Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

BENJAMIN MÜLLER, RENKE KUHLMANN, STEFAN VIGERSKE

On the performance of NLP solvers within global MINLP solvers

This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibilty for the content of this publication is assumed by the authors.

ZIB Report 17-35 (July 2017)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

On the performance of NLP solvers within global MINLP solvers

Benjamin Müller* Renke Kuhlmann† Stefan Vigerske‡

July 14, 2017

Abstract

Solving mixed-integer nonlinear programs (MINLPs) to global optimality efficiently requires fast solvers for continuous sub-problems. These appear in, e.g., primal heuristics, convex relaxations, and bound tightening methods. Two of the best performing algorithms for these sub-problems are Sequential Quadratic Programming (SQP) and Interior Point Methods. In this paper we study the impact of different SQP and Interior Point implementations on important MINLP solver components that solve a sequence of similar NLPs. We use the constraint integer programming framework SCIP for our computational studies.

Keywords. mixed-integer nonlinear programming, interior point, sequential quadratic programming, global optimization

1 Introduction

We consider nonconvex mixed-integer nonlinear programs (MINLPs) of the form

$$\min_{x\in[\ell,u]\subseteq\mathbb{R}^n} \left\{c^\top x \mid g_j(x) \leq 0 \; \forall j \in \mathcal{M}, x_i \in \mathbb{Z} \; \forall i \in \mathcal{I}\right\}, \tag{1}$$

where $c \in \mathbb{R}^n$, $\mathcal{M} := \{1,\dots,m\}$, $\mathcal{N} := \{1,\dots,n\}$, $\mathcal{I} \subseteq \mathcal{N}$, $\ell_i, u_i \in \mathbb{R} \cup \{\pm\infty\}$, $i \in \mathcal{N}$, and $g_j : [\ell, u] \to \mathbb{R}$, $j \in \mathcal{M}$, differentiable. MINLPs have applications in many areas, we refer to [4] for an overview. The state-of-the-art algorithm for solving MINLPs to global ϵ-optimality is spatial branch-and-bound, see, e.g., [5, 8, 9]. Solvers that implement this method typically need to compute local optimal solutions of nonlinear programs (NLPs). For example, primal heuristics [1] may require the solution of an NLP sub-problem of (1) and bounding methods may require the solution of a convex NLP relaxation [10, 13]. Two important solution methods for NLPs are the Inter-Point Method (IPM), which has been shown to be very efficient, and Sequential Quadratic Programming (SQP), which is said to be more robust and has better warm-starting properties than IPM.

The goal of this paper is to investigate the impact of different NLP solvers on the performance of an MINLP solver. For that, we consider the use of a portfolio of NLP solvers to solve a sequence of – sometimes very similar – NLPs as they arise in various components of an MINLP solver. With *dual components* we refer to algorithms that aim to strengthen a relaxation of the problem

*Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, benjamin.mueller@zib.de
†University Bremen, Bibliothekstr. 5, 28195 Bremen, Germany, renke.kuhlmann@math.uni-bremen.de
‡GAMS Software GmbH, c/o Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany, svigerske@gams.com

in each node of the spatial branch-and-bound tree, while with *primal components* we refer to algorithms that aim on finding an improving feasible solution. Naturally, for dual components, finding dual feasible solutions for convex NLPs is important, while for primal components finding a primal feasible solution of an NLP is sufficient, though the NLP might be nonconvex.

1.1 Dual Components

Often, a convex NLP relaxation of (1) is obtained by replacing constraints where $g_j(x)$ is nonconvex over $[\ell, u]$ by a convex relaxation, e.g., by using convex underestimators of $g_j(x)$ [7, 10, 11]. As the tightness of these underestimators depends on the variable bounds, branching decisions in the branch-and-bound tree search can allow to update the convex underestimators and thus improve the bound that the relaxation provides for the corresponding node in the tree.

The dual components that we consider in this paper are, first, the solution of convex nonlinear relaxations to bound the objective function $c^\top x$ in a node in the branch-and-bound tree. Thus, this component solves an NLP in potentially all nodes of the branch-and-bound tree, each being a convex relaxation of (1) when restricted to the variable bounds that are defining the node. Second, we consider Optimization-Based Bounds Tightening (OBBT), where possibly tighter bounds on selected variables are computed by minimizing and maximizing each of them over a convex relaxation of (1). Improved variable bounds can help to tighten the relaxation that is used to bound the optimal value of (1).

1.2 Primal Components

The NLPs that are solved by primal components are often obtained after fixing some or all of the integer variables x_i, $i \in \mathcal{I}$, in (1) to a given value and relaxing the integrality requirement on all non-fixed integer variables. A locally optimal (or at least feasible) solution to this NLP can be used to update the incumbent for the original MINLP, if all non-fixed integer variables take an integral value.

Many different strategies to find a good variable fixing have been developed. For our experiments, we consider an algorithm that uses the solution point computed by any primal heuristic applied to a MILP relaxation of (1) to fix all integer variables in (1) and to provide a starting point for the NLP solver. Additionally, we consider an NLP-diving heuristic, where first all integrality requirements are relaxed and then iteratively the NLP is solved and, based on its solution, additional integer variables are fixed, until either the NLP solution is feasible for the MINLP or the NLP solver fails to find a feasible solution to the NLP. In the latter case, a backtrack strategy may be applied to investigate an alternative for the latest variable fixing decision.

2 Computational Results

We used a development version of SCIP[1] [11] as MINLP solver. This version is based on SCIP 4.0, but next to the already existing interface to the IPM solver Ipopt [12], it includes new interfaces to the IPM and SQP solvers of WORHP [2] and the SQP solver FilterSQP [3]. For our experiments, we have used SCIP with the NLP solvers FilterSQP 20010817, Ipopt 3.12.7, WORHP-IP (IPM algorithm of WORHP 1.10.3), and WORHP-SQP (SQP algorithm of WORHP 1.10.3). Except for FilterSQP, all solvers use MA97 from HSL[2] to solve systems of linear equations. For all

[1] Solving Constraint Integer Programs, `http://scip.zib.de`
[2] Harwell Subroutine Library, `http://www.hsl.rl.ac.uk`

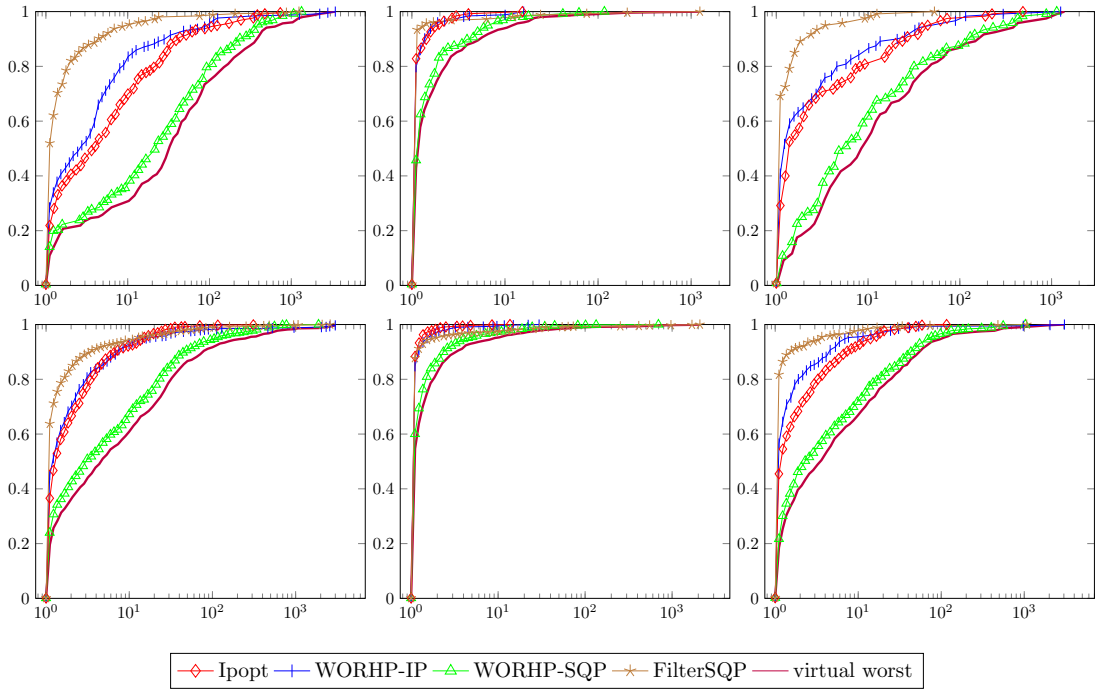

Figure 1: Performance profiles for dual (first row) and primal components (second row). Left: sum of NLP solving times for all NLPs per MINLP instance. Middle: shifted geometric mean of same solving times. Right: sum of solving times for NLPs where at least one solver returned a certificate of (local) infeasibility.

solvers, we disabled scaled termination tolerances and used a feasibility tolerance of 10^{-6}, an optimality tolerance of 10^{-7}, and equal limits on the number of NLP iterations. We used a time limit of one hour for SCIP.

As test set we consider all instances of MINLPLib2[3] (as of 2017/7/10) which can be handled by SCIP. When comparing the NLP solvers on dual components, we additionally discarded instances where SCIP does not detect any convex nonlinear constraint, since the convex relaxations would otherwise be linear (SCIP uses only polyhedral relaxations for nonconvex constraints). This leaves 327 instances. Further, when comparing on primal components, we disregard instances with only continuous variables, i.e., $\mathcal{I} = \emptyset$, since the primal components would not be applied otherwise. This leaves 938 instances.

The experiments were conducted on a cluster of 64bit Intel Xeon X5672 CPUs at 3.2 GHz with 12 MB cache and 48 GB main memory.

2.1 Dual and Primal Components

To ensure that all NLP solvers solve the same sequence of NLPs, we solve each NLP that occurs in the considered components of SCIP independently by all solvers, but pass only the result from the first solver back to SCIP. Table 1 contains aggregated results for the consumed time and the success of each NLP solver. Per instance, we collect the overall time spent in each solver and the

[3]MINLP Library 2, `http://www.gamsworld.org/minlp/minlplib2.html`

	dual components			primal components		
solver	time	nfeas	nopt	time	nfeas	nopt
Ipopt	37.0	663.4	613.4	4.4	65.5	58.5
FilterSQP	7.9	634.3	616.0	2.7	58.4	51.3
WORHP-IP	24.1	610.9	494.6	4.7	52.3	47.3
WORHP-SQP	179.2	568.3	235.9	21.7	57.7	31.7
virtual best	3.9	814.4	766.5	1.0	78.7	73.8

Table 1: Aggregated results for dual and primal components.

number of NLPs where a feasible or locally optimal solution has been found. Next, we compute the shifted geometric mean over all instances with a shift value of one second. To reduce the impact of trivial instances, we disregarded all instances where the sum of NLP solving times was at most one second for the virtual worst, which is the theoretical worst performing solver on each NLP– this leaves 607 instances for the primal and 201 instances for the dual components.

Comparing the running time of the NLP solvers, FilterSQP is on both, the dual and primal components, the fastest solver. Further, it is more than 3.1 times faster than the second fastest solver, WORHP-IP, on the dual components. Interestingly, WORHP-IP performs 34.9% faster than Ipopt on the dual components and 6.4% slower on the primal components. Furthermore, the variability in performance of the NLP solvers is quite large. Choosing the best performing solver for each NLP yields a speed-up of at least a factor of 2.0 compared to FilterSQP.

Regarding the solution quality, Ipopt found more often than all other solvers feasible and local optimal solutions. On dual components, Ipopt found between 4.4% and 14.3% more feasible and up to 61.5% more local optimal solutions than the other solvers. Even though WORHP-SQP finds many feasible points, it frequently fails to converge to a local optimum. Again, the variability with respect to the solution quality is large. Choosing the best NLP solver increases the success rate of finding a feasible solution by 17.7% on average, and finding a local optimal point by around 20.0% compared to Ipopt. This indicates that a dynamic and smart choice between a portfolio of NLP solvers could allow for a considerably better performance than deciding for a single NLP solver in advance.

Figure 1 shows different performance profiles comparing the sum and the shifted geometric mean of NLP solving times per instance. As already observed above, FilterSQP outperforms all solvers when considering the sum of NLP solving times on all NLPs. On the primal components, we see that Ipopt performs more robust than the other solvers because its worst case ratio to the virtual best is bounded by a factor of 100. A considerable part of the good performance of FilterSQP seems to come from NLPs that might be infeasible. The speed-up on instances for which at least one solver provided a certificate of infeasibility is much higher than on all NLPs. This phenomena is more distinct on the dual components than on the primal components.

Due to fixing integer variables heuristically, many NLPs that appear in the primal components turn out to be infeasible. Quickly detecting their infeasibility is important. FilterSQP seems to be the fastest solver on these NLPs, too, but WORHP-IP performs significantly better than Ipopt. This is due to the Penalty-IP approach of WORHP-IP, which is able to converge to infeasible stationary points quickly without the necessity of a separate feasibility restoration phase [6].

In contradiction to the previous results, Ipopt and WORHP-IP perform better than FilterSQP when considering the shifted geometric mean of solving times. The performance profiles in the second column of Figure 1 show that the difference between the solvers is less distinct as when considering the sum of solving times. This can be explained by the reduced impact of outliers in

	all		all optimal		
setting	# solved	time	# solved	nodes	time
SCIP + Ipopt	55	984.0s	41	108312	227.7s
SCIP + FilterSQP	53	0.92%	41	0.93%	0.90%
SCIP + WORHP-IP	50	1.06%	41	1.06%	0.98%
SCIP + WORHP-SQP	49	1.17%	41	0.95%	1.09%

Table 2: Aggregated results for SCIP using different NLP solvers. The entries of the *time* and *nodes* columns are relative to the first row.

the shifted geometric mean. Thus, the superior performance of FilterSQP could be caused by the absence of sometimes expensive fallback strategies, which are implemented by the other solvers. Within a MINLP solver, where not every NLP needs to be solved to optimality, such a "fast fail" strategy seems to be advantageous. This presumption is reinforced by our observation that tuning a solver to find more local optimal points decreased its average performance considerably.

Finally, we want to emphasize that the NLPs that arise within our experiments are typically small. This might be a disadvantage for solvers like Ipopt and WORHP, which are designed to solve large-scale NLPs.

2.2 Overall Performance

The impact of using different NLP solvers in SCIP is summarized in Table 2. For this comparison, we used the selection of 115 instances from MINLPLib2 that is also used in a publicly available MINLP benchmark[4] and set a gap limit of 10^{-3}.

Choosing a different NLP solver has a large impact on the performance and solvability of MINLPs. Table 2 shows that SCIP with Ipopt could solve the largest number of instances. However, SCIP performed fastest when using FilterSQP. On all instances the speed-up is 8% compared to using Ipopt, and on all instances that could be solved by all settings, the speed-up is even larger, namely 10%.

Acknowledgments

This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM).

References

[1] Berthold, T.: Heuristic algorithms in global MINLP solvers. Ph.D. thesis, Technische Universität Berlin (2014)

[2] Büskens, C., Wassel, D.: The ESA NLP solver WORHP. In: G. Fasano, J.D. Pintér (eds.) Modeling and Optimization in Space Engineering, vol. 73, pp. 85–110. Springer New York (2013)

[4] H. Mittelmann MINLP Benchmark, `http://plato.asu.edu/ftp/minlp.html`

[3] Fletcher, R., Leyffer, S.: User manual for filterSQP. Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Scotland (1998)

[4] Grossmann, I.E., Sahinidis, N.V.: Special issue on mixed integer programming and its application to engineering, part I. Optimization and Engineering **3**(4) (2002)

[5] Horst, R., Tuy, H.: Global optimization: Deterministic approaches. Springer (1996)

[6] Kuhlmann, R., Büskens, C.: A primal-dual augmented Lagrangian penalty-interior-point filter line search algorithm. Tech. rep., Universität Bremen (2017)

[7] McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. Mathematical Programming B **10**(1), 147–175 (1976)

[8] Quesada, I., Grossmann, I.E.: A global optimization algorithm for linear fractional and bilinear programs. Journal of Global Optimization **6**, 39–76 (1995)

[9] Ryoo, H.S., Sahinidis, N.V.: Global optimization of nonconvex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering **19**(5), 551–566 (1995)

[10] Ryoo, H.S., Sahinidis, N.V.: A branch-and-reduce approach to global optimization. Journal of Global Optimization **8**(2), 107–138 (1996)

[11] Vigerske, S., Gleixner, A.: SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods & Software (to appear)

[12] Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (2006)

[13] Zamora, J.M., Grossmann, I.E.: A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. Journal of Global Optimization **14**, 217–249 (1999)