Juniper: An Open-Source Nonlinear Branch-and-Bound Solver in Julia

Ole Kröger(✉), Carleton Coffrin, Hassan Hijazi, and Harsha Nagarajan

Los Alamos National Laboratory, Los Alamos, NM, USA
o.kroeger@wikunia.de

Abstract. Nonconvex mixed-integer nonlinear programs (MINLPs) represent a challenging class of optimization problems that often arise in engineering and scientific applications. Because of nonconvexities, these programs are typically solved with global optimization algorithms, which have limited scalability. However, nonlinear branch-and-bound has recently been shown to be an effective heuristic for quickly finding high-quality solutions to large-scale nonconvex MINLPs, such as those arising in infrastructure network optimization. This work proposes Juniper, a Julia-based open-source solver for nonlinear branch-and-bound. Leveraging the high-level Julia programming language makes it easy to modify Juniper's algorithm and explore extensions, such as branching heuristics, feasibility pumps, and parallelization. Detailed numerical experiments demonstrate that the initial release of Juniper is comparable with other nonlinear branch-and-bound solvers, such as Bonmin, Minotaur, and Knitro, illustrating that Juniper provides a strong foundation for further exploration in utilizing nonlinear branch-and-bound algorithms as heuristics for nonconvex MINLPs.

1 Introduction

Many of the optimization problems arising in engineering and scientific disciplines combine both nonlinear equations and discrete decision variables. Notable examples include the blending/pooling problem [1,2] and the design and operation of power networks [3–5] and natural gas networks [6]. All of these problems fall into the class of mixed-integer nonlinear programs (MINLPs), namely,

$$\begin{aligned} &\text{minimize: } f(x,y) \\ &\text{s.t.} \\ &g_c(x,y) \leq 0 \quad \forall c \in \mathcal{C} \\ &x \in \mathbb{R}^m, y \in \mathbb{Z}^n \end{aligned} \tag{MINLP}$$

where f and g are twice continuously differentiable functions and x and y represent real and discrete valued decision variables, respectively [7]. Combining nonlinear functions with discrete decision variables makes MINLPs a broad and challenging class of mathematical programs to solve in practice. To address this

© Springer International Publishing AG, part of Springer Nature 2018
W.-J. van Hoeve (Ed.): CPAIOR 2018, LNCS 10848, pp. 377–386, 2018.
https://doi.org/10.1007/978-3-319-93031-2_27

challenge, algorithms have been designed for special subclasses of MINLPs, such as when f and g are convex functions [8,9] or when f and g are nonconvex quadratic functions [10,11]. For generic nonconvex functions, global optimization algorithms [12–15] are required to solve MINLPs with a proof of optimality. However, the scalability of such algorithms is limited and remains an active area of research. Although global optimization algorithms have been widely successful at solving industrial MINLPs with a few hundred variables, their limited scalability precludes application to larger real-world problems featuring thousands of variables and constraints, such as AC optimal transmission switching [16].

One approach to addressing the challenge of solving large-scale industrial MINLPs is to develop heuristics that attempt to quickly find high-quality feasible solutions without guarantees of global optimality. To that end, it has been recently observed that nonlinear branch-and-bound (NLBB) algorithms can be effective heuristics for the nonconvex MINLPs arising in infrastructure systems [4–6] and that they present a promising avenue for solving such problems on real-world scales. To the best of our knowledge, Bonmin and Minotaur are the only open-source solvers that implement NLBB for the most general case of MINLP, which includes nonlinear expressions featuring transcendental functions. Both Bonmin and Minotaur provide optimized high-performance C++ implementations of NLBB with a focus on convex MINLPs.

The core contribution of this work is Juniper, a minimalist implementation of NLBB that is designed for rapid exploration of novel NLBB algorithms. Leveraging the high-level Julia programming language makes it easy to modify Juniper's algorithm and explore extensions, such as branching heuristics, feasibility pumps, and parallelization. Furthermore, the solver abstraction layer provided by JuMP [17] makes it trivial to change the solvers used internally by Juniper's NLBB algorithm. Detailed numerical experiments on 300 challenging MINLPs are conducted to validate Juniper's implementation. The experiments demonstrate that the initial release of Juniper has comparable performance to other established NLBB solvers, such as Bonmin, Minotaur, and Knitro, and that Juniper finds high-quality solutions to problems that are challenging for global optimization solvers, such as Couenne and Scip. These results illustrate that Juniper's minimalist implementation provides a strong foundation for further exploration of NLBB algorithms for nonconvex MINLPs.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the NLBB algorithms. Section 3 introduces the Juniper NLBB solver. The experimental validation is conducted in Sect. 4, and Sect. 5 concludes the paper.

2 The Core Components of Nonlinear Branch-and-Bound

To provide context for Juniper's implementation, we begin by reviewing the core components of an NLBB algorithm. NLBB is a natural extension of the well-known branch-and-bound algorithm for mixed-integer linear programs (MIPs) to MINLPs. The algorithm implicitly represents all possible discrete variable

assignments in a MINLP by a decision tree that is exponential in size. The algorithm then searches through this implicit tree (i.e., *branching*), looking for the best assignment of the discrete variables and keeping track of the best feasible solution found thus far, the so-called incumbent solution. At each node in the tree, the partial assignment of discrete variables is fixed and the remaining discrete variables are relaxed to continuous variables, resulting in a nonlinear program (NLP) that can be solved using an established solver, such as Ipopt [18]. If the solution to this NLP is globally optimal, then it provides a lower bound to the MINLP's objective function, $f(x, y)$. Furthermore, if this NLP *bound* is worse than the best solution found thus far, then the NLP relaxation proves that the children of the given node can be ignored. If this algorithm is run to completion, it will provide the globally optimal solution to the MINLP. However, if the MINLP includes nonconvex constraints, the NLP solver provides only local optimality guarantees, and the NLBB algorithm will be only a heuristic for solving the MINLP. The key to designing this kind of NLBB algorithm is to develop generic strategies that find feasible solutions quickly and direct the tree search toward higher-quality solutions. We now briefly review some of the core approaches to achieve these goals.

Branching Strategy: In each node of the search tree, the branching strategy defines the order in which the children (i.e., variable/value pairs) of that node should be explored. The typical branching strategies are (1) *most infeasible*, which branches on the variables that are farthest from an integer value in the NLP relaxation; (2) *pseudo cost*, which tracks how each variable affects the objective function during search and then prioritizes variables with the best historical record of improving the objective value [19]; (3) *strong*, which tests all branching options by brute-force enumeration and then takes the branch with the most promising NLP relaxation [20]; and (4) *reliability*, which uses a threshold parameter to limit strong branching to a specified amount of times for each variable [21].

Traversal Strategy: At any point during the tree search there are a number of *open* nodes that have branches that remain to be explored. The traversal strategy determines how the next node will be selected for exploration. The typical traversal strategies include (1) *depth first*, which explores the most recent open node first; and (2) *best first*, which explores the open node with the best NLP bound first. The advantage of depth first search is that it only requires a memory overhead that is linear in the number of discrete variables. In contrast, best first search results in the smallest number of nodes explored but can consume an exponential amount of memory.

Incumbent Heuristics: In some classes of MINLPs, finding an initial feasible solution can be incredibly difficult, and the NLBB algorithm can spend a prohibitive amount of time in unfruitful parts of the search tree. Running dedicated feasiblity heuristics at the root of the search tree is often effective in mitigating this issue. The most popular such heuristic is the *feasibility pump*, which is a

fixed-point algorithm that alternates between solving an NLP relaxation of the MINLP for assigning the continuous variables and solving a MIP projection of the NLP solution for assigning the discrete variables [22,23].

Code Block 1 Installing and Solving a MINLP with JuMP and Juniper

```
Pkg.add("JuMP"); Pkg.add("Ipopt"); Pkg.add("Cbc"); Pkg.add("Juniper")
using JuMP, Ipopt, Cbc, Juniper

ipopt = IpoptSolver(print_level=0); cbc = CbcSolver()
m = Model(solver=JuniperSolver(ipopt, mip_solver=cbc))

v = [10,20,12,23,42]; w = [12,45,12,22,21]
@variable(m, 0 <= x[1:5] <= 10, Int)

@objective(m, Max, dot(v,x))
@constraint(m, sum(x[i] for i=1:5) <= 6)
@NLconstraint(m, sum(w[i]*x[i]^2 for i=1:5) <= 300)

status = solve(m); getvalue(x)
```

Relaxation Restarts: In traditional branch-and-bound algorithms, the continuous relaxation is convex and guaranteed to converge to the global optimum or prove that the relaxation is infeasible. However, in the case of nonconvex MINLPs, a local NLP solver provides no such guarantees. Thus, it can be advantageous to restart the NLP solver from a variety of different starting points in the hopes of improving the lower bound or finding a feasible solution [8,24].

3 The Juniper Solver

The motivation for developing Juniper [25] is to provide relatively simple and compact implementation of NLBB so that a wide variety of algorithmic modifications can be explored in the pursuit of developing novel heuristics for nonconvex MINLPs. To that end, Julia is a natural choice for the implementation for two reasons: (1) Julia provides high-level programming, similar to Matlab and Python, that is preferable for rapid prototyping; and (2) the mathematical programming package JuMP [17] provides an AMPL-like modeling layer, which makes it easy to state MINLP problems, and a solver abstraction layer, which makes a wide range of NLP and MIP solvers available for use in Juniper. To demonstrate these properties, Code Block 1 provides a simple Julia v0.6 example illustrating the software installation, stating a JuMP v0.18 MINLP model, and solving it with Juniper. In this example, the NLP solver Ipopt is used for solving the continuous relaxation subproblems and the MIP solver Cbc is used in the feasibility pump heuristic.

From Code Block 1, it is clear how Juniper can be reconfigured to use different NLP and MIP solvers at runtime. As is typical for solvers, Juniper also features a wide variety of parameters for augmenting the NLBB algorithm. These

include options for selecting the branching strategy, tree traversal strategy, feasibility pump, parallelized tree search, and numerical tolerances, among others. A complete list of algorithm parameters is available in Juniper's documentation. After rigorous testing on hundreds of MINLP problems, the following default settings were identified: Strong branching is performed at the root node, and pseudo-cost branching is used afterward. Typically, complete strong branching is conducted; however, if the NLP runtime combined with the number of branches will require more than 100 s, the number of branches explored is reduced to meet this time limit. If the NLP relaxation fails in the root node, it will be restarted up to three times. Best first search is used for exploring the decision tree, and the runtime of the feasibility pump is limited to 60 s.

4 Experimental Evaluation

This section conducts a detailed numerical study of Juniper's performance under a variety of configurations and compares its performance to established MINLP solvers. Five points of comparison were considered for solving MINLPs. Bonmin v1.8 [8], Minotaur v0.2 [26], and Knitro v10.3 [24] were included as alternative NLBB implementations, whereas Couenne v0.5 [13] and Scip v5.0 [10,27] were used for a global optimization reference. All of the open-source solvers utilize Ipopt v3.12 [18] compiled with HSL [28] for solving NLP subproblems and their respective default LP and MIP solvers. All of the solvers, except Juniper, were accessed through their AMPL NL file interface. All of the computations were conducted on a cluster of HPE ProLiant XL170r servers featuring two Intel 2.10 GHz 16 Core CPUs and 128 GB of memory. All solvers were configured with an optimality gap of 0.01% and a runtime limit of 1 h. It is important to note that Julia's JIT takes around 3–10 s the first time Juniper is run; this time is not reflected in the runtime results.

MINLP Problem Selection: The first step in performing this evaluation is to select an appropriate collection of MINLP test problems. We began with 1500 MINLP problems from MINLPLIB2 [29], which are available in Julia via the MINLPLibJuMP package [30]. Second, all of the problems with no discrete variables or fewer than ten constraints were eliminated, resulting in about 700 problems that focus on the constrained mixed-integer problems that Juniper is intended for. Through a preliminary study, it was observed that more than half of these cases are solved to global optimality or are proven to be infeasible by Scip or Couenne in less than 60 s, suggesting that these are relatively easy cases for state-of-the-art global optimization methods and that they are not of interest to this work. The final collection of test problems consists of 298 MINLPs that are challenging for both NLBB and global optimization solvers.

Solver Comparison: The first and foremost goal is to demonstrate that Juniper has comparable computational performance to Bonmin and Minotaur. Figure 1 (top) provides an overview of the runtime for each solver to

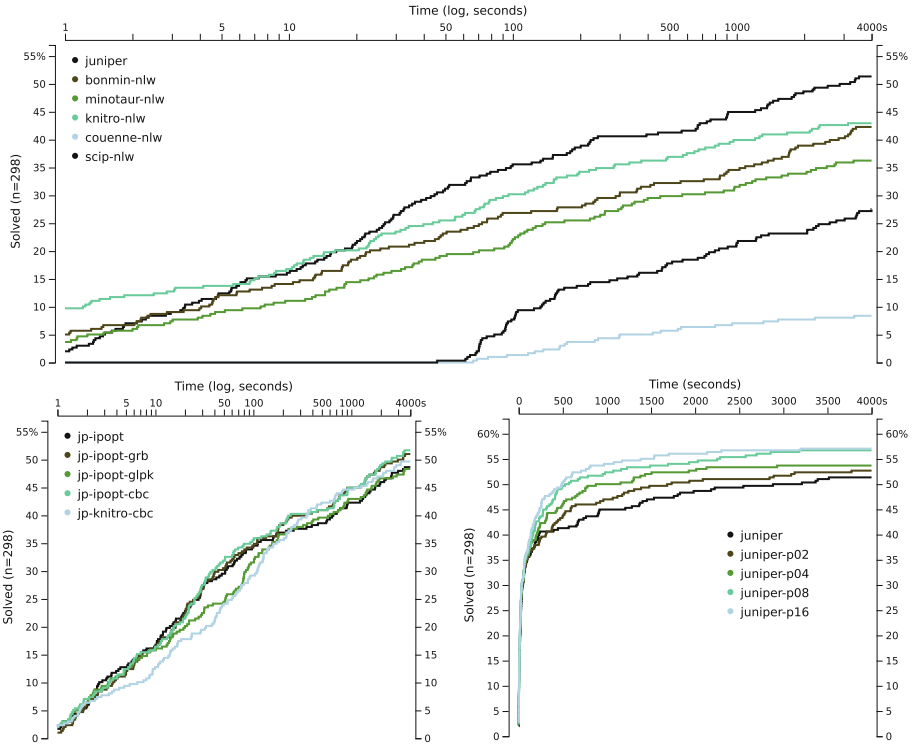

Fig. 1. Runtime profiles on 298 instances for all different solvers (top) using different MIP solvers (bottom left) and parallelized tree search (bottom right).

complete its tree search procedure. This figure highlights two key points: (1) Juniper is slower for small models that can be solved in less than 30 s; however, it consistently solves more models after 30 s; and (2) the search completes in no more than 50% of the cases considered, demonstrating that the selected MINLP instances present challenging tree search problems for both the NLBB and global optimization solvers.

Table 1 provides further details on the performance of each solver, including problem sizes, objective gaps from the best-known solution, and runtime results. The table begins with summary statistics. The first row shows the number of feasible solutions found by each solver as well as the number of test cases where the runtime limit was reached. The following three rows show the average optimality gaps and runtime for each solver. The first average is for all instances where the specific solver was able to find a feasible solution. The second average is for instances where all six solvers were able to find feasible solutions. The third average is for instances where all four NLBB solvers were able to find feasible solutions. These summary results indicate two key points: (1) Juniper is one of the most robust solvers (only Scip had a higher feasible solution count); and (2) for cases where all NLBB algorithms have feasible solutions, Bonmin has the

Table 1. Quality and runtime results for various instances

						juniper	bon	minot	knitro	coue	scip	juniper	bon	minot	knitro	coue	scip
Feasible instances / Time limit reached						228	196	193	187	183	257	114	105	134	144	247	215
						Gap (%)						Runtime (seconds)					
Average solver feasible						324.51	6.34	25.52	1365.77	395.68	1e+04	1425	1672	1789	1226	3285	2853
Average all solvers feasible (n=85)						21.43	0.86	3.64	24.57	16.19	2e+04	929	834	1558	1002	2788	2661
Average all NLBB solvers feasible (n=113)						18.48	0.94	11.19	20.69	-	-	918	845	1420	1112	-	-
Instance	$\|V\|$	$\|C\|$	$\|I\|$	$\|NC\|$	best obj.	juniper	bon	minot	knitro	coue	scip	juniper	bon	minot	knitro	coue	scip
oil2	937	927	2	284	-0.73	**0.00**	**0.00**	**0.00**	**0.00**	-	-	7	3	**2**	8	-	-
bchoco08	169	191	9	79	0.98	-	-	**0.00**	0.71	-	-	-	-	**46**	153	-	-
squfl015-080persp	2416	2481	15	1200	402.48	**0.00**	**0.00**	4.61	**0.00**	0.93	**0.00**	27	**23**	259	**23**	T.L	94
transswitch0014p	139	278	20	121	8082.58	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	-	2	14	**< 1**	32	T.L	-
squfl030-100persp	6031	6101	30	3000	363.09	**0.00**	**0.00**	43.71	**0.00**	137.24	**0.00**	683	919	T.L	**5**	T.L	489
FLay05M	63	66	40	5	64.50	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	1283	561	1213	1679	117	**71**
ndcc13	631	255	42	42	84.63	5.80	**0.00**	**0.00**	-	-	3.30	T.L	1721	**1172**	-	-	T.L
CLay0205H	261	366	50	40	8092.50	**0.00**	**0.00**	891.18	1.61	-	**0.00**	2548	1091	T.L	T.L	-	**71**
procurement1mot	785	750	60	1	291.54	**0.00**	**0.00**	**0.00**	**0.00**	8.19	1.95	445	**204**	385	2345	T.L	T.L
fo9_ar2_1	181	436	72	18	32.62	-	-	115.90	-	56.21	**0.00**	-	-	T.L	-	T.L	**3454**
crudeoil_pooling_ct1	311	566	80	37	210537.49	43.82	**0.00**	3.06	-	2.34	**0.00**	**T.L**	**T.L**	**T.L**	-	**T.L**	**T.L**
multiplants_mtg1a	194	257	93	28	391.61	**0.00**	**0.00**	3.08	-	2.46	**0.00**	909	**901**	T.L	-	T.L	997
multiplants_mtg2	230	307	112	37	7099.19	**0.00**	**0.00**	0.14	-	**0.00**	27.54	3390	**1877**	T.L	-	T.L	T.L
crudeoil_pooling_ct3	730	1199	128	211	287000.00	32.69	-	-	-	16.82	**0.00**	**T.L**	-	-	-	**T.L**	**T.L**
pooling_epa3	1105	1718	150	274	-14948.12	**0.00**	-	-	0.17	-	-	**T.L**	-	-	**T.L**	-	-
transswitch0118r	1240	1764	179	1311	129457.85	0.45	-	-	**0.00**	-	-	**181**	-	-	491	-	-
multiplants_stg1	415	262	198	34	355.09	24.36	**0.00**	1.03	-	-	-	**T.L**	**T.L**	**T.L**	-	-	-
qap	226	31	225	1	389718.00	0.50	0.38	6.79	**0.00**	10.94	6.53	1404	428	T.L	**137**	T.L	T.L
csched2	401	138	308	1	-166102.00	-	**0.00**	4.16	-	14.05	6.69	-	**18**	T.L	-	T.L	T.L
edgecross22-048	463	6161	462	1	84.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**12**	121	98	23	162	2201
crudeoil_pooling_dt1	3713	5779	916	570	209585.27	368.25	-	4.59	-	-	**0.00**	T.L	-	T.L	-	-	**3429**

highest solution quality, and Juniper, Minotaur, and Knitro have similar quality, on average. The remaining rows in the table provide a representative sample of the 298 problems considered. The first five columns describe each problem by name, number of variables V, number of constraints C, number of discrete variables I, and number of nonlinear constraints NC. The general trends are summarized as follows: (1) there is a great diversity among which solver is the best on the MINLP instances considered; and (2) in most cases, the solutions found by the NLBB solvers tie or improve those found by the global solvers; however, there are a few notable cases where global solvers find the best solutions. Overall, these results indicate that Juniper in its default configuration is comparable with the NLBB solvers considered here.

Subsolver Selection: One of the key features of Juniper is that it can use different solvers for the NLP relaxation and for the MIP aspect of the feasibility pump heuristic. Figure 1 (bottom left) shows a performance profile for a number of subsolver variants of Juniper, both with and without a MIP solver (i.e., Glpk [31], Cbc [32], Gurobi [33]), as indicated by Juniper-ipopt. Juniper-Knitro-Cbc shows the result of using Knitro as the NLP solver instead of Ipopt. The runtime difference between using the feasibility pump and using no heuristic is quite notable in some cases; however, given sufficient time, Juniper solves a similar number of cases even without a feasibility pump. To our surprise, there was little difference in using Cbc as the MIP solver compared to using Gurobi, suggesting that Cbc is a suitable default solver. We also observed that Glpk is not a suitable choice because it was typically unable to terminate in less than 60 s, which is the preferred feasibility pump time limit.

Parallel Tree Search: A key feature of Julia is native and easy-to-use support for parallel processing. Juniper leverages this capability to implement a parallel tree search algorithm. Figure 1 (bottom right) illustrates the benefits from this simple parallelization of the algorithm, where the first thread orchestrates the computation and all additional worker threads process open nodes in the search tree. The figure indicates that having two worker threads (instead of using the sequential algorithm) is about 1.7 times faster and having four worker threads is about 3.3 times faster. The difference between eight and sixteen worker threads is not that notable (both increase speed by around 5.8 times).

5 Conclusion

This work has highlighted the potential for leveraging NLBB algorithms as heuristics for solving challenging nonconvex MINLPs. To assist in the design of such algorithms, a new Julia-based solver, Juniper, is proposed as the base implementation for future exploration in this area. A detailed experimental study demonstrated that, despite its minimalist implementation, Juniper performs comparably to established NLBB solvers on the class of MINLPs for which it was designed. We hope that Juniper will provide the community with a valuable reference implementation for collaborative open-source research on heuristics for large-scale nonconvex MINLPs.

References

1. Audet, C., Brimberg, J., Hansen, P., Digabel, S.L., Mladenovic, N.: Pooling problem: alternate formulations and solution methods. Manage. Sci. **50**(6), 761–776 (2004)
2. Trespalacios, F., Kolodziej, S.P., Furman, K.C., Sawaya, N.W.: Multiperiod blend scheduling problem. Cyber Infrastructure for MINLP, June 2013. www.minlp.org/library/problem/index.php?i=168
3. Jabr, R.A.: Optimization of AC transmission system planning. IEEE Trans. Power Syst. **28**(3), 2779–2787 (2013)
4. Coffrin, C., Hijazi, H.L., Lehmann, K., Hentenryck, P.V.: Primal and dual bounds for optimal transmission switching. In: 2014 Power Systems Computation Conference, pp. 1–8, August 2014
5. Coffrin, C., Hijazi, H.L.: Heuristic MINLP for optimal power flow problems. In: 2014 IEEE Power & Energy Society General Meetings (PES) Application of Modern Heuristic Optimization Algorithms for Solving Optimal Power Flow Problems Competition (2014)
6. Borraz-Sanchez, C., Bent, R., Backhaus, S., Hijazi, H., Hentenryck, P.V.: Convex relaxations for gas expansion planning. INFORMS J. Comput. **28**(4), 645–656 (2016)
7. Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., Mahajan, A.: Mixed-integer nonlinear optimization. Acta Numer. **22**, 1–131 (2013)
8. Bonami, P., Biegler, L.T., Conn, A.R., Cornuéjols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optim. **5**(2), 186–204 (2008). In memory of George B. Dantzig
9. Lubin, M., Yamangil, E., Bent, R., Vielma, J.P.: Extended formulations in mixed-integer convex programming. In: Louveaux, Q., Skutella, M. (eds.) IPCO 2016. LNCS, vol. 9682, pp. 102–113. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33461-5_9
10. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. **1**(1), 1–41 (2009)
11. Bonami, P., Gunluk, O., Linderoth, J.: Solving box-constrained nonconvex quadratic programs (2016). http://www.optimization-online.org/DB_HTML/2016/06/5488.html
12. Ryoo, H., Sahinidis, N.: A branch-and-reduce approach to global optimization. J. Global Optim. **8**(2), 107–138 (1996)
13. Belotti, P.: Couenne: user manual (2009). https://projects.coin-or.org/Couenne/. Accessed 04 Oct 2015
14. Nagarajan, H., Lu, M., Wang, S., Bent, R., Sundar, K.: An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. arXiv preprint arXiv:1707.02514 (2017)
15. Nagarajan, H., Lu, M., Yamangil, E., Bent, R.: Tightening McCormick relaxations for nonlinear programs via dynamic multivariate partitioning. In: Rueher, M. (ed.) CP 2016. LNCS, vol. 9892, pp. 369–387. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44953-1_24
16. Sahraei-Ardakani, M., Korad, A., Hedman, K.W., Lipka, P., Oren, S.: Performance of AC and DC based transmission switching heuristics on a large-scale polish system. In: 2014 IEEE PES General Meeting—Conference Exposition, pp. 1–5, July 2014

17. Dunning, I., Huchette, J., Lubin, M.: JuMP: a modeling language for mathematical optimization. SIAM Rev. **59**(2), 295–320 (2017)
18. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
19. Benichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribiere, G., Vincent, O.: Experiments in mixed-integer linear programming. Math. Program. **1**(1), 76–94 (1971)
20. Applegate, D., Bixby, R., Chvatal, V., Cook, B.: Finding cuts in the TSP (a preliminary report). Technical report (1995)
21. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**(1), 42–54 (2005)
22. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. **104**(1), 91–104 (2005)
23. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: A storm of feasibility pumps for nonconvex MINLP. Math. Program. **136**(2), 375–402 (2012)
24. Byrd, R.H., Nocedal, J., Waltz, R.A.: KNITRO: an integrated package for nonlinear optimization. In: Di Pillo, G., Roma, M. (eds.) Large-Scale Nonlinear Optimization. Nonconvex Optimization and Its Applications, vol. 83, pp. 53–59. Springer, Boston (2006). https://doi.org/10.1007/0-387-30065-1_4
25. Kröger, O., Coffrin, C., Hijazi, H., Nagarajan, H.: Juniper (2017). https://github.com/lanl-ansi/Juniper.jl. Accessed 14 Dec 2017
26. Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., Munson, T.: Minotaur: a mixed-integer nonlinear optimization toolkit (2017)
27. Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Serrano, F., Shinano, Y., Viernickel, J.M., Vigerske, S., Weninger, D., Witt, J.T., Witzig, J.: The SCIP optimization suite 5.0. Technical report 17–61, ZIB, Takustr. 7, 14195 Berlin (2017)
28. Research Councils UK: The HSL mathematical software library. http://www.hsl.rl.ac.uk/. Accessed 30 Oct 2017
29. Vigerske, S.: MINLP Library 2 (2017). http://www.gamsworld.org/minlp/minlplib2/html/. Accessed 17 Dec 2017
30. Wang, S.: MINLPLibJuMP (2017). https://github.com/lanl-ansi/MINLPLibJuMP.jl. Accessed 14 Dec 2017
31. Free Software Foundation Inc.: GNU linear programming kit (2017). https://www.gnu.org/software/glpk/
32. The COIN-OR Foundation: COIN-OR CBC (2017). https://projects.coin-or.org/Cbc
33. Gurobi Optimization Inc.: Gurobi optimizer reference manual (2014). http://www.gurobi.com