BRANCHING STRATEGIES FOR NONLINEAR BRANCH-AND-BOUND

Leon Stanzel
Technische Universität Berlin
stanzel@campus.tu-berlin.de

ABSTRACT

1 Introduction

2 Contribution

3 MINLP

3.1 General MINLP

In general mixed-integer nonlinear programming (MINLP) problems are of the following form:

$$\min_{x \in \mathcal{X}} f(x)$$

where $\mathcal{X}$ is the feasible region made up of both nonlinear and discrete structures.
Moreover $f : \mathbb{R}^n \to \mathbb{R}$ could be any function.
This class of problems is NP-hard as it contains the class of mixed-integer linear programming problems which are well known to be NP-hard Kannan and Monma [1978]. It also contains problems which are undecidable Jeroslow [1973]. Commonly the class of MINLP problems is separated in convex and nonconvex subclasses. Note that convex MINLP are MINLP for which dropping integrality constraints results in a convex nonlinear programming problem (NLP). Real world applications of MINLP find themselves for example in manufacturing, portfolio optimization and logistics Lee and Leyffer [2011]. Applications in machine learning include the well known Support Vector Machines and Bayesian Network LearningManzour et al. [2021]. At present a large number of problems which can be modeled using MINLP are still unsolvable in practice, thus motivating further research.

3.2 Convex MINLP with boundable LMO

In this thesis we restrict to the setting of convex MINLP problems used in Hendrych et al. [2022], i.e. firstly we assume $\mathcal{X} = \bar{\mathcal{X}} \cup \mathbb{Z}_j$ is a compact non-convex set admitting a boundable linear minimization oracle (B-LMO), where $\mathbb{Z}_j = \{x \in \mathbb{R}^n | x_j \in \mathbb{Z} \forall j \in J\}, J \subseteq \{1, ..., n\}$ and $\bar{\mathcal{X}}$ denotes the continuous relaxation of $\mathcal{X}$. This means that optimizing a linear function over $\mathcal{X}$ can be done at the very least more efficiently than optimizing the original problem even when bounds are updated or added. Let **(l,u)** be new bounds, **d** $\in \mathbb{R}^n$ a direction, then the B-LMO is defined as the following function

$$(l, u, d) \in \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \arg\min_{v \in \mathbb{R}^n} \langle v, d \rangle \text{ s.t. } v \in \mathcal{X} \cap [l, u]$$

Secondly $f : \text{conv}(\mathcal{X}) \to \mathbb{R}$ is a Lipschitz-smooth convex function and we have access to its gradient without requiring an expression graph. $\text{conv}(\mathcal{X})$ refers to the convex hull of $\mathcal{X}$.
The last assumption made is that at each node given local bounds $[l, u]$ we have that

$$x \in \text{conv}(\mathcal{X}) \cap \mathbb{Z}_j \cap [l, u] \Rightarrow x \in X.$$

Be aware that in the context of optimization a differentiable function $f : \mathcal{X} \to \mathbb{R}$ is called Lipschitz-smooth or simply L-smooth if it holds that

$$f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2, \ \forall x, y \in \mathcal{X}.$$

4 Branch-and-Bound

One algorithm which is used as a basis for modern solvers of not only convex MINLP problems but also general MINLP problems is Branch-and-Bound (BnB). One example of a MINLP solver using Bnb is BARON which requires that both the objective function and the constraints can be expressed as factorable functions Tawarmalani and Sahinidis [2004]. Other examples include the convex MINLP solver Bonmin Bonami et al. [2013] and MINOTAUR which is a solver for decidable MINLP problemsMahajan et al. [2021]. Note that Branch-and-Bound was originally introduced by Land and Doig Land and Doig [2010] over 60 years ago in order to solve mixed-integer linear programming prolems (MIP) and still adaptations are a core component of many modern MIP solversBonami et al. [2013].
Given a convex MINLP P with objective function f and feasible region $\mathcal{X}$ the Branch-and-Bound algorithm works as follows Bonami et al. [2013]:

Algorithm 1 Branch-and-Bound Algorithm

```
Require: Set of current restrictions $\mathcal{R}$ initialized to contain the original problem $P$
 1: $U \leftarrow \infty$                                ▷ Current upper bound on objective function value
 2: if $\mathcal{R}$ is empty then
 3:     if a candidate solution x exists then
 4:         return x
 5:     else
 6:         return $P$ is infeasible
 7:     end if
 8: else
 9:     choose a restriction $Q$ from $\mathcal{R}$ and consider its (continuous) relaxation $Q^R$
10:     $x^R \leftarrow$ solve_to_optimality($Q^R$)
11:     remove $Q$ from $\mathcal{R}$
12:     if $x^R$ is infeasible for $Q^R$ then
13:         go back to 2.
14:     else if $f(x^R) \geq U$ then
15:         go back to 2.
16:     else
17:         $\mathcal{B} \leftarrow \{j \in J | x_j^R \notin \mathbb{Z}\}$          ▷ these are the branching candidates
18:         if $\mathcal{B}$ is not empty then
19:             choose $j \in \mathcal{B}$
20:             create a new child restriction $Q_1$ by modifying the upper bound on $x_j$ to $\lfloor x_j^R \rfloor$
21:             create a new child restriction $Q_2$ by modifying the lower bound on $x_j$ to $\lceil x_j^R \rceil$
22:             add the child restrictions to $\mathcal{R}$
23:         else
24:             $x^R$ is a feasible solution for $P$
25:             if $f(x^R) < f(x)$ or no candidate solution exists then
26:                 $x \leftarrow x^R$                         ▷ candidate solution is updated
27:             end if
28:             go to 2.
29:         end if
30:     end if
31: end if
```

Throughout each iteration of the algorithm the candidate solution maintains feasibility with regards to the original problem P and provides an upper bound on the optimal function value as each optimal solution to a relaxation is an upper bound on the optimal value of the original problem. As the current candidate solution is only changed if a strictly better solution is found, the sequence $(f(x_i))_{i=1,...,k}$ where x_i represents the i´th candidate solution found, is strictly decreasing. Note that also the lower bounds get increased when creating the second child node. Thus throughout the algorithm upper and lower bounds converge. From the algorithm it is clear that its performance primarily depends on three factors

- The order with which restriction are chosen from $\mathcal{R}$
- The choice of which variable to branch on also known as the branching strategy
- The quality of the relaxation used

as these directly impact the bounds computed and how many different restrictions have to be explored. The restrictions are also often called nodes as the original restriction and the added child restrictions make up a tree since there is one root and subsequently there exists exactly one path from the root to each leave. The length of a path from the root to a leave represent how many variables have been branched on to find the corresponding feasible solution. Most of the MINLP solvers use a continuous relaxation during the BnB process, i.e. they drop the integrality requirements.

4.1 Other solutions strategies

5 Branching Strategies

As the branching strategy has a large impact on the number of nodes processed before an optimal solution has been found and verified quite a few strategies have been tested since Branch-and-Bound was first applied to MINLP problems. Many of these strategies were first developed for mixed-integer linear problems and then applied to MINLP problems Bonami et al. [2011]. Ordered by perceived complexity some of the well known branching-strategies include:

- Random
- Most-Fractional(sometimes also known under Most-Infeasible)
- Lowest-Index-First
- Pseudocost
- Strong-Branching

Random branching as the name suggests is the strategy of simply choosing at random where usually this means according to the uniform distribution. The Most-Fractional strategy is slightly more complex as in this strategy for each branching candidate j one first has to calculate $\min\{\lceil x_j \rceil - x_j, x_j - \lfloor x_j \rfloor\}$ and then chooses the branching candidate where this is largest Bonami et al. [2013]. An algorithmically not more complex rule is Lowest-Index-First where at the very beginning the indexes of all integer variables are ordered by their somehow already known importance and then the selection of branching variable is made by always choosing the one with lowest index Gupta and Ravindran [1985]. Pseudocost branching is a strategy where one chooses to branch on the candidate variable with the largest expected improvement. This strategy however requires that one first follows an alternative strategy until the pseudocosts, the estimators of improvement, have stabilized. Stabilization is met when each candidate variable has been branched on a set number of times. The most powerful strategy of those mentioned above in terms of fewest nodes explored is Strong-Branching. Here for each branching candidate the relaxations of both child nodes are solved and then based on this a branching score is calculated via a weighted average. Finally one branches on the variable with the highest score computed. Since in Strong-Branching one has to solve each of the child node relaxations this is computationally very expensive, thus Strong-Branching is often only used for the first few iterations and then an alternative strategy is used. This is justified by the fact that the earlier one cuts branches the fewer nodes are explored. Another approach to make Strong-Branching less expensive involves approximating the nonlinear functions in the child nodes by e.g. quadratic functions Bonami et al. [2013]. Currently Boscia
[Comment to myself: Highlight importance of Lower and Upper bound improvement while writing this]

5.1 Pseudocost Branching in the context of Boscia

As prevously mentioned the pseudocost branching strategy requires that one first follows an alternative strategy until the pseudocosts have stabilized. In the case of this thesis we first choose to branch on the most fractional branching candidate until the pseudocosts have stabilized. We shall denote the number of times each branching candidate has to be branched on until the strategy is switched with $s \in \mathbb{N}$. The choices investigated are $s \in \{$enter numbers$\}$. As Boscia uses Frank-Wolfe methods to (approximately) solve nodes we obtain the local Frank-Wolfe dual gap at each of the nodes visited. Since the Frank-Wolfe dual gap upper bounds the optimality gap, we can make use of it to estimate the improvement achieved by branching. For this consider a parent node and its down- and up-branch child nodes with Frank-Wolfe dual gaps g, g_d, g_u, respectively. Let $j \in J$ be the branching index chosen via most fractional branching to obtain said child nodes. [Question: What do we do if not all of the child nodes are feasible? Is there another case that is problematic?] Then we can define the pseudocosts as $P_j^d := \frac{g_d - g}{x_j - \lfloor x_j \rfloor}$ and $P_j^u := \frac{g_u - g}{\lceil x_j \rceil - x_j}$ the first time j is the branching index. Subsequently if we have already branched on j for a total of m-times, we use the following rule to update the pseudocosts and keep their average

$$P_j^u = \frac{1}{m+1} \cdot (m \cdot P_j^u + \frac{g_u - g}{\lceil x_j \rceil - x_j}).$$

The analogous rule is followed for the down branch. Note that it is a well known result from statistics that the pair (m, P_j^u) is a sufficient statistic for the average. The aim of dividing by $\lceil x_j \rceil - x_j$ or $x_j - \lfloor x_j \rfloor$ is to determine the per unit increase of Frank-Wolfe dual gap changes. Motivated by this for $\mu \in [0,1]$ and each $i \in J$ we compute the so called branching score

$$s_i := (1-\mu)\min\{P_i^d \cdot (x_j - \lfloor x_j \rfloor), P_i^u \cdot (\lceil x_j \rceil - x_j)\} + \mu \max\{P_i^d \cdot (x_j - \lfloor x_j \rfloor), P_i^u \cdot (\lceil x_j \rceil - x_j)\}.$$

The choice of branching once pseudocosts have stabilized is then made by

$$j = \arg\max_{i \in J}\{s_i\},$$

an approach already followed in Bonami et al. [2013]. There however they used the difference in objective function value at the optimal solutions of continuous relaxations of parent and child nodes instead of the difference in Frank-Wolfe dual gap. Also they did not not use a Frank-Wolfe based method to solve nodes and in the context of Boscia at each node a mixed-integer problem is (approximately) solved. [Question: Do I need to highlight/ describe the difference more, especially why they can do that and we can not?] In Bonami et al. [2013] they experimented with an adaptive μ where it was chosen to be 0.7 before an integer feasible solution has been found and then lowering it to 0.1 henceforth which is a setting also used in other solvers. [Question: Does this setting make sense for Boscia?("optimizing over the integer hull at each node automatically makes all vertices computed across FW iterations are feasible for the original problem" Hendrych et al. [2022])– I´m interpreting this in the sense that we get an integer feasible solution right at the start, i.e. when solving the first node] For our experimental results in Section TBD we test the performance of $\mu \in \{\text{enter numbers}\}$. [Note to myself: The parameter μ and its role should be described in more detail, i.e. low and high μ represent what?]

6 Optimality Certificates for Frank-Wolfe algorithms combined with Branch-and-Bound

For the definitions and statements in this section we follow the terminology and notation of Braun et al. [2022] which gives an in depth overview of Frank-Wolfe methods.

A differentiable function $f : \mathcal{X} \to \mathbb{R}$ is called μ-strongly convex if

$$f(y) - f(x) \geq \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2, \ \forall x, y \in \mathcal{X}.$$

It is called L-smooth if

$$f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2, \ \forall x, y \in \mathcal{X}.$$

One measure commonly used to evaluate the quality of a solution x in the context of Frank-Wolfe algorithms is known as the primal gap

$$h(x) := \max_{y \in \mathcal{X}} f(x) - (y) = f(x) - f(x^*). \tag{1}$$

Here and throughout this thesis $x^* \in \mathcal{X}$ refers to an optimal solution. Note that problems can have more than one optimal solution and any such solution is unknown until the problem has been solved. Thus instead one considers the Frank-Wolfe gap

$$g(x) := \max_{v \in \mathcal{X}} \langle \nabla f(x), x - v \rangle$$

as it is an upper bound on the primal gap in the case of convexity. Note that $\langle \nabla f(x^*), x - x^* \rangle$ is what Braun et al. [2022] call the dual gap. These two gaps are of great importance for differentiable convex functions $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a convex set, as it holds that

$$x^* \in \mathcal{X} \text{ is optimal} \iff \langle \nabla f(x^*), x^* - x \rangle \leq 0, \ \forall x \in \mathcal{X}.$$

The backward direction of this equivalency is obvious as

$$0 \leq h(x^*) \leq \langle \nabla f(x^*), x^* - x \rangle \leq 0$$

implies that $h(x^*) = 0$. The other direction follows by assuming the existence of $x \in \mathcal{X}$ such that $\langle \nabla f(x^*), x^* - x \rangle > 0$. Now let $v = x^* - x$, then by convexity $\lambda x + (1-\lambda)x^* = x^* - \lambda v \in \mathcal{X}$. Lastly using the fact that

$$\lim_{\lambda \to 0} \frac{f(x^* - \lambda v) - f(x^*)}{\lambda} = \langle \nabla f(x^*), v \rangle > 0,$$

we know that for sufficiently small λ it holds that $f(x^*) - f(x^* - \lambda v) > 0$, i.e. x^* can not be optimal. Thus we have found a contradiction and proven the statement.
It follows from the definition of the Frank-Wolfe gap that

$$g(x) = 0 \iff x = \arg\min_{y \in \mathcal{X}} f(y).$$

for any $x \in \mathcal{X}$.
To summarize the above the Frank-Wolfe gap in the case of differentiable convex functions is not only an upper bound on the primal gap but also an optimality criterion.

7 Empirical results

7.1 Computational experiments

For our computational experiments we have modified the existing branch-and-bound framework of Boscia.jl Hendrych et al. [2022] and added a pseudocost branching option. Boscia.jl is an open-source Julia package implemented in Bonobo.jl and it makes use the Frankwolfe.jl framework Besançon et al. [2022] in order to solve each of the node subproblems encountered during the branch-and-bound process. For our experiments we have chosen to use (TBD SOLVER) as MIP solver. All experiments were run on [Comment to myself: write down the hardware specifics once experimental setup has been completed]. The package versions used are [TBD]. The test problems used for carrying out experiments are all available from the Boscia.jl package repository, with only the branching strategy modified to use different branching strategies.

7.2 KPI

For our experiments we aim to investigate how the choice of pseudocost branching with the settings described in Section TBD as the branching strategy affects the performance of Boscia in comparison to the branching strategies discussed in Hendrych et al. [2022] which were Most-Infeasible, Strong-Branching and Hybrid. In Hybrid, Strong-Branching is performed until depth $k \in \mathbb{N}$ of the branch-and-bound tree has been reached, then the strategy is changed to Most-Infeasible. In the following we list which KPI we choose to investigate and why we choose them.

The first and probably most important indicator of how well the algorithm performs is the number of solved instances. We shall investigate this by comparing how many problems are solved to optimality for each of the branching strategies and also within what time these problems are solved.

The next indicator which is of great importance is the number of nodes visited in the branch-and-bound process. As already mentioned in previous sections the number of nodes visited represents the number of sub problems solved. For this we shall compare how many nodes are visited for each strategy and how the lower and upper bounds evolve, i.e. what the convergence rate is.

To highlight results We furthermore investigate the proportion of problems where one method performs at least x times worse than the best in terms of nodes/ time.

7.3 Results

8 Conclusion

References

Ravindran Kannan and Clyde L Monma. On the computational complexity of integer programming problems. In *Optimization and Operations Research: Proceedings of a Workshop Held at the University of Bonn, October 2–8, 1977*, pages 161–172. Springer, 1978.

Robert C Jeroslow. There cannot be any algorithm for integer programming with quadratic constraints. *Operations Research*, 21(1):221–224, 1973.

Jon Lee and Sven Leyffer. *Mixed integer nonlinear programming*, volume 154. Springer Science & Business Media, 2011.

Hasan Manzour, Simge Küçükyavuz, Hao-Hsiang Wu, and Ali Shojaie. Integer programming for learning directed acyclic graphs from continuous data. *INFORMS journal on optimization*, 3(1):46–73, 2021.

Deborah Hendrych, Hannah Troppens, Mathieu Besançon, and Sebastian Pokutta. Convex integer optimization with frank-wolfe methods. *arXiv preprint arXiv:2208.11010*, 2022.

Mohit Tawarmalani and Nikolaos V Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical programming*, 99(3):563–591, 2004.

Pierre Bonami, Jon Lee, Sven Leyffer, and Andreas Wächter. On branching rules for convex mixed-integer nonlinear optimization. *Journal of Experimental Algorithmics (JEA)*, 18:2–1, 2013.

Ashutosh Mahajan, Sven Leyffer, Jeff Linderoth, James Luedtke, and Todd Munson. Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation*, 13:301–338, 2021.

Ailsa H Land and Alison G Doig. *An automatic method for solving discrete programming problems*. Springer, 2010.

Pierre Bonami, Mustafa Kilinç, and Jeff Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In *Mixed integer nonlinear programming*, pages 1–39. Springer, 2011.

Omprakash K Gupta and Arunachalam Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management science*, 31(12):1533–1546, 1985.

Gábor Braun, Alejandro Carderera, Cyrille W Combettes, Hamed Hassani, Amin Karbasi, Aryan Mokhtari, and Sebastian Pokutta. Conditional gradient methods. *arXiv preprint arXiv:2211.14103*, 2022.

Mathieu Besançon, Alejandro Carderera, and Sebastian Pokutta. Frankwolfe. jl: A high-performance and flexible toolbox for frank–wolfe algorithms and conditional gradients. *INFORMS Journal on Computing*, 34(5):2611–2620, 2022.

9 Appendix