Parameter Tuning of Linear Programming Solvers

Nikolaos Ploskas
Department of Electrical & Computer Engineering
University of Western Macedonia
Kozani, Greece
nploskas@uowm.gr

***Abstract*—Linear programming solvers include various options that can be used to control algorithmic aspects and considerably impact the solver performance. As it is obvious, manually finding optimal parameters is a very difficult task and sometimes impossible. For this reason, it is necessary to implement smart techniques that will automate this process. Other works have utilized derivative-free optimization solvers to tune solver parameters. In this work, eight open-source derivative-free optimization solvers are utilized for finding (near) optimal tuning parameters of state-of-the-art linear programming solvers. We investigate how sensitive linear programming solvers are to a parameter tuning process. Extensive computational results are presented on tuning four linear programming solvers (CLP, CPLEX, GUROBI, and XPRESS) over a set of 70 benchmark problems. We find better parameters for all linear programming solvers, achieving a reduction in execution time over their default parameters up to 26%. We conclude that several derivative-free optimization solvers outperform others on finding optimal optimal tuning parameters for linear programming solvers.**

***Index Terms*—Optimization, linear programming, performance, parameter learning**

I. Introduction

The software automatic tuning research area has been recognized as an important research area, especially in the last two decades, and works on finding better values for the parameters of an algorithm or a software package have increased significantly. Various approaches, including optimization algorithms, heuristic methods, and deep learning methods have been applied to tune software packages. The success of these methods is mainly based on the following factors: (i) how sophisticated the method is, (ii) the type of the software used to tune, and (iii) the training library.

Tuning the options of optimization solvers is of great importance since this can considerably impact a solver's performance. Even though there are various performance metrics for assessing the tuning process of an optimization solver, two are the most important ones: (i) decreasing its execution time, and (ii) finding a better solution (sometimes the optimal one) than the optimization solver with the default parameters. Optimization solvers include a variety of parameters that control various algorithmic aspects during the solution process. However, even if only a few parameters exist, it is usually impossible to list all parameter settings because the combinations are too many. Therefore, software developers perform extensive computational experiments on a dataset and set default option values based on these results (and on their experience). However, the selected option values may not be adequate for other unseen instances, especially large ones, and may lead the optimization solvers to poor performance. As a result, various works [1]– [4] have attempted to design and implement a systematic method for finding better option values of optimization solvers.

The following optimization problem can describe the parameter tuning problem:

$$\begin{aligned} \min \quad & f(O, P) \\ \text{s.t.} \quad & O \in R \end{aligned}$$

where f is the metric for assessing the solver performance (the execution time is usually used as the metric), O is the set of the option values (independent of each other), P is the set of the problem instances, and R is the feasible region, i.e., it includes all possible parameter combinations. The options can take any value between their bounds.

It is very difficult to find optimal solutions for this problem since the relationship between the parameters and the performance of the solver is implicit, i.e., there is no algebraic form to express the performance function. A few works (e.g., [4] [5]) have focused on finding a relation between the execution time of an optimization solver and various problem parameters, e.g., number of variables, constraints, and nonzeros, but the results were not very accurate when the training library included instances of various classes and sizes. In addition, some parameters may take discrete values, resulting in a complex and nonsmooth surface of the objective function, implying that derivative information obtained through local approximation may be inaccurate [6]. This problem can be solved by optimization algorithms that do not require explicit functional representations of the objective function or gradients [7]. Therefore, we have selected to use derivative-free optimization (DFO) solvers to speedup the optimization solvers' execution time, and more specifically, linear optimization solvers, by tuning their option settings. We treat the tuning problem as a black-box optimization problem, where the algebraic form of the objective function is not known, and we solve it through performing simulations of different option combinations. Each simulation includes solving the problem with a different set of parameters.

Various works have utilized DFO solvers as tuning strategies [1] [3] [6] [8]. Most of them use DFO solvers that can only handle continuous variables, and thus, they use rounding techniques for integer option parameters. This can lead to suboptimal results. In this work, we utilize seven DFO solvers

that are capable of dealing with both continuous and discrete variables (for a recent review on mixed-integer DFO, see [9]). We have also included in our computational study a DFO solver that can handle only continuous variables.

In addition, all previously tuning methods targeted mixed-integer programming solvers as well as local and global nonlinear programming solvers. The changes on the parameter values on these solvers can lead to great speedups. Since no previous work has been conducted to investigate whether option tuning has a significant impact on continuous linear optimization algorithm, we aim to investigate the applicability of option tuning on continuous linear optimization solvers. More specifically, we tune four linear programming (LP) solvers, namely CLP [10], CPLEX [11], GUROBI [12], and XPRESS [13]. We opted to tune the dual simplex algorithm since it is typically used in branch-and-bound frameworks to solve mixed-integer LP solvers [14]. Thus, tuning the dual simplex algorithm will have a great impact not only for solving continuous LP problems but also for mixed-integer ones. Even though LP algorithms are regarded mature, it will make a huge impact in practical problems even a small reduction on the execution time.

The rest of this paper is structured as follows. Section II includes a brief description of the DFO algorithms and software used, while Section III presents the framework used to automate the option tuning of optimization solvers. In Section IV, numerical results based on 70 LP problems from Kennington [15], Mészáros [16], Mittelmann [17], and Netlib [18] libraries are conducted in order to investigate whether or not DFO algorithms have affect LP solvers' performance. Finally, conclusions are provided in Section V.

II. Derivative-free optimization solvers

DFO algorithms are intended to solve optimization problems where gradient information is unavailable or unreliable. The first DFO algorithms were proposed by Spendley et al. [19] and Nelder and Mead [20]. As a result of the need to solve practical problems, DFO has emerged as an appealing area of study. Many real-world engineering problems involve complex computer simulations, which are being used as a partial substitute for laboratory experiments. Such problems are known as expensive black-box optimization systems since the objective and constraints of the problem cannot be obtained as closed-form equations. On top of the above challenges, real-world optimization problems have multiple optimal solutions, non-smoothness, and discontinuities. Under these circumstances, classical optimization algorithms may perform poorly and DFO algorithms are used in these cases.

The number of studies focused on new DFO algorithms and applications have rapidly increased. Various new algorithms have been developed, including generalized pattern search methods [21], trust-region methods [22], radial basis functions [23], and hybrid algorithms combining various algorithmic approaches [24]. The increase in computational power has also given rise to new parallel implementations of DFO algorithms that can also perform multiple simulation in parallel.

The mixed-integer DFO problem is of interest since there exist many real-world applications that require some or all variables to be discrete, including the software tuning problem [6] [8], the optimization of the circuitry design of heat exchangers [25], and the hydraulic capture community problems [26]. The first DFO algorithm handling discrete variables was proposed by Audet and Dennis [27]. Various new algorithms have been developed, including extensions to the mesh adaptive direct search method [28], line-search methods [29], surrogate models [30], and heuristic approaches [31].

Most recently, DFO methods have been incorporated in software packages and made it easier to use in various applications. Many software packages that implement DFO algorithms have been developed. In this work, we have selected seven open source DFO algorithms, which can handle both continuous and integer variables, for tuning LP solvers. In addition, we also selected MCS, which cannot handle integer variables explicitly, since it performed well on the computational study in [7]. Table II presents eight DFO solvers that were utilized to tune LP solvers' performance.

III. A framework for tuning optimization algorithms

Figure 1 depicts the framework design that we implemented to tune optimization algorithms. The inputs to the framework are:

- the LP solver that will be tuned.
- parameter information of the selected LP solver. For each parameter, the following information is needed: (i) the type of the variable (continuous, discrete, or categorical), (ii) the lower bound, (iii) the upper bound, and (iv) the default value.
- the DFO solvers that will be used to tune the optimization algorithms. The user can select many DFO solvers that will be executed in parallel and the best performance will be displayed at the end of the tuning process
- the training library in which the parameter tuning will take place.

Initially, the first step involves the execution of the LP solver with the default option values over the training library, i.e., a set of problems that is used for finding the best parameter values. Then, the framework runs in parallel all selected DFO solvers. In this step, each DFO solver produces a new feasible solution in each iteration. Each solution is evaluated on the training library using GAMS. HTCondor is also automatically utilized for running multiple GAMS jobs and thus allowing to save much computational time by performing various runs in parallel.

The impact of each option on the solver's performance is calculated by computing $optvalue/defvalue$, where $optvalue$ is the execution time of the solver where a non-default value when a specific option is used (various values are tested for each option and the $optvalue$ is extracted from the best performance of all values) and $defvalue$ is the execution time of the solver for the default value (from Step 1). Therefore, if $optvalue/defvalue < 1$, then there exists a non-default value

BFO 2.0 [32]	https://github.com/m01marpor/BFO
DAKOTA/MADS 6.10 [33]	https://dakota.sandia.gov/
DAKOTA/SOGA 6.10 [33]	https://dakota.sandia.gov/
DFLBOX [34]	http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3/14-mixed-integer-nonlinear-programming/16-dflbox
MCS 2.0 [35]	www.mat.univie.ac.at/~neum/software/mcs/
MISO [36]	https://ccse.lbl.gov/people/julianem/
NOMAD 4.1.0 [37]	https://www.gerad.ca/en/software/nomad
SNOBFIT [38]	www.mat.univie.ac.at/~neum/software/snobfit/

TABLE I
DFO SOLVERS USED

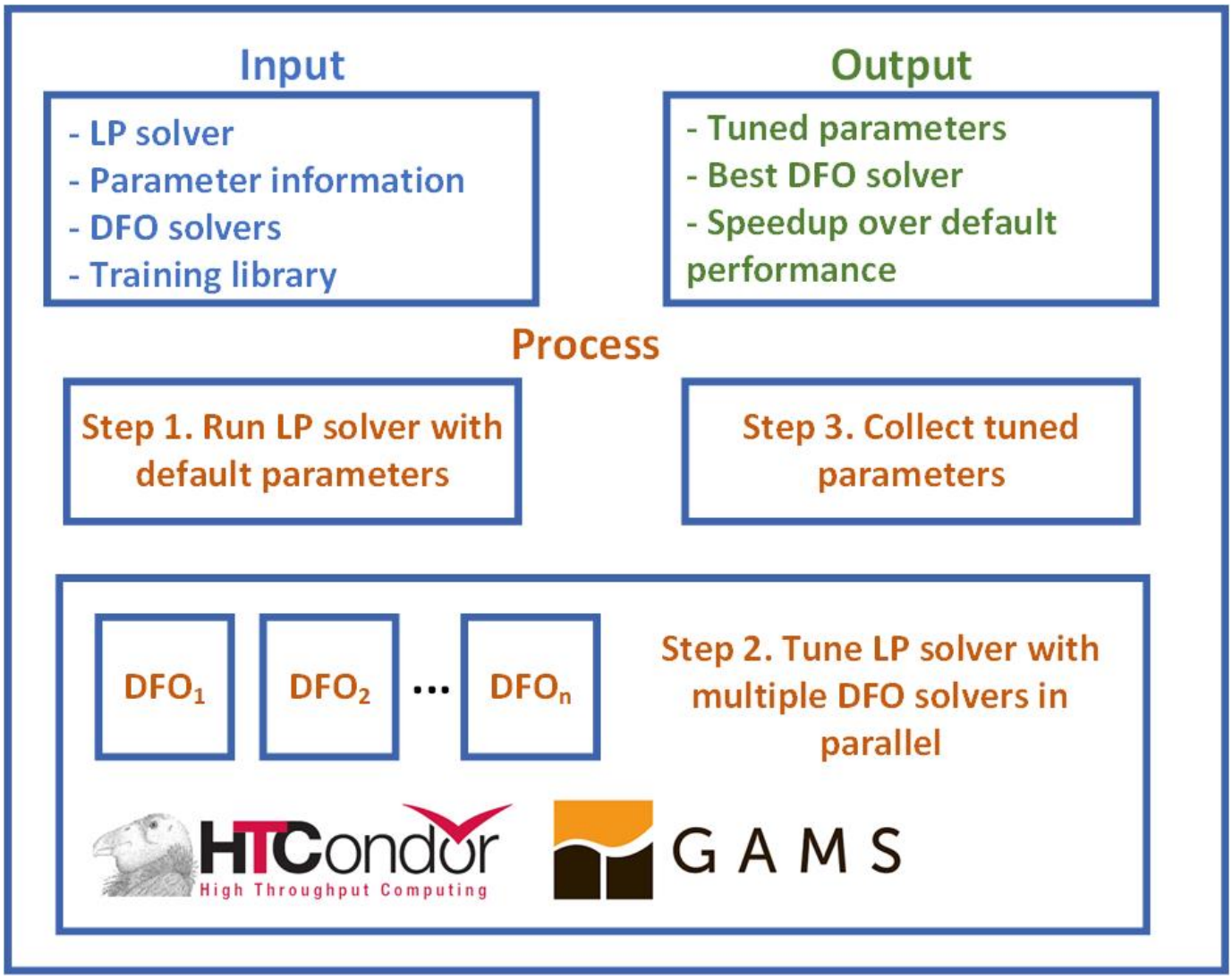


Fig. 1. Framework used to tune LP solvers using DFO solvers.

that leads the solver to perform better on the specific instances. On the other hand, if $optvalue/defvalue > 1$, then all non-default values lead the solver to perform worse on the specific instances). This information is communicated to the DFO solvers in order to produce the next feasible solution. After a predefined number of simulations performed by the DFO solvers, we terminate the parameter tuning process. Typically, we want to obtain the best parameter combination in a small number of simulations.

In this step, other performance metrics (other than the reduction of the execution time) can be used. For example, in minimization (maximization) problems, the decrease (increase) of the objective value can be aimed when there exist problems that cannot be optimally solved in a given time. However, since most LP algorithms find the optimal solutions on the majority of benchmark instances in a reasonable amount of time, we opted to use here the solver's execution time as the performance metric.

Each DFO solver's optimization results are saved to a file. The best optimization results from all DFO solvers are also stored into separate files. All this information is collected in the third step and the following output is provided to the user:

- the best combination of values found for the option setting.
- the best DFO solver.
- the speedup over the default solver's performance.

IV. Computational results

In this computational study, we apply the eight aforementioned DFO solvers to tune the execution time of the dual simplex algorithm of the following LP solvers: (i) CLP, (ii) CPLEX, (iii) GUROBI, and (iv) XPRESS. Computational results are based on a set of 70 LP problems from Kennington, Mészáros, Mittelmann, and Netlib libraries. The computational comparison has been performed on an Intel Core i7-8700 3.2 GHz with 16 GB of main memory and 12 cores, running under CentOS 8.4.

Most of the LP solvers have numerous options mainly divided into the following categories: (i) preprocessing, (ii) algorithmic, (iii) limit, (iv) and tolerance. Five options were selected for each LP solver; all options are important option settings in presolve, scaling, pricing, pivot, and crash procedures of the LP solvers. The options selected for each LP solver are the following:

- CLP: maxFactor, passPresolve, scaling, dualPivot, and crash
- CPLEX: aggfill, scaind, dpriind, folding, and reduce
- GUROBI: sifting, simplexpricing, aggfill, prepasses, and presparsify
- XPRESS: presolve, presolvePasses, crash, dualGradient, and autoScaling

Table IV presents the execution times with default parameters (column 2) and tuned parameters by each DFO solver (columns 3–10). First of all, all DFO solvers achieved to decrease the execution time of all LP solvers. The best runtime reductions for each LP solver are the following:

- 18% for CLP, achieved by DAKOTA/MADS.
- 16% for CPLEX, achieved by MISO.
- 25% for GUROBI, achieved by NOMAD.
- 26% for XPRESS, achieved by DAKOTA/SOGA.

It is particularly interesting that the best performance for each LP solver was achieved by a different DFO algorithm. Figure 2 presents the comparison of the different DFO solvers on this tuning option benchmark. Various DFO solvers have a good performance on tuning the four different LP solvers.

The number of the problems that the DFO solvers improved the default LP performance is one of the most straightforward assessment criterion to compare the ability of DFO solvers to tune the LP solvers. A DFO solver that can improve the execution time of a LP solver on more instances suggests a greater ability to find optimal option values. In the computational study, we also consider the effect of the disturbances from the computational environment to the execution time of the optimization solvers. Thus, runtime reductions between -5% and 5% are considered insignificant. As a result, a significant improvement corresponds to a runtime reduction greater than 5%. The percentage of insignificant and significant runtime reductions for each solver is shown in Figure 3. To obtain these results, we used the best results from all DFO solvers. The DFO solvers reduced CLP's performance on 23% of the problems while having no effect on 43% of the problems. Furthermore, on 36% of the problems, the DFO solvers improved CLP's performance. All other LP solvers produce similar results.

V. Conclusions

Hyper-parameter tuning is an important field that received considerable attention in the last decade. Various approaches, including DFO solvers and deep learning approaches, have been successfully applied. However, the tuning of optimization solvers was a hard task for these approaches since the search space is very large and the optimization problems in the training library have usually different characteristics. Some works have previously applied tuning methods on mixed-integer programming solvers and local and global nonlinear programming solvers. There has been no similar work to investigate whether significant speedups can be obtained by tuning continuous linear optimization solvers. This work aims to fill this gap and present computational evidence showing the ability of DFO solvers to tune continuous linear optimization solvers. Continuous linear optimization solvers are utilized in various applications and even a small reduction on their execution time would have a huge impact in practical problems.

In the computational study, we tuned four LP solvers, namely CLP, CPLEX, GUROBI, and XPRESS, over a set of 70 benchmark instances. Important option settings in presolve, scaling, pricing, pivot, and crash procedures of the LP solvers have been used in the tuning process. We developed a framework to automate this procedure and utilized GAMS and HTCondor. Eight open source DFO solvers were used to tune the LP solvers. We found better parameters for all LP solvers, achieving a reduction in execution time over their

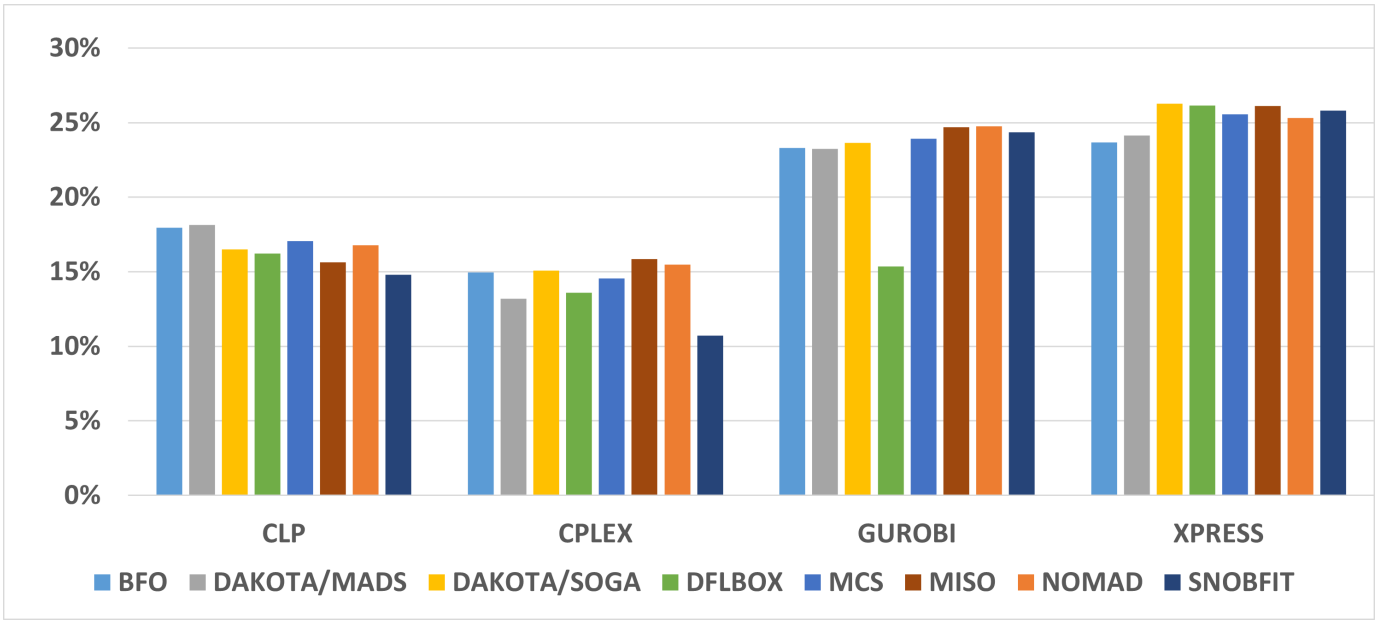
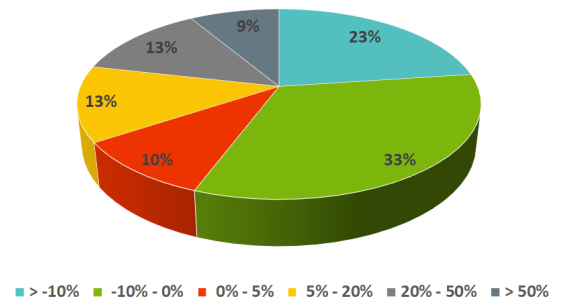
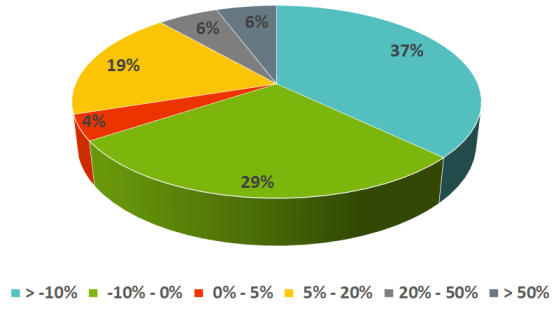


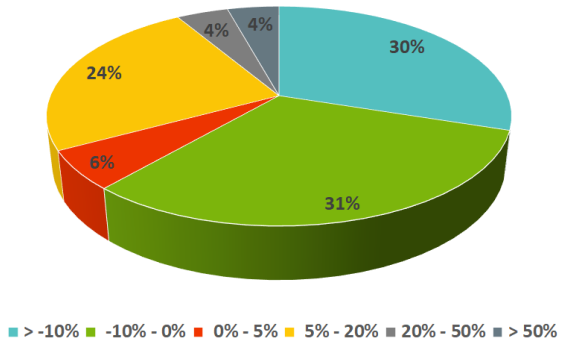
Fig. 2. Results on tuning the performance of LP solvers.



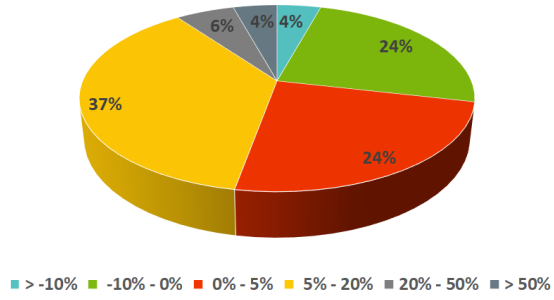
(a) Results for CLP



(b) Results for CPLEX



(c) Results for GUROBI



(d) Results for XPRESS

Fig. 3. Number of improvements for the group of 70 problems.

Solver	Default	BFO	DFLBOX	DAKOTA/MADS	DAKOTA/SOGA	MCS	MISO	NOMAD	SNOBFIT
CLP	32.78	26.90	27.46	**26.84**	27.37	27.19	27.66	27.28	27.93
CPLEX	16.55	14.08	14.30	14.36	14.05	14.14	**13.93**	13.99	14.78
GUROBI	17.49	13.42	14.81	13.43	13.36	13.31	13.18	**13.16**	13.23
XPRESS	19.11	14.59	14.11	14.50	**14.09**	14.22	14.12	14.28	14.18

TABLE II
EXECUTION TIMES WITH DEFAULT PARAMETERS AND TUNED PARAMETERS BY EACH DFO SOLVER

default parameters up to 18% for CLP, 16% for CPLEX, 25% for GUROBI, and 26% for XPRESS. The computational experiments show that several derivative-free optimization solvers outperform others on finding optimal optimal tuning parameters for LP solvers.

REFERENCES

[1] C. Audet, and D. Orban, "Finding optimal algorithmic parameters using derivative-free optimization," SIAM Journal on Optimization, vol. 17, no. 3, pp. 642–664, 2006.

[2] W. Chen, Z. Shao, K. Wang, X. Chen, and L. T. Biegler, "Random sampling-based automatic parameter tuning for nonlinear programming solvers," Industrial & Engineering Chemistry Research, vol. 50, no. 7, pp. 3907–3918, 2011.

[3] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: an automatic algorithm configuration framework," Journal of Artificial Intelligence Research, vol. 36, no. 7, pp. 267–306, 2009.

[4] S. Voulgaropoulou, N. Samaras, and N. Ploskas, "Predicting the execution time of the interior point method for solving linear programming problems using artificial neural networks," presented at the International Conference on Learning and Intelligent Optimization, Athens, Greece, May 2019, pp. 319-324.

[5] S. Voulgaropoulou, N. Samaras, and N. Ploskas, "Predicting the execution time of the primal and dual simplex algorithms using artificial neural networks," Mathematics, vol. 10, no. 7, pp. 1038, 2022.

[6] J. Liu, N. Ploskas, and N. V. Sahinidis, "Tuning BARON using derivative-free optimization algorithms," Journal of Global Optimization, vol. 74, no. 4, pp. 611–637, 2019.

[7] L. M. Rios, and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," Journal of Global Optimization, vol. 56, no. 3, pp. 1247–1293, 2013.

[8] B. Sauk, N. Ploskas, and N. V. Sahinidis, "GPU parameter tuning for tall and skinny dense linear least squares problems," Optimization Methods and Software, vol. 35, no. 3, pp. 638–660, 2020.

[9] N. Ploskas, and N. V. Sahinidis, "Review and comparison of algorithms and software for mixed-integer derivative-free optimization," Journal of Global Optimization, vol. 82, pp. 433—462, 2021.

[10] COIN-OR, "CLP homepage." https://www.coin-or.org/Clp/ (accessed May 25, 2022).

[11] IBM, "IBM ILOG CPLEX Optimizer." https://www.ibm.com/analytics/cplex-optimizer (accessed May 25, 2022).

[12] GUROBI Optimization, "Gurobi optimizer." https://www.gurobi.com/ (accessed May 25, 2022).

[13] FICO, "FICO Xpress Solver." https://www.fico.com/en/products/fico-xpress-solver (accessed May 25, 2022).

[14] N. Ploskas, and N. Samaras, "Linear Programming Using MATLAB," Switzerland: Springer, 2017.

[15] Netlib Repository, "The Kennington LP test problem set." https://www.netlib.org/lp/data/kennington/index.html (accessed May 25, 2022).

[16] C. Mészáros, "The Mészáros LP test problem set." http://old.sztaki.hu/ meszaros/public_ftp/lptestset/ (accessed May 25, 2022).

[17] H. Mittelmann, "The Mittelmann LP test problem set." http://plato.asu.edu/ftp/lptestset/ (accessed May 25, 2022).

[18] Netlib Repository, "The NETLIB LP test problem set." https://www.netlib.org/lp/ (accessed May 25, 2022).

[19] W. Spendley, G. R. Hext, and F. R. Himsworth, "Sequential application for simplex designs in optimisation and evolutionary operation," Technometrics, vol. 4, pp. 441—461, 1962.

[20] J. A. Nelder, and R. Mead, "A simplex method for function minimization," The Computer Journal, vol. 7, no. 4, pp. 308—313, 1965.

[21] V. Torczon, "On the convergence of pattern search algorithms," SIAM Journal on optimization, vol. 7, no. 1, pp. 1—25, 1997.

[22] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," In Advances in optimization and numerical analysis, S. Gomez, and J. P. Hennart, Eds. Dordrecht: Springer, 1994, ch. 4, pp. 51–67.

[23] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," Journal of Optimization Theory and Applications, vol. 79, no. 1, pp. 157—181, 1993.

[24] S. K. S. Fan, and E. Zahara, "A hybrid simplex search and particle swarm optimization for unconstrained optimization," European Journal of Operational Research, vol. 181, no. 2, pp. 527—548, 2007.

[25] N. Ploskas, C. Laughman, A. Raghunathan, and N. V. Sahinidis, "Optimization of circuitry arrangements for heat exchangers using derivative-free optimization," Chemical Engineering Research and Design, vol. 131, pp. 16—28, 2018.

[26] K. R. Fowler et al., "Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems," Advances in Water Resources, vol. 31, no. 5, pp. 743—757, 2008.

[27] C. Audet, and J. E. Dennis Jr, "Pattern search algorithms for mixed variable programming," SIAM Journal on Optimization, vol. 11, no. 3, pp. 573—594, 2001.

[28] M. A. Abramson, C. Audet, J. W. Chrissis, and J. G. Walston, "Mesh adaptive direct search algorithms for mixed variable optimization," Optimization Letters, vol. 3, no. 1, pp. 35—47, 2009.

[29] G. Liuzzi, S. Lucidi, and F. Rinaldi, "An algorithmic framework based on primitive directions and nonmonotone line searches for black-box optimization problems with integer variables," Mathematical Programming Computation, vol. 12, no. 4, pp. 673—702, 2020.

[30] A. Costa, and G. Nannicini, "RBFOpt: an open-source library for black-box optimization with costly function evaluations," Mathematical Programming Computation, vol. 10, no. 4, pp. 597—629, 2018.

[31] M. Laguna, F. Gortázar, M. Gallego, A. Duarte, and R. Marti, "A black-box scatter search for optimization problems with integer variables," Journal of Global Optimization, vol. 58, no. 3, pp. 497—516, 2014.

[32] M. Porcelli, and P. L. Toint, "BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations with continuous and discrete variables," ACM Transactions on Mathematical Software, vol. 44, no. 1, pp. 1—25, 2017.

[33] B. Adams et al., "Dakota, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis," Sandia National Lab, Albuquerque, United States.

[34] G. Liuzzi, S. Lucidi, and F. Rinaldi, "Derivative-free methods for bound constrained mixed-integer optimization," Computational Optimization and Applications, vol. 53, no. 2, pp. 505—526, 2012.

[35] W. Huyer, and A. Neumaier, "Global optimization by multilevel coordinate search," Journal of Global Optimization, vol. 14, no. 4, pp. 331—355, 1999.

[36] J. Müller, "MISO: mixed-integer surrogate optimization framework," Optimization and Engineering, vol. 17, no. 1, pp. 177—203, 2016.

[37] S. Le Digabel, "Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm," ACM Transactions on Mathematical Software (TOMS), vol. 37, no. 4, pp. 1—15, 2011.

[38] W. Huyer, and A. Neumaier, "SNOBFIT–stable noisy optimization by branch and fit," ACM Transactions on Mathematical Software (TOMS), vol. 35, no. 2, pp. 1—25, 2008.