Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Análise de desempenho dos parâmetros de um resolvedor de Programação Linear Inteira

Lorielem de Carvalho Domingues

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Samuel Souza Brito

Janeiro, 2022
João Monlevade–MG

Lorielem de Carvalho Domingues

Análise de desempenho dos parâmetros de um resolvedor de Programação Linear Inteira

Orientador: Samuel Souza Brito

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Janeiro de 2022

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

D671a Domingues, Lorielem de Carvalho.
Análise de desempenho dos parâmetros de um resolvedor de Programação Linear Inteira. [manuscrito] / Lorielem de Carvalho Domingues. - 2022.
33 f.: il.: , gráf., tab..

Orientador: Prof. Dr. Samuel Souza Brito.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Algoritmos computacionais. 2. Modelos matemáticos. 3. Programação linear. I. Brito, Samuel Souza. II. Universidade Federal de Ouro Preto. III. Título.

CDU 519.852

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Lorielem de Carvalho Domingues

Análise de desempenho dos parâmetros de um resolvedor de Programação Linear Inteira

Monografia apresentada ao Curso de Engenharia de Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de bacharel em Engenharia de Computação

Aprovada em 11 de janeiro de 2022

Membros da banca

Doutor - Samuel Souza Brito - Orientador - Universidade Federal de Ouro Preto
Doutor - George Henrique Godim da Fonseca - Universidade Federal de Ouro Preto
Doutora - Janniele Aparecida Soares Araujo - Universidade Federal de Ouro Preto

Samuel Souza Brito, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 15/01/2022

Documento assinado eletronicamente por **Samuel Souza Brito**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 01/02/2022, às 12:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0274046** e o código CRC **9F112C0B**.

Referência: Caso responda este documento, indicar expressamente o Processo nº 23109.001175/2022-45 SEI nº 0274046

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: - www.ufop.br

Dedico este trabalho a Deus, que me presenteia todos os dias com a energia da vida, que me dá forças e coragem para atingir os meus objetivos. Sem ele em minha vida, nada disso seria possível.

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado forças para chegar até aqui. Ao meu orientador, que se fez presente e sempre pronto a ajudar em tudo o que precisei. Aos meus familiares e amigos que caminharam comigo em oração e sempre compreenderam minhas ausências. E por fim, agradeço aos que desacreditaram de mim, pois eles me deram ainda mais forças para continuar.

"Science is more than a body of knowledge; it is a way of thinking."

— Carl Sagan (1934 – 1996),
in: The Demon-Haunted World: Science as a Candle in the Dark.

Resumo

Este trabalho trata-se de um estudo e análise do impacto dos parâmetros no desempenho do resolvedor de Programação Linear Inteira COIN-OR Branch-and-Cut (CBC). Para tal, foi utilizada uma base de dados composta por diversos modelos matemáticos reais e acadêmicos encontrados na literatura, sendo eles pertencentes a *Mixed-Integer Programming Library*. O pacote `irace` foi utilizado para a definição dos valores de parâmetros que melhoram o desempenho do CBC. A utilização dos parâmetros sugeridos pelo `irace` gerou para o CBC uma melhoria média de 33% em comparação com os resultados obtidos quando esse resolvedor foi executado com os valores predefinidos dos parâmetros.

Palavras-chaves: Programação Linear Inteira. COIN-OR Branch-and-Cut. Configuração automática de algoritmos.

Abstract

This work is a study and analysis of the impact of parameters on the performance of the COIN-OR Branch-and-Cut (CBC) Integer Linear Programming solver. For this purpose, a database composed of several real and academic mathematical models found in the literature was used, which belong to the *Mixed-Integer Programming Library*. The package `irace` was used to define the parameter values that improve the performance of the CBC. Using the parameters suggested by irace contributed to an average improvement on CBC of 33% compared to the results obtained when this solver was run with the default parameter values.

Key-words: Integer Linear Programming. COIN-OR Branch-and-Cut Solver. Automatic configuration of algorithms.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CBC	COIN-OR Branch-and-Cut
MIPLIB	Mixed-Integer Programming Library
PL	Programação Linear
PLI	Programação Linear Inteira

Lista de símbolos

$\leq$	Menor do que ou igual a
$\geq$	Maior do que ou igual a
$\in$	Pertence
$\mathbb{R}$	Conjunto dos números reais
$\mathbb{Z}$	Conjunto dos números inteiros
?	Aciona o menu de ajuda do resolvedor CBC
??	Visualização do valor atual de cada parâmetro do resolvedor CBC

Sumário

1 Introdução

A Programação Linear (PL) é um técnica de Pesquisa Operacional empregada para encontrar a forma ótima de alocação de recursos escassos entre atividades que dividem esses recursos entre si. O termo "programação" é referente ao planejamento e o termo "linear" delimita que para este tipo de problema todas as funções devem ser lineares. A PL, portanto, é a busca de resultados ótimos por meio do planejamento de atividades (HILLIER; LIEBERMAN, 2013).

A Programação Linear Inteira (PLI) é uma particularidade da PL na qual algumas variáveis de decisão podem assumir somente valores inteiros. Um modelo de PLI pode ser formalmente definido como:

$$\text{Max(ou Min):} \quad c^T x + h^T y \tag{1.1}$$

$$\text{sujeito a:} \quad Ax + Gy \leq b \tag{1.2}$$

$$x \geq 0, y \geq 0 \tag{1.3}$$

$$x \in \mathbb{R}^n, y \in \mathbb{Z}^p \tag{1.4}$$

onde x representa o conjunto de variáveis de decisão contínuas e y um conjunto de variáveis de decisão que devem assumir valores inteiros. Um modelo de PLI possui uma função objetivo, ou seja, uma função linear que deve ser maximizada ou minimizada (por exemplo, maximização de lucros ou minimização de custos) e que envolve as variáveis de decisão. Além disso, modelos de PLI possuem uma série de restrições representadas por equações e inequações lineares que devem ser satisfeitas.

De acordo com Brito e Santos (2021), a Programação Linear Inteira é uma técnica poderosa para modelar e resolver uma ampla variedade de problemas de otimização combinatória, muitos desses problemas com um interesse prático, como por exemplo para a maximização de lucros, a redução de custos, entre vários outros. Portanto, melhorias obtidas em resolvedores de PLI contribuem diretamente para resolução de diversos problemas de otimização.

O presente estudo tem a finalidade de analisar o desempenho do resolvedor de Programação Linear Inteira COIN-OR Branch-and-Cut (CBC). Considerando uma base de dados de problemas existentes, são utilizadas ferramentas estatísticas para analisar e obter as melhores configurações de parâmetros para o CBC.

1.1 Aplicações da Programação Linear Inteira

De acordo com Loesch e Hein (2009), as áreas que mais utilizam PLI são as de administração, produção, organização e planejamento, pois está associado a gestão de compras, a organização e o controle de estoque. Com o auxílio da PLI é possível, por exemplo, desenvolver um modelo matemático capaz de dar suporte à uma empresa na hora da compra, de tal forma que, inserindo informações reais da empresa, ele obtém uma decisão de compra otimizada, atribuindo menores custos à gestão de compras e estoque dos produtos.

Fernandes et al. (2013) afirmam que uma pesquisa aplicada a um caso real em um setor da siderurgia buscou melhorias no gerenciamento da área de produção, onde havia o maior gargalo da empresa. Dessa forma, foi possível otimizar o processo de produção, reduzir filas de espera e consequentemente melhorar sua relação com os clientes.

A PLI tem aplicações em diversas áreas, como visto acima. Essa técnica possui resultados satisfatórios, desde que aplicada de forma correta.

1.2 Identificação do problema

O COIN-OR Branch-and-Cut (CBC) é um resolvedor de modelos de PLI altamente configurável, visto que possui um grande conjunto de parâmetros. A escolha de valores para esses parâmetros influencia diretamente na forma como o CBC atua durante a resolução de um modelo. Nesse sentido, a determinação das melhores configurações de parâmetros a serem utilizados por esse resolvedor é uma tarefa fundamental para melhorar seu desempenho.

Este estudo sobre o desempenho do resolvedor CBC investiga e determina os melhores conjuntos e valores de parâmetros para acelerar o processo de busca pelas soluções ótimas de problemas de PLI. Para investigar o impacto dos parâmetros no CBC e descobrir as melhores configurações, foi utilizada a ferramenta estatística `irace`, um framework para configuração automática de algoritmos.

1.3 Objetivos propostos

O principal objetivo deste trabalho é estabelecer um conjunto de parâmetros e seus respectivos valores que maximizem o desempenho do CBC na resolução de problemas de PLI. Para isso, é necessário identificar e analisar os parâmetros e estabelecer as relações de causa-efeito através de experimentos computacionais.

Para atender as necessidades do proposto, foi utilizada a ferramenta de configuração automática de algoritmos conhecida como `irace` e uma base de dados composta por diversos

problemas de Programação Linear Inteira encontrados na literatura, sendo a maioria deles pertencente a *Mixed-Integer Programming Library* (MIPLIB). A MIPLIB é uma biblioteca padrão para testar e comparar a performance de resolvedores de PLI, pois contém uma coleção de problemas desafiadores que envolvem aplicações reais e acadêmicas (GLEIXNER et al., 2021).

1.4 Estrutura do trabalho

Os passos para execução deste trabalho são assim definidos. No Capítulo 2 são abordados os conceitos gerais, a ferramenta de configuração automática de algoritmos utilizada, o resolvedor ao qual o estudo analisa e a revisão da literatura base deste trabalho. No Capítulo 3 é apresentado o desenvolvimento do trabalho, onde os parâmetros são definidos, testados e analisados. As considerações finais são apresentadas no Capítulo 4.

2 Conceitos Gerais e Revisão da Literatura

Neste capítulo são apresentados os principais conceitos e trabalhos relacionados ao tema proposto.

2.1 Configuração automática de algoritmos

Muitos algoritmos de resolução de problemas têm uma demanda grande para determinar as melhores configurações de parâmetros a serem utilizadas dentro do contexto trabalhado. Existem ferramentas que auxiliam na escolha das melhores combinações de parâmetros para algoritmos. Conhecidos como configuração automática de algoritmos, essas ferramentas fazem uma espécie de triagem inicial dos dados a serem analisados, executam experimentos computacionais e testes estatísticos, retornando os melhores valores de parâmetros para o algoritmo analisado. Dessa forma, é poupado o esforço de quem utiliza o algoritmo para encontrar os melhores valores de parâmetros, além de permitir que soluções melhores possam ser encontradas. Um exemplo de ferramenta dessa natureza é o `irace`, cujas características são descritas a seguir.

2.1.1 Pacote `irace`

De acordo com López-Ibánez et al. (2016), o `irace` implementa um procedimento de corrida iterada. Seu principal uso é a configuração automática de algoritmos de otimização e decisão. Entretanto, o `irace` também pode ser empregado para configurar outros tipos de algoritmos, especialmente quando o desempenho é influenciado pelos valores dos parâmetros.

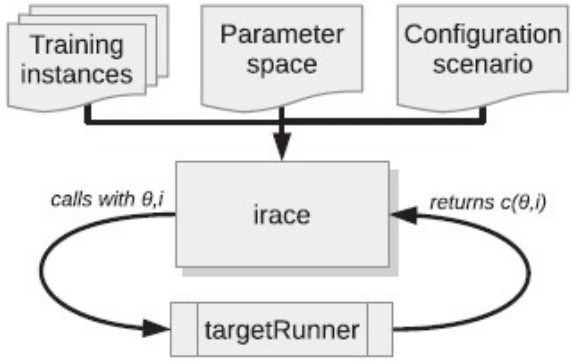
Conforme apresentado na Figura 1, o framework recebe como entrada:

1. um conjunto contendo os parâmetros que serão analisados, informando seu tipo, domínio e restrições;
2. um conjunto de instâncias para as quais os parâmetros devem ser ajustados;
3. um conjunto de configurações que serão aplicadas sobre esse cenário.

A partir dos dados de entrada, o `irace` procura por configurações que levam a um bom desempenho do algoritmo analisado, considerando o espaço de busca definido pelos valores de parâmetros. O algoritmo a ser configurado é executado com diferentes instâncias e configurações de parâmetros. Na Figura 1, *targetRunner* é um script (ou programa)

que é utilizado para executar o algoritmo analisado com uma configuração específica de parâmetros (θ) e uma instância (i). Ao final de sua execução, o *targetRunner* retorna a avaliação da execução do algoritmo considerando θ e i. Normalmente, o valor retornado pelo *targetRunner* é o tempo de execução do algoritmo.

Figura 1 – Esquema de funcionamento do `irace`. Fonte: (LÓPEZ-IBÁNEZ et al., 2016)



2.2 Resolvedor COIN-OR Branch-and-Cut

O COIN-OR Branch-and-Cut (CBC)[1] é um resolvedor de Programação Linear Inteira inicialmente projetado para ser usado como uma biblioteca, mas, apesar de ainda ser utilizado como uma, ele acabou tornando-se um potente resolvedor independente. O CBC inclui em seu pacote rotinas de pré-processamento, planos de corte, heurísticas e estratégias de *branching*. Ele é executado via terminal de linha de comando e pode ser operado no modo paralelo, para que se aproveite os vários núcleos do computador.

Escrito na linguagem de programação C/C++, o CBC possui código aberto, permitindo que desenvolvedores implementem algoritmos customizados a partir dele. Além disso, ele é um dos resolvedores de código aberto mais rápidos hoje em dia e também é um componente fundamental usado por resolvedores de Programação Não-Linear, como SHOT (KRONQVIST; LUNDELL; WESTERLUND, 2016) e Bonmin (BELOTTI et al., 2009).

2.3 Trabalhos relacionados

Existem na literatura diversos trabalhos que abordam a configuração automática de algoritmos. Entretanto, poucos trabalhos tratam da configuração automática de algoritmos para resolver problemas de PLI. Alguns dos principais trabalhos relacionados a esse tema são brevemente descritos a seguir.

[1] <https://github.com/coin-or/Cbc>

Xu et al. (2008) apresentam uma nova versão do SATzilla, uma abordagem automatizada para construir portfólios de algoritmos para problemas de Satisfatibilidade Booleana (SAT). Essa abordagem tem como entrada um conjunto de instâncias de problemas SAT e um conjunto de algoritmos solucionadores e constrói um portfólio por meio da otimização de uma determinada função objetivo (tempo de execução médio, porcentagem de instâncias resolvidas, etc.). Os autores mostram que a partir da construção do portfólio é possível aumentar consideravelmente o número de problemas resolvidos. Os portfólios produzem funções de *scoring* complexas e utilizam algoritmos de busca local. Após extensos experimentos computacionais os autores obtiveram melhorias substanciais no resolvedor SATzilla.

Ansótegui, Sellmann e Tierney (2009) propuseram um algoritmo baseado em gênero para configuração automática de algoritmos. Os autores mostraram que o algoritmo proposto fornece um conjunto de parâmetros de alta qualidade de forma robusta, podendo assim melhorar significativamente o algoritmo pioneiro. Esse trabalho realizou uma separação baseada em gênero, permitindo que o algoritmo focasse em conjuntos de alta qualidade e assim retornasse os resultados em uma velocidade substancial.

Hutter, Hoos e Brown (2010) estudaram a aplicação de procedimentos de configuração de diferentes algoritmos automatizados para diferentes resolvedores de PLI. Através disso, o estudo mostrou que esse processo produz melhorias consideráveis nos desempenhos dos algoritmos. Os autores argumentam que o sucesso na abordagem depende da disponibilidade de conjuntos de treinamento disponibilizadas. Para conjuntos de benchmark pequenos, essas melhorias não são significativas.

Xu et al. (2011) fizeram um estudo sobre a configuração e seleção automatizada de algoritmos para PLI. Trabalhando com um método automatizado para configuração de algoritmos conhecido como Hydra, os autores mostraram que é possível alcançar um desempenho melhor para problemas de PLI. Nesse trabalho foi descrito um novo algoritmo de seleção baseado na classificação com uma função de perda disforme, que obteve uma melhoria significativa no desempenho da seleção de algoritmos para PLI. Os autores também modificaram o Hydra para a seleção de configurações candidatas e com isso obtiveram um melhor desempenho em função do tempo de treinamento.

Martins et al. (2016) propuseram recomendações de parâmetros baseadas em técnicas de classificação para o resolvedor CBC na resolução de problemas de PLI. Foram selecionadas dezesseis combinações de parâmetros, obtendo-se uma taxa de 98% de assertividade utilizando algoritmos de árvore de decisão.

Boas et al. (2021) propuseram uma abordagem baseada em um modelo de PLI para gerar árvores de decisão ótimas para o Problema de Seleção de Algoritmos (PSA). A formulação produz árvores de decisão com profundidade limitada. A principal vantagem dessa abordagem é que, apesar da construção da árvore em si ser potencialmente cara do ponto

de vista computacional, uma vez que a árvore tenha sido construída, as recomendações do algoritmo podem ser feitas de forma constante. Foram realizados experimentos em um conjunto com instâncias de várias aplicações, considerando o resolvedor linear COIN-OR Linear Programming (CLP) e seus parâmetros. Os resultados revelaram configurações de parâmetros aprimoradas para esse resolvedor, incluindo a descoberta de uma nova configuração que deixou o CLP 22% mais rápido do que as configurações padrão.

3 Análise e Desenvolvimento

Este capítulo apresenta a metodologia utilizada no desenvolvimento do trabalho. Inicialmente, são definidos os parâmetros do CBC que serão analisados e testados. Em seguida, uma base de dados contendo problemas de PLI é construída e dividida aleatoriamente em dois conjuntos: treinamento e teste. A partir da definição dos parâmetros de interesse e do conjunto de instâncias de treinamento, um script do `irace` é implementado e executado. Por fim, o CBC é executado considerando as melhores configurações de parâmetros retornadas pelo `irace` e as instâncias do conjunto de teste.

3.1 Definição dos parâmetros

A primeira etapa realizada para a definição dos parâmetros a serem investigados consistiu em listar todos os parâmetros que podem ser configurados pelo usuário do CBC. Esses parâmetros podem ser visualizados através do menu de ajuda. Para isso, é necessário entrar no modo interativo do resolvedor, executando-o pelo terminal de linha de comando sem passar nenhum parâmetro. Nesse modo de execução, o menu de ajuda é acionado ao digitar o símbolo de interrogação (?), exibindo os comandos, ações e parâmetros disponibilizados pelo CBC. A Figura 2 mostra o resultado dessa operação.

Figura 2 – Comandos, ações e parâmetros disponibilizados pelo CBC.

```
Enter ? for list of commands or help
Coin:?
In argument list keywords have leading - , -stdin or just - switches to stdin
One command per line (and no -)
abcd? gives list of possibilities, if only one + explanation
abcd?? adds explanation, if only one fuller help
abcd without value (where expected) gives current value
abcd value sets value
Commands are:
Double parameters:
 dualB(ound) dualT(olerance) primalT(olerance) primalW(eight) psi
 progress((Interval)) zeroT(olerance)
Branch and Cut double parameters:
 allow(ableGap) cuto(ff) inc(rement) integerT(olerance) preT(olerance)
 pumpC(utoff) ratio(Gap) sec(onds) secni(fs)
Integer parameters:
 force(Solution) idiot(Crash) maxF(actor) maxIt(erations) output(Format)
 randomS(eed) slog(Level) sprint(Crash)
Branch and Cut integer parameters:
 cutD(epth) cutL(ength) depth(MiniBab) bkpivot(ing) bkmaxcalls bkclqext(method)
 oddwext(method) hot(StartMaxIts) log(Level) maxN(odes) maxNI(FS)
 maxSaved(Solutions) maxSo(lutions) passC(uts) passF(easibilityPump)
 passT(reeCuts) pumpT(une) randomC(bcSeed) slow(cutpasses) strat(egy)
 strong(Branching) trust(PseudoCosts)
Keyword parameters:
 allC(ommands) chol(esky) crash cross(over) direction error(sAllowed)
 fact(orization) keepN(ames) mess(ages) perturb(ation) presolve
 printi(ngOptions) scal(ing) timeM(ode)
Branch and Cut keyword parameters:
 combine(Solutions) combine2(Solutions) constraint(fromCutoff) cost(Strategy)
 cplex(Use) cuts(OnOff) Dins DivingS(ome) DivingC(oefficient) DivingF(ractional)
 DivingG(uided) DivingL(ineSearch) DivingP(seudoCost) DivingV(ectorLength)
 dw(Heuristic) feas(ibilityPump) flow(CoverCuts) GMI(Cuts) gomory(Cuts)
 greedy(Heuristic) heur(isticsOnOff) clique(Cuts) cgraph oddwheel(Cuts)
 clqstr(engthen) knapsack(Cuts) lagomory(Cuts) latwomir(Cuts)
 lift(AndProjectCuts) local(TreeSearch) mixed(IntegerRoundingCuts)
 node(Strategy) PrepN(ames) pivotAndC(omplement) pivotAndF(ix) preprocess
 probing(Cuts) proximity(Search) randomi(zedRounding) reduce(AndSplitCuts)
 reduce2(AndSplitCuts) residual(CapacityCuts) Rens Rins round(ingHeuristic)
 sosO(ptions) sosP(rioritize) two(MirCuts) Vnd(VariableNeighborhoodSearch)
 zero(HalfCuts)
Actions or string parameters:
 allS(lack) barr(ier) basisI(n) basisO(ut) directory dualS(implex)
 either(Simplex) end exit export gsolu(tion) guess help import initialS(olve)
 max(imize) min(imize) para(metrics) primalS(implex) printM(ask) quit
 restoreS(olution) saveS(olution) solu(tion) stat(istics) stop
Branch and Cut actions:
 branch(AndCut) doH(euristic) mips(tart) nextB(estSolution) prio(rityIn) solv(e)
```

A definição e o valor atual de cada parâmetro do CBC podem ser visualizados ao digitar no modo interativo o comando referente ao parâmetro seguido de dois símbolos de interrogação (??). A Figura 3 apresenta um exemplo de utilização desse comando, mostrando a definição, os possíveis valores e o valor atual do parâmetro *clique*.

Figura 3 – Definição, possíveis valores e valor atual do parâmetro *clique*.

```
Coin:clique??
clique(Cuts) : Whether to use Clique cuts
This switches on clique cuts (either at root or in entire tree). An
improved version of the Bron-Kerbosch algorithm is used to separate
cliques.
<Possible options for cliqueCuts are: off on root ifmove forceOn onglobal;
        current  ifmove>
```

O procedimento descrito anteriormente foi executado para todos os parâmetros exibidos no menu de ajuda do CBC. Em seguida, essas informações foram analisadas e 25 parâmetros foram selecionados para compor o conjunto de parâmetros que foram analisados neste trabalho. Dada a limitação de tempo para o desenvolvimento deste trabalho, não foi possível analisar todos os parâmetros do resolvedor. A maioria dos parâmetros selecionados atuam na execução de rotinas de planos de corte e heurísticas, estratégias que visam a acelerar o processo de resolução de problemas de PLI. A Tabela 1 apresenta os parâmetros que foram selecionados, bem como os valores que eles podem assumir e as ações que executam.

Tabela 1 – Parâmetros do CBC que foram utilizados neste trabalho.

Parâmetro	**Valores**	**Ação**
clique	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes de clique
flow	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Flow Cover*
gomory	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes de *Gomory*
knapsack	off, on, root, ifmove, forceOn, onglobal, forceandglobal	ativa ou desativa cortes da mochila
mixed	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Mixed Integer Rounding*
oddwheel	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Odd Wheel*
probing	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Probing*
two	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Two Phase Mixed Integer Rounding*
zero	off, on, root, ifmove, forceOn, onglobal	ativa ou desativa cortes do tipo *Zero Half*
passF	0, 10, 30, 100	define o número de iterações a serem executadas pela heurística *Feasibility Pump*
DivingC	off, on, both, before	ativa ou desativa a heurística *Coefficient Diving*
DivingF	off, on, both, before	ativa ou desativa a heurística *Fractional Diving*
DivingG	off, on, both, before	ativa ou desativa a heurística *Guided Diving*
DivingL	off, on, both, before	ativa ou desativa a heurística *Linesearch Diving*
DivingP	off, on, both, before	ativa ou desativa a heurística *Pseudocost Diving*
DivingV	off, on, both, before	ativa ou desativa a heurística *Vectorlength Diving*
feas	off, on, both, before	ativa ou desativa a heurística *Feasibility Pump*
greedy	off, on, both, before	ativa ou desativa uma heurística gulosa baseada na fixação de variáveis
Rins	off, on, both, before, often	ativa ou desativa a heurística *Relaxed Induced Neighborhood Search* (RINS)
round	off, on, both, before	ativa ou desativa uma heurística simples (mas eficiente) de arredondamento
node	hybrid, fewest, depth, upfewest, downfewest, updepth, downdepth	método para escolher o próximo nó da árvore de *branch-and-cut*
strat	0, 2	seleciona um grupo de estratégias a serem utilizadas pelo resolvedor
strong	0, 5, 100, 1000, 10000	número de variáveis a serem consideradas no *Strong Branching*
clqstr	after, off, before	ativa ou desativa a rotina de preprocessamento *Clique Strengthening*
preprocess	off, on, equal, sos, trysos, strategy	ativa ou desativa o preprocessamento em problemas de PLI

3.2 Construção da base de dados

A base de dados utilizada neste trabalho é composta por diversos problemas de PLI disponibilizados pela *Mixed-Integer Programming Library* (MIPLIB). A MIPLIB foi introduzida na comunidade acadêmica em 1992, e desde então tornou-se uma biblioteca padrão para comparar o desempenho dos resolvedores de PLI. Ela contém uma coleção de problemas acadêmicos e reais, sendo a maior parte composta de aplicações industriais.

Inicialmente, foram coletadas todas as instâncias do conjunto *Benchmark* da MIPLIB 2010[1] e da MIPLIB 2017[2], totalizando 327 instâncias. Em seguida, foram removidas as instâncias duplicadas e aquelas que representavam modelos infactíveis, restando 304 instâncias. As 304 instâncias foram executadas no CBC com os valores padrão dos parâmetros e considerando um tempo limite de 600 segundos.

Dentro do tempo limite estabelecido, 64 instâncias foram resolvidas na otimalidade pelo CBC. Essas instâncias formam a base de dados utilizadas neste trabalho. De maneira análoga à definição de um conjunto reduzido de parâmetros para estudo, a decisão de utilizar problemas resolvidos em até 10 minutos pelo CBC foi tomada devido à limitação de tempo para finalização deste trabalho. Mesmo considerando um subconjunto reduzido de parâmetros e problemas de PLI, a execução dos experimentos computacionais consumiu uma quantidade significativa do tempo gasto na execução deste trabalho, conforme será apresentado na Seção 3.3.

Após a definição da base de dados, as instâncias foram divididas aleatoriamente em dois conjuntos de mesmo tamanho: 32 instâncias para *treinamento* e 32 instâncias para *teste*. O conjunto de treinamento contém as instâncias a serem resolvidas pelo CBC durante a execução do script do `irace`, para definir as melhores configurações de parâmetros. Por outro lado, o conjunto de teste é utilizado para avaliar as configurações de parâmetros retornadas pelo `irace`, comparando o desempenho com os valores padrão dos parâmetros do CBC.

Os problemas de PLI que compõem a base de dados utilizada neste trabalho são apresentados na Tabela 2. A coluna “Conjunto” define se o problema pertence à classe de treinamento ou de teste, enquanto as colunas “Variáveis”, “Restrições” e “Não-zeros” apresentam, respectivamente, o número de variáveis, restrições e elementos com valores diferentes de zero na matriz de restrições de cada problema.

[1] <http://miplib2010.zib.de/>
[2] <https://miplib.zib.de/>

Tabela 2 – Instâncias que compõem a base de dados utilizada neste trabalho.

Instância	**Conjunto**	**Variáveis**	**Restrições**	**Não-zeros**
acc-tight5	treino	1.339	3.052	16.134
air05	treino	7.195	426	52.121
bienst2	treino	505	576	2.184
binkar10_1	treino	1.026	1.026	4.496
bley_xl1	treino	5.831	175.620	869.391
cbs-cta	treino	24.793	10.112	64.388
dano3_3	treino	13.873	3.202	79.655
dano3_5	treino	13.873	3.202	79.655
drayage-100-23	treino	11.090	4.630	41.550
irp	treino	20.315	39	98.254
lectsched-4-obj	treino	14.163	14.163	82.428
markshare_4_0	treino	34	4	123
mik-250-20-75-4	treino	270	195	9.270
mine-166-5	treino	830	8.429	19.412
neos-1122047	treino	5.100	57.791	163.640
neos-1171448	treino	4.914	13.206	131.859
neos-1582420	treino	10.100	10.180	24.814
neos-4413714-turia	treino	190.402	2.303	761.756
neos-686190	treino	3.660	3.664	18.085
neos-827175	treino	32.504	14.187	110.790
neos-933966	treino	31.762	12.047	180.618
neos-934278	treino	23.123	11.495	125.577
nu25-pr12	treino	5.868	2.313	17.712
nursesched-sprint02	treino	10.250	3.522	204.000
physiciansched6-2	treino	111.827	168.336	480.259
qiu	treino	840	1.192	3.432
rmatr100-p5	treino	37.816	37.617	113.048
sp150x300d	treino	600	450	1.200
sp98ir	treino	1.680	1.531	71.704
swath1	treino	6.805	884	34.965
tanglegram2	treino	4.714	8.980	26.940
triptim1	treino	30.055	15.706	515.436
air04	teste	8.904	823	72.965
app1-1	teste	2.480	4.926	18.275
core2536-691	teste	15.293	2.539	177.739
decomp2	teste	14.387	10.765	64.073
eilB101	teste	2.818	100	24.120
fast0507	teste	63.009	507	409.349
fiball	teste	34.219	3.707	104.792
mas74	teste	151	13	1.706
mas76	teste	151	12	1.640
mik-250-1-100-1	teste	251	151	5.351
mzzv11	teste	10.240	9.499	134.603
mzzv42z	teste	11.717	10.460	151.261
neos-1109824	teste	1.520	28.979	89.528
neos-1171737	teste	2.340	4.179	58.620
neos-2987310-joes	teste	27.837	29.015	580.291
neos-3083819-nubu	teste	8.644	4.725	24.048
neos-3381206-awhea	teste	2.375	479	4.275
neos8	teste	23.228	46.324	313.180
neos-860300	teste	1.385	850	384.329
neos-957323	teste	57.756	3.757	499.656
neos-960392	teste	59.376	4.744	189.503
nw04	teste	87.482	36	636.666
pg	teste	2.700	125	5.200
piperout-08	teste	10.399	14.589	44.959
piperout-27	teste	11.659	18.442	54.662
pk1	teste	86	45	915
qap10	teste	4.150	1.820	18.200
ran16x16	teste	512	288	1.024
rmatr100-p10	teste	7.359	7.260	21.877
roll3000	teste	1.166	2.295	29.386
seymour1	teste	1.372	4.944	33.549
tbfp-network	teste	72.747	2.436	215.837

3.3 Etapa de treinamento: execução do *irace*

A maioria dos algoritmos que resolvem problemas de otimização são configuráveis. Isso acontece porque não existe apenas uma combinação ótima, já que cada problema possui sua particularidade. Assim, a definição dos melhores conjuntos de parâmetros e seus respectivos valores para um algoritmo também pode ser vista como um problema de otimização.

A ferramenta `irace` foi utilizada para encontrar configurações de parâmetros que melhorem o desempenho do resolvedor CBC na resolução de problemas de PLI. De acordo com López-Ibánez et al. (2016), a corrida iterativa implementada no `irace` se inicia com uma quantidade de configurações de parâmetro candidatas. A cada passo, as configurações são avaliadas em um único problema de entrada e, ao final desse passo, as piores configurações são identificadas e descartadas. A avaliação utilizada neste trabalho considera o tempo gasto pelo CBC para resolver um problema de PLI na otimalidade, sendo o objetivo encontrar configurações de parâmetros que minimizem esse tempo. Após o descarte das piores configurações, a corrida iterada continua avaliando as remanescentes, executando até analisar todas as configurações de parâmetros em todas as instâncias ou até alcançar um determinado critério de parada. O critério de parada pode ser o tempo total gasto pelo `irace` ou o número de experimentos realizados.

Foram gerados alguns *scripts* para configurar e executar o `irace`. Esses *scripts* contêm as informações relacionadas ao critério de parada, número de *threads* utilizadas, conjunto de instâncias e execução do CBC. Neste trabalho optou-se por limitar o número de experimentos realizados em 50.000. Para acelerar o processo de execução foram utilizadas 8 *threads* e cada execução do CBC foi limitada a no máximo 600 segundos. O conjunto de instâncias utilizado pelo `irace` compreende os problemas da classe de treinamento apresentados na Tabela 2.

O *script* que controla a execução do CBC considera as situações onde determinadas configurações de parâmetros podem gerar resultados inconsistentes, como soluções incorretas ou execuções interrompidas prematuramente. Essas situações podem ser causadas por problemas de instabilidade numérica ou falhas nos algoritmos do resolvedor. Uma configuração de parâmetros que gera resultados inconsistentes é penalizada e o valor recebido em sua avaliação é o dobro do tempo limite utilizado no CBC (1.200). Assim, configurações dessa natureza são descartadas ao final de cada passo.

Definidas as configurações necessárias, o `irace` foi executado em um computador com 96 GB de RAM, processador Intel® Core™ i9-7900X de 3.30 GHz e Sistema Operacional Linux Ubuntu 20.04 de 64bits. Foram necessários aproximadamente 40 dias para finalizar a execução. Ao final do *log* de execução, a ferramenta apresentou uma lista com as melhores configurações de parâmetros encontradas para o cenário considerado. Para fins de

análise e comparação dos resultados, este trabalho considerou as 3 melhores configurações retornadas pelo `irace`. Essas configurações são apresentadas na Tabela 3, onde *Config1* é a melhor configuração encontrada para os parâmetros analisados, seguida por *Config2* e *Config3*, respectivamente. A tabela apresenta também os valores de parâmetros definidos como padrão pelo CBC, representados pela configuração *Default*.

Tabela 3 – Configuração padrão e melhores configurações para os parâmetros do CBC.

Param	**Default**	**Config1**	**Config2**	**Config3**
clique	ifmove	root	root	root
flow	ifmove	forceOn	off	off
gomory	ifmove	onglobal	onglobal	ifmove
knapsack	ifmove	root	root	root
mixed	ifmove	onglobal	ifmove	onglobal
oddwheel	ifmove	root	ifmove	root
probing	on	forceOn	forceOn	forceOn
two	root	on	on	on
zero	ifmove	ifmove	onglobal	onglobal
passF	30	10	10	30
DivingC	on	off	off	off
DivingF	off	off	off	off
DivingG	off	off	off	off
DivingL	off	off	off	off
DivingP	off	off	off	off
DivingV	off	off	off	off
feas	on	on	on	on
greedy	on	both	on	on
Rins	on	on	on	on
round	on	on	both	on
node	fewest	fewest	fewest	fewest
strat	1	1	1	1
strong	5	5	5	5
clqstr	after	off	off	off
preprocess	sos	sos	sos	sos

A partir da análise da Tabela 3 é possível perceber que 11 dos 25 parâmetros analisados possuem valores iguais àqueles definidos como padrão pelo CBC. Considerando as 3 melhores configurações retornadas pelo `irace`, apenas 8 parâmetros apresentam diferenças em seus valores: *flow*, *gomory*, *mixed*, *oddwheel*, *zero*, *passF*, *greedy* e *round*.

Após a definição das configurações de parâmetros, foram realizados experimentos computacionais para analisar o desempenho do CBC. A seção a seguir apresenta os resultados obtidos.

3.4 Etapa de teste

As três melhores configurações de parâmetros retornadas pelo `irace` foram submetidas a um experimento para analisar o desempenho do CBC, utilizando as instâncias do conjunto de teste (ver Tabela 2) e considerando um tempo limite de 600 segundos

por execução. Para fins de comparação, considerou-se os valores de parâmetros definidos como padrão pelo CBC. O computador utilizado nessa etapa foi o mesmo utilizado na etapa de treinamento. Os resultados são reportados na Tabela 4, onde as colunas *Default*, *Config1*, *Config2* e *Config3* apresentam, respectivamente, os tempos de execução do CBC considerando as configurações de parâmetros padrão e as três melhores configurações de parâmetros retornadas pelo `irace` (ver Tabela 3). Os valores destacados em negrito apresentam os melhores tempos de execução encontrados para cada instância.

Tabela 4 – Tempo gasto (em segundos) pelo CBC para resolver as instâncias do conjunto de teste.

Instância	**Default**	**Config1**	**Config2**	**Config3**
air04	46,65	**36,99**	45,71	53,56
app1-1	61,61	9,72	**9,32**	10,60
core2536-691	369,65	**296,92**	303,67	417,71
decomp2	20,98	19,84	**19,47**	19,75
eilB101	460,77	221,90	**211,96**	229,03
fast0507	600,00	421,95	**377,04**	600,00
fiball	168,41	**128,88**	168,74	137,63
mas74	457,80	**389,57**	406,50	458,17
mas76	53,91	23,39	**21,82**	33,56
mik-250-1-100-1	146,14	**144,21**	158,47	149,27
mzzv11	148,61	**112,89**	143,30	147,18
mzzv42z	53,19	53,28	**49,96**	68,06
neos-1109824	506,78	**156,96**	377,88	158,26
neos-1171737	600,00	**227,08**	264,63	266,97
neos-2987310-joes	8,76	**6,18**	6,21	9,14
neos-3083819-nubu	27,76	25,04	**24,44**	24,75
neos-3381206-awhea	38,13	**9,20**	16,65	16,16
neos-860300	63,41	**48,70**	86,45	78,76
neos-957323	353,66	**57,37**	232,67	335,32
neos-960392	181,18	105,46	98,16	**80,34**
neos8	45,37	**34,15**	36,60	46,23
nw04	**15,92**	22,49	21,55	21,74
pg	34,74	**30,19**	31,68	32,15
piperout-08	138,69	**59,56**	142,57	69,14
piperout-27	600,00	469,28	**132,89**	139,71
pk1	69,19	63,73	61,89	**52,46**
qap10	81,56	70,61	**70,51**	76,18
ran16x16	542,83	519,66	**242,99**	296,33
rmatr100-p10	107,41	**71,66**	92,99	97,11
roll3000	**489,07**	600,00	600,00	600,00
seymour1	553,87	308,43	284,27	**283,65**
tbfp-network	311,99	224,09	**220,58**	224,26
Média	229,94	155,29	155,05	163,54

Em algumas execuções, o tempo limite de 600 segundos foi atingido. É o caso das execuções do CBC considerando a configuração *default* nas instâncias *fast0507*, *neos-1171737* e *piperout-27*. Além disso, as configurações *Config1*, *Config2* e *Config3* também atingiram o tempo limite na instância *roll3000*.

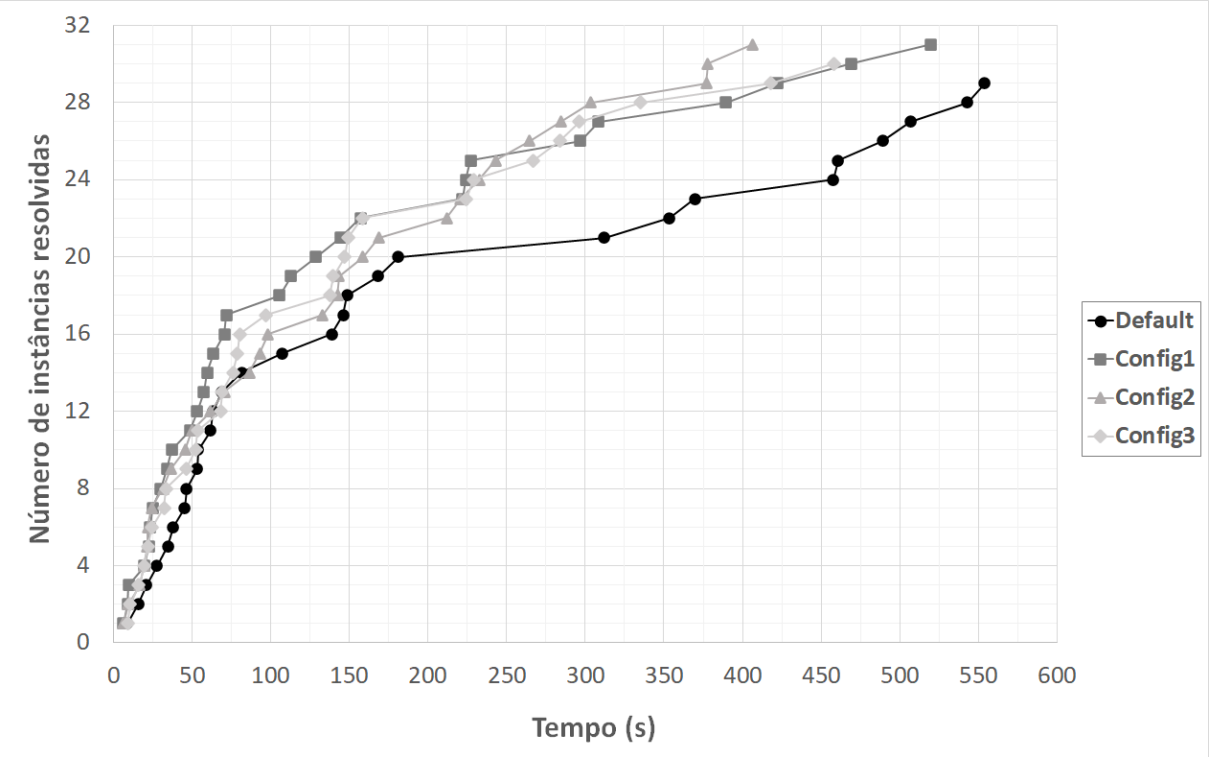
A configuração de parâmetros *Default* do CBC apresentou um tempo de execução

médio de 229, 94 segundos. Por outro lado, as configurações *Config1*, *Config2* e *Config3* retornadas pelo `irace` obtiveram, respectivamente, tempos de execução médios de 155, 29, 155, 05 e 163, 54 segundos. Esses resultados representam uma melhoria significativa no desempenho do CBC, diminuindo em até 33% o tempo médio gasto para resolver os problemas do conjunto de teste.

O melhor ganho de desempenho ocorreu na instância *app1-1*, onde as configurações retornadas pelo `irace` diminuíram significativamente o tempo gasto pelo CBC. Nessa instância, o tempo de execução foi diminuído em 84% utilizando *Config1*, 85% considerando *Config2* e 83% empregando os valores de parâmetros de *Config3*.

A Figura 4 apresenta, para cada configuração, o número de instâncias resolvidas ao longo do tempo. Analisando o gráfico é possível perceber que as configurações que possibilitaram a resolução de um número maior de instâncias foram as configurações retornadas pelo `irace`. Nesse caso, a segunda configuração foi a que resolveu mais instâncias e também foi a que gastou menos tempo, resolvendo 31 das 32 instâncias do conjunto de teste em apenas 406 segundos.

Figura 4 – Número de instâncias resolvidas por tempo de execução (em segundos).



A configuração *Default* teve resultados piores em praticamente toda a linha de tempo de execução da Figura 4, quase sempre apresentando os piores tempos e resolvendo menos instâncias do que as demais. Através do gráfico também é possível perceber que, na maior parte da resolução das instâncias, a utilização das configurações retornadas pelo

`irace` obtiveram tempos semelhantes entre si.

Em geral, é possível perceber a melhoria no desempenho do CBC gerada a partir da aplicação das configurações de parâmetros retornadas pelo `irace` na etapa de treinamento. Essas novas configurações permitiram que o CBC gastasse menos tempo na resolução de problemas e também resolvesse mais instâncias de acordo com o tempo considerado.

4 Conclusão e Trabalhos Futuros

Este trabalho teve como premissa avaliar o comportamento do resolvedor CBC de acordo com a análise de um conjunto de parâmetros. O intuito foi comparar o desempenho desse resolvedor utilizando os valores padrão para os seus parâmetros com os valores de parâmetros retornados por uma ferramenta de configuração automática de algoritmos. Para isso, um conjunto de parâmetros de interesse foi definido e uma base de dados foi construída. Com auxílio da ferramenta `irace`, foram encontradas três configurações de parâmetros com valores diferentes daqueles definidos como padrão pelo CBC. Essas configurações foram utilizadas para testar o CBC e os resultados foram satisfatórios, alcançando até 85% de melhoria no desempenho do resolvedor. Na média, a melhoria do desempenho obtido foi de até 33%.

Como trabalhos futuros destacam-se o estudo e análise dos demais parâmetros do CBC, utilização de técnicas de validação cruzada e inclusão de mais instâncias nos experimentos computacionais. A utilização de uma outra ferramenta de configuração automática de algoritmos, como o ParamILS (HUTTER et al., 2009), também pode ser empregada para realizar um estudo comparativo com os resultados obtidos pelo `irace`.

Referências

ANSÓTEGUI, C.; SELLMANN, M.; TIERNEY, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In: SPRINGER. *International Conference on Principles and Practice of Constraint Programming*. [S.l.], 2009. p. 142–157. Citado na página 19.

BELOTTI, P. et al. Branching and bounds tightening techniques for non-convex minlp. *Optimization Methods and Software*, Taylor & Francis, v. 24, n. 4-5, p. 597–634, 2009. Citado na página 18.

BOAS, M. G. V. et al. Optimal decision trees for the algorithm selection problem: integer programming based approaches. *International Transactions in Operational Research*, v. 28, n. 5, p. 2759–2781, 2021. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12724>. Citado na página 19.

BRITO, S. S.; SANTOS, H. G. Preprocessing and cutting planes with conflict graphs. *Computers & Operations Research*, v. 128, p. 105176, 2021. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054820302938>. Citado na página 14.

FERNANDES, L. J. et al. Planejamento e controle da produção de cilindros para laminação: um estudo de caso quantitativo. *Production*, scielo, v. 23, p. 120 – 134, 03 2013. ISSN 0103-6513. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-65132013000100009&nrm=iso>. Citado na página 15.

GLEIXNER, A. et al. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. Disponível em: <https://doi.org/10.1007/s12532-020-00194-3>. Citado na página 16.

HILLIER, F. S.; LIEBERMAN, G. J. *Introdução à pesquisa operacional*. [S.l.]: McGraw Hill Brasil, 2013. Citado na página 14.

HUTTER, F.; HOOS, H. H.; BROWN, K. L. *Automated configuration of mixed integer programming solvers*. Springer and berlin and heidelberg. [S.l.]: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, 2010. Citado na página 19.

HUTTER, F. et al. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, v. 36, p. 267–306, October 2009. Citado na página 31.

KRONQVIST, J.; LUNDELL, A.; WESTERLUND, T. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, Springer, v. 64, n. 2, p. 249–272, 2016. Citado na página 18.

LOESCH, C.; HEIN, N. *Pesquisa operacional: fundamentos e modelos*. [S.l.]: Saraiva, 2009. Citado na página 15.

LÓPEZ-IBÁNEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, Elsevier, v. 3, p. 43–58, 2016. Citado 4 vezes nas páginas 9, 17, 18 e 26.

MARTINS, R. d. S. O. et al. Recomendação de parâmetros para o coin-or branch and cut. In: *Anais do XLVIII Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2016. p. 2508–2514. Citado na página 19.

XU, L. et al. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, v. 32, p. 565–606, 2008. Citado na página 19.

XU, L. et al. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In: *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*. [S.l.: s.n.], 2011. p. 16–30. Citado na página 19.