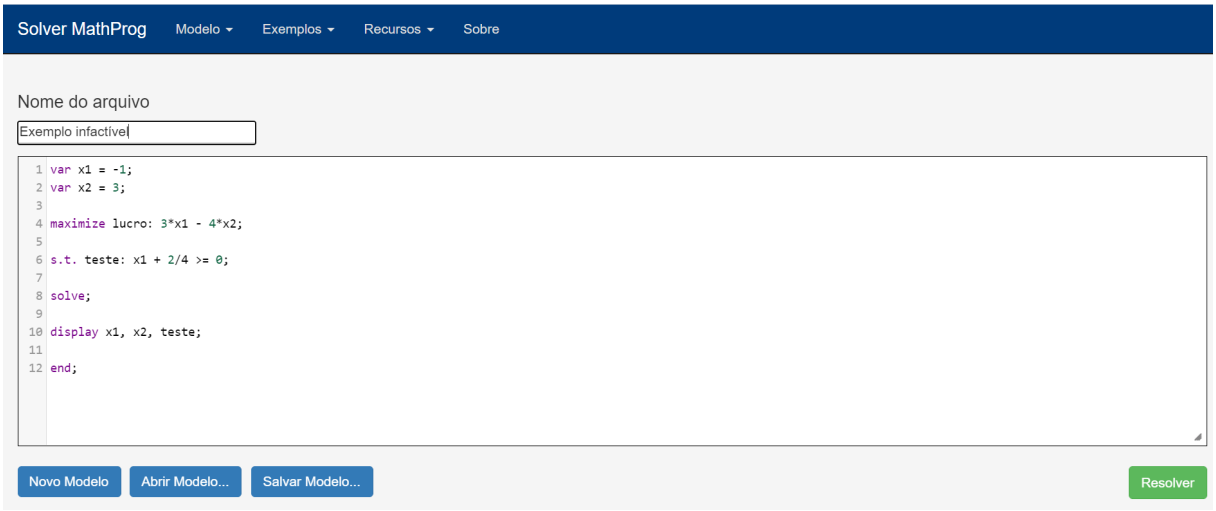
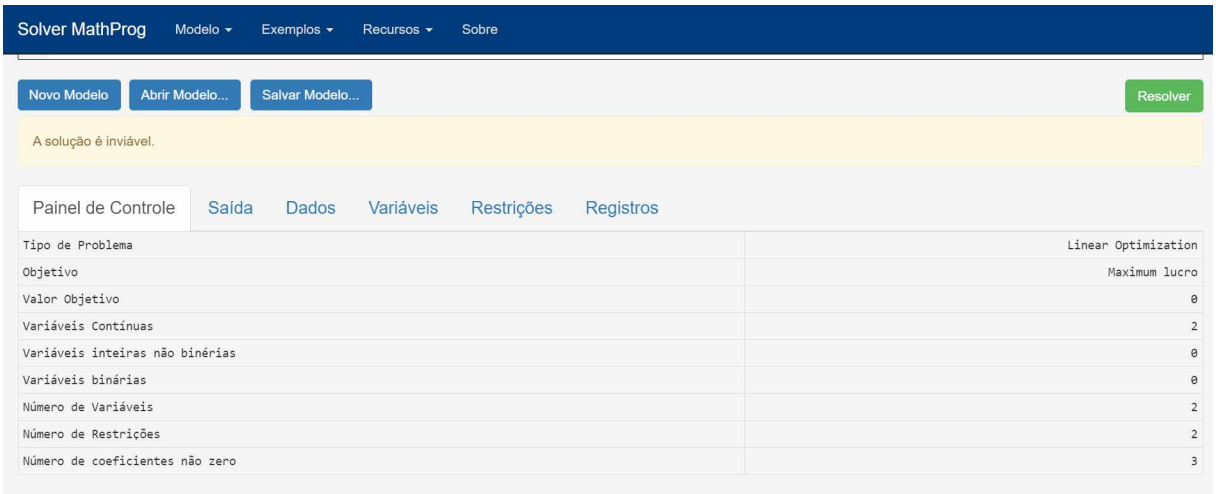


Figura 26 – Solução inviável para o problema.



encontrados no site validador de HTML[1] para nosso arquivo.

Outra grande dificuldade foi a politica de segurança dos navegadores, que não permitem executar algumas tarefas apenas no front-end. Navegadores esperam que haja

[1] <validator.w3.org>

Figura 27 – Saída.

Figura 28 – Registro da resolução do problema.

um servidor que acesse os dados do computador para evitar que haja algum risco para o usuário, como tentativas de invasões e acesso a arquivos sem permissão. Como neste projeto temos a opção de abrir arquivos do sistema, foi necessário criar um mecanismo que seja seguro e que possibilite a ação de ler o arquivo em tempo real e transfira para o console. No caso da aba "Exemplos", era preciso acessar a pasta contendo os exemplos nos arquivos do usuário; porém, com a política de segurança, isso não é mais possível. Para corrigir esse problema, deveria ser criado um servidor web para que o back-end fizesse o acesso ao arquivo no diretório. Como isso viola osn requisitos deste TCC, fizemos a inserção dos arquivos dentro do HTML para que, quando o usuário selecionar o exemplo, possamos pegar via *id* da *tag* e inserir no console da aplicação. Outro problema encontrado foi que a propriedade *chooseEntry*, do pacote *fileSystem* do Google Chrome, muito utilizada no projeto inicial, mas não era mais suportada pelos navegadores.

As considerações feitas acima, convergem para a complexidade adquirida neste projeto, foram feitas grandes mudanças na estrutura do projeto para que atendesse os requisitos sem sair dos padrões das boas práticas de programação bem como a boa experiência do usuário. Apesar das dificuldades, acredita-se ter alcançado bons resultados.

5 Conclusões

Entender e se desenvolver na área de Pesquisa Operacional é de suma importância para todos os estudantes de computação. É preciso ter consciência de que no mundo atual há uma grande necessidade de evolução tecnológica, e isso incita o desenvolvimento de soluções mais eficientes. Neste TCC, desenvolveu-se uma ferramenta de auxílio estudantes de PO, que permite a validação online de modelos em GNU MathProg, além de trazer exemplos que colaborem com o estudo dessa linguagem[1]

No desenvolvimento do projeto, adicionamos funcionalidades a fim de atender às necessidades principais e características consideradas primordiais no primeiro momento. Contudo, existem considerações a serem implementadas no futuro com o objetivo de plurificar o uso do software. Dentre elas, destacam-se:

- Implementar a responsividade do site;
- Adicionar gráficos para melhor visualização das soluções;
- Atualizar versão do Bootstrap para a mais atual;
- Inserir linguagem *server-side* para auxiliar na funcionalidade de abertura de arquivos e armazenamento dos exemplos;
- Traduzir para Português os comentários dos exemplos disponíveis na aplicação;
- Inserir apoio teórico para linguagem GNU MathProg ajudando aos usuários caso tenha alguma dúvida;
- Inserir uma aba de relatório de sensibilidade do GLPK.

[1] O sistema está disponível para uso no endereço: <http://gvdmathprog.atwebpages.com/> e seu código-fonte também está disponível para download ou colaboração na plataforma GitHub através do endereço: <https://github.com/GabrielVieiraDias/MathProg-Solver.git>

Referências

BOOTSTRAP TEAM. *Build fast, responsive sites with Bootstrap*. 2020. Disponível em: <https://getbootstrap.com/>. Acesso em: 23 nov. 2020. Citado na página 22.

CERON, R. *The GNU Linear Programming Kit*: Part 1: Introduction to linear optimization. 2006. IBM Developer. Citado 4 vezes nas páginas 10, 11, 12 e 13.

FOURER, R.; GAY, D. M.; KERNIGHAN, B. W. *AMPL*: A modelling language for mathematical programming. 2. ed. Duxbury: Thonson, 2003. ISBN 0-534-38809-4. Citado na página 10.

FREE SOFTWARE FOUNDATION. *GNU Linear Programming Kit*. 2012. Disponível em: <https://www.gnu.org/software/glpk/>. Acesso em: 18 nov. 2020. Citado 2 vezes nas páginas 10 e 29.

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à Pesquisa Operacional*. 9. ed. Porto Alegre, RS: AMGH, 2013. ISBN 978-8580551181. Citado na página 8.

KANTOR, J. *MathProg Solver*. 2016. Disponível em: <https://github.com/jckantor/MathProg-Solver>. Acesso em: 18 nov. 2020. Citado 7 vezes nas páginas 8, 9, 14, 20, 22, 25 e 26.

MAKHORIN, A. *Modeling Language GNU MathProg*: Language reference for GLPK version 4.58. Moscow, 2016. 74 p. Citado na página 8.

MOZILLA DEVELOPER NETWORK. *CSS*. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em: 23 nov. 2020. Citado na página 20.

_____. *Linguagem de Marcação de Hipertexto*. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 23 nov. 2020. Citado na página 19.

_____. *JavaScrip*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 23 nov. 2020. Citado na página 21.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software*: Uma abordagem profissional. 8. ed. Porto Alegre, RS: AMGH, 2016. 968 p. ISBN 978-8580555332. Citado 3 vezes nas páginas 15, 16 e 17.

SOCIEDADE BRASILEIRA DE PESQUISA OPERACIONAL. *O que é pesquisa operacional?* 2014. Disponível em: <https://www.sobrapo.org.br/o-que-e-pesquisa-operacional>. Acesso em: 18 nov. 2020. Citado na página 8.

SOTTINEN, T. *Operations Research with GNU Linear Programming Kit*. 2009. 200 p. Notas de aula da disciplina ORMS 1020. Citado 2 vezes nas páginas 12 e 13.

THE JQUERY FOUNDATION. *What is jQuery?* 2020. Disponível em: <https://jquery.com/>. Acesso em: 23 nov. 2020. Citado na página 21.

WINSTON, W. L. *Introduction to Mathematical Programming*: Operations research. Stamford: Thomson Learning, 2002. v. 1. 924 p. ISBN 978-0534359645. Citado na página 11.