UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Layza Nauane de Paula Silva

# Comparação de desempenho entre diferentes solvers de Otimização Linear e Inteira

**Uberlândia, Brasil**

**2026**

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Layza Nauane de Paula Silva

# Comparação de desempenho entre diferentes solvers de Otimização Linear e Inteira

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2026

# Agradecimentos

Primeiramente, agradeço a Deus por me fortalecer, guiar e sustentar ao longo desta jornada. Em todos os momentos, pude perceber Seu cuidado e Sua presença.

Agradeço à minha mãe, Elisangela, ao meu pai, José Edgardo, ao meu noivo, Emanuel, aos meus irmãos, Yasmin e Carlos, aos meus queridos avós, Rita e Sebastião, e a toda a minha família, pelo apoio, incentivo e compreensão ao longo desta jornada. Vocês sempre foram minha base.

Aos meus amigos Jaqueline, Isabella, Paula e Mario, agradeço pela amizade, parceria e pelos momentos compartilhados durante essa etapa tão importante. Com vocês, essa caminhada se tornou mais leve e especial.

Por fim, expresso meus sinceros agradecimentos ao meu orientador, Paulo Henrique Ribeiro Gabriel, pela orientação, dedicação e pelos conhecimentos compartilhados ao longo deste trabalho. Sua contribuição foi essencial para o desenvolvimento e a realização desta pesquisa.

# Resumo

Este trabalho apresenta uma análise comparativa de desempenho entre quatro solvers open-source para otimização linear e inteira: GLPK (GNU Linear Programming Kit), CBC (COIN-OR Branch and Cut), HiGHS e SCIP (Solving Constraint Integer Programs). O objetivo consiste em avaliar a eficiência computacional desses solvers quando integrados à biblioteca de modelagem Pyomo, em Python, considerando critérios como tempo de execução, qualidade da solução, escalabilidade e estabilidade. A metodologia compreende duas etapas experimentais: validação inicial com problemas clássicos de pequena escala; e avaliação de desempenho em 36 instâncias de maior porte, organizadas simetricamente em três tipos estruturais por problema e seis dimensões crescentes, de 100 a 10000 variáveis. Foram utilizadas instâncias sintéticas geradas de forma controlada para o Problema da Dieta (Programação Linear) e instâncias padronizadas da literatura para o Problema da Mochila (Programação Inteira). Cada solver foi executado 10 vezes por instância, com limite de tempo de 5 segundos por execução, e os resultados foram analisados em termos de média, desvio padrão e taxa de obtenção de soluções ótimas. Os resultados indicam que não existe um solver universalmente superior para todas as classes de problemas. Para Programação Linear, GLPK e HiGHS apresentaram desempenho consistente nas instâncias esparsas, enquanto o CBC se destacou nas instâncias mais densas. Para Programação Inteira, o desempenho variou significativamente conforme a estrutura das instâncias: o GLPK foi o mais rápido em instâncias não correlacionadas e fracamente correlacionadas, mas tornou-se o único solver incapaz de provar otimalidade nas instâncias fortemente correlacionadas em escala alta – caso em que o CBC se mostrou o mais robusto. O SCIP apresentou maior custo computacional nos problemas puramente lineares, comportamento provavelmente relacionado à sua arquitetura orientada à Programação Inteira.

**Palavras-chave**: Otimização Matemática; Programação Linear; Programação Inteira; Solvers Open-Source; Pyomo; Python.

# Abstract

This work presents a comparative performance analysis between four open-source solvers for linear and integer optimization: GLPK (GNU Linear Programming Kit), CBC (COIN-OR Branch and Cut), HiGHS, and SCIP (Solving Constraint Integer Programs). The objective is to evaluate the computational efficiency of these solvers when integrated with the Pyomo modeling library in Python, considering execution time, solution quality, scalability, and stability. The methodology comprises two experimental stages: initial validation with classical small-scale problems; and performance evaluation across 36 larger instances, symmetrically organized into three structural types per problem and six increasing dimensions, ranging from 100 to 10000 variables. We used synthetic instances generated under controlled conditions for the Diet Problem (Linear Programming), and standardized instances from the literature for the Knapsack Problem (Integer Programming). Each solver was run 10 times per instance, with a time limit of 5 seconds per run, and the results were analyzed in terms of mean, standard deviation, and the rate of obtaining optimal solutions. The results indicate that there is no universally superior solver across all problem classes. For Linear Programming, GLPK and HiGHS performed consistently on sparse instances, while CBC stood out on denser instances. For Integer Programming, performance varied significantly according to the structure of the instances: GLPK was the fastest in uncorrelated and weakly correlated instances, but became the only solver unable to prove optimality in strongly correlated instances at a high scale – a case in which CBC proved to be the most robust. SCIP showed higher computational cost in purely linear problems, a behavior probably related to its Integer Programming-oriented architecture.

**Keywords**: Mathematical Optimization; Linear Programming; Integer Programming; Open-Source Solvers; Pyomo; Python.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| B&B | Branch-and-Bound |
| CBC | COIN-OR Branch and Cut |
| CPL | Common Public License |
| GLPK | GNU Linear Programming Kit |
| GNU | GNU's Not Unix |
| GPL | GNU General Public License |
| HiGHS | Solver de otimização linear e inteira de alto desempenho |
| MIP | Mixed Integer Programming (Programação Inteira Mista) |
| MIPLIB | Mixed Integer Programming Library |
| MINLP | Mixed Integer Nonlinear Programming (Programação Inteira Não Linear Mista) |
| OSI | Open Solver Interface |
| PI | Programação Inteira |
| PIB | Programação Inteira Binária |
| PIP | Programação Inteira Pura |
| PL | Programação Linear |
| QP | Quadratic Programming (Programação Quadrática) |
| RAM | Random Access Memory |
| SCIP | Solving Constraint Integer Programs |
| WSL2 | Windows Subsystem for Linux 2 |

# Sumário

# 1 Introdução

Problemas de otimização estão presentes em diversos contextos reais, tanto no ambiente industrial quanto em aplicações computacionais, nas quais é necessário tomar decisões eficientes diante de restrições como tempo, custo, capacidade ou disponibilidade de recursos. De forma geral, otimizar consiste em identificar a melhor solução possível para um problema, levando-o a uma condição considerada ideal segundo critérios previamente estabelecidos (ANDRÉASSON; EVGRAFOV; PATRIKSSON, 2020).

Nesse contexto, a partir da década de 1950, com o desenvolvimento da Pesquisa Operacional, a otimização matemática passou a ganhar destaque como uma ferramenta essencial para apoio à tomada de decisão em áreas como logística, economia, engenharia e saúde.

Com o avanço da capacidade computacional, modelos de Programação Linear (PL) e Programação Inteira (PI) tornaram-se amplamente utilizados para representar problemas reais de diferentes níveis de complexidade. A resolução desses modelos depende do uso de algoritmos especializados, implementados em softwares conhecidos como solvers, responsáveis por explorar o espaço de soluções e encontrar respostas ótimas ou próximas do ótimo em tempo viável.

O desempenho desses solvers está diretamente relacionado ao tempo de execução, à escalabilidade, à estabilidade e, em alguns casos, à qualidade das soluções obtidas, especialmente em problemas de maior porte ou com variáveis inteiras. Em problemas de natureza combinatória, esse desempenho torna-se ainda mais crítico, devido ao crescimento exponencial do espaço de soluções, o que impõe desafios computacionais significativos (COSTA, 2008).

Estudos como o de Avella, Calamita e Palagi (2023) indicam que o desempenho dos solvers pode variar consideravelmente conforme a natureza e a complexidade do problema tratado. Nesse contexto, torna-se relevante a realização de análises comparativas que permitam identificar o comportamento dessas ferramentas em diferentes cenários.

A motivação deste trabalho está associada ao crescente interesse pelo uso de solvers open-source, impulsionado pela ausência de custos de licenciamento, pela transparência dos algoritmos e pela facilidade de integração com linguagens de programação, como o Python. Apesar de sua ampla utilização, ainda há uma escassez de estudos comparativos conduzidos em ambientes controlados, utilizando diferentes classes de problemas e critérios de avaliação padronizados.

Diante desse cenário, este trabalho tem como objetivo comparar o desempenho de

quatro solvers open-source – GLPK, CBC, HiGHS e SCIP – na resolução de problemas de Programação Linear e Inteira, utilizando a biblioteca Pyomo em Python. A avaliação considera critérios como tempo de execução, qualidade da solução, escalabilidade e estabilidade.

Como objetivos específicos, busca-se (*i*) validar o funcionamento dos solvers por meio de problemas de pequena escala; (*ii*) avaliar o desempenho em instâncias de maior complexidade; e (*iii*) comparar os resultados obtidos entre os solvers.

A hipótese que orienta este estudo é que solvers mais recentes, como HiGHS e SCIP, possam apresentar melhor desempenho em instâncias maiores e em problemas de Programação Inteira. Essa suposição baseia-se nos avanços algorítmicos incorporados a essas ferramentas ao longo dos últimos anos, bem como nos resultados observados em estudos anteriores, como o de Machado (2024), no qual o HiGHS e o SCIP obtiveram desempenho competitivo mesmo em comparação com solvers comerciais. Os resultados, no entanto, confirmaram parcialmente essa hipótese: o desempenho relativo dos solvers variou conforme a classe e a estrutura do problema, e os solvers mais tradicionais, como GLPK e CBC, mostraram-se competitivos ou superiores em diversos cenários avaliados.

Os resultados dos experimentos conduzidos permitem identificar padrões de desempenho que orientam a escolha de solvers open-source, oferecendo evidências empíricas para que pesquisadores e profissionais selecionem a ferramenta mais adequada a cada contexto de otimização.

As próximas seções deste trabalho estão organizadas da seguinte forma: O Capítulo 2 apresenta a fundamentação teórica, abordando conceitos de modelagem matemática, programação linear e inteira, principais algoritmos utilizados na resolução dos problemas e os solvers avaliados, além de discutir trabalhos relacionados. O Capítulo 3 descreve o desenvolvimento do trabalho, incluindo a configuração do ambiente experimental, os critérios de avaliação, os problemas utilizados e a apresentação e discussão dos resultados obtidos nas instâncias de pequena e grande escala. Por fim, o Capítulo 4 apresenta as conclusões, as limitações identificadas e as sugestões de trabalhos futuros.

# 2 Fundamentação Teórica

## 2.1 Modelagem Matemática

A Pesquisa Operacional (PO) pode ser compreendida como uma abordagem científica voltada ao apoio à tomada de decisão, baseada na construção de modelos matemáticos que representam problemas reais de forma estruturada (SOUTO-MAIOR, 2014). Nesse contexto, a modelagem consiste em traduzir uma situação prática do dia a dia para a linguagem matemática, identificando variáveis de decisão, função objetivo e restrições.

De forma geral, um problema de otimização pode ser representado na forma padrão:

$$\text{minimizar ou maximizar} \quad f(x) \tag{2.1}$$

$$\text{sujeito a} \quad g_i(x) \leq 0, \quad i = 1, \ldots, m \tag{2.2}$$

$$h_j(x) = 0, \quad j = 1, \ldots, p \tag{2.3}$$

$$x \in \Omega \tag{2.4}$$

onde:

- $x = (x_1, x_2, \ldots, x_n)$ representa o vetor de variáveis de decisão;
- $f(x)$ é a função objetivo, responsável por expressar o critério de desempenho a ser maximizado ou minimizado;
- $h_j(x)$ e $g_i(x)$ representam, respectivamente, as restrições de igualdade e de desigualdade, que delimitam o conjunto de soluções possíveis;
- $\Omega$ define o domínio das variáveis, especificando os valores que podem ser assumidos por cada componente de x.

A modelagem matemática é uma etapa fundamental, pois a forma como o problema é representado influencia diretamente a eficiência da solução obtida e sua aplicabilidade prática. No contexto deste trabalho, a modelagem assume papel central, uma vez que os modelos são definidos de forma padronizada por meio da biblioteca Pyomo, permitindo a comparação direta entre diferentes solvers.

## 2.2 Programação Linear

A Programação Linear (PL), também conhecida como Otimização Linear, é um dos principais ramos da otimização matemática, sendo aplicada a problemas em que tanto

a função objetivo quanto as restrições são representadas por relações lineares. Devido à sua estrutura bem definida e ampla aplicabilidade, a PL é utilizada em diversas áreas, como logística, produção e finanças.

Um problema de Programação Linear pode ser formulado como (MACULAN; FAMPA, 2004):

$$\text{minimizar ou maximizar} \quad z = \sum_{j=1}^{p} c_j x_j \tag{2.5}$$

$$\text{sujeito a} \quad \sum_{j=1}^{p} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, q \tag{2.6}$$

$$x_j \geq 0, \quad j = 1, \ldots, p. \tag{2.7}$$

Nessa formulação, $c_j$, $a_{ij}$ e $b_i$ são parâmetros reais do modelo, enquanto $x_j$ representam as variáveis de decisão. A expressão linear definida na função objetivo estabelece o critério a ser minimizado ou maximizado. As restrições garantem que as decisões respeitem os limites impostos pelo problema, e as condições $x_j \geq 0$ correspondem às restrições de não negatividade.

Quando as variáveis são contínuas, o conjunto de soluções viáveis forma uma região convexa, o que garante que, caso exista uma solução ótima finita, ela estará localizada em um dos vértices dessa região. Essa característica torna a PL computacionalmente mais tratável, sendo frequentemente utilizada como base para a resolução de problemas mais complexos, incluindo aqueles com variáveis inteiras.

## 2.3 Programação Inteira

A Programação Inteira (PI) é uma extensão da Programação Linear na qual algumas ou todas as variáveis de decisão são restritas a assumir valores inteiros. Essa característica é essencial para modelar problemas em que as decisões são discretas, como seleção de itens ou alocação de recursos indivisíveis.

Segundo Alves e Delgado (1997), um problema de PI pode ser classificado em três categorias principais: Programação Inteira Pura, quando todas as variáveis são inteiras; Programação Inteira Mista, quando apenas parte das variáveis é inteira; e Programação Inteira Binária, quando as variáveis assumem valores 0 ou 1.

A Programação Inteira mantém a estrutura da Programação Linear, adicionando apenas a restrição de integralidade sobre parte ou todas as variáveis. No geral, pode ser

representada por (MACULAN; FAMPA, 2004):

$$\text{minimizar ou maximizar} \quad z = \sum_{j=1}^{p} c_j x_j \tag{2.8}$$

$$\text{sujeito a} \quad \sum_{j=1}^{p} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, q \tag{2.9}$$

$$x_j \geq 0, \quad j = 1, \ldots, p, \tag{2.10}$$

$$x_j \in \mathbb{Z}, \quad j \in I, \tag{2.11}$$

onde $I$ indica o conjunto das variáveis que devem assumir valores inteiros.

A presença de variáveis inteiras torna o problema significativamente mais complexo do ponto de vista computacional, uma vez que o espaço de soluções deixa de ser contínuo e passa a ser discreto. Como consequência, a resolução desses problemas pode demandar esforço computacional elevado, especialmente em instâncias de maior porte. No contexto deste trabalho, essa característica é especialmente relevante, pois as diferenças de desempenho entre os solvers tendem a se tornar mais evidentes em problemas de Programação Inteira.

## 2.4 Algoritmos para Programação Linear

### 2.4.1 Método Simplex

O Método Simplex foi desenvolvido por Dantzig (1963) e tornou-se um dos algoritmos mais conhecidos para resolver problemas de Programação Linear. Seu surgimento foi decisivo para viabilizar a aplicação prática da PL em problemas de maior porte, marcando um avanço importante na consolidação da Pesquisa Operacional.

O funcionamento do Simplex é iterativo: em vez de analisar todos os vértices da região factível, o método percorre apenas alguns pontos estratégicos até alcançar a solução ótima (TAHA, 2007). Ele parte de um vértice inicial e se desloca para vértices vizinhos, seguindo pelas arestas da região viável. Esse processo continua enquanto houver possibilidade de melhorar o valor da função objetivo; quando não há mais melhoria possível, a solução ótima é atingida.

Com o tempo, surgiram variações que ampliaram sua aplicação. O **Simplex Dual** inicia com uma solução ótima, porém inviável, e conduz o processo até torná-la factível, sendo útil em análises pós-ótimas. O **Simplex Revisado** utiliza formulação matricial, tornando os cálculos mais eficientes e numericamente estáveis em problemas maiores. Já o método com **variáveis limitadas** adapta o algoritmo para tratar limites superiores e inferiores nas variáveis, recurso comum em modelos de Programação Inteira (TAHA, 2007).

O Simplex permanece amplamente utilizado em implementações modernas, sendo frequentemente empregado na resolução de problemas de Programação Linear e nas relaxações lineares de problemas inteiros, o que impacta diretamente o desempenho de diversos solvers analisados neste trabalho.

### 2.4.2 Método dos Pontos Interiores

O interesse pelo Método dos Pontos Interiores se intensificou a partir do trabalho de Karmarkar (1984), que apresentou um algoritmo de complexidade polinomial para Programação Linear, impulsionando o desenvolvimento de toda uma família de métodos baseados nessa abordagem.

Ao contrário do Método Simplex, que percorre as bordas da região viável, os métodos de pontos interiores avançam pelo interior do conjunto de soluções factíveis em direção ao ótimo. São teoricamente elegantes e convergem em tempo polinomial (CELIS, 2018).

Na prática, para problemas de grande porte, o número de iterações tende a ser menor do que no Simplex, porém cada passo é computacionalmente mais custoso, pois exige a resolução de sistemas lineares. Entre as principais variantes desenvolvidas estão o método projetivo, o método afim-escala e o método de caminho central, este último responsável por manter os iterados próximos a uma trajetória interior parametrizada que converge para a solução ótima (CELIS, 2018).

Esse método é especialmente eficiente em problemas de grande escala, sendo utilizado em alguns solvers modernos como alternativa ao Simplex, dependendo da estrutura do problema.

## 2.5 Algoritmos para Programação Inteira

### 2.5.1 Branch-and-Bound

O método Branch-and-Bound (Ramificação e Limitação), ou simplemente B&B, foi criado por Land e Doig (1960), sendo o principal algoritmo para a resolução de problemas de PI, sendo amplamente empregado em modelos inteiros puros e mistos (PAIXÃO; DANIELSSON; ABAIDE, 2025).

Segundo Alves e Delgado (1997), o método B&B é um dos principais algoritmos utilizados para resolver problemas de Otimização Inteira. Ele inicia resolvendo a relaxação linear do problema. Se a solução obtida já for inteira, ela é ótima. Caso contrário, o problema é dividido em subproblemas menores, adicionando restrições que forçam valores inteiros às variáveis fracionárias. A cada divisão, são calculados limites para a função

objetivo, permitindo descartar subproblemas que não podem gerar soluções melhores. Esse processo continua até que a melhor solução inteira seja encontrada.

Como problemas de PI são, em geral, NP-difíceis, no pior caso, o método B&B apresenta complexidade exponencial. Ainda assim, é muito usado como abordagem para a resolução de modelos inteiros, principalmente quando combinado a boas estratégias de ramificação e avaliação.

O desempenho desse método está diretamente relacionado ao número de nós explorados na árvore de busca, o que influencia significativamente o tempo de execução dos solvers em problemas de Programação Inteira.

### 2.5.2 Método dos Planos de Corte

O método dos planos de corte foi desenvolvido por Gomory (1958). Segundo Alves e Delgado (1997, p. 14), o algoritmo consiste em:

> Introduzir sucessivamente novas restrições na relaxação linear do PLI, restrições essas que cortam o conjunto das soluções possíveis eliminando algumas delas e a própria solução óptima do PL (por isso se chamam planos de corte), sem contudo eliminar qualquer solução inteira.

Na prática, os planos de corte são frequentemente combinados ao método Branch-and-Bound, dando origem ao algoritmo Branch-and-Cut, amplamente implementado em solvers modernos de programação inteira.

Esses algoritmos constituem a base das técnicas implementadas nos solvers modernos, sendo frequentemente combinados com heurísticas, estratégias de ramificação e mecanismos de otimização adicionais. Dessa forma, o desempenho de cada solver está diretamente relacionado à forma como essas técnicas são integradas e ajustadas internamente.

## 2.6 Solvers

Na prática, a resolução de problemas de otimização é realizada por meio de solvers, que implementam algoritmos como Simplex, pontos interiores e Branch-and-Bound, além de heurísticas e técnicas de otimização adicionais.

Dessa forma, o desempenho de um solver não depende apenas do algoritmo base utilizado, mas também de estratégias internas de otimização, o que justifica a realização de estudos comparativos como o proposto neste trabalho.

Nas subseções seguintes, são apresentados os solvers open-source que foram utilizados nos experimentos deste trabalho.

### 2.6.1 GLPK

O GLPK (GNU Linear Programming Kit) é um solver open-source desenvolvido pelo Projeto GNU para a resolução de problemas de Programação Linear (LP) e Programação Inteira Mista (MIP) (GNU Project, 2024). É implementado em linguagem C e pode ser utilizado tanto como biblioteca quanto por meio do executável standalone *glpsol* (GNU, 2024).

O GLPK implementa métodos clássicos de otimização, como o método Simplex, o método de ponto interior e técnicas de *branch-and-cut* para problemas inteiros. Também oferece suporte à linguagem de modelagem GNU MathProg.

O projeto é mantido pela Free Software Foundation e distribuído sob a licença GNU General Public License (GPL), o que permite seu uso, modificação e redistribuição em projetos acadêmicos e de software livre, desde que mantidas as mesmas condições de licença.

### 2.6.2 HiGHS

O HiGHS é um solver open-source voltado para problemas de Programação Linear (LP), Programação Inteira Mista (MIP) e Programação Quadrática (QP), com foco em instâncias de grande porte e estrutura esparsa (HiGHS, 2024). É desenvolvido em C++ e pode ser utilizado como biblioteca ou executável standalone.

O solver implementa variações do método Simplex, métodos de ponto interior e técnicas de *branch-and-cut*. O HiGHS é distribuído sob a licença MIT, uma licença permissiva que permite uso, modificação e redistribuição do código, inclusive em projetos proprietários.

### 2.6.3 SCIP

O SCIP (Solving Constraint Integer Programs) é um solver direcionado principalmente à Programação Inteira Mista (MIP) e à Programação Inteira Não Linear Mista (MINLP) (SCIP, 2024). Ele possui uma estrutura avançada que inclui técnicas de ramificação, geração de planos de corte, propagação, precificação e decomposição de Benders.

A partir da versão 8.0.3, o SCIP é distribuído sob a licença Apache 2.0, que permite uso e modificação com poucas restrições. Já as versões anteriores (até 8.0.2) utilizam a Licença Acadêmica ZIB, que permite uso gratuito apenas para fins de pesquisa e ensino.

### 2.6.4 CBC

O CBC (COIN-OR Branch and Cut) é um solver open-source para Programação Inteira Mista (MIP), desenvolvido no âmbito do projeto COIN-OR (COIN-OR, 2024). É implementado em C++ e utiliza a abordagem de *branch-and-cut* para a busca de soluções inteiras. Pode ser utilizado como biblioteca ou por meio de executável standalone, integrando-se a diferentes ferramentas de modelagem através da Open Solver Interface (OSI).

O CBC é distribuído sob a Common Public License (CPL), uma licença open-source que permite uso e modificação do código, exigindo que eventuais alterações no próprio solver permaneçam sob a mesma licença.

## 2.7 Trabalhos Relacionados

A comparação de desempenho entre solvers de otimização linear e inteira tem sido objeto de diversos estudos, especialmente em contextos onde a eficiência computacional impacta diretamente a viabilidade prática dos modelos.

Um estudo relevante é o de Meindl e Templ (2013), que compararam solvers comerciais e open-source para problemas de otimização linear e inteira, motivados pelo problema de supressão de células, utilizado na proteção de dados estatísticos. Os autores combinaram duas abordagens: por um lado, reuniram resultados de benchmarks já consolidados – baseados em instâncias da biblioteca MIPLIB, nos quais os solvers comerciais resolviam mais problemas dentro de um limite de tempo; por outro, conduziram um estudo de caso próprio sobre esse problema. De modo geral, os solvers comerciais apresentaram melhor desempenho, enquanto os open-source mostravam maior limitação nos problemas mais complexos. Esse trabalho é importante por apresentar uma metodologia clara de comparação, baseada em critérios objetivos.

Em outra linha, Beltramin (2015) realizou uma comparação computacional entre dois solvers comerciais amplamente utilizados, analisando diferenças de desempenho em problemas de Programação Inteira Mista. O estudo mostrou que, embora ambos apresentem resultados semelhantes em muitos casos, pequenas diferenças nas estratégias internas podem influenciar o tempo de resolução. Esse tipo de análise contribui para compreender como aspectos algorítmicos impactam o desempenho prático dos solvers.

O estudo de Koch et al. (2022) olhou para a evolução dos solvers ao longo do tempo, analisando o quanto eles melhoraram entre 2001 e 2020. Os autores observaram que boa parte desse avanço veio de melhorias nos próprios algoritmos, e não apenas do hardware mais rápido. Isso ajuda a explicar por que se espera que solvers mais novos, como o HiGHS e o SCIP, sejam mais eficientes que os mais antigos. Essa é justamente uma das

ideias que este trabalho busca colocar à prova, mas focando nos solvers open-source usados junto com o Pyomo.

Mais recentemente, Alves et al. (2024) também compararam três ferramentas open-source de otimização usando o problema clássico das N-Rainhas, modelado como Programação Inteira Mista. O objetivo era identificar qual delas seria mais robusta para aplicações reais, avaliando desempenho, escalabilidade e eficiência computacional à medida que o tamanho do problema crescia. Esse estudo é especialmente próximo do presente trabalho, pois também parte de um problema clássico para comparar solvers livres, reforçando que esse tipo de avaliação continua sendo relevante e atual.

Além do desempenho computacional, a literatura também aponta questões relacionadas à confiabilidade dos resultados. Neumaier e Shcherbina (2004) demonstraram que, em alguns casos, erros numéricos podem levar solvers a retornarem conclusões incorretas, especialmente em problemas mal condicionados. Esse estudo destaca que a análise de desempenho não deve considerar apenas o tempo de execução, mas também a estabilidade e a precisão das soluções obtidas.

Entre os trabalhos apresentados, o de Machado (2024) foi o que mais orientou esta pesquisa. Nele, os autores desenvolveram um benchmark comparando diferentes solvers aplicados a modelos metabólicos em larga escala, avaliando tempo de execução, estabilidade e capacidade de resolver instâncias maiores. Os solvers open-source foram comparados diretamente com solvers comerciais, e os resultados mostraram que ferramentas livres, como o HiGHS e o SCIP, já alcançam um desempenho competitivo. Foi a partir dessa observação que surgiu a hipótese deste trabalho: se os solvers open-source mais recentes se mostram competitivos até diante de opções comerciais, era razoável esperar que, em uma comparação apenas entre solvers open-source da mesma categoria, os mais modernos também levassem vantagem sobre os mais tradicionais.

De modo geral, os trabalhos analisados mostram que a comparação entre solvers é uma prática consolidada e necessária para avaliar eficiência, robustez e aplicabilidade. No entanto, muitos estudos concentram-se em solvers comerciais ou em contextos específicos de aplicação, não considerando, em geral, uma comparação padronizada em ambiente controlado. Assim, o presente trabalho contribui ao realizar uma análise comparativa focada exclusivamente em solvers open-source amplamente utilizados, integrados ao ambiente Pyomo em Python, considerando critérios como tempo de execução, qualidade da solução, escalabilidade e estabilidade computacional.

# 3 Desenvolvimento

## 3.1 Configuração do Ambiente Experimental

A fim de garantir a reprodutibilidade dos experimentos e evitar interferências externas nos resultados, como a influência que versões de software, bibliotecas ou sistema operacional pode ter nas soluções, o ambiente utilizado neste trabalho foi previamente definido e documentado.

Os experimentos foram realizados utilizando a linguagem Python, com o apoio da biblioteca Pyomo, responsável pela modelagem dos problemas de otimização. Neste trabalho optou-se pelo Pyomo por três motivos. Primeiro, sua interface unificada com diversos solvers, que permite resolver o mesmo modelo com diferentes solvers sem necessidade de reescrever a formulação, o que é essencial para uma comparação justa entre as ferramentas avaliadas. Segundo, trata-se de uma biblioteca open-source desenvolvida em Python, alinhada à proposta do trabalho de avaliar exclusivamente ferramentas gratuitas e de código aberto. Terceiro, é uma biblioteca consolidada na literatura científica e em aplicações práticas de Pesquisa Operacional. Porém, cabe destacar que a modelagem também poderia ser realizada por meio de outras bibliotecas, como o PuLP, que oferece sintaxe mais simples e uma menor curva de aprendizado.

Os experimentos foram conduzidos em um computador com processador Intel Core i5-1334U de 13ª geração (10 núcleos físicos, 12 threads), arquitetura x86_64, 8 GB de memória RAM e sistema operacional Windows 11. Para a execução, foi utilizado um ambiente Linux por meio do WSL2 (Windows Subsystem for Linux). A escolha pelo Linux se deve à sua estabilidade e ao seu amplo uso em ambientes de pesquisa e computação científica.

Todos os testes foram executados no mesmo computador e nas mesmas condições, a fim de evitar que fatores externos influenciassem os resultados. Além disso, não foram utilizadas configurações avançadas específicas de cada solver, buscando manter uma comparação mais justa e padronizada.

### 3.1.1 Dependências e Versões Utilizadas

A linguagem de programação utilizada foi Python 3.12.12. Para a modelagem dos problemas de otimização, foi utilizada a biblioteca Pyomo (versão 6.10.0), que é amplamente empregada na formulação e resolução de problemas de otimização em Python (HART; WATSON; WOODRUFF, 2011; BYNUM et al., 2021).

O gerenciamento das dependências foi feito por meio de um ambiente virtual criado com o módulo `venv`, o que permite isolar o projeto do sistema operacional e manter a consistência das versões das bibliotecas em futuras reproduções dos experimentos.

Além disso, as dependências foram registradas em um arquivo `requirements.txt`, contendo as versões exatas utilizadas. A Tabela 1 apresenta a configuração completa do ambiente experimental.

Tabela 1 – Configuração do ambiente experimental

| **Componente** | **Versão / Especificação** |
|---|---|
| Sistema Operacional | Linux 6.6.87.2-microsoft-standard-WSL2 |
| Processador | 13th Gen Intel(R) Core(TM) i5-1334U |
| Arquitetura | x86_64 |
| Memória RAM | 8 GB |
| Python | 3.12.12 |
| Pyomo | 6.10.0 |
| HiGHS (highspy) | 1.13.1 |
| SCIP | 10.0.1 |
| GLPK | 5.0 |
| CBC | 2.10.11 |
| matplotlib | 3.10.8 |
| numpy | 2.4.2 |

Fonte: Elaboração própria.

Os solvers GLPK e CBC foram instalados por meio do gerenciador de pacotes do sistema operacional. O solver HiGHS foi utilizado por meio do pacote Python `highspy`, enquanto o SCIP foi instalado como executável a partir de sua distribuição oficial.

Todos os solvers foram acessados de forma padronizada por meio da interface do Pyomo, o que permitiu a execução dos modelos de maneira uniforme, independentemente da ferramenta utilizada. O código-fonte desenvolvido neste trabalho está disponível em: <https://github.com/LayzaDev/solver-benchmark-pyomo>

### 3.1.2 Verificação do Ambiente

Antes da execução dos experimentos, foi realizado um procedimento de verificação do ambiente computacional, com o objetivo de garantir que todos os solvers estavam corretamente instalados e acessíveis por meio da interface Pyomo.

Para essa verificação, foi desenvolvido um script em Python que executa um modelo linear simples utilizando cada um dos solvers avaliados. Esse teste permite confirmar a comunicação entre o Pyomo e os solvers, bem como verificar se o processo de resolução ocorre sem erros.

## 3.2 Metodologia Experimental

Esta seção descreve a metodologia experimental adotada neste trabalho, incluindo os critérios utilizados na avaliação dos solvers, as instâncias selecionadas para os experimentos, o procedimento de execução e a forma de análise dos resultados obtidos.

### 3.2.1 Critérios de Avaliação

Os solvers foram comparados com base nos seguintes critérios:

- **Tempo de execução**: quanto tempo cada solver levou para encontrar a solução;
- **Qualidade da solução**: verificação se a solução encontrada é correta e ótima;
- **Escalabilidade**: comportamento do solver quando o problema aumenta de tamanho;
- **Estabilidade**: se o solver conclui a execução sem erros ou interrupções.

### 3.2.2 Instâncias de Menor Porte

Antes dos experimentos de maior porte, que constituem o foco deste trabalho, foi realizada uma etapa preliminar com instâncias de pequena escala. Seu objetivo não foi comparar o desempenho dos solvers, mas validar o ambiente experimental e verificar se os modelos do Problema da Dieta e do Problema da Mochila estavam corretamente formulados, produzindo soluções consistentes em todos os solvers avaliados. Os problemas selecionados foram:

- Problema da Dieta, que consiste em otimizar o custo financeiro de um conjunto de alimentos, respeitando restrições nutricionais. Trata-se de um clássico problema de Programação Linear (DANTZIG, 1990).
- Problema da Mochila, que consiste em selecionar um conjunto de itens para serem alocados em uma mochila, respeitando a capacidade dessa mochila. Esse é um problema bastante estudado de Programação Inteira (KELLERER; PFERSCHY; PISINGER, 2004).

Confirmada essa consistência, a análise de desempenho propriamente dita foi conduzida sobre as instâncias de maior escala, que serão apresentadas posteriormente na seção 3.4

### 3.2.3 Instâncias de Maior Porte

Na segunda etapa, foram utilizadas instâncias de maior porte, organizadas de forma simétrica para os dois problemas avaliados: dezoito instâncias para o Problema da Dieta e dezoito instâncias para o Problema da Mochila, totalizando trinta e seis experimentos de maior escala.

Para o Problema da Dieta, foram geradas instâncias sintéticas com semente aleatória fixa, agrupadas em três tipos estruturais que variam a densidade da matriz de restrições nutricionais: Tipo 1 (*esparso*), com 10 nutrientes; Tipo 2 (*médio*), com 30 nutrientes; e Tipo 3 (*denso*), com 50 nutrientes. Cada tipo foi avaliado em seis dimensões crescentes: 100, 500, 1000, 2000, 5000 e 10000 alimentos, totalizando dezoito instâncias.

Para o Problema da Mochila, foram utilizadas instâncias padronizadas de Pisinger (2005), agrupadas em três tipos de correlação entre pesos e valores dos itens: Tipo 1 (não correlacionado), Tipo 2 (fracamente correlacionado) e Tipo 3 (fortemente correlacionado). Cada tipo foi avaliado nas mesmas seis dimensões: 100, 500, 1000, 2000, 5000 e 10000 itens, totalizando também dezoito instâncias.

Essa organização simétrica – três tipos por problema, seis tamanhos por tipo – permite comparar o comportamento dos solvers tanto em função do aumento de escala quanto em função das diferentes estruturas internas de cada classe de problema.

### 3.2.4 Procedimento de Execução

Para cada problema e para cada solver foi realizada inicialmente uma execução de aquecimento, cujo objetivo foi reduzir a influência de custos iniciais associados ao carregamento do solver e do ambiente de execução. Essa execução inicial foi descartada da análise.

Após o aquecimento, cada solver foi executado dez vezes por instância, sendo registrado o tempo de resolução em milissegundos em cada execução. A partir desses valores foram calculadas a média e o desvio padrão dos tempos de execução.

Foi definido um limite de tempo de 5 segundos por execução, aplicado a ambos os problemas e configurado individualmente em cada solver por meio dos parâmetros correspondentes: `tmlim` no GLPK, `sec` no CBC, `time_limit` no HiGHS e `limits/time` no SCIP. Quando o limite é atingido, o solver interrompe a busca e retorna a melhor solução encontrada até aquele momento, sem garantia de otimalidade. Nesse caso, o status da execução é registrado como *feasible*, em contraposição ao status *optimal*, que indica que a otimalidade foi provada dentro do tempo disponível.

Também foi realizada uma verificação de consistência das soluções obtidas. Para o Problema da Dieta, confirmou-se que todos os solvers retornaram o mesmo valor de custo

mínimo em cada instância. Para o Problema da Mochila, o valor encontrado por cada solver foi comparado ao valor ótimo conhecido, registrado no próprio arquivo de instância por Pisinger (2005), calculando-se o gap percentual conforme a expressão:

$$\text{gap} = \frac{v^* - v_{\text{encontrado}}}{v^*} \times 100\% \tag{3.1}$$

onde $v^*$ é o valor ótimo conhecido e $v_{\text{encontrado}}$ é o valor retornado pelo solver.

### 3.2.5 Análise dos Resultados

Após a coleta dos dados, foram calculados:

- médias de tempo de execução;
- desvio padrão entre as repetições;
- análise do comportamento em problemas de maior porte.

Os resultados são apresentados em tabelas e gráficos, permitindo melhor visualização das diferenças de desempenho entre os solvers.

## 3.3 Problemas de Pequena Escala

### 3.3.1 Problema da Dieta

O Problema da Dieta é um modelo clássico de PL que busca determinar a combinação de alimentos de menor custo capaz de satisfazer um conjunto mínimo de requisitos nutricionais. Na instância considerada neste trabalho foram utilizados 12 alimentos e 6 nutrientes. Cada alimento possui um custo associado e valores nutricionais correspondentes, incluindo calorias, proteínas, carboidratos, gorduras, fibras e cálcio. Os dados nutricionais e os valores de custo utilizados nesta instância foram adaptados do trabalho de Tirone (2019).

O modelo tem como objetivo minimizar o custo total da dieta, garantindo que os requisitos mínimos de cada nutriente sejam atendidos. As variáveis de decisão representam a quantidade de cada alimento incluída na dieta e são modeladas como variáveis contínuas não negativas. A Tabela 2 apresenta os dados utilizados na instância do problema.

### 3.3.2 Problema da Mochila

O Problema da Mochila é um modelo clássico de Programação Inteira cujo objetivo é selecionar um conjunto de itens que maximize o valor total transportado sem

Tabela 2 – Dados da instância do Problema da Dieta

| **Alimento** | **Custo** | **Calorias** | **Proteína** | **Carboidratos** | **Gordura** | **Fibra** | **Cálcio** |
|---|---|---|---|---|---|---|---|
| Arroz | 1.50 | 200 | 4 | 45 | 1 | 1 | 10 |
| Feijão | 1.20 | 150 | 10 | 28 | 1 | 7 | 50 |
| Frango | 4.00 | 250 | 30 | 0 | 8 | 0 | 15 |
| Brócolis | 2.00 | 55 | 4 | 11 | 1 | 3 | 100 |
| Leite | 1.80 | 120 | 8 | 12 | 5 | 0 | 300 |
| Ovo | 0.80 | 80 | 6 | 1 | 5 | 0 | 28 |
| Batata | 1.00 | 160 | 3 | 37 | 0 | 3 | 12 |
| Atum | 3.50 | 130 | 28 | 0 | 1 | 0 | 20 |
| Cenoura | 1.10 | 40 | 1 | 9 | 0 | 2 | 33 |
| Banana | 0.90 | 90 | 1 | 23 | 0 | 3 | 5 |
| Pão Integral | 1.40 | 130 | 5 | 24 | 2 | 4 | 40 |
| Queijo | 2.50 | 110 | 7 | 1 | 9 | 0 | 200 |

Fonte: Adaptado de Tirone (2019).

ultrapassar a capacidade máxima da mochila. Na instância considerada neste trabalho foram utilizados oito itens, cada um associado a um peso e a um valor. A mochila possui capacidade máxima de 4 kg.

O modelo tem como objetivo maximizar o valor total dos itens selecionados, enquanto a restrição principal garante que o peso total não ultrapasse a capacidade estabelecida. As variáveis de decisão são modeladas como variáveis binárias, assumindo valor 1 quando o item é selecionado e 0 caso contrário. A Tabela 3 apresenta os itens utilizados na instância do problema.

Tabela 3 – Itens do Problema da Mochila

| **Item** | **Peso (kg)** | **Valor** |
|---|---|---|
| Notebook | 2.5 | 80 |
| Câmera | 1.2 | 60 |
| Livro | 0.8 | 20 |
| Garrafa | 0.6 | 15 |
| Fone | 0.3 | 40 |
| Carregador | 0.4 | 35 |
| Lanche | 0.5 | 25 |
| Casaco | 1.0 | 30 |

Fonte: Elaboração própria.

### 3.3.3 Resultados e Discussão

No problema da dieta, todos os solvers analisados encontraram a mesma solução ótima para a instância considerada, com custo mínimo igual a R$ 17.0572. A verificação de consistência entre os solvers foi realizada admitindo uma tolerância numérica de $10^{-4}$, para acomodar eventuais diferenças de arredondamento em ponto flutuante entre as

implementações. A solução obtida inclui os alimentos feijão, leite, ovo e batata, em quantidades suficientes para satisfazer as restrições nutricionais do modelo. As quantidades de cada alimento presentes na solução ótima são apresentadas na Tabela 4.

Tabela 4 – Resultados do Problema da Dieta

| **Solver** | **Status** | **Custo (R$)** | **Tempo médio (ms)** | **Desvio (ms)** |
|---|---|---|---|---|
| GLPK | Optimal | 17.0572 | 3.10 | 0.17 |
| HiGHS | Optimal | 17.0572 | 3.43 | 0.36 |
| SCIP | Optimal | 17.0572 | 15.85 | 2.69 |
| CBC | Optimal | 17.0572 | 17.99 | 2.16 |

Fonte: Elaboração própria.

No problema da Mochila, todos os solvers analisados também encontraram a mesma solução ótima para a instância considerada, com valor total igual a 205 e peso total de 4 kg. Os itens selecionados na solução ótima foram câmera, garrafa, fone, carregador, lanche e casaco, respeitando a capacidade máxima de peso da mochila. A Tabela 5 apresenta a média e o desvio padrão dos tempos de execução observados para cada solver.

Tabela 5 – Resultados do Problema da Mochila

| **Solver** | **Status** | **Valor (pts)** | **Tempo médio (ms)** | **Desvio (ms)** |
|---|---|---|---|---|
| GLPK | Optimal | 205 | 4.57 | 1.87 |
| HiGHS | Optimal | 205 | 8.05 | 1.91 |
| SCIP | Optimal | 205 | 17.02 | 2.38 |
| CBC | Optimal | 205 | 22.73 | 3.80 |

Fonte: Elaboração própria.

Os resultados obtidos mostram que todos os solvers analisados foram capazes de encontrar a mesma solução ótima para ambos os problemas considerados. Esse resultado indica que os modelos foram implementados corretamente e que a interface de modelagem utilizada permite a execução consistente dos experimentos em diferentes solvers. Em relação ao tempo de execução, observa-se que as diferenças entre os solvers são relativamente pequenas, uma vez que os problemas possuem dimensão reduzida e baixo custo computacional. Ainda assim, nota-se que os solvers GLPK e HiGHS apresentaram os menores tempos médios de execução nas instâncias analisadas, enquanto CBC e SCIP apresentaram tempos ligeiramente superiores.

No entanto, esses experimentos de pequena escala têm caráter exclusivamente validatório, sendo utilizados para verificar a consistência dos modelos e o correto funcionamento dos solvers. Dessa forma, não são considerados para fins de comparação de desempenho, a qual é realizada apenas nas instâncias de maior escala. Como o tamanho das instâncias é reduzido, os problemas são resolvidos rapidamente por qualquer algoritmo de otimização moderno, o que limita a capacidade de diferenciar o comportamento

dos solvers. Dessa forma, torna-se necessário avaliar os solvers em instâncias de maior porte, nas quais o aumento do número de variáveis e restrições tende a tornar o esforço computacional mais significativo. Esses experimentos são apresentados na seção seguinte.

## 3.4 Problemas de Maior Escala

### 3.4.1 Problema da Dieta

Para o Problema da Dieta, foram geradas instâncias sintéticas de maior porte seguindo a mesma estrutura do problema original: minimizar o custo total de uma dieta que satisfaça requisitos nutricionais mínimos. As instâncias foram criadas com dados nutricionais gerados de forma controlada, com semente aleatória fixa (semente = 42) para garantir a reprodutibilidade dos experimentos. Os valores de custo por porção foram sorteados no intervalo [R$ 0,50; R$ 5,00], e os conteúdos nutricionais por porção foram gerados em intervalos típicos para cada categoria de nutriente, conforme padrões nutricionais de referência. Os requisitos mínimos de cada nutriente foram definidos como uma fração entre 5% e 15% da oferta total disponível, garantindo que todas as instâncias possuam solução factível.

As instâncias foram organizadas em três tipos estruturais, que variam o número de nutrientes e, consequentemente, a densidade da matriz de restrições do modelo:

- **Tipo 1** (*esparso*): 10 nutrientes por instância. Representa problemas com poucas restrições ativas, nos quais a matriz de coeficientes nutricionais é esparsa.
- **Tipo 2** (*médio*): 30 nutrientes por instância. Representa um cenário intermediário, com densidade moderada de restrições.
- **Tipo 3** (*denso*): 50 nutrientes por instância. Representa problemas com maior número de restrições ativas, impondo maior esforço ao solver na fase de resolução do sistema linear.

Cada tipo foi avaliado em seis dimensões crescentes de alimentos disponíveis: 100, 500, 1000, 2000, 5000 e 10000, totalizando dezoito instâncias para o Problema da Dieta. Essa progressão permite observar como o tempo de resolução de cada solver escala tanto com o aumento do número de variáveis quanto com o aumento da densidade das restrições.

### 3.4.2 Problema da Mochila

Para o Problema da Mochila, foram utilizadas instâncias padronizadas disponibilizadas por Pisinger (2005), amplamente empregadas na literatura para avaliação de

algoritmos de otimização inteira. As instâncias foram obtidas a partir de um repositório espelho de Martins (2024), uma vez que os links de download originais do autor encontravam-se indisponíveis no momento da realização dos experimentos. As instâncias seguem a nomenclatura `knapPI_T_N_R_I`, onde `T` indica o tipo de correlação entre pesos e valores dos itens, `N` o número de itens, `R` o intervalo de geração dos coeficientes e `I` o índice da instância. Nos experimentos deste trabalho, foram utilizadas as instâncias com $R = 1000$ e $I = 1$ para cada combinação de tipo e tamanho, totalizando dezoito instâncias.

Foram selecionados três tipos de instâncias, que diferem pelo grau de correlação entre os pesos e os valores dos itens:

- **Tipo 1** (não correlacionado): pesos e valores dos itens são gerados de forma independente. Instâncias desse tipo tendem a ser mais tratáveis pelos algoritmos de *branch-and-cut*, pois a relaxação linear fornece limites de boa qualidade.
- **Tipo 2** (fracamente correlacionado): o valor de cada item é aproximadamente proporcional ao seu peso, com uma variação adicionada. Essa correlação parcial aumenta a dificuldade em relação ao Tipo 1, pois os itens tornam-se mais difíceis de discriminar pela relaxação linear.
- **Tipo 3** (fortemente correlacionado): o valor de cada item é definido como $v_i = w_i + k$, onde $k$ é uma constante fixa (no caso das instâncias utilizadas neste trabalho, k=R/10=100k = R/10 = 100 k=R/10=100). Essa estrutura é reconhecida na literatura como a classe mais difícil para algoritmos de *branch-and-bound*, pois a relaxação linear fornece limites muito fracos, exigindo exploração extensa da árvore de busca para provar otimalidade (PISINGER, 2005).

Cada tipo foi avaliado em seis dimensões crescentes, totalizando outras dezoito instâncias, com a mesma progressão de tamanhos do experimento da Dieta para permitir comparação direta entre as duas classes de problema. Em todos os casos, o valor ótimo já está registrado no próprio arquivo de instância, o que permitiu verificar, além do tempo de execução, se cada solver foi capaz de provar a otimalidade da solução encontrada dentro do limite de tempo estabelecido.

A inclusão do Tipo 3 é metodologicamente relevante porque permite observar o comportamento dos solvers sob condições de alta dificuldade combinatória – cenário em que o limite de tempo de 5 segundos passa a ter papel ativo no experimento, podendo impedir que alguns solvers concluam a prova de otimalidade em instâncias maiores.

### 3.4.3 Resultados e Discussão

#### 3.4.3.1 Problema da Dieta

As Tabelas 6, 7 e 8 apresentam os resultados obtidos para o Problema da Dieta nas instâncias de maior escala, organizadas por tipo estrutural. Em todos os casos e para todos os solvers, o status retornado foi *optimal* e o valor de custo mínimo foi idêntico entre os solvers para cada instância, confirmando a consistência das soluções obtidas. Nenhuma instância do Problema da Dieta atingiu o limite de tempo de 5 segundos, o que confirma a tratabilidade da Programação Linear mesmo em grande escala.

Tabela 6 – Resultados do Problema da Dieta – Tipo 1 (*esparso*, 10 nutrientes)

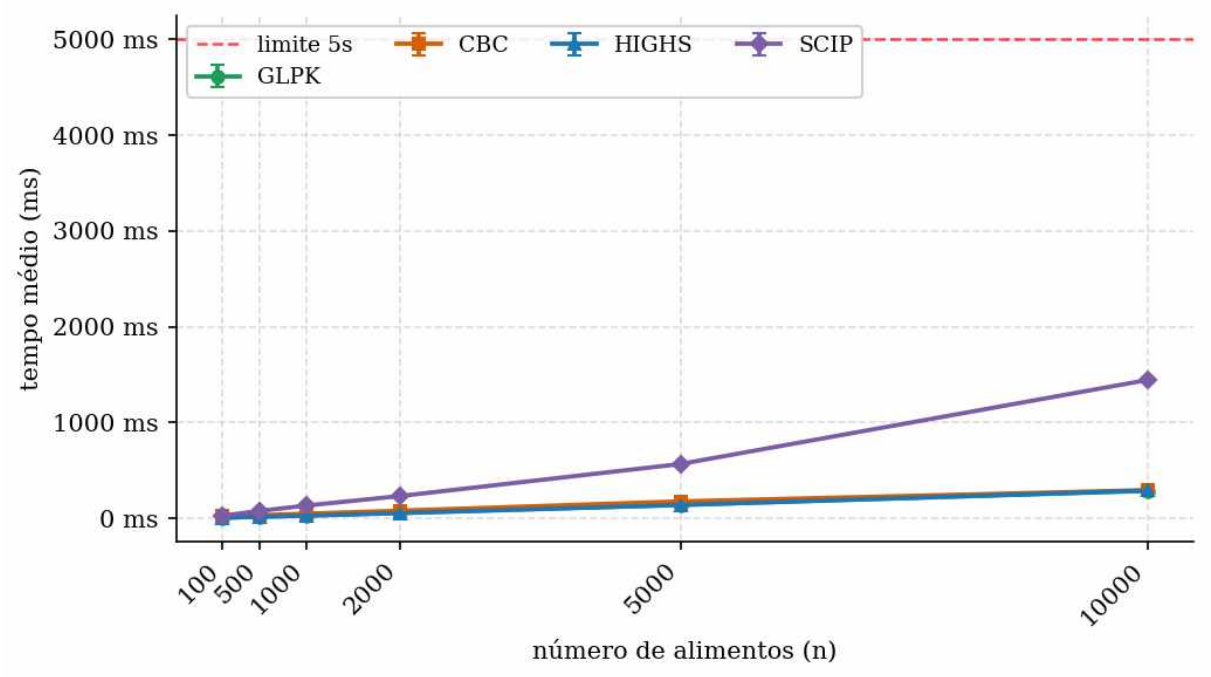
| **Instância** | **Solver** | **Custo ótimo (R$)** | **Tempo médio (ms)** | **Desvio (ms)** | **Exec. ótimas** |
|---|---|---|---|---|---|
| 100 × 10 | GLPK | 7,1882 | 6,617 | 0,392 | 10/10 |
| | HiGHS | | 6,166 | 0,302 | 10/10 |
| | SCIP | | 28,445 | 4,637 | 10/10 |
| | CBC | | 25,779 | 3,530 | 10/10 |
| 500 × 10 | GLPK | 26,0727 | 18,538 | 1,291 | 10/10 |
| | HiGHS | | 16,794 | 0,731 | 10/10 |
| | SCIP | | 78,301 | 5,377 | 10/10 |
| | CBC | | 33,475 | 3,870 | 10/10 |
| 1000 × 10 | GLPK | 52,5122 | 29,848 | 1,394 | 10/10 |
| | HiGHS | | 28,076 | 0,984 | 10/10 |
| | SCIP | | 133,942 | 15,160 | 10/10 |
| | CBC | | 49,528 | 6,780 | 10/10 |
| 2000 × 10 | GLPK | 104,6703 | 58,426 | 2,582 | 10/10 |
| | HiGHS | | 54,889 | 8,572 | 10/10 |
| | SCIP | | 233,260 | 10,426 | 10/10 |
| | CBC | | 80,416 | 14,214 | 10/10 |
| 5000 × 10 | GLPK | 207,7283 | 141,374 | 3,642 | 10/10 |
| | HiGHS | | 139,830 | 14,650 | 10/10 |
| | SCIP | | 567,654 | 13,275 | 10/10 |
| | CBC | | 177,533 | 16,378 | 10/10 |
| 10000 × 10 | GLPK | 417,8554 | 286,998 | 9,323 | 10/10 |
| | HiGHS | | 291,639 | 6,708 | 10/10 |
| | SCIP | | 1445,088 | 25,589 | 10/10 |
| | CBC | | 294,717 | 6,297 | 10/10 |

Fonte: elaboração própria.

As Figuras 1 e 2 apresentam, respectivamente, a escalabilidade e o desvio padrão dos tempos de execução para as instâncias do Tipo 1 (*esparso*, 10 nutrientes).

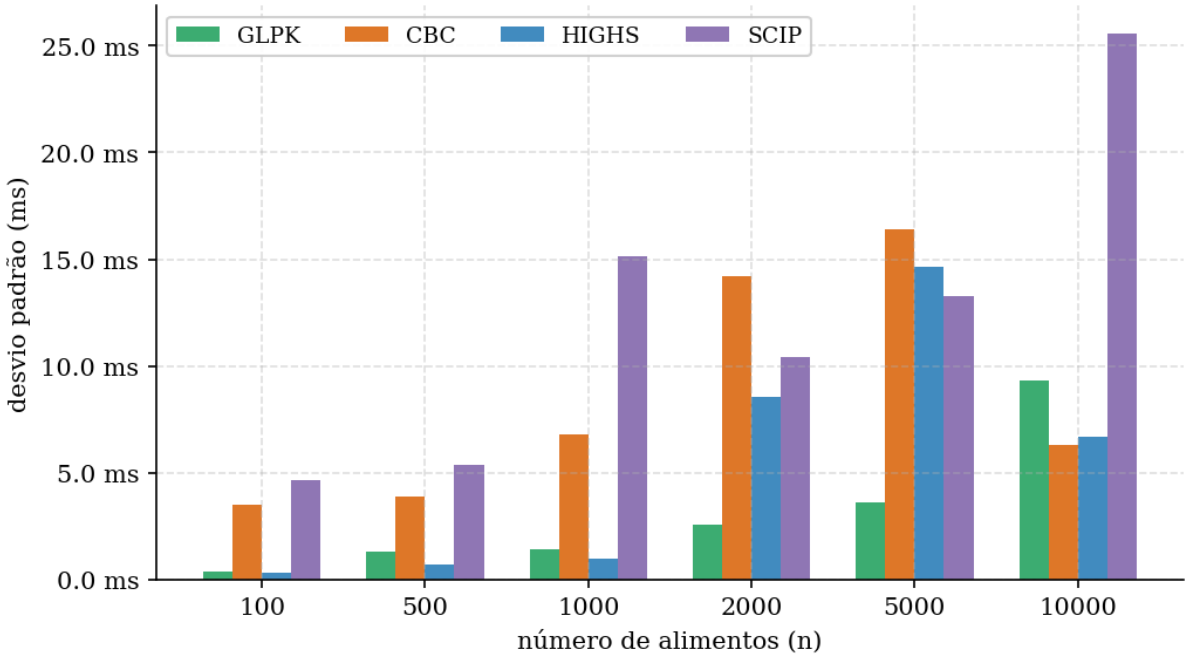
Observa-se que GLPK e HiGHS apresentaram desempenho equivalente ao longo de todas as dimensões, com os menores tempos de execução entre os solvers avaliados. Em $n = 10000$, ambos ficaram abaixo de 300 ms – o GLPK com 287 ms e o HiGHS com 292 ms. O CBC manteve desempenho intermediário, alcançando 295 ms em $n = 10000$, valor próximo aos dois líderes. O SCIP apresentou crescimento significativamente mais acentuado, atingindo aproximadamente 1445 ms em $n = 10000$, valor cerca de cinco vezes superior aos demais.

Figura 1 – Escalabilidade – Dieta (PL) – Tipo 1 (*esparso*, 10 nutrientes)




Fonte: elaboração própria.

Figura 2 – Desvio padrão – Dieta (PL) – Tipo 1 (*esparso*, 10 nutrientes)




Fonte: elaboração própria.

Em relação à estabilidade, o GLPK e o HiGHS apresentaram os menores desvios padrão em todas as instâncias, indicando comportamento consistente entre as repetições. O CBC também manteve desvios reduzidos. O SCIP foi o solver com maior variabilidade nas instâncias maiores, com desvio de aproximadamente 26 ms em $n = 10000$.

Tabela 7 – Resultados do Problema da Dieta – Tipo 2 (*médio*, 30 nutrientes)

| **Instância** | **Solver** | **Custo ótimo (R$)** | **Tempo médio (ms)** | **Desvio (ms)** | **Exec. ótimas** |
|---|---|---|---|---|---|
| 100 × 30 | GLPK | 8,6473 | 11,087 | 0,961 | 10/10 |
| | HiGHS | | 10,700 | 0,883 | 10/10 |
| | SCIP | | 45,234 | 8,822 | 10/10 |
| | CBC | | 29,307 | 2,391 | 10/10 |
| 500 × 30 | GLPK | 34,8419 | 39,349 | 1,992 | 10/10 |
| | HiGHS | | 38,688 | 1,592 | 10/10 |
| | SCIP | | 105,407 | 13,668 | 10/10 |
| | CBC | | 58,956 | 7,181 | 10/10 |
| 1000 × 30 | GLPK | 64,3235 | 73,509 | 3,861 | 10/10 |
| | HiGHS | | 70,220 | 2,338 | 10/10 |
| | SCIP | | 173,377 | 9,209 | 10/10 |
| | CBC | | 101,869 | 10,892 | 10/10 |
| 2000 × 30 | GLPK | 133,5259 | 143,979 | 6,969 | 10/10 |
| | HiGHS | | 139,416 | 12,002 | 10/10 |
| | SCIP | | 268,310 | 11,672 | 10/10 |
| | CBC | | 171,527 | 12,260 | 10/10 |
| 5000 × 30 | GLPK | 280,5808 | 383,352 | 13,329 | 10/10 |
| | HiGHS | | 346,765 | 16,951 | 10/10 |
| | SCIP | | 520,517 | 21,694 | 10/10 |
| | CBC | | 320,344 | 14,636 | 10/10 |
| 10000 × 30 | GLPK | 549,9233 | 929,759 | 40,846 | 10/10 |
| | HiGHS | | 744,945 | 89,389 | 10/10 |
| | SCIP | | 1032,071 | 89,133 | 10/10 |
| | CBC | | 581,223 | 11,600 | 10/10 |

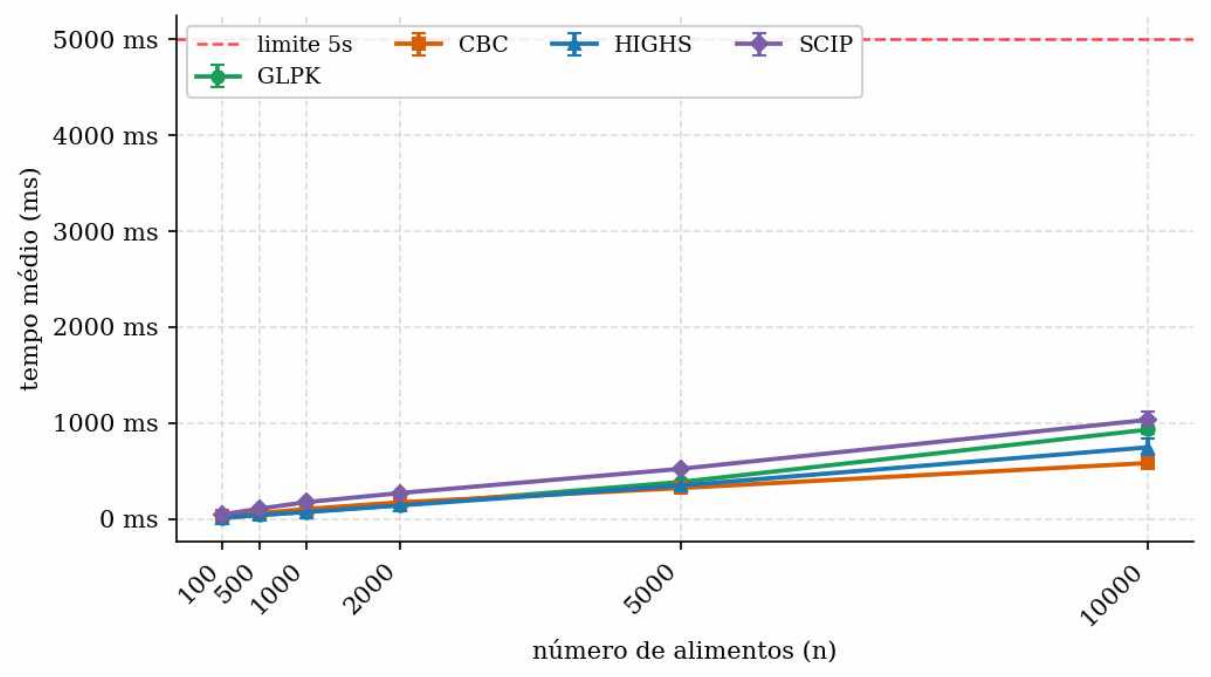
Fonte: elaboração própria.

As Figuras 3 e 4 apresentam os resultados para as instâncias do Tipo 2 (*médio*, 30 nutrientes).

Com o aumento da densidade de restrições, o comportamento dos solvers começa a se diferenciar. O HiGHS mantém-se competitivo com o GLPK até $n = 2000$, com tempos de 139 ms e 144 ms, respectivamente. A partir de $n = 5000$, porém, o CBC passa a apresentar os menores tempos de execução: 320 ms em $n = 5000$ e 581 ms em $n = 10000$. O HiGHS atingiu 745 ms, o GLPK 930 ms e o SCIP 1032 ms na instância de maior dimensão. Esse resultado indica que, com maior densidade de restrições, o CBC torna-se comparativamente mais eficiente na resolução do sistema linear.

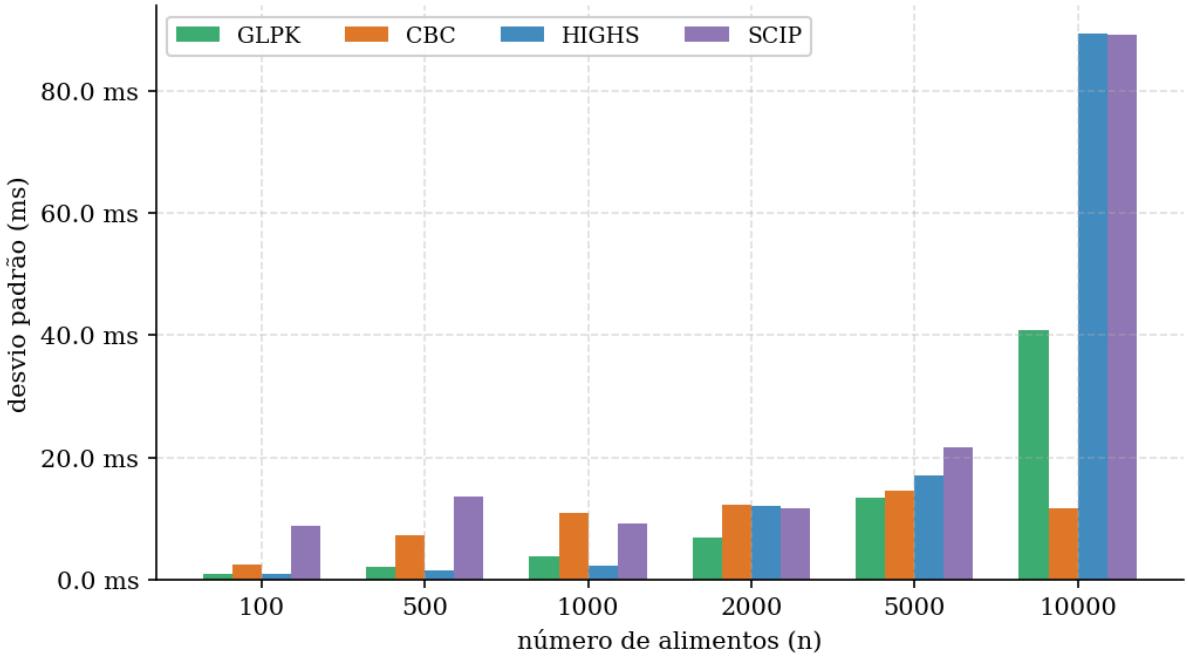
Em relação ao desvio padrão, o CBC apresentou a menor variabilidade em $n = 10000$, com desvio de apenas 12 ms. O HiGHS e o SCIP apresentaram os maiores desvios nessa instância, com valores de 89 ms cada, indicando maior instabilidade nas execuções de maior porte.

Figura 3 – Escalabilidade – Dieta (PL) – Tipo 2 (*médio*, 30 nutrientes)




Fonte: Elaboração própria.

Figura 4 – Desvio padrão – Dieta (PL) – Tipo 2 (*médio*, 30 nutrientes)



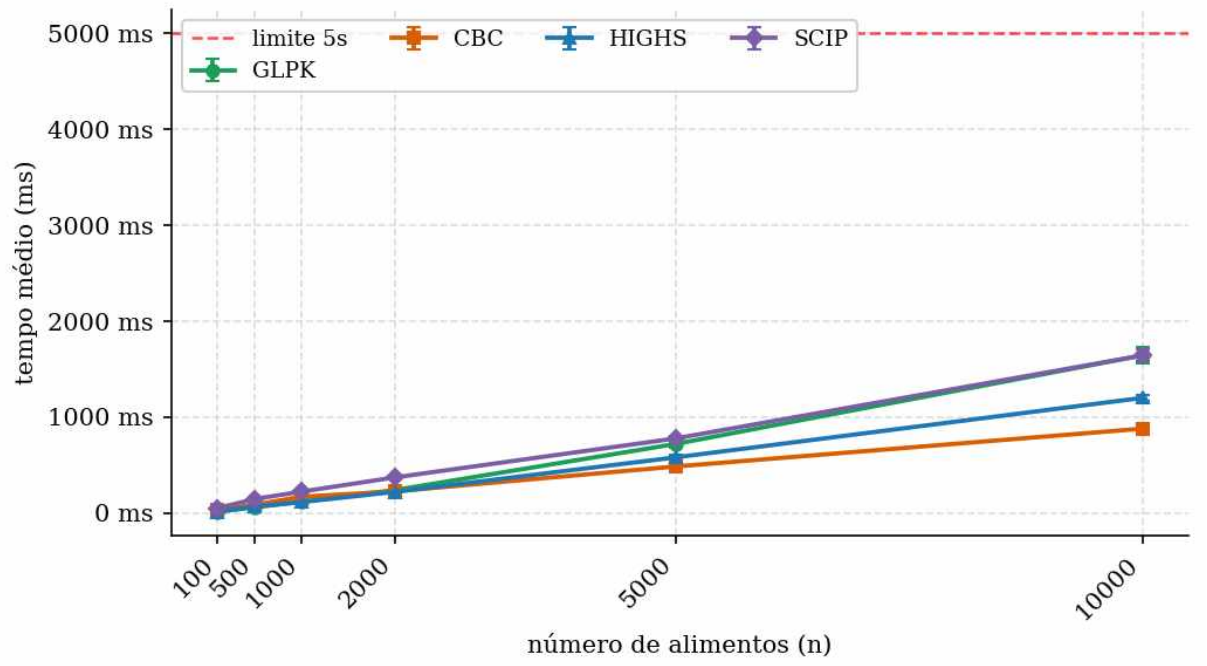

Fonte: Elaboração própria.

Tabela 8 – Resultados do Problema da Dieta – Tipo 3 (*denso*, 50 nutrientes)

| Instância | Solver | Custo ótimo (R$) | Tempo médio (ms) | Desvio (ms) | Exec. ótimas |
|---|---|---|---|---|---|
| 100 × 50 | GLPK | 9,1416 | 14,545 | 0,759 | 10/10 |
| | HiGHS | | 14,046 | 0,710 | 10/10 |
| | SCIP | | 51,171 | 7,715 | 10/10 |
| | CBC | | 32,075 | 7,004 | 10/10 |
| 500 × 50 | GLPK | 33,2519 | 60,348 | 2,485 | 10/10 |
| | HiGHS | | 68,871 | 23,678 | 10/10 |
| | SCIP | | 147,411 | 14,658 | 10/10 |
| | CBC | | 88,182 | 13,744 | 10/10 |
| 1000 × 50 | GLPK | 71,7483 | 118,485 | 4,689 | 10/10 |
| | HiGHS | | 115,091 | 3,016 | 10/10 |
| | SCIP | | 224,093 | 15,068 | 10/10 |
| | CBC | | 170,329 | 28,714 | 10/10 |
| 2000 × 50 | GLPK | 136,1075 | 240,464 | 12,784 | 10/10 |
| | HiGHS | | 221,346 | 10,155 | 10/10 |
| | SCIP | | 372,567 | 14,006 | 10/10 |
| | CBC | | 226,764 | 9,761 | 10/10 |
| 5000 × 50 | GLPK | 291,5805 | 720,206 | 44,643 | 10/10 |
| | HiGHS | | 580,833 | 22,931 | 10/10 |
| | SCIP | | 777,903 | 30,631 | 10/10 |
| | CBC | | 485,795 | 29,297 | 10/10 |
| 10000 × 50 | GLPK | 574,8198 | 1644,342 | 84,190 | 10/10 |
| | HiGHS | | 1201,577 | 33,419 | 10/10 |
| | SCIP | | 1644,333 | 62,509 | 10/10 |
| | CBC | | 880,626 | 18,199 | 10/10 |

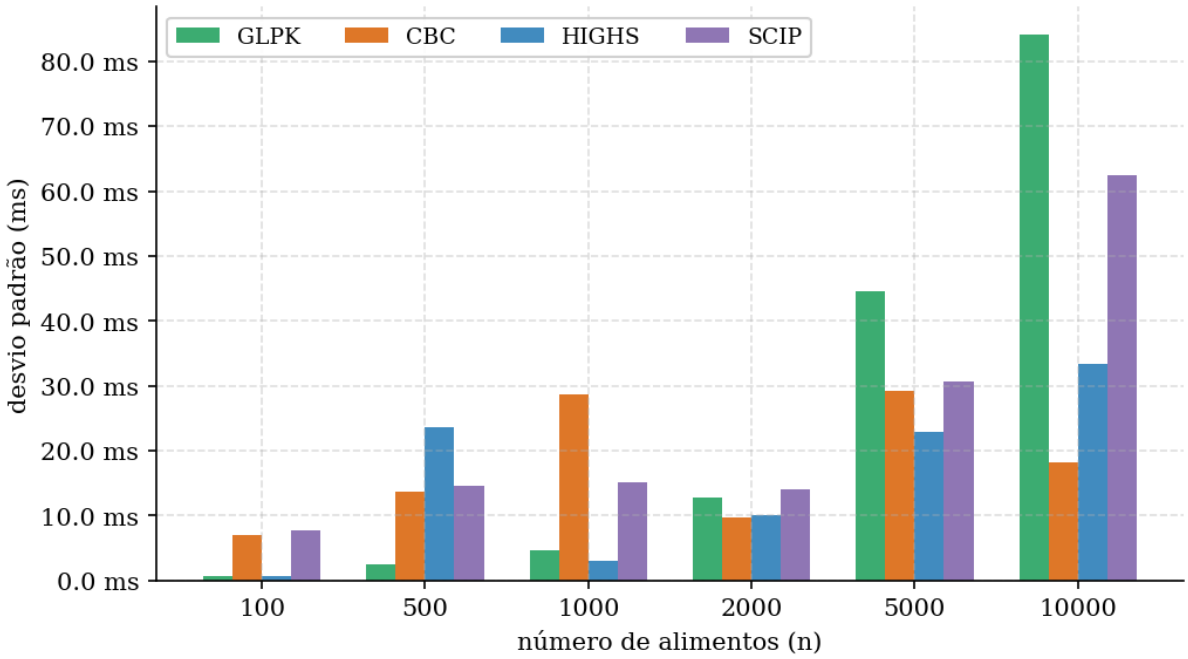
Fonte: Elaboração própria.

Figura 5 – Escalabilidade – Dieta (PL) – Tipo 3 (*denso*, 50 nutrientes)




Fonte: Elaboração própria.

Figura 6 – Desvio padrão – Dieta (PL) – Tipo 3 (*denso*, 50 nutrientes)




Fonte: Elaboração própria.

As Figuras 5 e 6 apresentam os resultados para as instâncias do Tipo 3 (*denso*, 50 nutrientes), que representa o cenário de maior densidade de restrições avaliado.

O CBC consolidou-se como o solver mais rápido nas instâncias de maior porte, com tempo de aproximadamente 881 ms em $n = 10000$. O HiGHS ocupou a segunda posição com 1202 ms. O GLPK e o SCIP apresentaram tempos equivalentes entre si – 1644 ms cada –, evidenciando que a vantagem do GLPK observada nas instâncias esparsas desaparece progressivamente com o aumento da densidade das restrições.

Em relação ao desvio padrão, o GLPK passou a apresentar a maior variabilidade em $n = 10000$, com desvio de 84 ms, enquanto o CBC manteve o menor desvio, de apenas 18 ms. Esse comportamento sugere que o GLPK torna-se menos estável à medida que a densidade do problema aumenta.

De forma geral, os resultados da Dieta indicam que o padrão de desempenho relativo entre os solvers não é fixo: varia conforme a densidade das restrições. Em instâncias esparsas, GLPK e HiGHS lideram. Com o aumento da densidade, o CBC torna-se progressivamente mais competitivo, assumindo a liderança nas instâncias mais densas.

#### 3.4.3.2 Problema da Mochila

Conforme descrito na Seção 3.4.2, as instâncias da Mochila foram avaliadas em três tipos de correlação – não correlacionadas (Tipo 1), fracamente correlacionadas (Tipo 2) e fortemente correlacionadas (Tipo 3) –, para evidenciar como a estrutura interna das

instâncias impacta o desempenho dos algoritmos de *branch-and-cut*. Os resultados são apresentados separadamente para cada tipo.

Tabela 9 – Resultados do Problema da Mochila – Instâncias Não Correlacionadas (Tipo 1)

| **Instância** | **Solver** | **Tempo médio (ms)** | **Desvio (ms)** | **Exec. ótimas** |
|---|---|---|---|---|
| knapPI_1_100 | GLPK | 5,040 | 0,200 | 10/10 |
| | HiGHS | 14,197 | 1,721 | 10/10 |
| | SCIP | 22,179 | 4,818 | 10/10 |
| | CBC | 35,220 | 4,100 | 10/10 |
| knapPI_1_500 | GLPK | 9,087 | 0,392 | 10/10 |
| | HiGHS | 42,352 | 1,160 | 10/10 |
| | SCIP | 45,084 | 9,294 | 10/10 |
| | CBC | 66,885 | 11,245 | 10/10 |
| knapPI_1_1000 | GLPK | 14,036 | 0,452 | 10/10 |
| | HiGHS | 105,126 | 9,109 | 10/10 |
| | SCIP | 82,457 | 11,432 | 10/10 |
| | CBC | 77,662 | 11,028 | 10/10 |
| knapPI_1_2000 | GLPK | 39,709 | 0,938 | 10/10 |
| | HiGHS | 252,837 | 11,046 | 10/10 |
| | SCIP | 162,262 | 5,646 | 10/10 |
| | CBC | 141,730 | 14,067 | 10/10 |
| knapPI_1_5000 | GLPK | 121,636 | 8,098 | 10/10 |
| | HiGHS | 961,038 | 14,203 | 10/10 |
| | SCIP | 555,384 | 48,882 | 10/10 |
| | CBC | 217,263 | 13,990 | 10/10 |
| knapPI_1_10000 | GLPK | 836,092 | 35,045 | 10/10 |
| | HiGHS | 3328,923 | 36,032 | 10/10 |
| | SCIP | 2434,873 | 47,711 | 10/10 |
| | CBC | 351,219 | 14,588 | 10/10 |

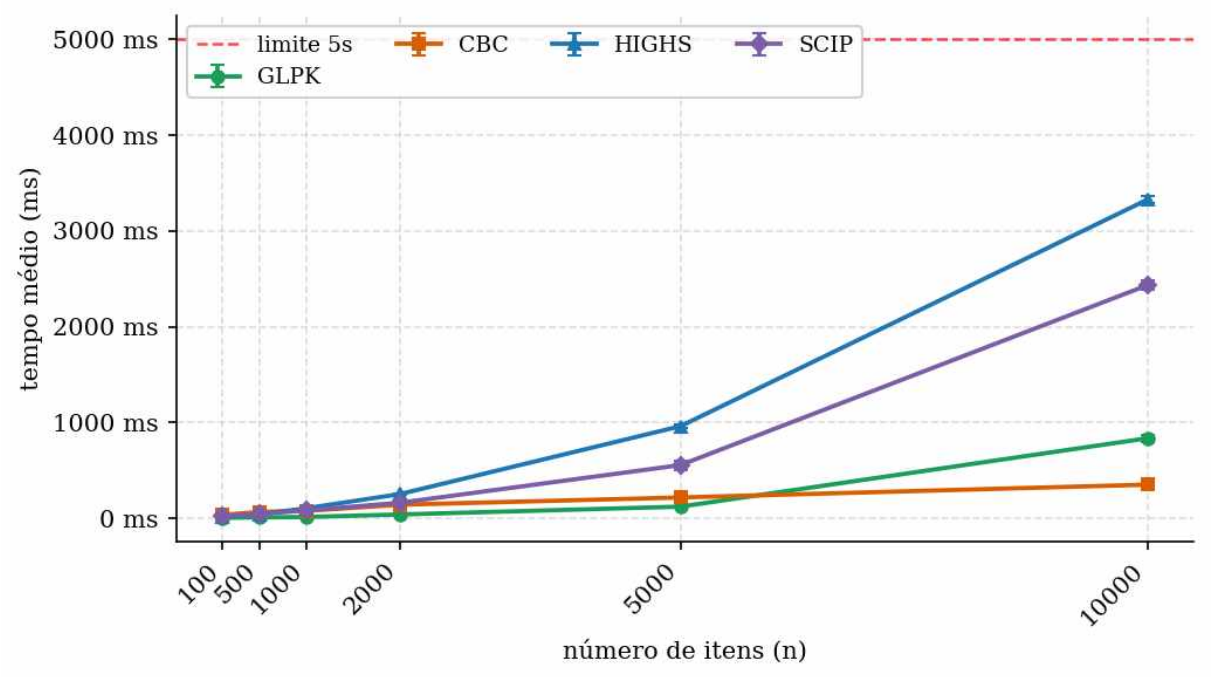
Fonte: Elaboração própria.

As Figuras 7 e 8 apresentam, respectivamente, a escalabilidade e o desvio padrão para as instâncias do Tipo 1 (não correlacionadas). Em todas as dimensões avaliadas, os quatro solvers encontraram a solução ótima em 100% das execuções.

O GLPK foi o solver mais rápido em cinco das seis dimensões avaliadas, com tempos que variaram de aproximadamente 5 ms em $n = 100$ a 122 ms em $n = 5000$. Em $n = 10000$, porém, ocorreu uma inversão no ranking: o tempo do GLPK saltou para 836 ms, enquanto o CBC manteve crescimento mais moderado e atingiu 351 ms, tornando-se o solver mais rápido nessa dimensão. O CBC, aliás, foi o solver com escalabilidade mais regular do conjunto, passando de 35 ms em $n = 100$a 351 ms em $n = 10000$ – um crescimento de aproximadamente dez vezes ao longo de toda a escala.

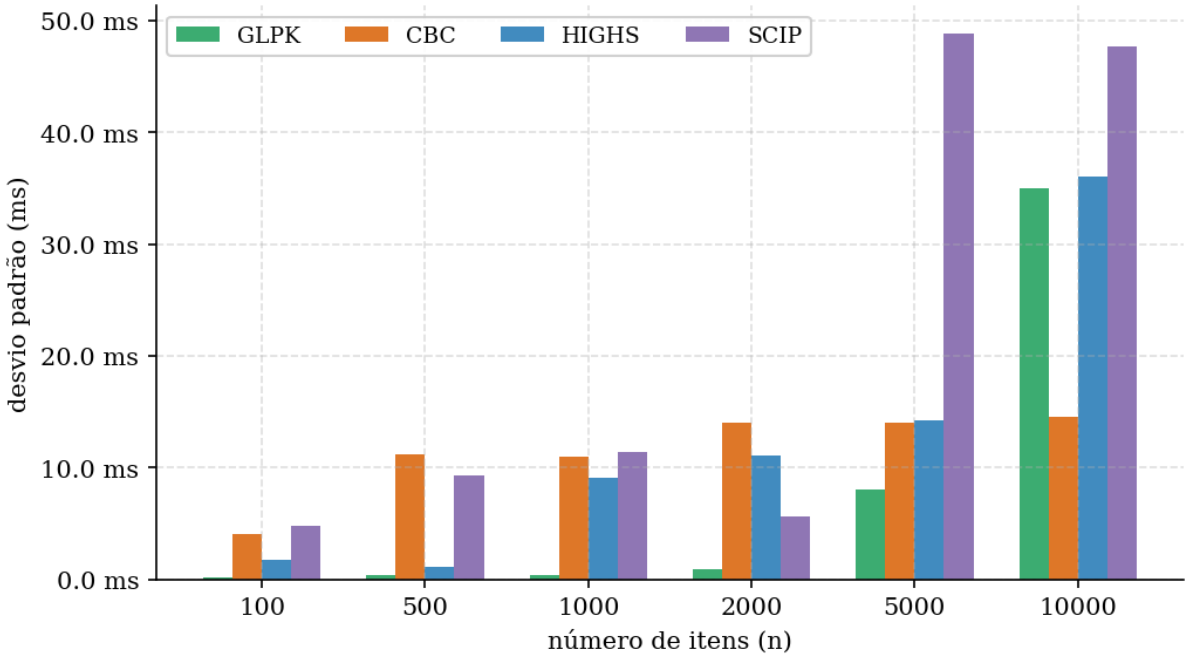
O HiGHS e o SCIP apresentaram crescimento substancialmente mais acentuado nas instâncias maiores. Em $n = 10000$, o HiGHS atingiu 3329 ms e o SCIP 2435 ms –

Figura 7 – Escalabilidade – Mochila (PI) – Tipo 1 (não correlacionado)




Fonte: Elaboração própria.

Figura 8 – Desvio padrão – Mochila (PI) – Tipo 1 (não correlacionado)




Fonte: Elaboração própria.

valores ainda dentro do limite de 5 segundos, porém há indicativos de maior sensibilidade desses solvers ao aumento da dimensão em instâncias não correlacionadas.

Em relação ao desvio padrão, o GLPK foi o solver mais estável até $n = 5000$, com desvios consistentemente abaixo de 9 ms. Em $n = 10000$, porém, sua variabilidade aumentou para aproximadamente 35 ms, em linha com a degradação observada no tempo médio. O CBC apresentou o comportamento mais regular ao longo de toda a escala, com desvios entre 4 ms e 15 ms. O HiGHS e o SCIP registraram os maiores desvios nas instâncias de maior porte, com 36 ms e 48 ms em $n = 10000$, respectivamente.

Tabela 10 – Resultados do Problema da Mochila – Instâncias Fracamente Correlacionadas (Tipo 2)

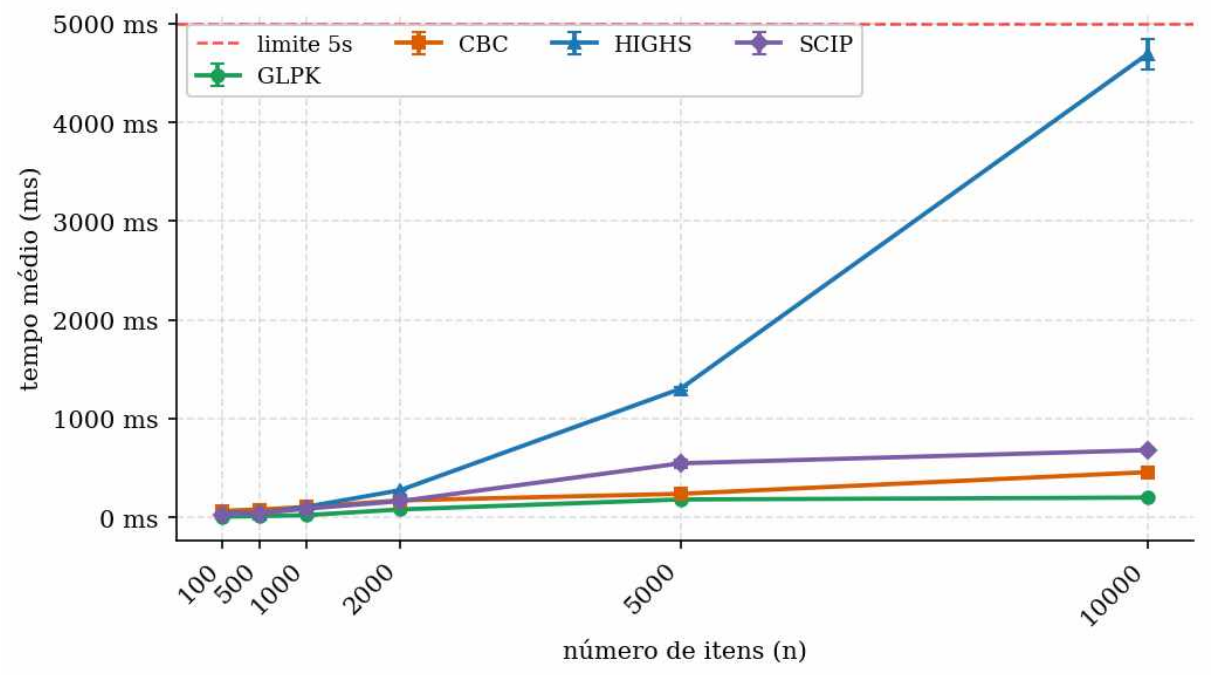
| **Instância** | **Solver** | **Tempo médio (ms)** | **Desvio (ms)** | **Exec. ótimas** |
|---|---|---|---|---|
| knapPI_2_100 | GLPK | 5,391 | 0,475 | 10/10 |
| | HiGHS | 48,868 | 2,059 | 10/10 |
| | SCIP | 29,837 | 7,734 | 10/10 |
| | CBC | 65,836 | 16,223 | 10/10 |
| knapPI_2_500 | GLPK | 12,160 | 0,527 | 10/10 |
| | HiGHS | 40,355 | 0,999 | 10/10 |
| | SCIP | 41,337 | 12,082 | 10/10 |
| | CBC | 78,750 | 3,892 | 10/10 |
| knapPI_2_1000 | GLPK | 21,374 | 0,781 | 10/10 |
| | HiGHS | 106,257 | 2,140 | 10/10 |
| | SCIP | 86,892 | 9,833 | 10/10 |
| | CBC | 104,553 | 11,098 | 10/10 |
| knapPI_2_2000 | GLPK | 79,439 | 1,656 | 10/10 |
| | HiGHS | 274,049 | 4,074 | 10/10 |
| | SCIP | 163,156 | 13,734 | 10/10 |
| | CBC | 170,934 | 6,196 | 10/10 |
| knapPI_2_5000 | GLPK | 180,533 | 2,798 | 10/10 |
| | HiGHS | 1301,034 | 12,621 | 10/10 |
| | SCIP | 546,697 | 42,395 | 10/10 |
| | CBC | 237,685 | 15,729 | 10/10 |
| knapPI_2_10000 | GLPK | 199,769 | 11,017 | 10/10 |
| | HiGHS | 4693,709 | 157,149 | 8/10 |
| | SCIP | 680,211 | 18,824 | 10/10 |
| | CBC | 457,232 | 18,383 | 10/10 |

Fonte: Elaboração própria.

As Figuras 9 e 10 apresentam os resultados para as instâncias do Tipo 2 (fracamente correlacionadas).

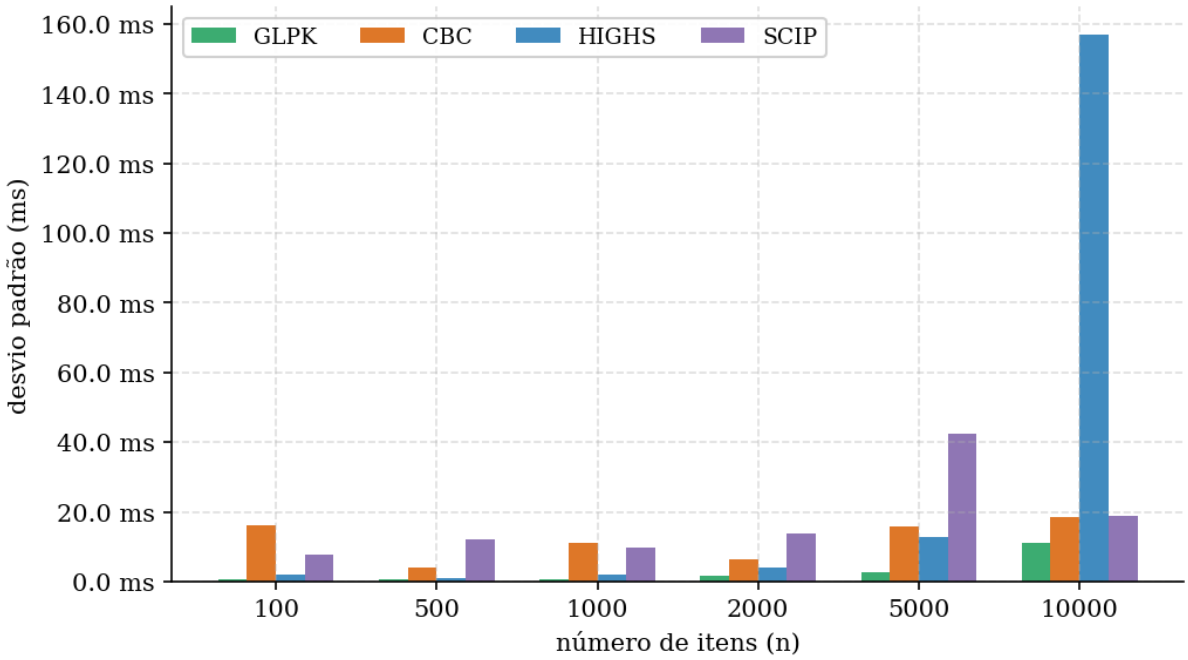
Na instância do tipo 2 o GLPK foi o solver mais rápido em todas as dimensões avaliadas, com tempos que variaram de aproximadamente 5 ms em $n = 100$ a 200 ms em $n = 10000$. Diferentemente do Tipo 1, em que houve inversão de ranking na maior

Figura 9 – Escalabilidade – Mochila (PI) – Tipo 2 (fracamente correlacionado)




Fonte: Elaboração própria.

Figura 10 – Desvio padrão – Mochila (PI) – Tipo 2 (fracamente correlacionado)




Fonte: Elaboração própria.

dimensão, o GLPK manteve a liderança em toda a escala, com vantagem expressiva sobre os demais solvers a partir de $n = 5000$, nesse ponto, seu tempo é cerca de 30% do tempo do segundo colocado.

A segunda posição não foi ocupada pelo mesmo solver ao longo da escala. Nas dimensões menores ($n = 100$ a $n = 2000$), o SCIP ficou em segundo lugar na maioria das instâncias, com tempos que variaram de 30 ms a 163 ms. A partir de $n = 5000$, porém, o CBC passou a ocupar essa posição, com 238 ms em $n = 5000$ e 457 ms em $n = 10000$, contra 547 ms e 680 ms do SCIP. Esse comportamento sugere que o CBC escala melhor que o SCIP nesse tipo de instância, embora seja mais lento nas dimensões menores. O HiGHS apresentou comportamento distinto nesse tipo de instância. Seus tempos cresceram de forma muito mais acentuada com o aumento da dimensão, atingindo aproximadamente 4694 ms em $n = 10000$ – muito próximo ao limite de 5 segundos, conforme ilustrado na Figura 9. Em 2 das 10 execuções nessa instância, o limite de tempo foi atingido, resultando em 8/10 execuções com status optimal. Ainda assim, o valor encontrado pelo HiGHS apresentou gap inferior a 0,01% em relação ao ótimo conhecido nas execuções concluídas dentro do limite de tempo, indicando que o solver encontra soluções de alta qualidade mas apresenta forte sensibilidade à estrutura das instâncias fracamente correlacionadas em grande escala.

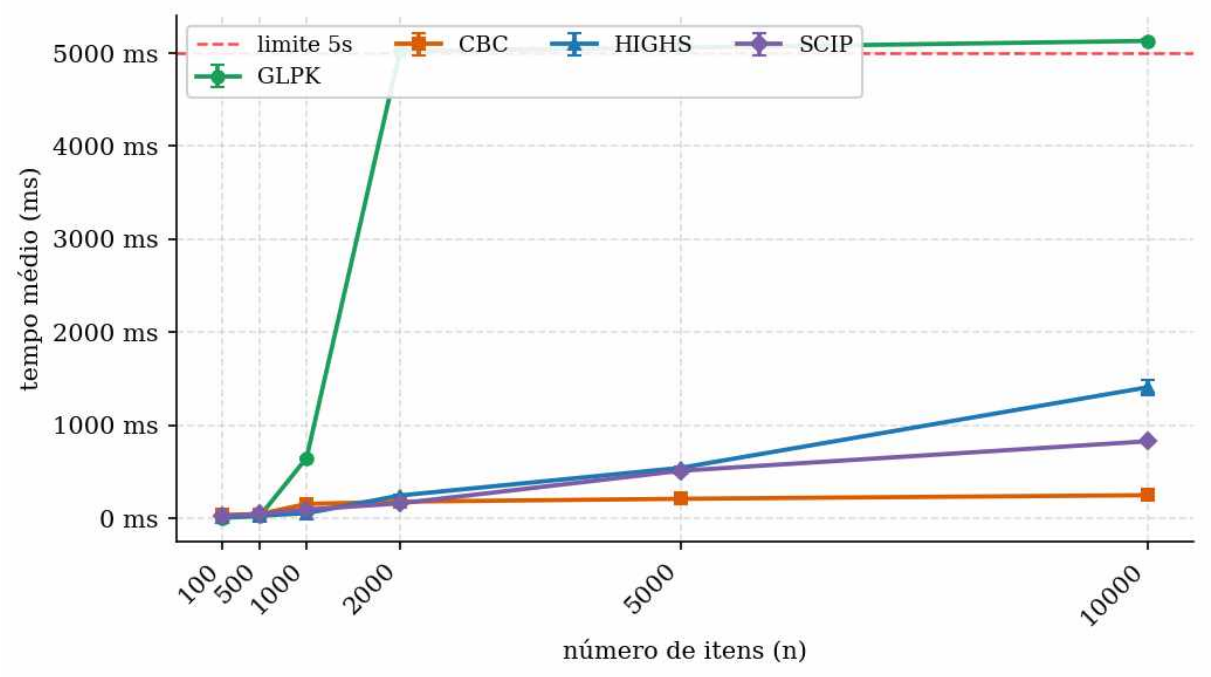
Em relação ao desvio padrão, o GLPK manteve a maior estabilidade ao longo de toda a escala, com desvios abaixo de 12 ms em todas as dimensões. O CBC e o SCIP apresentaram desvios moderados, da ordem de 10 a 20 ms na maioria das instâncias. O HiGHS destacou-se negativamente em $n = 10000$, com desvio de aproximadamente 157 ms – o maior registrado em todo o experimento da Mochila.

As Figuras 11, 12 e 13 apresentam os resultados para as instâncias do Tipo 3 (fortemente correlacionadas), que correspondem à classe de maior dificuldade combinatória do benchmark de Pisinger (2005).

Esses resultados revelam o achado mais significativo deste trabalho: a degradação progressiva do GLPK conforme a dimensão aumenta, culminando em timeout sistemático nas instâncias maiores. Nas dimensões menores ($n = 100$ e $n = 500$), o GLPK manteve a liderança, com tempos de aproximadamente 5 ms e 23 ms, respectivamente. Em $n = 1000$, o GLPK ainda prova otimalidade, mas seu tempo salta para 648 ms – valor já significativamente superior aos demais solvers nessa instância: HiGHS com 53 ms, SCIP com 94 ms e CBC com 154 ms. Nesse ponto, o GLPK passa de mais rápido a mais lento do grupo em uma única dimensão.

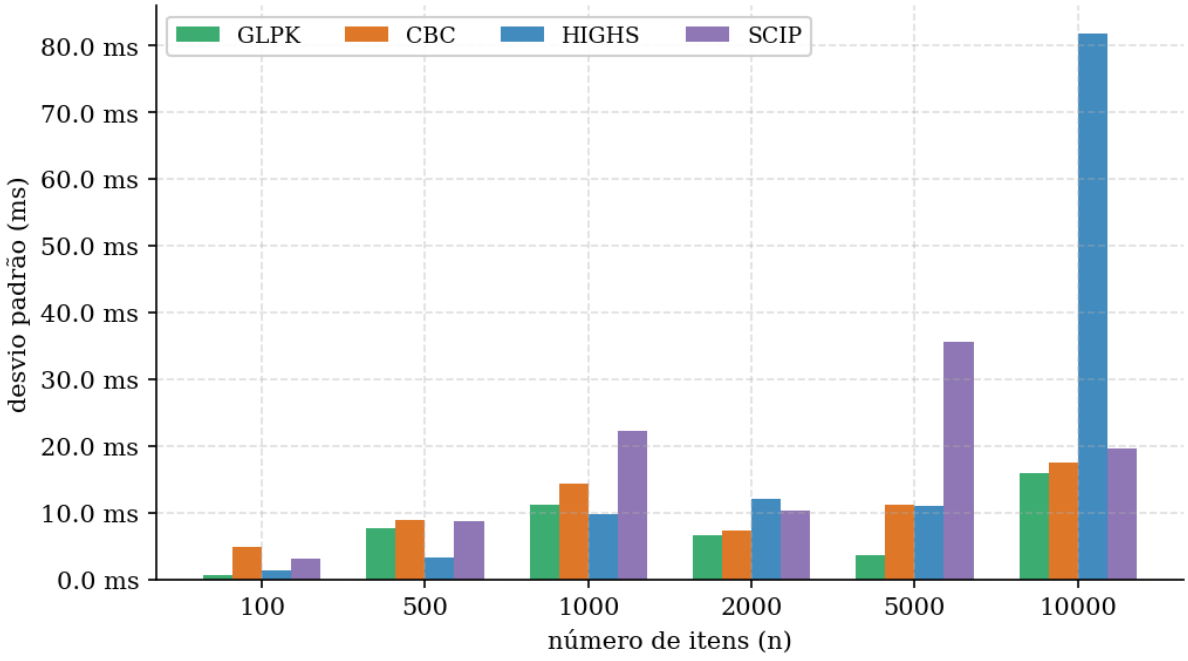
A partir de $n = 2000$, conforme ilustrado na Figura 13, o GLPK passa a atingir o limite de 5 segundos em todas as execuções, retornando status feasible com 0/10 execuções ótimas em $n = 2000$, $n = 5000$ e $n = 10000$. Apesar disso, o valor encontrado pelo GLPK coincidiu com o ótimo conhecido em todas as execuções (gap = 0%). Esse resultado indica

Figura 11 – Escalabilidade – Mochila (PI) – Tipo 3 (fortemente correlacionado)




Fonte: Elaboração própria.

Figura 12 – Desvio padrão – Mochila (PI) – Tipo 3 (fortemente correlacionado)




Fonte: Elaboração própria.

Tabela 11 – Resultados do Problema da Mochila – Instâncias Fortemente Correlacionadas (Tipo 3)

| **Instância** | **Solver** | **Tempo médio (ms)** | **Desvio (ms)** | **Exec. ótimas** |
|---|---|---|---|---|
| knapPI_3_100 | GLPK | 4,895 | 0,740 | 10/10 |
| | HiGHS | 13,271 | 1,360 | 10/10 |
| | SCIP | 20,873 | 3,136 | 10/10 |
| | CBC | 35,441 | 4,971 | 10/10 |
| knapPI_3_500 | GLPK | 22,597 | 7,772 | 10/10 |
| | HiGHS | 27,878 | 3,291 | 10/10 |
| | SCIP | 45,430 | 8,775 | 10/10 |
| | CBC | 44,572 | 8,958 | 10/10 |
| knapPI_3_1000 | GLPK | 647,574 | 11,219 | 10/10 |
| | HiGHS | 53,453 | 9,745 | 10/10 |
| | SCIP | 93,849 | 22,309 | 10/10 |
| | CBC | 153,656 | 14,311 | 10/10 |
| knapPI_3_2000 | GLPK | 5022,816 | 6,675 | 0/10 |
| | HiGHS | 244,429 | 12,105 | 10/10 |
| | SCIP | 159,637 | 10,267 | 10/10 |
| | CBC | 177,942 | 7,392 | 10/10 |
| knapPI_3_5000 | GLPK | 5059,255 | 3,727 | 0/10 |
| | HiGHS | 541,376 | 11,066 | 10/10 |
| | SCIP | 509,708 | 35,569 | 10/10 |
| | CBC | 208,855 | 11,231 | 10/10 |
| knapPI_3_10000 | GLPK | 5131,636 | 16,030 | 0/10 |
| | HiGHS | 1407,858 | 81,887 | 10/10 |
| | SCIP | 827,250 | 19,627 | 10/10 |
| | CBC | 247,533 | 17,603 | 10/10 |

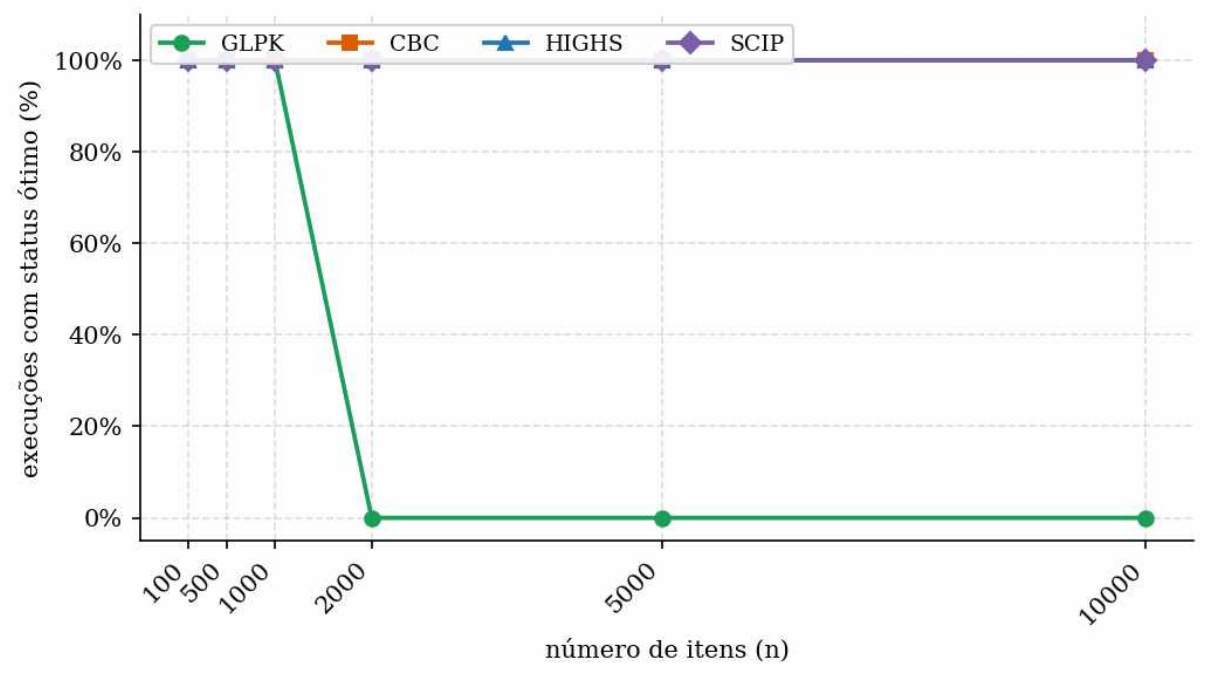
Fonte: Elaboração própria.

que o solver localiza a solução ótima durante a exploração da árvore de busca, mas não consegue concluir a prova de otimalidade – ou seja, não consegue fechar o gap entre o limite inferior e o limite superior – dentro do tempo disponível.

Os demais solvers provaram otimalidade em 100% das execuções em todas as dimensões. O CBC destacou-se como o mais eficiente nesse tipo de instância, com tempos que variaram de 35 ms em $n = 100$ a 248 ms em $n = 10000$, mantendo crescimento moderado e desvios reduzidos ao longo de toda a escala. O SCIP foi o segundo mais rápido nas instâncias maiores, com 827 ms em $n = 10000$. O HiGHS, embora também tenha provado otimalidade em todas as instâncias, apresentou crescimento mais acentuado, com 1408 ms em $n = 10000$.

Em relação ao desvio padrão, o CBC manteve comportamento estável em toda a escala, com desvios entre 5 ms e 18 ms. O SCIP apresentou maior variabilidade nas instâncias intermediárias, com 36 ms em $n = 5000$. O HiGHS registrou o maior desvio do Tipo 3 em $n = 10000$, com aproximadamente 82 ms. Os desvios do GLPK aparecem

Figura 13 – Taxa de execuções com status *optimal* – Mochila (PI) – Tipo 3 (fortemente correlacionado)




Fonte: Elaboração própria.

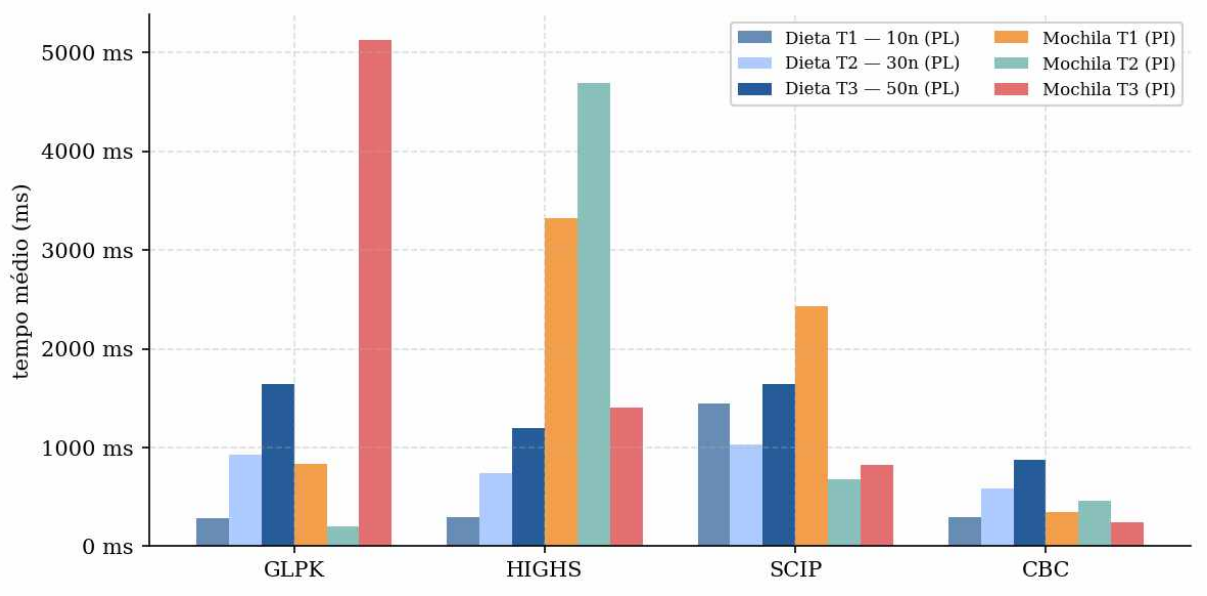
reduzidos a partir de $n = 2000$ (entre 4 ms e 16 ms), porém esse comportamento não reflete estabilidade real do solver: como o GLPK atinge o limite de 5 segundos de forma sistemática nessas instâncias, o tempo medido é determinado pelo timeout e não pela conclusão da busca.

Esse resultado evidencia que o grau de correlação entre pesos e valores dos itens tem impacto determinante no comportamento dos solvers de Programação Inteira. o GLPK, que se mostrou competitivo nos Tipos 1 e 2, torna-se o único solver a não provar otimalidade no Tipo 3, enquanto o CBC mesmo mais lento nas dimensões menores, revela-se o mais robusto diante da maior dificuldade combinatória, mantendo crescimento moderado em toda a escala.

A Figura 14 apresenta uma comparação do tempo médio de execução de todos os solvers para instâncias de $n = 10000$, cobrindo os três tipos da Dieta (PL) e os três tipos da Mochila (PI), permitindo uma visão consolidada do comportamento de cada solver sob diferentes classes e estruturas de problema.

No Problema da Dieta, observa-se que os tempos crescem progressivamente do Tipo 1 para o Tipo 3 conforme a densidade das restrições aumenta, porém todos os solvers concluem as execuções bem abaixo do limite de 5 segundos. No Tipo 1 (*esparso*), GLPK, HiGHS e CBC apresentam tempos equivalentes – entre 287 ms e 295 ms –, enquanto o SCIP se destaca negativamente com 1445 ms. No Tipo 3 (*denso*), o CBC passa a liderar

Figura 14 – Tempo médio a $n = 10000$ – Dieta (PL) vs. Mochila (PI) – todos os tipos




Fonte: Elaboração própria.

com 881 ms, seguido do HiGHS com 1202 ms, enquanto GLPK e SCIP convergem para 1644 ms cada.

No Problema da Mochila, o contraste entre os tipos de instância é substancialmente mais acentuado. No Tipo 1 (não correlacionado), o CBC se destaca com 351 ms, o GLPK ocupa a segunda posição com 836 ms, e o HiGHS e o SCIP apresentam tempos elevados de 3329 ms e 2435 ms, respectivamente. No Tipo 2 (fracamente correlacionado), o GLPK lidera com aproximadamente 200 ms e o CBC mantém 457 ms, enquanto o HiGHS atinge 4694 ms – muito próximo ao limite de 5 segundos, com 8/10 execuções concluindo dentro do tempo.

No Tipo 3 (fortemente correlacionado), a inversão de ranking do GLPK torna-se evidente na figura: sua barra ultrapassa o limite de 5000 ms, correspondendo ao timeout sistemático observado em todas as execuções (0/10 com status *optimal*). Os demais solvers provam otimalidade em 100% das execuções, com o CBC apresentando o menor tempo com 248 ms, seguido do SCIP com 827 ms e do HiGHS com 1408 ms.

A Tabela 12 consolida o ranking final de desempenho dos solvers em cada classe e estrutura de problema. Os valores entre parênteses indicam o tempo médio de execução (em ms) na maior instância avaliada ($n = 10000$), e a posição no ranking considera esse tempo em conjunto com a taxa de execuções com prova de otimalidade.

De forma geral, o CBC destacou-se pela maior robustez ao longo do conjunto de experimentos, mantendo tempos competitivos em todos os cenários avaliados – em particular, foi o mais rápido nas instâncias mais densas da Dieta e nos três tipos da

Tabela 12 – Ranking final de desempenho dos solvers por classe e estrutura de problema

| **Classe** | **Estrutura** | **1º** | **2º** | **3º** | **4º** |
|---|---|---|---|---|---|
| PL (Dieta) | Esparso (Tipo 1) | GLPK (287) | HiGHS (292) | CBC (295) | SCIP (1445) |
| | Médio (Tipo 2) | CBC (581) | HiGHS (745) | GLPK (930) | SCIP (1032) |
| | Denso (Tipo 3) | CBC (881) | HiGHS (1202) | GLPK (1644) | SCIP (1644) |
| PI (Mochila) | Não corr. (Tipo 1) | CBC (351) | GLPK (836) | SCIP (2435) | HiGHS (3329)[a] |
| | Fraca corr. (Tipo 2) | GLPK (200) | CBC (457) | SCIP (680) | HiGHS (4694)[b] |
| | Forte corr. (Tipo 3) | CBC (248) | SCIP (827) | HiGHS (1408) | GLPK (5132)[c] |

Fonte: elaboração própria.

Mochila em $n = 10000$. O GLPK apresentou desempenho consistente em instâncias de Programação Linear e foi o mais rápido em quase todas as dimensões dos Tipos 1 e 2 da Mochila, mas revelou limitações significativas em escala alta: perdeu a liderança para o CBC em $n = 10000$ do Tipo 1 e não conseguiu provar otimalidade no Tipo 3 a partir de $n = 2000$. O HiGHS mostrou-se eficiente nas instâncias menores e médias de Programação Linear, mas apresentou crescimento acentuado em Programação Inteira, ficando entre os solvers mais lentos da Mochila em todos os três tipos. O SCIP manteve comportamento consistente na Mochila, mas apresentou maior custo computacional nos problemas de Programação Linear. Esses resultados reforçam a importância de avaliar solvers em múltiplas estruturas de problema, e não apenas em diferentes escalas: como evidenciado na Figura 14, dois problemas resolvidos sob a mesma dimensão ($n = 10000$) podem produzir rankings de desempenho completamente distintos conforme a classe e a estrutura interna da instância.

# 4 Conclusão

Este trabalho teve como objetivo comparar o desempenho de quatro solvers open-source – GLPK, CBC, HiGHS e SCIP – quando integrados à biblioteca de modelagem Pyomo em Python, na resolução de problemas de Programação Linear e Inteira. A avaliação foi conduzida em duas etapas experimentais: uma fase inicial com problemas clássicos de pequena escala, destinada à validação dos modelos e à verificação da consistência das soluções, e uma fase de maior porte, com trinta e seis instâncias organizadas simetricamente entre os dois problemas – três tipos estruturais por problema, em seis dimensões crescentes que vão de 100 a 10000 variáveis.

Os experimentos permitem responder à hipótese que orientou a pesquisa. A hipótese inicial previa que os solvers de desenvolvimento mais recente, como HiGHS e SCIP, apresentariam desempenho superior aos solvers mais tradicionais, especialmente em instâncias de maior porte. Os resultados não confirmaram essa expectativa de forma geral. No Problema da Dieta, formulado como Programação Linear, GLPK e HiGHS apresentaram desempenho equivalente e consistentemente superior nas instâncias esparsas, enquanto o CBC se destacou nas instâncias mais densas. No Problema da Mochila, formulado como Programação Inteira, o GLPK liderou na maioria das instâncias dos Tipos 1 e 2, mas perdeu a liderança para o CBC em $n = 10000$ do Tipo 1 e não foi capaz de provar otimalidade em nenhuma execução do Tipo 3 a partir de $n = 2000$, atingindo o limite de tempo em todas as repetições.

O achado mais relevante é o comportamento diferenciado dos solvers conforme a classe e a estrutura do problema. Na Programação Linear, o ranking variou conforme a densidade da matriz de restrições, mas todos os solvers concluíram a execução dentro do tempo disponível em todas as instâncias. Na Programação Inteira, o desempenho relativo variou de forma mais acentuada conforme o grau de correlação entre pesos e valores: o GLPK foi o mais rápido em instâncias não correlacionadas e fracamente correlacionadas até 5000 itens, mas tornou-se o único solver incapaz de provar otimalidade nas instâncias fortemente correlacionadas em escala alta – caso em que o CBC se mostrou o mais robusto. Esse contraste indica que a escolha do solver deve ser orientada pela classe e pela estrutura do problema, e não apenas pela data de lançamento ou pela reputação do software. O SCIP, em particular, apresentou overhead superior nos problemas puramente lineares, comportamento provavelmente relacionado ao fato de sua arquitetura ser orientada à Programação Inteira.

Como contribuição, este trabalho apresenta uma análise comparativa conduzida em ambiente controlado, com critérios padronizados e problemas clássicos de otimização,

oferecendo uma base de referência para a escolha de solvers open-source integrados via Pyomo.

O trabalho possui limitações que devem ser consideradas na interpretação dos resultados. As instâncias avaliadas chegam a 10000 variáveis, dimensão suficiente para evidenciar diferenças de escalabilidade entre os solvers, mas ainda inferior à de problemas industriais reais, nos quais o comportamento relativo dos solvers pode se alterar. Além disso, os tempos de execução incluem o overhead da interface Pyomo, que representa uma fração não desprezível do tempo total nas instâncias menores. Por fim, os experimentos foram executados em um único ambiente computacional, e variações de hardware podem influenciar os resultados absolutos, embora as tendências relativas tendam a se manter.

Como trabalhos futuros, sugere-se a ampliação do conjunto de instâncias para dimensões superiores a 10000 variáveis, a inclusão de outros solvers open-source ou até mesmo comerciais, e a investigação do impacto de diferentes configurações e parâmetros internos no desempenho observado. Especificamente em relação ao Problema da Mochila, sugere-se a avaliação de classes adicionais do benchmark de Pisinger, como as instâncias do tipo spanner, nas quais os itens são gerados como múltiplos de um pequeno conjunto-base de itens-chave. Essa estrutura tende a tornar o comportamento dos algoritmos de branch-and-cut ainda mais sensível à organização interna do problema. Por fim, pode-se ampliar o tempo limite de execução para investigar se o GLPK seria capaz de provar otimalidade nas instâncias fortemente correlacionadas em escala alta dado tempo suficiente, ou se a degradação observada no Tipo 3 é de natureza estrutural.

# Referências

ALVES, F.; PETIZ, M.; RIBEIRO, R.; ABBASI, A.; CARVALHO, P. N.; RODRIGUES, R. An n-queens benchmark using MILP solvers: Comparison between open-source optimization tools. In: PEREIRA, A. I. et al. (Ed.). **Optimization, Learning Algorithms and Applications (OL2A 2024)**. Cham: Springer, 2024. (Communications in Computer and Information Science, v. 2281), p. 97–108. Citado na página 19.

ALVES, R.; DELGADO, C. **Programação Linear Inteira**. Faculdade de Economia, Universidade do Porto, 1997. Disponível em: <https://repositorio-aberto.up.pt/handle/10216/74369>. Acesso em: 18 fev. 2026. Citado 3 vezes nas páginas 13, 15 e 16.

ANDRÉASSON, N.; EVGRAFOV, A.; PATRIKSSON, M. **An Introduction to Continuous Optimization**: Foundations and fundamental algorithms. Garden City, NY: Dover Publications, 2020. 390 p. (Dover Books on Mathematics). ISBN 9780486802879. Citado na página 10.

AVELLA, P.; CALAMITA, A.; PALAGI, L. **A computational study of off-the-shelf MINLP solvers on a benchmark set of congested capacitated facility location problems**. 2023. Disponível em: <https://arxiv.org/abs/2303.04216>. Acesso em: 20 fev. 2026. Citado na página 10.

BELTRAMIN, A. **Modern branch-and-cut solvers for Mixed-Integer Linear Programming**: A computational comparison. 133 p. Tese (Doutorado) — Università degli studi di Padova, Pádua, 2015. Citado na página 18.

BYNUM, M. L.; HACKEBEIL, G. A.; HART, W. E.; LAIRD, C. D.; NICHOLSON, B. L.; SIIROLA, J. D.; WATSON, J.-P.; WOODRUFF, D. L. **Pyomo**: Optimization modeling in Python. 3. ed. Cham: Springer, 2021. v. 67. 225 p. (Springer Optimization and Its Applications, v. 67). Citado na página 20.

CELIS, A. M. V. **Algoritmo de Ponto Interior para Programação Linear Baseado no FDIPA**. 39 p. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2018. Disponível em: <https://pesc.coppe.ufrj.br/uploadfile/publicacao/2826.pdf>. Acesso em: 18 fev. 2026. Citado na página 15.

COIN-OR FOUNDATION. **CBC**: COIN-OR branch and cut. 2024. Disponível em: <https://www.coin-or.org/Cbc/>. Acesso em: 20 fev. 2026. Citado na página 18.

COSTA, J. de S. **Técnicas de otimização aplicadas a sistemas elétricos de distribuição**. 107 p. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Juiz de Fora, 2008. Disponível em: <https://repositorio.ufjf.br/jspui/handle/ufjf/2861>. Acesso em: 17 fev. 2026. Citado na página 10.

DANTZIG, G. B. **Linear Programming and Extensions**. Princeton, NJ: Princeton University Press, 1963. 625 p. Citado na página 14.

_____. The diet problem. **Interfaces**, v. 20, n. 4, p. 43–47, 1990. Disponível em: <https://doi.org/10.1287/inte.20.4.43>. Acesso em: 20 fev. 2026. Citado na página 22.

GNU PROJECT. **GLPK**: GNU linear programming kit. Boston, MA, 2024. Disponível em: <https://www.gnu.org/software/glpk/>. Acesso em: 20 fev. 2026. Citado na página 17.

GOMORY, R. E. Outline of an algorithm for integer solutions to linear programs. **Bulletin of the American Mathematical Society**, v. 64, n. 5, p. 275–278, 1958. Citado na página 16.

HART, W. E.; WATSON, J.-P.; WOODRUFF, D. L. Pyomo: Modeling and solving mathematical programs in Python. **Mathematical Programming Computation**, v. 3, p. 219–260, ago. 2011. Citado na página 20.

HiGHS DEVELOPERS. **HiGHS Optimization Software**. 2024. Disponível em: <https://highs.dev/>. Acesso em: 20 fev. 2026. Citado na página 17.

KARMARKAR, N. A new polynomial-time algorithm for linear programming. **Combinatorica**, v. 4, p. 373–395, dez. 1984. Citado na página 15.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. **Knapsack problems**. Berlin, Heidelberg: Springer, 2004. 563 p. ISBN 9783642073113. Citado na página 22.

KOCH, T.; BERTHOLD, T.; PEDERSEN, J.; VANARET, C. Progress in mathematical programming solvers from 2001 to 2020. **EURO Journal on Computational Optimization**, v. 10, p. 100031, 2022. Citado na página 18.

LAND, A. H.; DOIG, A. G. An automatic method of solving discrete programming problems. **Econometrica**, v. 28, n. 3, p. 497–520, jul. 1960. Citado na página 15.

MACHADO, D. A benchmark of optimization solvers for genome-scale metabolic modeling of organisms and communities. **mSystems**, v. 9, n. 2, p. e00833–23–1–8, fev. 2024. Citado 2 vezes nas páginas 11 e 19.

MACULAN, N.; FAMPA, M. H. C. **Otimização Linear**. Rio de Janeiro: Universidade Federal do Rio de Janeiro, 2004. 263 p. Citado 2 vezes nas páginas 13 e 14.

MARTINS, D. F. **0-1 Knapsack Problem Instances**. 2024. Repositório GitHub. Disponível em: <https://github.com/dnlfm/knapsack-01-instances>. Acesso em: 17 fev. 2026. Citado na página 28.

MEINDL, B.; TEMPL, M. Analysis of commercial and free and open source solvers for the cell suppression problem. **Transactions on Data Privacy**, v. 6, n. 2, p. 147–159, ago. 2013. Citado na página 18.

NEUMAIER, A.; SHCHERBINA, O. Safe bounds in linear and mixed-integer linear programming. **Mathematical Programming**, v. 99, p. 283–296, maio 2004. Citado na página 19.

PAIXÃO, J. L. da; DANIELSSON, G. H.; ABAIDE, A. da R. Otimização via branch-and-bound: Teoria, algoritmos e exemplos de aplicação. In: **Ciência, Tecnologia e Inovação na Engenharia Elétrica**. Ponta Grossa: Atena Editora, 2025, (Coleção Engenharias). cap. 2. ISBN 978-65-258-3358-3. Citado na página 15.

PISINGER, D. **Knapsack problem instances and optimization codes**. 2005. Disponível em: <http://hjemmesider.diku.dk/~pisinger/codes.html>. Acesso em: 17 fev. 2026. Citado 5 vezes nas páginas 23, 24, 27, 28 e 39.

SCIP OPTIMIZATION SUITE. **SCIP**: Solving constraint integer programs. 2024. Disponível em: <https://www.scipopt.org/>. Acesso em: 20 fev. 2026. Citado na página 17.

SOUTO-MAIOR, C. D. **Pesquisa Operacional**. 3. ed. Florianópolis: Universidade Federal de Santa Catarina, 2014. 94 p. ISBN 978-85-7988-151-0. Citado na página 12.

TAHA, H. A. **Operations Research**: An introduction. 8. ed. Upper Saddle River: Pearson Prentice Hall, 2007. 838 p. ISBN 0-13-188923-0. Citado na página 14.

TIRONE, G. S. P. de C. **Programação Linear**: Uma aplicação ao problema da dieta. 71 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Federal do Rio Grande, Rio Grande, RS, 2019. Citado 2 vezes nas páginas 24 e 25.