

Branch-and-Bound Library: Um Framework C++ livre para Implementação de Algoritmos de Branch-And-Bound Paralelo

Wendel Melo

Departamento de Engenharia de Telecomunicações

Escola de Engenharia

Universidade Federal Fluminense

wendelx@id.uff.br

RESUMO

Neste trabalho apresentamos um *framework* para implementação de algoritmos de *Branch-and-Bound* paralelo denominado *Branch-and-Bound Library* ou BBL. BBL é desenvolvida sobre o paradigma de orientação a objetos em C++ e pode ser utilizada para a construção de metodologias baseadas em *Branch-and-Bound* para otimização inteira, tanto para o caso linear quanto para o caso não linear e para otimização global por meio de *Branch-and-Bound* espacial. BBL é distribuída sob a licença MIT de *software* livre, possui uma série de parâmetros para customização e permite que suas estruturas de dados (classes) sejam derivadas, possibilitando assim um maior nível de especialização das abordagens construídas. Como resultado deste trabalho, esperamos fornecer suporte para pesquisa e desenvolvimento de aplicações especializadas de *Branch-and-Bound*, tanto em domínio discreto quanto em domínio contínuo.

PALAVRAS CHAVE. *Branch-and-Bound*, Otimização Computacional, *Software* livre.

Branch-and-Bound; Desenvolvimento de *Software*.

ABSTRACT

In this work, we present a framework for the implementation of Branch-and-Bound algorithms called *Branch-and-Bound Library* or BBL. BBL is developed using the object-oriented paradigm and can be used to build methodologies based on Branch-and-Bound for integer optimization, for both linear and nonlinear cases, and for global optimization by means of spatial Branch-and-Bound. BBL is released under the MIT free software license; it has several parameters for customization and it allows derivation of its data structures (classes), thereby enabling a high level of specialization in the developed approaches. As a result of this work, we hope to support research and development of specialized applications of Branch-and-Bound in both continuous and discrete domains.

KEYWORDS. *Branch-and-Bound*, Computational Optimization, Free Software.

Branch-and-Bound; Software Development.

1. Introdução

Land e Doig propuseram o que parece ser o primeiro algoritmo de *Branch-and-Bound* (BB) em Land e Doig [1960]. Desde então, algoritmos de *Branch-and-Bound* se consolidaram como uma das principais técnicas para a resolução de problemas de otimização que incorporam variáveis

em domínios discretos, seja em contexto linear ou não linear, e também para a resolução de problemas de otimização global em domínio contínuo através da variação conhecida como *Branch-and-Bound* espacial (mais informações podem ser encontradas em Morrison et al. [2016]; Huang et al. [2021]; Liberti [2008]). Neste trabalho, consideramos o seguinte problema geral de otimização:

$$(P) \quad \begin{array}{ll} \text{minimizar}_x & f(x) \\ \text{sujeito a:} & x \in X \end{array} \quad (1)$$

onde $f(x)$ é uma função possivelmente não linear e possivelmente não convexa, e X é uma região viável definida por um conjunto de restrições, possivelmente incluindo restrições não convexas e/ou de integralidade de variáveis. Em geral, algoritmos de BB são adotados para tratar problemas não convexos, seja pela incorporação de restrições de integralidade, seja pela presença de funções não convexas na sua função objetivo ou no seu conjunto de restrições. Em geral, há três elementos principais presentes em uma típica aplicação de algoritmo de BB:

1. Um esquema de relaxação convexa para o problema (P) , com o qual se pode obter limites duais (inferiores para o caso de minimização) para (P) ;
2. Um esquema de particionamento da região viável de (P) que permita melhorar a relaxação convexa em cada subpartição gerada, de modo que os limites duais possam ser melhorados a medida que o algoritmo progride e explora novas subpartições;
3. Mecanismos que permitam obter limites primais para (P) , em geral através de soluções viáveis que por ventura são encontradas pela metodologia. Dessa forma, pode-se adotar um critério de parada principal baseado na nulidade da diferença entre o melhor limite primal conhecido e o melhor limite dual conhecido. Limites primais também são utilizados para o descarte de subpartições onde a busca por uma solução de melhora não se mostre promissora.

Os principais pacotes de otimização linear inteira, tanto comerciais quanto abertos, possuem implementações engenhosas de algoritmos de BB, como, por exemplo Cplex (IBM [2015]), Gurobi (Gu et al. [2011]), Xpress (FICO [2025]), Mosek (APS [2025]), GLPK (Makhorin [2012]), CBC (Forrest et al. [2020]), dentre outros. Todavia, mesmo com os avançados recursos do estado da arte na área, ainda existe dificuldade marcante para a resolução de diversas aplicações do mundo real. De forma similar, os principais pacotes de otimização global, por exemplo, Baron (Tawarmalani e Sahinidis [2005]), SCIP (Bulusani et al. [2024]) e Couenne (Belotti et al. [2009]) possuem implementações amadurecidas de BB espacial. Entretanto, esses pacotes podem ter dificuldade em abordar problemas não convexos mesmo em dimensão reduzida. Por estas razões, pode ser necessário desenvolver algoritmos personalizados para abordar aplicações específicas de otimização. Por exemplo, em Bard e Moore [1990] é desenvolvido um algoritmo de BB especializado para o problema de otimização em dois níveis. Já em Fampa et al. [2016], é apresentado um algoritmo de BB especializado para o problema de Steiner Euclidiano. Dentre as dificuldades envolvidas na implementação de um algoritmo de BB paralelo, podemos mencionar:

- Implementação de exploração paralela na árvore de partições em sistemas com múltiplos processadores, com sincronização de *threads* de processamento e gerenciamento de zonas de exclusão mútua para atualização de dados de execução;
- Armazenamento de dados eficiente em memória para possibilitar o tratamento de problemas com árvores de enumeração extensas;

- Gerenciamento de estruturas de dados que representam a árvore de subpartições e os respectivos subproblemas envolvidos em cada uma delas;
- Descarte de partições não promissoras quando o limite primal é atualizado;

De modo apoiar a pesquisa e implementação eficiente de algoritmos de BB, tanto com objetivos gerais quanto para aplicações específicas, apresentamos nesse trabalho um *framework* aberto em C++ para o desenvolvimento de algoritmos de BB paralelo. Nosso *framework* é orientado a objetos e toma para si a complexidade envolvida no gerenciamento das *threads* de execução, das estruturas de dados envolvidas e da exploração apropriada da árvore de subpartições com descarte de subramos não promissores, verificação de critérios de parada e mecanismos para acompanhamento geral da execução do algoritmo. Nosso *framework*, denominado *Branch-and-Bound Library* (Biblioteca de *Branch-and-Bound*) ou BBL, é gratuito, de código aberto e está disponível em repositório público de códigos de programação. Ademais, além de BBL possui uma série de parâmetros customizáveis, a biblioteca ainda é implementada sob o paradigma da orientação a objetos usando a linguagem C++, o que permite que suas classes possam ser derivadas de modo a facilitar a modificação dos procedimentos implementados, caso seja necessário. Com a disponibilização de BBL, esperamos facilitar o trabalho de pesquisadores e técnicos em trabalhos relacionados ao desenvolvimento de algoritmos de *Branch-and-Bound* em aplicações específicas, liberando os mesmos de toda a complexidade envolvida em uma implementação eficiente dos pormenores de uma metodologia de BB, o que pode ser especialmente útil para usuários com pouca experiência em programação.

Este texto está organizado da seguinte forma: a Seção 2 discute aspectos gerais do algoritmo de BB, a Seção 3 apresenta a *Branch-and-Bound Library* de forma geral. Por fim, a seção 4 conclui o trabalho.

2. O Algoritmo de *Branch-and-Bound*

Em geral, algoritmos de *Branch-and-Bound* são aplicados na resolução de problemas como (P) onde algum tipo de não-convexidade esteja presente. Esta não convexidade pode ser oriunda de restrições de integralidade de um subconjunto das variáveis de decisão (otimização inteira mista), ou pode se originar da presença de funções não convexas na função objetivo ou no conjunto de restrições (otimização global). Para que seja exitosa, uma metodologia de BB deve se basear em algum um esquema de relaxação convexa a ser adotado nas partições do espaço que vão sendo dinamicamente geradas. Assim, o problema abordado (P) é aproximado por sua relaxação convexa, aqui denotada por $(\tilde{P})$. Adicionalmente, o esquema de relaxação convexa adotado deve ter o potencial de melhorar a aproximação do problema original à medida que novas subpartições do espaço são geradas e exploradas. Usualmente, em problemas de otimização linear inteira mista, a relaxação convexa é obtida através da relaxação das restrições de integralidade, gerando assim a chamada *relaxação contínua*. Esse mesmo esquema também pode ser adotado em problemas de otimização não linear inteira mista onde não haja qualquer função não convexa presente na função objetivo nem no conjunto de restrições. Em contra-partida, em problemas de otimização global contínua, o esquema de relaxação convexa deve ser capaz de construir uma envoltória convexa sobre a região viável do problema, e/ou encontrar uma função convexa que seja um minorante para a função objetivo, caso esta última seja não convexa. Em problemas de otimização global com variáveis em domínio discreto, pode ser preciso mesclar estes dois esquemas para a construção da almejada relaxação convexa. Em problemas de otimização global contínua, a metodologia de BB é denominada como *Branch-and-Bound espacial*. Por se basear no particionamento do espaço algébrico do problema tratado, podemos classificar a metodologia de BB como sendo um exemplar de algoritmo de divisão e conquista.

O Algoritmo 1 traz o algoritmo geral de BB. Aqui, definimos como $(\tilde{P})$ a relaxação convexa de (P) e como $\tilde{f}(x)$ a função objetivo de $(\tilde{P})$. X^k é a k -ésima partição do espaço, associada então ao k nó na chamada árvore de BB. Quando $k = 0$, temos o nó raiz desta árvore e a partição inicial do espaço, que é definida como o próprio espaço original do problema (linha 2). z^u é definido como o melhor limitante primal conhecido (para o caso de minimização, z^u é o melhor limitante superior conhecido), em geral dado por soluções viáveis para (P) . l_b^k é o limite inferior da partição k .

	Entrada: (P) : Problema de Otimização, ϵ_c : tolerância de convergência.
	Saída: (x^*) : solução ótima de (P) .
1	$z^u := +\infty$;
2	$X^0 := X$;
3	Seja $N := \{0\}$ a lista inicial de índices de nós em aberto;
4	$i := 0$;
5	Seja l_b^i o limitante inferior do nó i ;
6	$l_b^0 := -\infty$;
7	Seja $(\tilde{P}_{X^k})$ a relaxação convexa de (P) resolvida na partição (divisão) do espaço X^k ;
8	enquanto $N \neq \emptyset$ faça
9	Selecione um nó k de N ;
10	$N := N \setminus \{k\}$;
11	se $(\tilde{P}_{X^k})$ é inviável então
12	Seja $(\tilde{x}^k)$ uma solução ótima de $(\tilde{P}_{X^k})$ com valor objetivo ótimo $\tilde{f}(\tilde{x}^k)$;
13	se $\tilde{x}^k$ for viável para (P) e $f(\tilde{x}^k) < z^u$ então
14	$z^u := f(\tilde{x}^k)$;
15	$(x^*) := (\tilde{x}^k)$;
16	$N := N \setminus \{j : l_b^j \geq z^u - \epsilon_c\}$; // Podas por limite
17	se $\tilde{f}(\tilde{x}^k) < z^u - \epsilon_c$ então
	// Ramificação
18	Selecione uma variável x_j para ramificação;
19	Selecione os valores $\underline{v}_j$ e $\overline{v}_j$ para definir a divisão do espaço;
20	$X^{i+1} := X^k \cap \{x \in \mathbb{R}^n : x_j \leq \underline{v}_j\}$;
21	$X^{i+2} := X^k \cap \{x \in \mathbb{R}^n : x_j \geq \overline{v}_j\}$;
22	$l_b^{i+1} := l_b^{i+2} := \tilde{f}(\tilde{x}^k)$;
23	$N := N \cup \{i+1, i+2\} \setminus \{k\}$;
24	$i := i+2$;

Algoritmo 1: Branch-and-Bound básico.

Note que, no Algoritmo 1, uma árvore de busca é percorrida, onde, a cada nó k explorado, a relaxação convexa $(\tilde{P}_{X^k})$ na partição X^k é resolvida. Assim, tomando por $(\tilde{x}^k)$ a solução ótima encontrada para $(\tilde{P}_{X^k})$, caso este seja viável, temos as seguintes possibilidades a considerar:

1. *O problema $(\tilde{P}_{X^k})$ é inviável:* neste caso, pode-se descartar todas as subpartições de X^k , pois também serão inviáveis (*poda por inviabilidade*, linha 11 do Algoritmo 1);
2. $\tilde{f}(\tilde{x}^k) \geq z^u$: neste caso, como $\tilde{f}(\tilde{x}^k)$ é um limitante inferior para (P) em toda subpartição X^k , podemos então descartar esta partição, pois ela não pode abrigar solução que melhore o limitante superior (*poda por limite*, linha 17 do Algoritmo 1);
3. $\tilde{x}^k$ *é viável para* (P) *e* $f(\tilde{x}^k) < z^u$: neste caso, temos uma solução viável melhor que a melhor solução conhecida. Assim, podemos atualizar z^u e também descartar todas as demais partições em aberto que apresentam limitante inferior maior que o novo valor de z^u (*podas*

por limite, linha 16 do Algoritmo 1). Note que $\tilde{x}^k$ ser viável para (P) implica em $\tilde{x}^k$ atender possíveis restrições de integralidade ou restrições não convexas que possam estar presentes em (P) . Assim, é possível que $\tilde{x}^k$ seja viável para $(\tilde{P}_{X^k})$ (relaxação), mas não seja viável para (P) (o problema original);

4. $\tilde{f}(\tilde{x}^k) < z^u$: neste caso, precisamos avaliar subpartições de X^k , pois existe a possibilidade de alguma delas abrigar solução viável que melhore o limitante primal. Escolhemos uma variável x_j para particionar o espaço e ramificamos a árvore, criando assim duas novas subpartições (*ramificação*, linhas 18-24 do Algoritmo 1). É interessante notar que, em casos onde $(\tilde{P})$ é construído apenas relaxando as restrições de integralidade, como no caso onde (P) é um problema de otimização linear inteira mista, se $\tilde{x}^k$ for uma solução inteira e melhorar o limitante superior, ocorrerá poda por otimalidade na linha 17, pois $f(\tilde{x}^k)$ terá atualizado o limite superior z^u na linha 14.

No processo de ramificação (linhas 18-24), deve-se escolher os valores $\underline{v}_j$ e $\overline{v}_j$ (linha 19) para definir a divisão do espaço (linhas 20 e 21). No caso de (P) ser um problema de otimização linear inteira mista, classicamente, neste tipo de partição do espaço, x_j está restrita a valores inteiros e $\underline{v}_j = \lfloor \tilde{x}_j^k \rfloor$ e $\overline{v}_j = \lceil \tilde{x}_j^k \rceil$. Outros esquemas de ramificação mais complexos podem ser usados, todavia. Por outro lado, se (P) for um problema de otimização global contínua, em geral, $\underline{v}_j = \overline{v}_j$, pois não faz sentido o conceito de arredondamento. Note que, nessa situação, a rigor, a divisão do espaço não gera partições, pois as duas divisões geradas terão interseção não vazia.

De modo a acelerar a convergência de algoritmos de BB, diversos truques tem sido propostos na literatura, como, por exemplo, pré-processamento, heurísticas de viabilidade, pseudo-custos, *strong branching*, geração de colunas, geração de cortes, ramificação precoce (*early branching*), etc (o leitor interessado pode consultar Morrison et al. [2016]; Huang et al. [2021]; Borchers e Mitchell [1994]). O escopo deste trabalho se concentra em implementar um arcabouço geral de BB, que não dependa de qualquer dado oriundo do problema sendo resolvido. Assim, nosso *framework* BBL não contempla estes recursos. Todavia, os mesmos ainda podem ser incorporados em uma implementação de BB que seja construída sobre BBL, ficando então estes recursos a cargo de implementação do usuário, pois dependem de particularidades do problema sendo tratado.

3. O *framework Branch-and-Bound Library*

Nosso *framework*, denominado *Branch-and-Bound Library* implementa em C++ todo o arcabouço necessário para um algoritmo de BB paralelo conforme apresentado no Algoritmo 1, utilizando os conceitos do desenvolvimento orientado a objetos. A inspiração para o desenvolvimento de BBL surgiu da nossa experiência prévia em implementar algoritmos de *Branch-and-Bound* para diferentes contextos, como o algoritmo de BB do solver MINLP *Muriqui* (Melo et al. [2020]), o algoritmo de BB espacial do solver quadrático indefinido *Iquad* (Fampa et al. [2017]), o algoritmo heurístico IGMA (Algoritmo de Minimização do Gap de Integralidade) para MINLP Binária (Melo et al. [2018]) e o algoritmo especializado para a resolução do problema de Steiner euclidiano em Fampa et al. [2016]. Rapidamente ficou claro que seria de interesse concentrar todos os mecanismos internos relacionados ao funcionamento de BB em uma biblioteca computacional separada, de modo a facilitar a manutenibilidade, clareza, correção de erros e reaproveitamento, não apenas da nossa parte, mas também por parte de outros pesquisadores e técnicos. BBL é distribuída livremente sobre a licença MIT (MIT [1987]) e, correntemente, possui mais 6 mil linhas de código estrito (já excluindo linhas em branco e linhas de comentário). BBL está disponível no repositório <https://github.com/xmuriqui/bbl> onde pode ser baixada para utilização e estudo. Para usar BBL, todavia, o usuário deve implementar:

- A definição dos limites inferiores e superiores para as variáveis do problema;
- Um procedimento *callback* para a resolução das relaxações convexas em cada partição do espaço (linha 12 do Algoritmo 1). Por exemplo, para implementar um resolvidor para otimização linear inteira mista com BBL, será preciso utilizar algum método de resolução de otimização linear para construir este procedimento *callback*;
- Um procedimento para escolher a(s) variável(eis) x_j sobre a(s) qual(is) a ramificação será feita, com os respectivos valores $\underline{v}_j$ e $\overline{v}_j$ para definição das partições geradas.

Procedimentos adicionais para melhorar o desempenho de metodologias de BB ficam a cargo de usuário. O escopo de BBL é implementar de forma transparente o funcionamento geral interno de um algoritmo de BB paralelo. É válido ressaltar que BBL permite a total customização do particionamento ou divisão do espaço, inclusive permitindo a ramificação sobre múltiplas variáveis simultaneamente conforme discutido em Melo [2012]. Por exemplo, se for optado pelo particionando sobre duas variáveis binárias x_i e x_j , serão geradas quatro subpartições do espaço: a primeira com $x_i = 0, x_j = 0$, a segunda com $x_i = 0, x_j = 1$, a terceira com $x_i = 1, x_j = 0$ e a quarta $x_i = 1, x_j = 1$. Dentre as principais funcionalidades presentes podemos mencionar:

- Exploração das partições do espaço em paralelo com possibilidade de escolha do número de *threads* exploradoras;
- Estruturas de dados avançadas para gerenciamento eficiente dos nós da árvore de BB;
- Zona de exclusão mútua para garantir o correto gerenciamento e exploração das partições entre as diferentes *threads*;
- Procedimento *callback* chamado para resolução da relaxação convexa em cada partição. Com a implementação deste procedimento, o usuário pode adicionar diversas funcionalidades ao código como preprocessamento, adição de cortes, heurísticas, geração de colunas, ramificação precoce (*early branching*, proposta por Borchers e Mitchell [1994]), atualização de pseudo-custo, etc;
- Procedimento *callback* para o usuário escolher o(s) índice(s) da(s) variável(eis) x_j sobre as quais será feita ramificação, e os valores $\underline{v}_j$ e $\overline{v}_j$ usados para dividir o espaço;
- Um procedimento *callback* que permite ao usuário gerar as novas partições por conta própria como bem entender;
- Possibilidade de Incorporação de solução primal e/ou dual de cada relaxação solucionada, de modo a permitir o uso de técnicas de *warm start*;
- Possibilidade de derivar a classe BBL_Node, que carrega as informações de cada partição gerada. Dessa forma, é possível armazenar informações extras para cada partição e personalizar o método de geração de novas partições;
- Exploração da árvore pelas seguintes estratégias: “Largura”, “Profundidade”, “Melhor Limite”, “Profundidade até a primeira solução viável e após Melhor Limite”;
- Histórico de soluções viáveis encontradas;
- Procedimento *callback* chamado a cada vez que a melhor solução conhecida é encontrada;

- Procedimento *callback* que permite acompanhar o final de cada iteração;
- Contadores de podas de cada tipo realizadas ao longo da exploração;
- Possibilidade de contagem do número de podas por tipo em cada nível de profundidade da árvore;
- Critérios de parada alternativos como: “*número máximo de iterações*”, “*tempo máximo de CPU*”, “*tempo máximo de relógio*” e “*parada forçada pelo usuário*”;
- Possibilidade de definição de limites adicionais para as variáveis de decisão do problema abordado.
- Possibilidade de checagem dinâmica, por parte do usuário, de todos os nós de BB em aberto nos procedimentos *callback*.

É possível encontrar outros *frameworks* para implementação de *Branch-and-Bound*, como, por exemplo, *Kodiak* (Smith et al. [2015]), outro framework também implementado em C++ como um projeto suportado pela agência espacial dos Estados Unidos (NASA). Todavia, *Kodiak* possui menos opções customizáveis que BBL e implementa o BB de modo recursivo, o que traz limitações práticas pelo maior uso de memória do sistema necessário para realizar chamadas de função, limitação da profundidade máxima da árvore de BB devido ao tamanho máximo da pilha de chamadas de função e a possibilidade de exploração apenas por profundidade, o que pode comprometer seriamente a performance de um algoritmo de BB. Uma outra biblioteca, denominada *ABACUS* (Jünger e Thienel [2000]) oferece um framework avançado para BB já com integração com pacotes de otimização linear, e suporte direto a cortes e geração de colunas. Todavia, o projeto está restrito apenas ao âmbito de problemas de otimização com expressões lineares, e não pode ser usado para implementar um algoritmo de BB espacial que adote relaxação convexa não linear. Outro projeto similar ao *ABACUS*, é o *SYMPHONE* (Ralphs e Güzelsoy [2005]), que possui funcionalidades semelhantes, mas também possui seu escopo limitado apenas a subproblemas lineares. Outros *frameworks* para *Branch-and-Bound* incluem *BiCePS* (Ralphs et al. [2004]) e *Branch-and-Cut and Price Framework* (Ralphs e Ládanyi [2001]), sendo estes também com recursos avançados voltados para otimização linear inteira. BBL, por outro lado, tem o objetivo de trazer a separação total entre o problema sendo resolvido e as implementações relativas ao funcionamento de algoritmos de BB. Por essa razão, BBL não traz qualquer tipo de recurso para representação do problema, integração com pacotes resolvidores e suporte a recursos avançados como geração de cortes e de colunas. Em contrapartida, BBL pode ser utilizada em qualquer projeto geral que envolva BB, seja para o tratamento de problemas lineares ou não lineares, com a possível presença de variáveis inteiras. Ademais, BBL pode ser integrada com outros pacotes computacionais que implementam apenas os recursos adicionais presentes nos demais *frameworks* citados, como, por exemplo, o projeto *Open Solver Interface* (Forrest et al. [2024a]) que implementa integração com uma série de solvers de otimização linear, ou o projeto *Cut Generator Library* (Forrest et al. [2024b]) que implementa uma biblioteca com uma série de opções de cortes para otimização linear inteira mista. Através desses projetos externos, pode-se trazer ferramentas avançadas para um BB implementado com BBL, com a vantagem de BBL ser altamente customizável e possibilitar, por exemplo, a construção de procedimentos para resolução de problemas de otimização global.

4. Conclusões

O desenvolvimento de software aberto é uma tarefa importante para dar suporte a pesquisa e aplicações computacionais. Neste trabalho, disponibilizamos um *framework* aberto em C++ para a

implementação de algoritmos de BB paralelo denominado *Branch-and-Bound Library* (BBL). BBL tem o objetivo de suportar o desenvolvimento e aplicação de metodologias de BB especializadas a determinadas aplicações tornando transparente ao usuário a exploração paralela do espaço de busca. Dessa forma, usuários podem se focar na contribuição de suas metodologias em si, se liberando da preocupação com detalhes técnicos da implementação do procedimento base de BB. BBL está disponível sob a licença MIT no repositório <https://github.com/xmuriqui/bbl>, onde pode ser baixada livremente para utilização e estudo. Junto com o código e arquivos de compilação, também é possível baixar exemplos de uso da BBL neste mesmo repositório. Correntemente, BBL já suporta procedimentos de *Branch-and-Bound* nos pacotes computacionais Muriqui para otimização não linear inteira mista e Iquad para otimização quadrática global.

Referências

- APS, M. (2025). The mosek optimization toolbox for matlab manual - release 11.0.19. Software. URL <http://www.mosek.com/>.
- Bard, J. F. e Moore, J. T. (1990). A branch and bound algorithm for the bilevel programming problem. *SIAM Journal on Scientific and Statistical Computing*, 11(2):281–292. URL <https://doi.org/10.1137/0911017>.
- Belotti, P., Lee, J., Liberti, L., Margot, F., e and, A. W. (2009). Branching and bounds tightening-techniques for non-convex minlp. *Optimization Methods and Software*, 24(4-5):597–634. URL <https://doi.org/10.1080/10556780903087124>.
- Bolusani, S., Besançon, M., Bestuzheva, K., Chmiela, A., Dionísio, J., Donkiewicz, T., van Doornmalen, J., Eifler, L., Ghannam, M., Gleixner, A., Graczyk, C., Halbig, K., Hedtke, I., Hoen, A., Hojny, C., van der Hulst, R., Kamp, D., Koch, T., Kofler, K., Lentz, J., Manns, J., Mexi, G., Mühmer, E., Pfetsch, M. E., Schlösser, F., Serrano, F., Shinano, Y., Turner, M., Vigerske, S., Weninger, D., e Xu, L. (2024). The SCIP Optimization Suite 9.0. ZIB-Report 24-02-29, Zuse Institute Berlin. URL <https://nbn-resolving.org/urn:nbn:de:0297-zib-95528>.
- Borchers, B. e Mitchell, J. E. (1994). An improved branch and bound algorithm for mixed integer nonlinear programs. *Comput. Oper. Res.*, 21:359–367. ISSN 0305-0548.
- Fampa, M., Lee, J., e Melo, W. (2016). A specialized branch-and-bound algorithm for the euclidean steiner tree problem in n-space. *Computational Optimization and Applications*, p. 1–25. ISSN 1573-2894. URL <http://dx.doi.org/10.1007/s10589-016-9835-z>.
- Fampa, M., Lee, J., e Melo, W. (2017). On global optimization with indefinite quadratics. *EURO Journal on Computational Optimization*, 5(3):309–337. ISSN 2192-4414. URL <https://doi.org/10.1007/s13675-016-0079-6>.
- FICO (2025). Fico xpress optimization. Software. URL <https://www.fico.com/en/products/fico-xpress-optimization>.
- Forrest, J., Vigerske, S., Hafer, L., Ralphs, T., jpfasano, Saltzman, M., Santos, H. G., Jan-Willem, Hunsaker, B., Achterberg, T., Jensen, B., Andre, A., Gleixner, A., Gassmann, H. I., Lougee, R., Young, J., Kristjansson, B., King, A., Arevall, HÅ¥vard, Spoorendonk, S., e to st (2024a). coin-or/osi: Release 0.108.11. URL <https://doi.org/10.5281/zenodo.13345260>. Acessado em: 3 de maio de 2025.

- Forrest, J., Vigerske, S., Ralphs, T., Hafer, L., Santos, H. G., rlougee, Bonami, P., jpfasano, Jan-Willem, a andre, h-i gassmann, Saltzman, M., Kristjansson, B., jpgoncal1, Brito, S., to st, e tost-tost (2024b). coin-or/cgl: Release 0.60.9. URL <https://doi.org/10.5281/zenodo.13347302>. Acessado em: 3 de maio de 2025.
- Forrest, J., Vigerske, S., Santos, H. G., Ralphs, T., Hafer, L., Kristjansson, B., fasano, J. P., Straver, E., Lubin, M., Rlougee, Jpgoncal1, Gassmann, H. I., e Saltzman, M. (2020). coin-or/cbc: Version 2.10.5. URL <https://doi.org/10.5281/zenodo.3700700>.
- Gu, Z., Rothberg, E., e Bixby, R. (2011). Gurobi 4.6.1. Software. URL www.gurobi.com.
- Huang, L., Chen, X., Huo, W., Wang, J., Zhang, F., Bai, B., e Shi, L. (2021). Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. URL <https://arxiv.org/abs/2111.06257>.
- IBM. *IBM ILOG CPLEX V12.6 User's Manual for CPLEX*, 2015.
- Jünger, M. e Thienel, S. (2000). The abacus system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Software: Practice and Experience*, 30 (11):1325–1352. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-024X%28200009%2930%3A11%3C1325%3A%3AAID-SPE342%3E3.0.CO%3B2-T>.
- Land, A. H. e Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520. ISSN 00129682. URL <http://www.jstor.org/stable/1910129>.
- Liberti, L. (2008). *Introduction to Global Optimization*. URL https://www.lix.polytechnique.fr/~liberti/teaching/mpro/pma-12/nonconvex_optimization.pdf.
- Makhorin, A. *GLPK (GNU Linear Programming Kit)*, 2012. URL <https://www.gnu.org/software/glpk/>.
- Melo, W. (2012). Algoritmos para programação não linear inteira mista. dissertação de mestrado, COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.
- Melo, W., Fampa, M., e Raupp, F. (2018). Integrality gap minimization heuristics for binary mixed integer nonlinear programming. *Journal of Global Optimization*, 71(3):593–612. ISSN 1573-2916.
- Melo, W., Fampa, M., e Raupp, F. (2020). An overview of minlp algorithms and their implementation in muriqui optimizer. *Annals of Operations Research*, 286(1):217–241. ISSN 1572-9338.
- MIT (1987). The mit license (mit). License. URL <https://mit-license.org/>.
- Morrison, D. R., Jacobson, S. H., Sauppe, J. J., e Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102. ISSN 1572-5286. URL <https://www.sciencedirect.com/science/article/pii/S1572528616000062>.

- Ralphs, T. K. e Ládanyi, L. *COIN/BCP User's Manual*, 2001. URL <https://citeseerx.ist.psu.edu/document?doi=be5496effe9efcbeb94aa5cf08f4a7d2a2a2fc1c>.
- Ralphs, T. K., Ládanyi, L., e Saltzman, M. J. (2004). A library hierarchy for implementing scalable parallel search algorithms. *The Journal of Supercomputing*, 28:215–234. URL <https://link.springer.com/article/10.1023/B:SUPE.0000020179.55383.ad>.
- Ralphs, T. K. e Güzelsoy, M. (2005). The symphony callable library for mixed integer programming. In Golden, B., Raghavan, S., e Wasil, E., editors, *The Next Wave in Computing, Optimization, and Decision Technologies*, p. 61–76, Boston, MA. Springer US. ISBN 978-0-387-23529-5.
- Smith, A. P., Muñoz, C. A., Narkawicz, A. J., e Markevicius, M. (2015). Kodiak: An implementation framework for branch and bound algorithms. Technical Report NASA/TM-2015-218776, NASA Langley Research Center, Hampton, Virginia. URL <https://ntrs.nasa.gov/citations/20150014346>.
- Tawarmalani, M. e Sahinidis, N. V. (2005). A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249. ISSN 0025-5610. URL <http://dx.doi.org/10.1007/s10107-005-0581-8>.