Pesquisa Operacional para o Desenvolvimento
Vol. 14, e14004, p. 1-15, 2021

ISSN 1984-3534

PIFOP: UMA APLICAÇÃO WEB PARA DESENVOLVIMENTO COLABORATIVO DE MODELOS MATEMÁTICOS EM AMPL[1]

Davi Pereira *, **Ricardo Camargo**

Departamento de Engenharia de Produção
Universidade Federal de Minas Gerais - UFMG, Belo Horizonte - MG, Brasil

Recebido 10/07/2020, aceito 30/07/2021

RESUMO

IDE's (*Integrated Development Environments*) são ferramentas utilizadas por programadores e modeladores para desenvolver e executar seus programas. Muitas delas têm migrado do *desktop* para a *web*, dando origem às *Web* IDE's (WIDE's). Nesta categoria de aplicações *web*, encontramos apenas duas que são voltadas especificamente para o desenvolvimento de modelos de otimização. O PIFOP surge como mais uma alternativa à comunidade de pesquisa operacional, se diferenciando das outras opções no seu suporte à linguagem AMPL (*A Mathematical Programming Language*), na possibilidade de edição colaborativa e também na sua solução de execução remota de programas de otimização. O PIFOP tem sido muito bem recebido pelos nossos primeiros usuários. No presente trabalho apresentamos esta ferramenta e demonstramos como ela pode facilitar o trabalho de professores, alunos, pesquisadores e profissionais da área de otimização matemática em geral.

Palavras-chave: AMPL, Linguagens de modelagem, Online IDE, Web IDE.

ABSTRACT

IDE's (Integrated Development Environments) are tools used by programmers and modelers to develop and execute their programs. Many of them have been moving from the desktop to the web, originating the Web IDE's (WIDE's). In this category of web applications, we've found only two that are aimed specifically at the development of optimization models. PIFOP emerges as another alternative for the operations research community, differing from the other options in its support for AMPL (A Mathematical Programming Language), in the possibility of collaborative editing and also in its solution for remote execution of optimization programs. PIFOP has been well received by our first users. In this work we present PIFOP and demonstrate how it can facilitate the work of teachers, students, researchers and mathematical optimization professionals in general.

Keywords: AMPL, Modeling languages, Online IDE, Web IDE.

* Autor para correspondência. E-mail: davidoro@pifop.com
DOI: 10.4322/PODes.2021.013

[1] *Todos os autores assumem a responsabilidade pelo conteúdo do artigo.*

1. Introdução

PIFOP (**P**rogramming **I**nterface **F**or **O**ptimization **P**roblems) é uma aplicação *web*[1] desenvolvida pelos autores para a elaboração e execução de modelos de otimização em AMPL[2] (*A Mathematical Programming Language*, ou Uma Linguagem de Programação Matemática). O PIFOP foi concebido com o objetivo de preencher uma lacuna existente no conjunto de ferramentas disponíveis para o desenvolvimento em AMPL. Embora a comunidade de modelagem tenha acesso a diversas ferramentas que possibilitam o desenvolvimento de modelos local e individualmente, e também ambientes virtuais de desenvolvimento de modelos em outras linguagens de modelagem algébrica, até onde sabemos não existem ferramentas possam auxiliar o desenvolvimento de modelos de programação matemática em AMPL colaborativamente pela *web*.

A utilidade e a importância de ferramentas colaborativas podem ser verificadas na recepção e adoção delas pelos milhares de usuários individuais e empresariais. Entre os casos de sucesso em diferentes contextos podemos citar o *Overleaf*[3] para edição colaborativa de arquivos LaTeX, o *Repl.it*[4] para edição colaborativa de código em diversas linguagens de programação de computadores, e o *Google Docs*[5] para edição colaborativa de textos. Tais aplicações provaram-se especialmente úteis dentro de instituições de ensino ao facilitarem o trabalho em grupo. No entanto, não encontramos nada parecido com tais ambientes no contexto específico de modelagem matemática em AMPL.

O PIFOP foi desenvolvido tendo em vistas duas funcionalidades principais: 1) edição colaborativa de modelos em AMPL e 2) visualização em tempo real dos resultados da execução de resolvedores de otimização. A resolução dos problemas pode ser local ou remota. Para resolução local possibilitamos a utilização do GLPSOL[6], um resolvedor gratuito de problemas de programação linear inteira mista que disponibilizamos localmente no PIFOP utilizando a tecnologia *WebAssembly*. Para resolução remota, duas opções estão disponíves. Primeiro, possibilitamos a utilização do *NEOS server* (Gropp e Moré (1997), Czyzyk et al. (1998), Dolan (2001)), um servidor de otimização gratuito hospedado pelo *Wisconsin Institute for Discovery*[7]. Segundo, possibilitamos a utilização de servidores hospedados pelos próprios usuários do PIFOP. Nesta modalidade, usuários podem gerenciar seus próprios servidores, o que pode ser bastante útil para universidades e laboratórios que desejam tornar os recursos de uma máquina e seus resolvedores acessíveis aos alunos, professores e pesquisadores de maneira remota.

Esta última opção de execução remota é um diferencial importante do PIFOP porque muitos resolvedores de otimização são computacionalmente caros, consumindo grande quantidade de memória e de tempo de processamento. Resolver múltiplos modelos numa mesma máquina simultaneamente pode rapidamente exaurir os recursos computacionais, afetando negativamente o desempenho dos resolvedores. Isso pode ser particularmente prejudicial para os pesquisadores que estejam testando o desempenho de seus modelos. Ao dar ao usuário a possibilidade de hospedar seus próprios servidores de otimização, ele mesmo pode controlar quem poderá utilizar o servidor e estabelecer limites de tempo e memória para cada usuário. No Departamento de Engenharia de Produção da UFMG três máquinas hospedam servidores de otimização acessíveis aos alunos de graduação e pós-graduação.

Nosso objetivo neste trabalho é apresentar aos alunos, educadores e profissionais da área de pesquisa operacional esta nova ferramenta, que até onde sabemos é a primeira do tipo voltada especificamente para modelagem matemática colaborativa em AMPL. Os professores da área possuem agora uma maneira mais eficiente de trabalhar com seus alunos, e estes têm um modo

[1] https://pifop.com
[2] https://ampl.com
[3] https://overleaf.com
[4] https://repl.it
[5] https://docs.google.com
[6] https://en.wikibooks.org/wiki/GLPK/Using_GLPSOL
[7] https://neos-server.org

mais fácil de trabalhar em grupo. Além disso, pesquisadores podem facilmente compartilhar seus modelos com revisores e modeladores em geral e executar resolvedores remotamente de maneira menos trabalhosa.

No presente artigo, apresentamos brevemente trabalhos relacionados ao nosso na Seção 2, seguido de uma introdução concisa à linguagem AMPL na Seção 3, apresentando uma justificativa para a termos escolhido. Na Seção 4, demonstramos algumas das funcionalidades do PIFOP. A Seção 5 traz uma visão geral do sistema por detrás das facilidades oferecidas pela ferramenta. Por fim, na Seção 6, concluímos com algumas considerações sobre o PIFOP e caminhos futuros a serem trilhados.

2. Trabalhos Relacionados

Como um *software*, a aplicação por nós desenvolvida se encontra na categoria *web integrated development environment* (WIDE), ou ambiente virtual e integrado de desenvolvimento. Essa categoria de *software* não é conceitualmente nova (Yulianto et al., 2017), mas para maior clareza sobre o que consideramos ser trabalhos relacionados ao nosso, apresentamos nossa própria definição do termo WIDE. Uma IDE é um *software* que integra diversas funcionalidades e ferramentas úteis no desenvolvimento de programas de computador, tais como editor de código fonte, explorador de arquivos e terminal para impressão da saída do programa. Consideramos ser uma WIDE uma aplicação *web* que busca oferecer as mesmas funcionalidades de uma IDE, porém sendo disponibilizada *online*, sendo acessível através de um navegador e possuindo pelo menos quatro características:

1. Editor de código no navegador.
2. Armazenamento de arquivos em nuvem.
3. Funcionalidades básicas de gerenciamento de projetos (como a possibilidade de o usuário criar múltiplos projetos com múltiplos arquivos).
4. Execução/compilação remota e visualização da saída do programa.

Diversos sistemas já foram desenvolvidos para trazer para a *web* funcionalidades de IDE's que por muito tempo só estavam disponíveis em *desktop*. Na literatura, nós encontramos muitas propostas de WIDE's, mas a maioria delas foi descontinuada, sendo que nenhuma delas foi desenvolvida para modelagem matemática: Arvue (Aho et al., 2011), Web2Compile (Santos et al., 2014), IDE 2.0 (Itahriouan et al., 2014), CEclipse (Wu et al., 2011), Harmonik (Yulianto et al., 2017) e WeScheme (Yoo et al., 2011) são algumas dessas WIDE's. Arvue possibilitava o desenvolvimento e publicação de aplicações *web* baseadas em Vaadin (um *framework* de desenvolvimento de aplicações Java). Web2Compile foi uma ferramenta para desenvolvimento de aplicações para redes de sensores sem fio. IDE 2.0 oferecia um ambiente para desenvolvimento em PHP, HTML e Javascript. CEclipse era uma plataforma para desenvolvimento Java na nuvem. Harmonik possibilitava que seus usuários criassem programas em uma pseudo linguagem de programação, que era traduzida para linguagens como C++ e Java, compilada e executada no navegador. WeScheme é um ambiente para programação em Scheme e Racket, e é a única WIDE desta lista que ainda está disponível[8].

Podemos encontrar na *web* um número de WIDE's consideravelmente maior do que o da literatura. Para programação genérica, isto é, desenvolvimento de programas em linguagens genéricas como C++ e Java, podemos mencionar *Repl.it*[9], *Cloud9*[10], *GitLab's WIDE*[11],

[8] https://wescheme.org
[9] https://repl.it
[10] https://aws.amazon.com/cloud9
[11] https://about.gitlab.com

Codeanywhere[12] e *Eclipse Che*[13]. Para criação de documentos a partir de código fonte LaTeX, temos a aplicação *Overleaf*[14]. E para desenvolvimento de modelos de otimização matemática, encontramos *Watson Studio*[15] e *RASON*[16]. Essas duas últimas WIDE's são as únicas que encontramos que são voltadas especificamente para modelagem matemática.

Watson Studio oferece uma variedade de ferramentas para preparação, análise e modelagem de dados. Uma dessas ferramentas é o *Decision Optimization Model Builder*, ou Construtor de Modelos para Otimização de Decisões. Nela encontramos um ambiente para desenvolvimento de modelos em OPL — **O**ptimization **P**rogramming **L**anguage, uma linguagem de modelagem algébrica para resolvedores CPLEX. Os modelos podem ser resolvidos remotamente em uma máquina virtual usando CPLEX. Os recursos computacionais e de armazenamento da máquina virtual são disponibilizados pela IBM Cloud Services e depende do plano de assinatura do usuário. Usuários no nível gratuito podem usar máquinas com até 4 vCPUs e 16 GB de RAM por um tempo limitado.

RASON IDE permite o desenvolvimento de modelos na linguagem RASON (**R**estful **A**nalytic **S**olver **O**bject **N**otation). Os modelos são resolvidos utilizando o *Solver SDK* da *Frontline Solvers*, e a requisição para execução é realizada no servidor em `rason.net`, que executa os modelos em máquinas na *Microsoft Azure*. De maneira semelhante ao *Watson Studio*, os recursos computacionais disponíveis dependem do plano de assinatura do usuário. Usuários no nível gratuito podem resolver modelos limitados por um número máximo de variáveis e restrições, que utilizem até 7 GB de memória RAM e não ultrapassem 1 hora de execução por processo e 4 horas por mês.

Notamos que na área da otimização matemática outras aplicações *web* existem, mas sua relação com o nosso trabalho é menor porque não são WIDE's no sentido que estamos utilizando o termo. O já mencionado *NEOS server* é um servidor gratuito de otimização remota que executa modelos de otimização submetidos pelos usuários e retorna os resultados da execução, mas não oferece outras funcionalidades que o colocaria na categoria WIDE. Sierksma e Zwols (2015) disponibilizam uma ferramenta *web*[17] que se assemelha bastante a uma WIDE para otimização, mas com algumas limitações. Usando esta ferramenta é possível desenvolver e executar modelos em GMPL (*GNU Mathematical Programming Language*, ou Linguagem de Progração Matemática GNU) no navegador, bem como visualizar os exemplos do livro dos autores citados. No entanto, não a consideramos uma WIDE pelo fato de não oferecer armazenagem de arquivos em nuvem e nem gerenciamento básico de projetos.

Mencionamos também que alguns pacotes e bibliotecas de programação possibilitam o desenvolvimento de modelos de otimização usando linguagens de programação genérica. Sendo assim, as WIDE's voltadas para programação genérica também podem ser utilizadas para modelagem matemática. Por exemplo, o pacote Python-MIP permite modelagem e resolução de problemas de programação linear inteira mista em Python (Santos e Toffolo, 2019), e o pacote JuMP possibilita a otimização matemática em Julia (Dunning et al., 2017). A vantagem de se utilizar linguagens de programação para otimização é o ganho de flexibilidade naquilo que é possível fazer no programa de otimização, como por exemplo, visualização interativa e comunicação com resolvedores durante a sua execução. A desvantagem é ter que utilizar uma linguagem cuja sintaxe não foi desenvolvida especificamente para modelagem matemática, o que torna a implementação computacional de um modelo menos clara e mais complexa do que seria utilizando-se uma linguagem de modelagem algébrica.

[12] `https://codeanywhere.com`
[13] `https://github.com/eclipse/che`
[14] `https://overleaf.com`
[15] `https://dataplatform.cloud.ibm.com`
[16] `https://rason.com`
[17] `https://online-optimizer.appspot.com`

3. Por que AMPL?

A linguagem AMPL (***A** **M**athematical **M**odeling **L**anguage*) possui vários atrativos que influenciaram nossa decisão de adotá-la como a linguagem do PIFOP. Primeiro, a sua sintaxe expressiva é de fácil entendimento pelos iniciantes na área de modelagem matemática. Existe uma correspondência bastante clara entre os símbolos algébricos que estamos acostumados a usar em salas de aula e as palavras-chave da linguagem AMPL. Por exemplo, veja a seguir na Figura 1 como uma formulação simples do problema da mochila expressa em AMPL permanece bastante similar ao que encontraríamos em um livro ou artigo.

Figura 1: O mesmo modelo de otimização para o problema da mochila expresso usando (1) notação algébrica e (2) AMPL.

(1)

Seja I o conjunto de itens disponíveis
Seja W a capacidade da mochila
Seja v_i o valor associado ao item $i \in I$
Seja w_i o peso associado ao item $i \in I$

As variáveis de decisão são definidas como:

$$x_i = \begin{cases} 1, \text{ se o item } i \text{ é carregado} \\ 0, \text{ caso contrário} \end{cases}$$

O objetivo é maximizar $z = \sum_{i \in I} v_i x_i$

Sujeito a:

$$\sum_{i \in I} w_i x_i \leq W$$
$$x_i \in \{0, 1\}$$

(2)

```
set I;
param W;
param v{I};
param w{I};


var x{I} binary;

maximize z:  sum{i in I} v[i]*x[i];

subject to:

c1:  sum{i in I} w[i]*x[i] <= W;
# domínio definido na declaração
# das variáveis x acima
```

Fonte: Elaborada pelos autores.

Vale notar que, ao suportar a linguagem AMPL, implicitamente nós também estamos suportando a linguagem GNU MathProg (GMPL[18]) do GLPK[19], cuja sintaxe é um subconjunto da sintaxe da linguagem AMPL.

Segundo, de acordo com as estatísticas de uso do servidor remoto de otimização *NEOS server*, a linguagem é a mais popular entre seus usuários. Das 2.351.976 execuções ou *jobs* submetidos ao servidor entre junho de 2019 e junho de 2020, 55% utilizaram AMPL como dados de entrada, 42% utilizaram GAMS e 3% utilizaram outras linguagens (NEOS Statistics, 2021). Embora não seja possível concluir que o AMPL seja a linguagem mais popular entre o público que desejamos atingir, esta é pelo menos uma evidência de que a linguagem está ainda entre as mais utilizadas.

Terceiro, AMPL se comunica com a maioria dos resolvedores de otimização comerciais ou gratuitos. O mesmo poderia ser dito para GAMS, mas a compatibilidade do AMPL com os resolvedores é ainda maior, o que pode ser facilmente constatado na página do *NEOS server* que informa os resolvedores oferecidos e a forma de acessá-los. A liderança da linguagem AMPL é clara. Esta característica é vantajosa tanto para os usuários quanto para nós, desenvolvedores do PIFOP. Para os usuários é uma vantagem porque dessa forma possuem flexibilidade para alterar o resolvedor utilizado, ou mesmo usar diferentes resolvedores em um mesmo programa de otimização, sem que seja necessário usar diferentes linguagens para cada um. Para nós

[18] https://en.wikibooks.org/wiki/GLPK/GMPL_(MathProg)

[19] GNU Linear Programming Kit (GLPK) é uma biblioteca para resolução de problemas de programação linear, programação inteira mista e outros problemas relacionados utilizando a linguagem C. https://gnu.org/software/glpk

desenvolvedores é uma vantagem porque nos permite focar no suporte de um único interpretador sem perder em número de resolvedores suportados.

Por esses motivos consideramos que a linguagem AMPL é a melhor opção para fornecermos aos usuários do PIFOP a melhor experiência de modelagem matemática. Não descartamos a possibilidade de incluirmos outras linguagens em versões posteriores de nossa ferramenta, mas até o momento concentramos nossos esforços apenas ao suporte da linguagem AMPL.

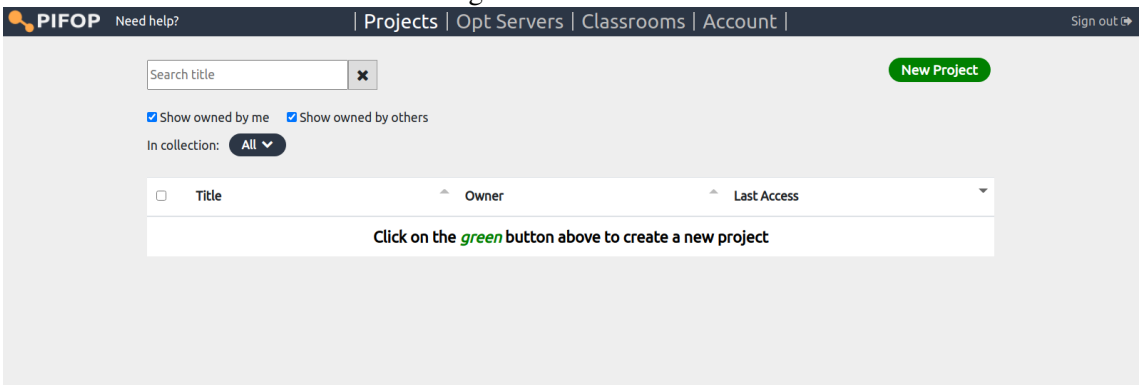
4. Utilizando o PIFOP

Nesta seção demonstraremos como utilizar o PIFOP para 1) criar projetos de modelagem, 2) executar resolvedores localmente e remotamente, 3) compartilhar projetos com outros usuários e colaborar com eles e 4) hospedar um servidor de otimização.

4.1. Criando Projetos

Poucos passos são necessários para começar a trabalhar em um modelo de otimização. O PIFOP possui basicamente duas telas: 1) a tela inicial (Figura 2), onde o usuário pode visualizar seus projetos, os servidores de otimização aos quais ele possui acesso e informações sobre sua conta e 2) a tela da aplicação em si (Figura 3), onde o usuário trabalha em seus modelos. Ao acessar o PIFOP pela primeira vez, o usuário entrará na tela inicial, onde ele encontrará o botão *New Project* para criar um novo projeto.

Figura 2: Tela inicial.



Fonte: Elaborada pelos autores.

Ao clicar nesse botão, será solicitado que o usuário dê um nome ao projeto e, opcionalmente, carregue os arquivos iniciais dele. Um projeto de otimização em AMPL é tipicamente composto de pelo menos três arquivos: um arquivo contendo o modelo, um arquivo contendo os dados de entrada e um arquivo contendo os comandos a serem executados. O leitor interessado em acompanhar este tutorial na prática pode acessar a nossa página de ajuda [20] e baixar os arquivos de um dos diversos exemplos retirados do livro referência da linguagem AMPL (Fourer et al., 2002) de projetos de otimização.

Concluída a criação do projeto e carregamento dos arquivos, o usuário será direcionado à tela da aplicação, onde o projeto recém criado estará pronto para ser editado e executado. Para editar seus arquivos, o usuário deverá abrí-los no editor clicando duas vezes neles no explorador de arquivos à esquerda. Finalizadas as edições desejadas, o próximo passo é a resolução do modelo implementado através de um resolvedor de otimização. Para isso utilizamos o terminal à direita, através do qual o usuário pode executar resolvedores com os arquivos de seu projeto.

[20] https://pifop.com/help/ampl_examples.html

4.2. Executando um Resolvedor

Atualmente existem três opções disponíveis para resolução de modelos de otimização. Primeiro, o usuário pode executar o resolvedor GLPSOL localmente no próprio navegador. GLPSOL é um resolvedor gratuito de problemas de programação linear inteira mista que disponibilizamos localmente no PIFOP utilizando a tecnologia *WebAssembly*. Para executá-lo, basta digitar o comando `glpsol` no terminal com os nomes dos arquivos contendo o modelo e dados do problema:

```
> glpsol -m <modelo> -d <dados>
```
[21]

Segundo, o usuário pode executar resolvedores remotamente no servidor de otimização *NEOS*. Diversos resolvedores são disponibilizados gratuitamente neste servidor, que pode ser utilizado através do comando `neos`. Mesmo resolvedores comerciais, como CPLEX, são disponibilizados gratuitamente, podendo ser selecionados através da opção `solver` (`-s`) tal como mostrado a seguir:

```
> neos -m <modelo> -d <dados> -c <comandos> -s <resolvedor>
```

Terceiro, o usuário pode executar resolvedores remotamente em um servidor de otimização hospedado por algum usuário do PIFOP. Falaremos sobre hospedagem de servidores de otimização na Seção 4.4. Um usuário que possui permissão para utilizar determinado servidor pode fazê-lo selecionando acima do terminal o servidor desejado e utilizando o comando `ampl` no terminal, exatamente como faria ao utilizar o interpretador da linguagem para *desktop*:

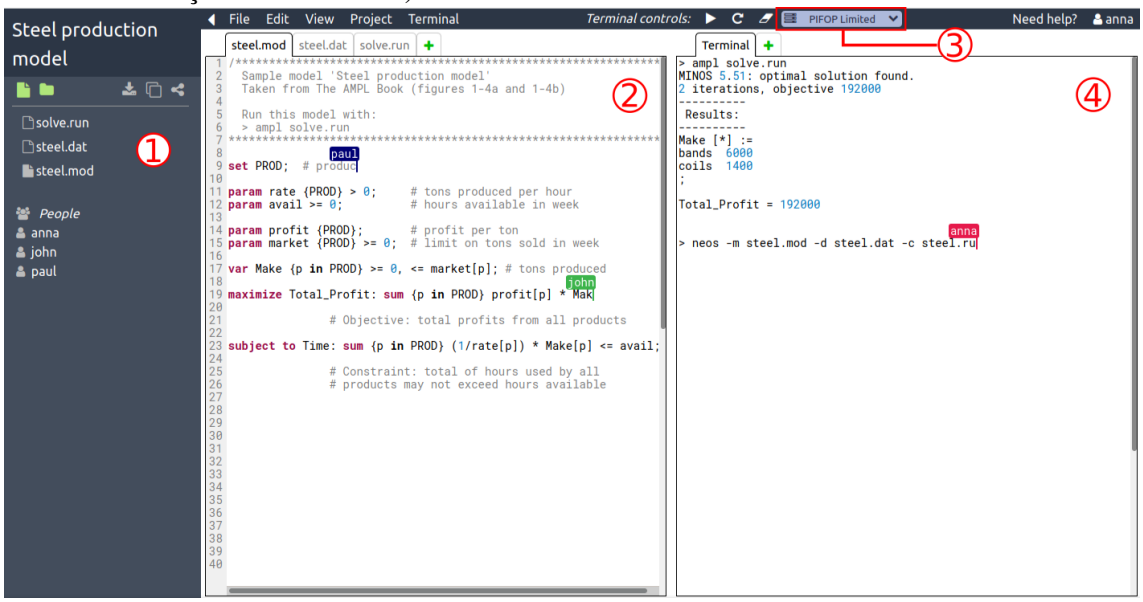
```
> ampl <arquivo>
```

Cada um desses três comandos possuem outras opções que podem ser configuradas na linha de comando. Para listar as opções disponíveis, o usuário deve entrar o comando com a opção `help` (por exemplo: `glpsol --help`).

Os resultados da execução são impressos no terminal em tempo real para todos os colaboradores do projeto. Se o modelo for complexo, requerendo várias horas ou mesmo dias para terminar a sua execução, o usuário pode fechar seu navegador sem que a execução seja interrompida e retornar em outro momento para ver os resultados.

[21] Os símbolos '<' e '>' foram utilizados para indicar nomes escolhidos pelo usuário.

Figura 3: Tela da aplicação. 1) Explorador de arquivos. 2) Editor de texto. 3) Servidor de otimização selecionado. 4) Terminal.



Fonte: Elaborada pelos autores.

4.3. Compartilhando Projetos

Para exemplificar o compartilhamento de projetos e a colaboração entre múltiplos usuários, propomos o seguinte cenário. Um professor deseja dar um exercício de modelagem para seus alunos. Neste exercício, os alunos devem completar um modelo com as restrições que lhe faltam para ficar correto. O PIFOP oferece duas opções de compartilhamento de projetos: a primeira permite que o usuário apenas visualize o projeto e a segunda permite que o usuário edite o projeto. Neste cenário, o que se deseja é que cada aluno ou grupo edite a sua própria cópia do projeto que contém o modelo incompleto e, depois de realizada a tarefa, compartilhe a sua solução com o professor. Portanto, o professor compartilhará o seu projeto com os alunos com permissão apenas para visualização, para que sua cópia do projeto não seja editada, mas apenas acessada para ser copiada. Para fazer isso, no menu à esquerda, o professor clicará no botão *Share Project* e copiará o endereço com permissão apenas para visualização (*view only*). Ele então compartilhará este endereço com seus alunos utilizando o canal de comunicação desejado.

Acessando o projeto com este endereço, os alunos poderão apenas visualizá-lo. Para criar uma cópia editável do projeto, o aluno clicará no botão *Copy* no menu à esquerda. Uma vez copiado, o projeto pode ser editado e executado conforme descrito nas Seções 4.1 e 4.2. Caso o trabalho seja em grupo, o endereço do projeto editável pode ser compartilhado com os alunos do grupo, e todos eles poderão trabalhar no projeto simultaneamente. Concluída a atividade, é a vez do aluno compartilhar seu projeto com seu professor. Usando o endereço compartilhado pelo aluno, o professor irá acessar o projeto e avaliar a corretude da solução proposta.

Um outro cenário de utilização do compartilhamento de projetos em modo *view only* pode ser vista em nossa página de ajuda. Ali o usuário encontrará vários exemplos do livro da linguagem AMPL prontos para serem acessados. Mesmo neste modo de acesso o usuário pode utilizar o terminal e ver os resultados da execução do modelo. Caso o usuário queira fazer alterações em um desses exemplos basta criar uma cópia e editar os arquivos. Um professor que queira demonstrar para seus alunos uma técnica de modelagem mais avançada, como inserção de cortes ou geração de colunas, pode facilmente criar um projeto exemplo e compartilhá-lo com seus alunos. E um pesquisador que propõe uma nova formulação matemática para um problema de otimização poderá facilmente compartilhar seu modelo com, por exemplo, os revisores da revista para a qual ele está

submetendo um artigo. Além de conter a sua própria formulação, o pesquisador poderá incluir no projeto as formulações já conhecidas da literatura, o que irá facilitar a comparação entre o desempenho de cada um por parte dos revisores.

4.4. Hospedando um Servidor de Otimização

Na Seção 4.2 apresentamos a opção que os usuários têm de executar resolvedores em servidores de otimização hospedados por usuários do PIFOP através do comando `ampl`. A hospedagem pode ser feita em qualquer máquina Linux que tenha alguma versão do AMPL nela instalada. Essa funcionalidade permite que o hospedeiro tenha total controle sobre quem pode acessar o seu servidor, o que permite que servidores sejam configurados tanto para uso particular quanto para uso compartilhado. Um exemplo do primeiro tipo de uso é quando o usuário já possui o AMPL instalado em seu próprio computador e deseja usá-lo através da interface do PIFOP. E um exemplo do segundo caso é em laboratórios de universidades, onde se deseja permitir que alunos utilizem os recursos de uma máquina e seus resolvedores sem dar a eles acesso direto e ilimitado a ela.

Para hospedar um servidor de otimização, os pré-requisitos básicos são 1) sistema operacional Linux, 2) alguma versão do AMPL (podendo ser tanto as versões comerciais quanto as gratuitas) e 3) o programa servidor em si. Destacamos que nosso serviço é oferecido à parte do AMPL. Os usuários interessados em adquirir o AMPL deverão fazê-lo acessando o site oficial do produto. Os comandos abaixo mostram como baixar e extrair nosso programa que faz a interface entre o AMPL e o PIFOP:

```
> wget https://pifop.com/download/opt-server.zip
> unzip opt-server.zip
```

Depois de baixado e extraído, o usuário deverá entrar na pasta criada e iniciar o servidor:

```
> cd opt-server
> ./opt-server
```

Na primeira vez que o servidor for executado será solicitado que o usuário dê um nome para o servidor e informe o caminho completo para a pasta contendo o programa `ampl`:

```
1) Give your server a name: Meu Servidor
2) Absolute path to the directory containing ampl: /path/to/ampl-directory
```

Uma vez estabelecida conexão entre o servidor de otimização e o servidor central do PIFOP, basta que o usuário entre com sua conta e o servidor estará pronto para ser utilizado por ele. Isso pode ser confirmado acessando a área de gerenciamento de servidores de otimização[22].

Caso o servidor seja para uso compartilhado, será necessário explicitar quais usuários podem utilizá-lo e quais as permissões e limites computacionais prescritos para eles. Isso é feito modificando o arquivo `config.json` que é gerado na pasta do servidor após a primeira execução. Primeiro, é necessário criar grupos de usuários e, segundo, adicionar usuários a esses grupos. Para isso, uma seção `user-groups` contendo as permissões de cada grupo é adicionada. Por exemplo, se quiséssemos definir um grupo chamado "Alunos" e estabelecer limites de uso de memória e de tempo dos processos iniciados por usuários neste grupo, nosso arquivo de configuração deverá conter o seguinte:

[22] https://pifop.com/app?area=userspace_opt_servers_area

```
{
    "name" : "Meu Servidor",
    "remember-me" : true,
    "ampl-directory" : "/path/to/ampl-directory",

    "user-groups" : {
        "Alunos" : {
            "max-memory" : 20,
            "max-time"   : 60
        }
    },

    "users" : {
        "Alunos": [
            "mariabazotte",
            "jvitor",
            "davidoro@ufmg.br"
        ]
    }
}
```

Exemplo de arquivo de configuração `config.json`.

Uma lista completa dos limites e permissões que podem ser configuradas para cada grupo de usuários pode ser encontrada em nosso *site*[23]. Por fim, basta reiniciar o servidor para que as alterações feitas tomem efeito.

4.5. Limites de Utilização

Assim como as WIDE's para otimização *Watson Studio* e *RASON*, o PIFOP foi planejado para oferecer diferentes níveis de serviço. O primeiro nível é gratuito e foi projetado com base em nossa experiência com alunos de graduação do curso de Engenharia de Produção da UFMG de forma a atender as suas demandas. Nesse nível, em que os usuários podem permanecer por tempo ilimitado, sempre será possível executar os modelos localmente, utilizando o GLPSOL, ou remotamente, utilizando o *NEOS server*. Também sempre será possível criar e compartilhar projetos. Os limites atualizados do nível gratuito podem ser encontrados em nosso *site*[24].

A utilização e hospedagem de servidores de otimização, que até o presente momento está disponível para todos os usuários, é uma funcionalidade que estará disponível apenas para usuários pagantes quando a fase *beta* do PIFOP terminar. Pretendemos oferecer planos especiais destinados a universidades e instituições de ensino, mas até o presente momento os detalhes de cada plano ainda estão sendo definidos.

5. O Sistema

Na seção anterior destacamos algumas funcionalidades do PIFOP e demonstramos a sua utilização. Nesta seção nosso objetivo é dar ao leitor uma visão geral do sistema que possibilita essas funcionalidades, na expectativa de que dessa forma possamos contribuir com aqueles que estão envolvidos na construção de sistemas ou aplicações similares ao PIFOP. Com isso não pretendemos afirmar que nosso sistema é o modelo a ser seguido. Antes, a descrição aqui feita deve ser tomada apenas como um exemplo de como estruturar um sistema para proporcionar as funcionalidades descritas na seção anterior. Embora nossa aplicação seja para o contexto específico da modelagem matemática, o sistema, em geral, independe dessa especialização que fizemos dele, e por isso acreditamos que conhecê-lo pode auxiliar desenvolvedores em suas decisões de projeto.

[23] https://pifop.com/help/hosting_opt_server.html
[24] https://pifop.com/pricing

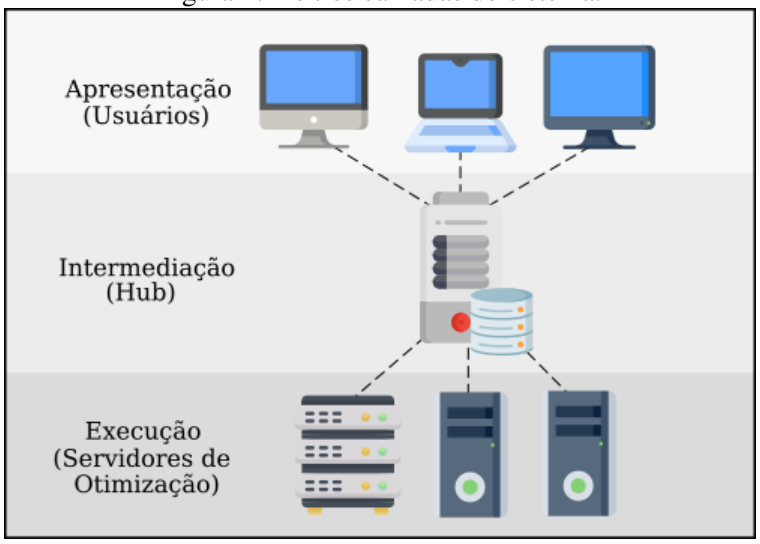
5.1. Arquitetura em Três Camadas

Nosso sistema possui uma arquitetura em três camadas: apresentação, intermediação e execução (Figura 4). A camada de *apresentação* consiste na interface gráfica com a qual o usuário interage usando seu navegador. Os modelos de otimização são executados nos servidores de otimização, que se encontram na camada de *execução*. Entre essas duas camadas está a camada de *intermediação*, responsável por transmitir os arquivos do usuário para a camada de execução e transmitir os resultados da execução para a camada de apresentação, e também por propagar as edições feitas em projetos para todos os colaboradores conectados.

Nas duas camadas extremas (apresentação e execução) encontram-se mais de um computador, enquanto que na camada do meio existe apenas um: o *hub*. Além das funções de intermediação entre usuários e servidores de otimização e entre os colaboradores de um projeto, o *hub* também realiza outras funções subsidiárias, como autenticação, armazenamento de arquivos e controle de acesso de usuários a servidores de otimização. Não existe nenhum tipo de comunicação direta nem entre colaboradores nem entre usuários e servidores de otimização. Dessa forma garantimos 1) que os únicos usuários que poderão acessar um projeto são os usuários com a chave de acesso daquele projeto e 2) que os únicos usuários que poderão utilizar um servidor de otimização são os usuários que o administrador do servidor tiver permitido. Entre servidores de otimização não há nenhum tipo de comunicação, seja de forma direta ou indireta através do *hub*.

Nesta descrição do sistema não estamos considerando a integração com o *NEOS server*. Quando falamos de servidores de otimização estamos nos referindo a servidores hospedados pelos próprios usuários do PIFOP, conforme descrito na Seção 4.4, e que são utilizados através do comando `ampl` no terminal.

Figura 4: As três camadas do sistema.



Fonte: Elaborada pelos autores[25].

A seguir veremos algumas das características mais importantes de cada uma dessas camadas.

5.2. Camada de Apresentação

Implementada em *javascript*, a camada de apresentação fornece a interface pela qual o usuário interage com o sistema. A comunicação entre a interface e o *hub* se dá por meio de uma conexão persistente e bidirecional utilizando o protocolo de comunicação *WebSocket* (Fette e

[25] Ícones criados por Freeplk do site `https://flaticon.com`

Melnikov, 2011), que permite que qualquer uma das duas partes transmita mensagens para a outra a qualquer momento. Das três camadas, essa é a mais simples, consistindo em grande parte de controles pelos quais o usuário envia requisições para o *hub*, tais como criar e acessar projetos, editar arquivos, mover pastas e executar modelos. Seu componente mais complexo é sem dúvida o editor de texto, principalmente por causa de três funcionalidades: 1) destaque de sintaxe AMPL 2) edição colaborativa e 3) suporte para arquivos longos, com centenas de milhares de linhas.

Essas funcionalidades de nosso editor não seriam possíveis utilizando-se o controle padrão para entrada de texto *textarea*. Embora pudéssemos ter optado por utilizar algum editor de texto de terceiros, tais como *Ace*[26], *Monaco*[27] e *CodeMirror*[28], de qualquer maneira teríamos que fazer as adaptações necessárias para integrá-los ao nosso sistema, e visto que nenhum deles possui nem suporte à colaboração nem destaque de sintaxe em AMPL nativas, preferimos desenvolver nós mesmos um editor com essas funcionalidades.

Na verdade, o suporte à edição colaborativa não é bem uma responsabilidade somente do editor em si, mas do subsistema que inclui tanto o editor quanto o *hub*. Este subsistema se baseia no mecanismo de colaboração conhecido na literatura como *Operations Transformation* (Ellis e Gibbs, 1989), no qual também se baseiam outros editores colaborativos, tais como os editores das aplicações *Overleaf* (Overleaf How-to Guides, 2020) e *Google Docs* (Xu et al., 2014). Esse mecanismo permite que as edições realizadas pelos usuários sejam imediatamente aplicadas localmente sem que haja divergência definitiva entre as visões dos diferentes colaboradores do projeto.

Para interagir com o servidor de otimização e executar seus modelos, o usuário utiliza um terminal. Da mesma forma que os arquivos são compartilhados entre os colaboradores de um projeto, também são os terminais. Isso permite que os programas de otimização sejam executados uma única vez e sua saída seja compartilhada entre os colaboradores, o que é especialmente útil quando a execução do programa possui horas de duração. Além disso, os usuários podem abrir múltiplos terminais e executar diferentes programas em cada um deles. Ou seja, não é necessário esperar o término de uma execução para dar início a outra, e diferentes colaboradores podem trabalhar de forma independente, cada um utilizando um terminal.

5.3. Camada de Intermediação

Tanto o *hub* na camada de intermediação quanto o servidor de otimização na camada de execução foram implementados em C++. Em ambos os casos a motivação para essa decisão foi evitar, pelo menos em parte, problemas de desempenho. O *hub* possui diversas responsabilidades, sendo a mais importante a de intermediar a comunicação entre usuários e seus colaboradores e entre usuários e servidores de otimização. Isso o torna um forte candidato a ser o gargalo de todo o sistema: enquanto as outras partes do sistema só precisam se comunicar com o *hub*, este precisa se comunicar com todas as partes, além de realizar autenticação de usuários e buscas no banco de dados.

Para estabelecimento e manutenção da comunicação em *WebSocket* utilizamos a biblioteca de código aberto *libwebsockets*[29], de Andy Green. Também decidimos manter o banco de dados na mesma máquina do *hub* e utilizar o gerenciador de banco de dados SQLite[30] para gerenciá-lo, o que nos permite acessá-lo diretamente do nosso programa utilizando a API em C do SQLite.

Quando um usuário solicita acesso a um projeto, os arquivos armazenados no banco de dados são carregados na memória do programa e é criada uma sessão de edição do projeto. Se outros usuários acessarem este projeto, eles serão vinculados à mesma sessão de edição. Esta sessão de edição contém as versões mais recentes dos arquivos do projeto, que são salvos

[26] https://ace.c9.io
[27] https://microsoft.github.io/monaco-editor
[28] https://codemirror.net
[29] https://libwebsockets.org
[30] https://sqlite.org

automaticamente a cada 5 segundos. Quando um usuário realiza alguma modificação em um arquivo, essa modificação é propagada para todos os usuários vinculados à sessão de edição. Como mencionamos na subseção anterior, utilizamos o mecanismo *Operational Transformation* para solucionar conflitos entre modificações realizadas por diferentes usuários ao mesmo tempo, de maneira que não há divergência definitiva entre as visões que os usuários possuem dos arquivos que estão editando (embora possa haver divergência temporária, enquanto as modificações não são propagadas). A sessão de edição permanece aberta enquanto houver usuários conectados ao projeto ou enquanto houver programas de otimização sendo executados em seus terminais.

Além de ser porta de entrada para projetos, o *hub* é também porta de entrada para os servidores de otimização. Quando um usuário submete um programa para ser executado em um servidor de otimização, o *hub* verifica se ele possui permissão para utilizar aquele servidor. Em caso afirmativo, um vínculo é criado entre o terminal pelo qual o usuário submeteu o programa e o processo executado no servidor de otimização, de maneira que as saídas daquele processo são impressas naquele terminal.

5.4. Camada de Execução

Como mencionamos na subseção anterior, o servidor de otimização foi implementado em C++ e utilizamos a biblioteca *libwebsockets* para a comunicação com o *hub*. É importante que o servidor de otimização possua uma boa performance porque, embora a comunicação seja de um para um, o *hub* na verdade transmite a mensagem de muitos. O servidor, portanto, receberá potencialmente muitas requisições de diferentes usuários e executará diversos programas em paralelo, enviando as saídas de cada programa em tempo real de volta para o *hub*.

Quando o servidor de otimização recebe uma requisição de execução de um programa de otimização, primeiramente ele salva os arquivos submetidos localmente, isto é, os arquivos do modelo, dos dados e de comandos AMPL. Em seguida o interpretador AMPL é chamado com esses arquivos como entrada de maneira que a saída, ao invés de ser escrita no dispositivo de saída padrão, é redirecionada para o próprio programa servidor.

O interpretador é executado dentro de um ambiente *sandbox* implementado via *nsjail*[31], uma ferramenta de código aberto não oficial da *Google* desenvolvida principalmente por Robert Swiecki. Esse *sandbox* é um ambiente que estabelece limites para os programas nele executado de modo a restringir o acesso de arquivos, uso de memória e tempo de execução. A configuração do *sandbox* depende dos limites do usuário que está utilizando o servidor. Na Seção 4.4 mostramos como esses limites são estabelecidos através da definição de grupos de usuários e as restrições impostas quanto ao uso de memória, tempo de execução, e outros parâmetros.

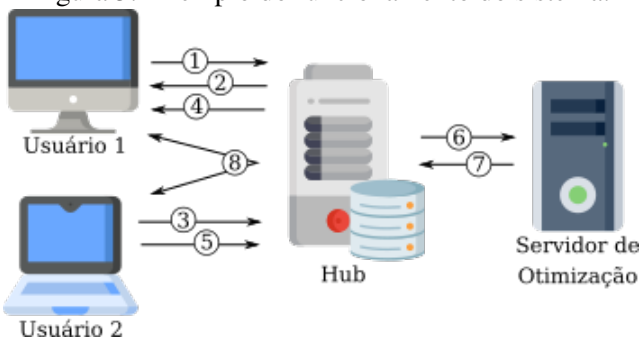
5.5. Exemplo de Funcionamento

A Figura 5 representa o fluxo de comunicação entre as camadas do sistema quando um usuário acessa um projeto, realiza modificações nele e executa um modelo em um servidor de otimização.

Primeiramente, o usuário 1 solicita acesso a um projeto para modificá-lo (1). Depois de adicionar o usuário 1 à sessão de edição do projeto, o *hub* responde com os arquivos e demais dados do projeto (2). Então o usuário 2 realiza uma modificação em algum arquivo, que é enviada para o servidor (3) e propagada para os demais colaboradores do projeto (4). Em seguida o usuário 2 decide executar o modelo modificado, e para isso submete os arquivos para execução no servidor de otimização selecionado (5). O *hub* cria um vínculo entre o terminal usado pelo usuário 2 e o processo que será executado no servidor de otimização e envia os arquivos do usuário para este servidor (6). Por fim, o servidor de otimização retorna a saída da execução para o *hub* (7), que por sua vez envia a saída para os usuários conectados ao projeto (8).

[31] https://github.com/google/nsjail

Figura 5: Exemplo do funcionamento do sistema.



Fonte: Elaborada pelos autores[32].

6. Conclusão

Até onde sabemos, o PIFOP é a primeira WIDE para otimização matemática em AMPL, e a primeira WIDE para otimização com colaboração em tempo real. Embora o sistema seja jovem, tendo apenas 2 anos de desenvolvimento com interrupções, temos obtido respostas bastante positivas dos alunos graduação e pós-graduação em Engenharia de Produção da UFMG que têm utilizado a ferramenta.

Existem ainda muitas funcionalidades que pretendemos implementar. Primeiro, queremos permitir a hospedagem de servidores de otimização em Windows e MacOS, já que o AMPL também está disponível nessas plataformas. Segundo, queremos suportar outras linguagens de modelagem algébrica, tais como GAMS e MPS. Terceiro, pretendemos implementar filas de requisições nos servidores de otimização para evitar sobrecargas indesejáveis. E quarto, queremos desenvolver interfaces próprias para dispositivos móveis (*tablets* e *smartphones*).

Em seu estado atual o PIFOP já atende satisfatoriamente as necessidades de alunos, professores e pesquisadores. Os modeladores possuem agora uma maneira mais produtiva de trabalhar colaborativamente, e universidades nacionais e internacionais dispõem de uma ferramenta que permite professores e alunos utilizarem os recursos computacionais de seus laboratórios remotamente para execução dos modelos. Nossa expectativa é que o PIFOP possa se consolidar como um ambiente colaborativo de fácil utilização para modelagem matemática.

Finalmente, agradecemos aos alunos que se dispuseram a testar o sistema durante seu desenvolvimento e ao Laboratório de Simulação e Otimização de Sistemas (LASOS) do Departamento de Engenharia de Produção da UFMG por nos ter permitido utilizar as suas máquinas em nossos testes.

Referências

Aho, T., Ashraf, A., Englund, M., Katajamäki, J., Koskinen, J., Lautamäki, J., Nieminen, A., Porres, I., e Turunen, I. Designing IDE as a service. *Communications of Cloud Software*, v. 1, n. 1, 2011.

Czyzyk, J., Mesnier, M. P., e Moré, J. J. The NEOS Server. *IEEE Journal on Computational Science and Engineering*, v. 5, n. 3, p. 68–75, 1998.

Dolan, E. D. *The NEOS Server 4.0 Administrative Guide*. Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[32]Ícones criados por Freeplk do site https://flaticon.com

Dunning, I., Huchette, J., e Lubin, M. JuMP: A modeling language for mathematical optimization. *SIAM Review*, v. 59, n. 2, p. 295–320, 2017.

Ellis, C. A. e Gibbs, S. J. Concurrency control in groupware systems. In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. Portland Oregon, USA, 1989. p. 399–407.

Fette, I. e Melnikov, A. *The WebSocket Protocol*, 2011. Disponível em: https://tools.ietf.org/html/rfc6455. Acesso em: 30/08/2021.

Fourer, R., Gay, D. M., e Kernighan, B. W. *The AMPL Book. AMPL: A Modeling Language for Mathematical Programming*. Australia: Duxbury/Thomson, 2002.

Gropp, W. e Moré, J. J. *Optimization environments and the NEOS server*. ANL/MCS-P-654-0397; CONF-9607197-1, Argonne National Lab., IL (United States), 1997.

Itahriouan, Z., Aknin, N., Abtoy, A., e El Kadiri, K. E. Building a Web-based IDE from Web 2. 0 perspective. *International Journal of Computer Applications*, v. 96, n. 22, 2014.

NEOS Statistics. *NEOS Solver Statistics*, 2021. Disponível em: https://neos-server.org/neos/report.html. Acesso em: 21/07/2021.

Overleaf How-to Guides. *Can multiple authors edit the same file at the same time?*, 2020. Disponível em: https://www.overleaf.com/learn/how-to/Can_multiple_authors_edit_the_same_file_at_the_same_time? Acesso em: 11/06/2020.

Santos, A., Farias, M., Rocha, G., Pereira, M. V., de Farias, C. M., França, T. C., Costa, R. O., e de Janeiro-RJ-Brazil, R. Web2Compile: uma web IDE para Redes de sensores sem fio. *Simpósio Brasileiro de Redes de Computadores e Sistema Distribuídos (SBRC)*, v. p. 1037–1044, 2014.

Santos, H. G. e Toffolo, T. A. Tutorial de desenvolvimento de métodos de programação linear inteira mista em Python usando o pacote Python-MIP. *Pesquisa Operacional para o Desenvolvimento*, v. 11, n. 3, p. 127–138, 2019.

Sierksma, G. e Zwols, Y. *Linear and Integer Optimization: theory and practice*. Boca Raton, FL: CRC Press, 2015.

Wu, L., Liang, G., Kui, S., e Wang, Q. CEclipse: An online IDE for programing in the cloud. In: *2011 IEEE World Congress on Services*. Washington, DC. IEEE, 2011. p. 45–52.

Xu, Y., Sun, C., e Li, M. Achieving convergence in operational transformation: conditions, mechanisms and systems. In: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. Maryland, USA, 2014. p. 505–518.

Yoo, D., Schanzer, E., Krishnamurthi, S., e Fisler, K. WeScheme: the browser is your programming environment. In: *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*. New York, NY, 2011. p. 163–167.

Yulianto, B., Prabowo, H., Kosala, R., e Hapsara, M. Harmonik=++(Web IDE). *Procedia Computer Science*, v. 116, p. 222–231, 2017.