Learning To Scale Mixed-Integer Programs

Timo Berthold, Gregor Hendel*

December 18, 2020

Abstract

Many practical applications require the solution of numerically challenging linear programs (LPs) and mixed integer programs (MIPs). Scaling is a widely used preconditioning technique that aims at reducing the error propagation of the involved linear systems, thereby improving the numerical behavior of the dual simplex algorithm and, consequently, LP-based branch-and-bound. A reliable scaling method often makes the difference whether these problems can be solved correctly or not. In this paper, we investigate the use of machine learning to choose at the beginning of the solution process between two common scaling methods: Standard scaling and Curtis-Reid scaling. The latter often, but not always, leads to a more robust solution process, but may suffer from longer solution times.

Rather than training for overall solution time, we propose to use the attention level of a MIP solution process as a learning label. We evaluate the predictive power of a random forest approach and a linear regressor that learns the (square root of the) difference in attention level. It turns out that the resulting classification not only reduces various types of numerical errors by large margins, but it also improves the performance of the dual simplex algorithm.

The learned model has been implemented within the FICO Xpress MIP solver and it is used by default since release 8.9, May 2020, to determine the scaling algorithm Xpress applies before solving an LP or a MIP.

Introduction

Numerics play an important role in solving linear and mixed integer programs (LPs and MIPs) [25]. All major solvers for LPs and MIPs rely on floating-point arithmetic, hence numerical round-off errors and cancellation effects might occur and therefrom, numerical errors might propagate. This affects the solution process in various critical ways: first of all, the dual simplex algorithm might be more prone to cycling, leading to a longer runtime or even an unsuccessful termination. The computed optimal solution vectors might have residual errors larger than acceptable and consequently, the solution might be deemed wrong.

*FICO Xpress Optimization, Fair Isaac Germany GmbH Takustr. 7, 14195 Berlin, Germany timoberthold@fico.com, gregorhendel@fico.com

In a worst case, instances might be declared infeasible by mistake or seemingly neutral changes in the model, like changing the order of constraints, might lead to different optimal solutions being reported [24]. Since the two standard methods to solve MIPs, namely LP-based branch-and-bound and cutting plane algorithms, both rely heavily on the dual simplex algorithm as a subroutine, they directly inherit its numerical properties and issues. To make things worse, tableau-based cuts like Gomory cuts [18] can amplify numerical issues, unless these cuts are carefully filtered and selected [4].

Obviously, cycling, residual errors and other numerical problems are undesirable and various methods exist to mitigate them, see, e.g., [9, 12, 17]. One of the earliest and to date still most effective way to prevent numerical issues from occuring in LP and MIP solvers is the use of various scaling algorithms. Scaling methods for linear programming have first been suggested in the early 1960's [15]. In 1972, Curtis and Reid introduced a scaling algorithm that solves a least-squares problem, which effectively minimizes the sum of the squares of the logarithms of the matrix elements, plus some correction terms [11].

Various different scaling algorithms have been suggested, each of them beneficial for some classes of instances. However, it is not clear a priori which scaling algorithm is best-suited for a given LP or MIP instance. To this end, we suggest a method to predict which of two of the most common scaling methods will lead to a numerically more robust solution process.

In this paper, we consider general *mixed integer programs (MIPs)* of the form

$$\min\{c^\mathsf{T} x \,|\, Ax \geq b,\ \ell \leq x \leq u,\ x_j \in \mathbb{Z}\ \forall j \in \mathcal{I}\}, \tag{1}$$

with objective coefficient vector $c \in \mathbb{Q}^n$, constraint coefficient matrix $A \in \mathbb{Q}^{m \times n}$, constraint right-hand side $b \in \mathbb{Q}^m$, and variable bounds $\ell, u \in \overline{\mathbb{Q}}^n$, where $\overline{\mathbb{Q}} := \mathbb{Q} \cup \{\pm\infty\}$. Furthermore, the set $\mathcal{I} \subseteq \{1, \ldots, n\}$ denotes the indices of variables that are required to take integer values. If $\mathcal{I} = \varnothing$, (1) is called a *linear program (LP)*. By omitting the integrality requirements for a given MIP, we obtain its *LP relaxation*.

We address the solution of LPs and MIPs by a general purpose solver, in our case FICO Xpress [13]. Recently, the use of Machine Learning methods to take criticial decisions within MIP solvers gained a lot of interest. Examples include learning when to run primal heuristics [21], the addition of new cutting planes [27] or one of the most challenging tasks in MIP solving, choosing a variable to branch on [5, 20]. For an overview, see [7]. However, to the best of our knowledge, no publication so far considered learning scaling methods or other features addressing the numerical stability of a solve. As an important observation, Tang et al. [27] pointed out that learning methods that solely minimize running time might tend to prefer a less stable alternative, given that solvers often terminate faster when they produce a wrong result due to numerical errors.

The contribution of this paper is to introduce machine learning models to choose between different scalings during the runtime of an LP or MIP solve. This is not a proof of concept; the technique has been implemented within the state-of-the-art MIP solver FICO Xpress and is used by default in its latest

release. To this end, we use the *attention level* as a learning label when using ML to take decisions for a mathematical optimization process. The attention level is a measure between 0 and 1 that aims at estimating how likely numerical errors are to occur during an LP or MIP solve; for more details see next section. While numerical robustness of an algorithm represents a valuable goal of its own, this avoids getting the learning process flawed by wrong results. Moreover, we demonstrate that this can still lead to a significant speed improvement, at least for LPs, which is mainly due to the fact that the solver can save on costly recovering procedures.

Background: Scaling, Attention Level

Formally, scaling refers to the multiplication of each row and column in a linear system $Ax \geq b$ by a non-negative scalar, hence transforming the system to an equivalent system $RACx^* \geq Rb$, $x^* = C^{-1}x$, with $R \in \mathbb{Q}_{\geq 0}^{m \times m}, C \in \mathbb{Q}_{\geq 0}^{n \times n}$, with C, R being diagonal matrices, i.e., $C_{ij} = 0$ and $R_{ij} = 0$ for $i \neq j$. In other words, the diagonals of R and C contain the scaling factors for the rows and columns, respectively. Note that for a MIP, integer columns are not scaled, i.e. if $j \in \mathcal{I}$, then $C_{jj} = 1$.

Within FICO Xpress, only powers of two are considered as potential values C_{jj} and R_{ii} for scaling. This has the added advantage that no errors in the system itself will be introduced by scaling, since multiplication and division by powers of two can be done exactly in floating point representation, cf. [6].

In this paper, we use two different scaling approaches. The first one is *standard scaling (ST)*, also referred to as equilibration scaling [29]. ST first scales rows by dividing each row by the absolute value of the nearest power of two of its largest element and then scales columns in the same fashion. For many years, this has been the default scaling procedure of FICO Xpress.

The second scaling approach that we consider is *Curtis-Reid scaling (CR)* [11]. CR solves the least squares problem

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (\log_2 |A_{ij}| - r_i - q_j)^2 \tag{2}$$

Then, CR scales the matrix by $R_{ii} = 2^{\lfloor r_i + 0.5 \rfloor}$ and $C_{jj} = 2^{\lfloor q_j + 0.5 \rfloor}$. The rationale is that this method takes all elements into account and not only extreme ones. Generally, CR is known to work best on numerically challenging problems and is consequently by far the most commonly used non-default scaling technique within FICO Xpress. However, always using CR comes at the price of a slight slowdown of 2–3% on average, according to our computational study. Therefore it is not enabled by default.

Other approaches have been suggested, with no clear evidence that any of them is superior [22, 26]. Up to the experience of the authors, it is rare that another method significantly outperforms the better of ST and CR. The next

common scaling options to make a difference are to apply scaling before presolving instead of after presolving and the question of whether to use dedicated special rules for so-called big-M constraints. This, however, is beyond the scope of the present paper.

Scaling has different goals. Its main purpose is to improve the numerical behavior of the solver; a secondary goal is to improve the runtime of the simplex algorithm. In fact, the observed runtime improvements mostly are a direct consequence of the improved numerical stability. However, runtime improvements might be counteracted by the afore-mentioned effect that producing wrong results due to a numerical error might lead to faster termination. Hence, runtime improvements from scaling might not be as pronounced as improvements in the stability.

There are various ways to measure the numerical stability of an LP or MIP solve. One way is to directly observe the actual number of failures when solving a problem. There are three main types of failures that FICO Xpress tracks for the simplex algorithm: dual failures, primal failures and singular inverts. Note that during a MIP solve, LPs are typically solved via dual simplex, since this allows for warm-starting after adding cuts or branching restrictions [28]. Interior point algorithms, the most common alternative to the simplex method, are are not considered in the present work as they are invariant to scaling [2].

FICO Xpress first attempts to solve each node LP relaxation via dual simplex. Primal simplex serves as fallback strategy if dual simplex fails to maintain dual feasibility due to numerical errors. *Dual and primal failures* occur if the corresponding simplex algorithm (dual or primal) fails to maintain its respective (dual and primal) feasibility. [1]

For performance reasons, the basis inverse (more precisely: its factorization) is incrementally updated after most iterations [14], and only infrequently recomputed from scratch to get rid of accumulated errors. A *singular invert* refers to a situation where refactorization cannot be computed correctly because the current basis matrix is found not to be of full rank. In this case, the solver tries to repair the basis by replacing dependent columns, which often leads to a loss of dual feasibility, comparable to a dual failure. Singular inverts as well as dual and primal failures are mostly performance issues, with primal failures being worse: since primal simplex is already the back-up strategy, a primal failure results in an unsolved node LP relaxation. At such a point, the solver has to conduct a branching without LP information. which is most likely poor and might lead to a significantly larger branch-and-bound (sub-)tree.

Other kinds of errors can be observed a posteriori. One of the most typical issues with numerically challenging instances is that a solution which was feasible in the presolved space violates constraints in the original space by more than an acceptable threshold. Also sometimes it can be observed that due to numerical errors, different solvers finish with contradicting primal and dual bounds. It should be noted that such inconsistent results do not necessarily mean that one

[1] In either case, various strategies are executed to recover feasibility before giving up and deeming the attempt as failure.

of them is wrong. Since both feasibility and optimality of a MIP solve are defined w.r.t. numerical thresholds, multiple answers can be "correct within tolerances" at the same time.

In Numerical Analysis, *condition numbers* represent upper bounds of the change in the output value of a function given a marginal change in the input value. Wilkinson showed that the condition number of a square matrix $\kappa(A_B) = ||A_B|| \cdot ||A_B^{-1}||$ can be used to calculate a bound on the relative error obtained when multiplying this matrix with a perturbed vector [30]. In our application, A_B is the basis matrix of an optimal LP solution. To compute the *attention level* we sample those basis condition numbers $\kappa(A_{B_i})$ once at each search node with optimal LP relaxation and count the relative frequency of the following four buckets: each $\kappa(A_{B_i}) < 10^7$ is counted as stable, each $\kappa(A_{B_i}) < 10^{10}$ is counted as suspicious, each $\kappa(A_{B_i}) < 10^{14}$ is counted as unstable, and bases for which $\kappa(A_{B_i}) \geq 10^{14}$ are counted as ill-posed. These four percentages add up to one: $p_{stab} + p_{susp} + p_{unst} + p_{ill} = 1$. From the three critical categories, the attention level $\alpha = 0.01p_{susp} + 0.3p_{unst} + p_{ill}$ is defined, such that $\alpha \in [0, 1]$. The attention level assumes zero or one if and only if all sampled bases are stable or ill-posed, respectively. The higher the attention level, the more likely numerical issues occur. The attention level was first introduced in Cplex [10] and is also available in FICO Xpress [31].

As a rule of thumb, instances are said to be numerically suspicious if their attention level is larger than 0.1. In this case, it is advisable to alter the model formulation itself to improve numerics, to use control settings that address numerical behavior (e.g., scaling) or to activate non-default procedures that aim for higher solution accuracy, e.g., iterative refinement [17] or MIP refinement.

It should be noted that, in our experience, CR often leads to better numerical behavior, but not always. On the data set in the subsequent sections, ST turns out to be the more stable scaling option (has the smaller attention level) in 24 % of the cases, whereas CR reduces the attention level in 30 % of the cases. The remaining cases are ties.

The advantage of the attention level is that it is also meaningful for numerically less challenging instances that do not lead to actual failures. It has a fine granularity to measure the grade of numerical conditions, taking the whole solution process into account. We therefore decided to use the attention level as a label to design ML models that have numerical stability as a learning goal.

An obvious alternative would be using time. This would carry a peculiar risk. For some numerically challenging problems, numerical errors might not lead to a wrong result, but seemingly prove optimality by a wrong conclusion. Consider the situation when the optimal solution has been found early in the search. Then, numerical errors in techniques that improve the dual bound may close the remaining gap prematurely without the error being noticed. As a consequence, a worse scaling might seemingly speed up the solver. This is a major motivation why we train for the attention level, not for running time.

Methodology

Learning Task/Feature Space Let $S = \{\text{st}, \text{cr}\}$ denote the set of available scalings. We aim to train a classifier $y : \mathbb{R}^d \mapsto S$ that labels a d-dimensional input vector of features $f = (f_i, \ldots, f_d)$ onto one of our two mentioned scaling choices such that the resulting attention level $\alpha(y(f))$ is minimized. In our case, the feature vector is 3-dimensional, where each feature describes one difference between the two scaling methods regarding the obtained matrix coefficients, right-hand side, and objective coefficients after presolving. Our feature space measures for each scaling the orders of magnitude between, for example, the smallest and largest nonzero coefficient of the matrix A. In preliminary experiments, we also tested larger feature spaces, including, for example, the original (unpresolved) coefficient ranges of the matrix. We found that training models on the unpresolved coefficients of the matrix leads to significantly worse training and test performance. This confirms our expectation that presolving transformations alone may increase numerical stability even for ill-posed input data.

For the feature computation, we apply each of the two scaling methods $s \in S$ to the presolved matrix A to obtain the corresponding scaling matrices R_s, C_s that contain the row and column multipliers along their respective diagonals. We obtain minimum and maximum nonzero entries of the scaled matrix coefficients, right-hand side, and objective coefficients, denoted by symbols d and D, respectively:

- $d_s^{\text{coef}}, D_s^{\text{coef}} = \min, \max\{|(R_s A C_s)_{ij}|; (R_s A C_s)_{ij} \neq 0\}$,
- $d_s^{\text{obj}}, D_s^{\text{obj}} = \min, \max\{|(C_s c)_j|; (C_s c)_j \neq 0\}$,
- $d_s^{\text{rhs}}, D_s^{\text{rhs}} = \min, \max\{|(R_s b)_i|; (R_s b)_i \neq 0\}$.

If all objective coefficients or right-hand side entries are 0, we define their corresponding minimum/maximum entry to be 0, where finite bounds of the variables are treated as part of the right-hand side.

We then capture the range in the orders of magnitude of the scaled MIP problem by three separate logarithms,

- $L_s^{\text{coef}} = \log_{10}(D_s^{\text{coef}}/d_s^{\text{coef}})$,
- $L_s^{\text{obj}} = \log_{10}(D_s^{\text{obj}}/d_s^{\text{obj}})$, and
- $L_s^{\text{rhs}} = \log_{10}(D_s^{\text{rhs}}/d_s^{\text{rhs}})$.

Intuitively, numerically stable MIPs feature small coefficient ranges, whereas MIPs that combine coefficients at vastly different orders of magnitude are more often ill-posed and prone to numerical issues. Note that $L_s^{\text{coef}}, L_s^{\text{obj}}, L_s^{\text{rhs}} \geq 0$. We capture the special case of an all-zero component by setting the corresponding L to 0, just like in the case that all values are equal to the same nonzero value. The L-values represent how many orders of magnitude the coefficients of a given MIP

span. Our feature vector consists of the difference in the L-vectors of standard and Curtis-Reid scaling:

$$f = \left(L_{\text{cr}}^{\text{coef}} - L_{\text{st}}^{\text{coef}}, L_{\text{cr}}^{\text{obj}} - L_{\text{st}}^{\text{obj}}, L_{\text{cr}}^{\text{rhs}} - L_{\text{st}}^{\text{rhs}}\right).$$

We approach the classification problem to choose a scaling from S by a regression problem, for which we use the attention level difference as label: $\Delta = \sqrt{\alpha_{\text{cr}}} - \sqrt{\alpha_{\text{st}}}$. We chose the square root to artificially inflate notable differences in attention level close to 0. This is mainly motivated by the empirical observation that for our training (and test) data, the majority of values are rather close to zero than close to one.

We will train a regression model $\tilde{\Delta}(f)$ that computes (predicts) the expected Δ based on the coefficient statistics captured by f. The classification algorithm uses $\tilde{\Delta}(f)$ and a threshold value τ. If $\tilde{\Delta}(f) \leq \tau$, it selects Curtis-Reid scaling, otherwise it selects standard scaling. The additional control parameter τ allows to bias the selection towards standard scaling, i.e., to switch to CR only if a substantial reduction in attention level is expected. This conservative approach may prevent performance degradations due to switching to CR unnecessarily.

Data collection The regression is trained offline on data collected by running FICO Xpress on a large test bed of 1199 diverse MIP problems. This test bed comprises publicly available problems from the MIPLIB 2017 [16] benchmark set as well as confidential customer problems. A subset of these problems is part of a numerically challenging test set. We solve all problems twice using FICO Xpress, once with standard (`st`) and again with Curtis-Reid (`cr`) scaling. We enable the computation of the attention level at the largest supported sampling frequency by setting the parameter `MIPKAPPAFREQ` to 1, such that the condition number κ is computed after every feasible node LP relaxation throughout the search. In addition, we diversify our data even more by running each problem/scaling combination with three (default + two) different permutation seeds. FICO Xpress uses a nondefault permutation seed for a cyclic reordering of the rows and columns of a problem. While the problem stays unchanged mathematically, it is well-known that such problem permutations can have a dramatic impact on the performance [24]. We use the term "instance" to denote an input problem together with a seed.

We filter out unsuitable records from our data set: some of the instances are solved in presolving or with the initial LP, whereas others do not finish the initial LP within the time limit, such that no κ can be computed. We identify 362 such cases, such that we are left with 3235 training examples.

The attention levels measured with both scaling techniques are depicted in Figure 1. Each point corresponds to the pair $(\sqrt{\alpha_{\texttt{st}}}, \sqrt{\alpha_{\texttt{cr}}})$ of one data record. We observe horizontal and vertical concentrations of points along a grid at values $\sqrt{0.01} = 0.1$, $\sqrt{0.3} \approx 0.55$, and 1. Those are exactly the attention level values if all measured values of κ fall into a single category "suspicious", "unstable" or "ill-posed", respectively. The considerable number of points below the diag-

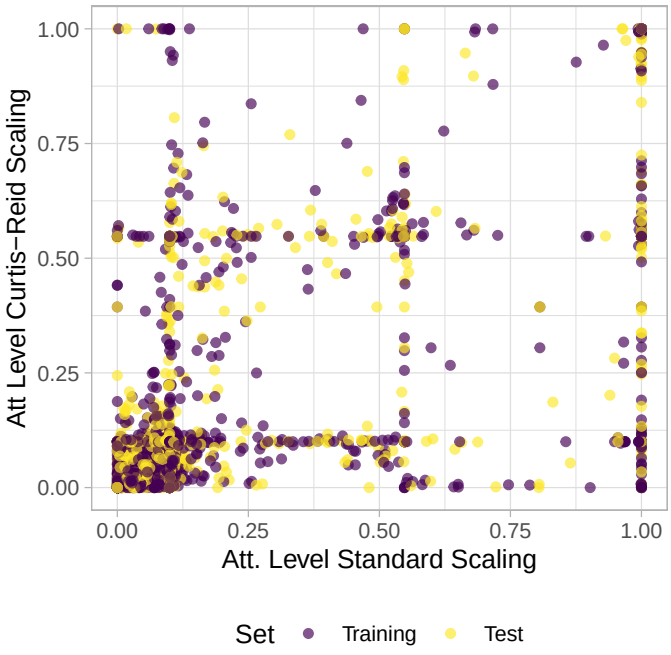

Figure 1: Square root of attention level with standard and Curtis-Reid scaling on our data set.

onal shows that there is room for improving (reducing) the attention level by switching the used scaling.

Training For the training of the regression models, we split our data set randomly into 60 % (1941) training and 40 % (1294) test set. Since each problem has been solved multiple times with different seeds, a single problem may be present in both the training and the test set, but with different seeds. Since the size and coefficients of the presolved model can be different between seeds, also the scaling applied after presolving is affected by permutations. The data reveals that there are 598 matrices for which the fastest and slowest seed differ by at least 5 %, and there are about the same number of instances (560/600 for Standard/Curtis Reid) that differ in attention level across seeds.

On the training set, we train a linear regression (see, e.g., [8]) and a random forest regression [23]. We perform the training in the `R` programming language, which provides linear regression through its base function `lm` and random forest regression from the add-on package `randomForest`. We train a random forest with 500 trees. After the model training, we pick the thresholds τ by searching the domain $[-1, 1]$ in 0.01-steps to finish our classifier individually for the linear regression and random forest models on the training set. The values of τ that

minimize the attention level on the training set are -0.05 for the linear model and 0.01 for the random forest model. The negative tau for the linear model yields a conservative switch that avoids unnecessary switches to CR with potential performance degradations.

We evaluate all models on the training and test data in Figure 2. For the continuous regression problem, a common quality measure is the mean squared error (MSE) $\frac{1}{k^{\text{test}}} \sum_{i=1}^{k^{\text{test}}} (\tilde{\Delta}(f_i) - \Delta_i)^2$. In addition, we report the average attention level separately for the training and test set. We compute the attention level achieved by the classifiers that combine the trained regression models and the optimal thresholds. Figure 2 shows the average attention level on all instances for which at least one scaling has a nonzero attention level.

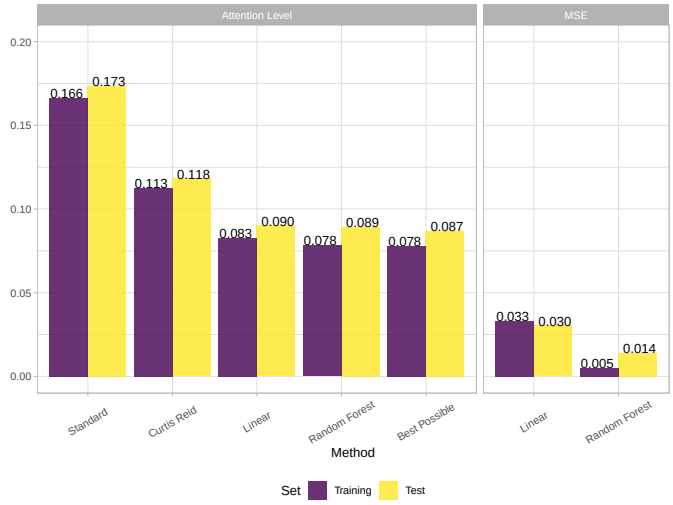

Figure 2: Performance of the trained models in attention level and mean squared error.

In terms of MSE, the random forest model is clearly better than the linear model. On the test set, the random forest model achieves an MSE of 0.014, which is a reduction compared to the linear model by more than 50 %. However, the random forest model performs much better on the training than on the test set, which can be seen as an indication for overfitting. The linear model achieves an MSE of approx. 0.03 on both the training and the test set. While this error is not as small as the one for the random forest model, the linear model shows no tendency of overfitting to the training set.

Concerning the attention level, we see that the standard scaling method has the highest average attention level on both training and test sets. By using Curtis-Reid scaling, the average attention level can be reduced by more than 30 % compared to standard scaling. The trained models reduce the attention level on the test set even further. The linear model achieves an average attention level of 0.09 on the test set, and the random forest achieves an average attention

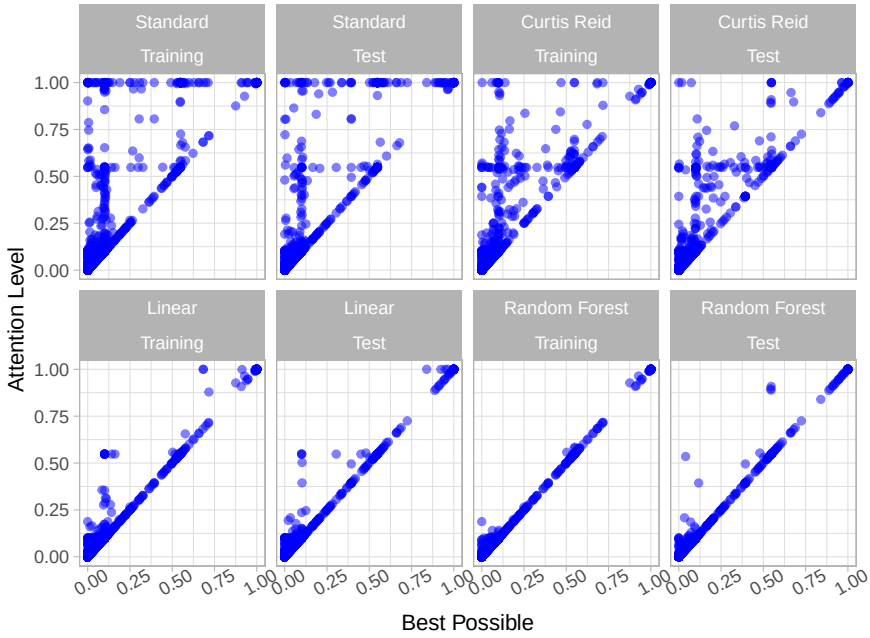

Figure 3: Comparison of Attention Level $\sqrt{\alpha}$ between available and learned scaling methods on training and test set

level of 0.089. Both learned models exhibit a performance close to the best possible average attention level of 0.087 on the test set, with a relative deviation of only 3 % (random forest) and 4 % (linear).

Figure 3 provides a visual comparison between the two individual, basic scaling methods standard and Curtis-Reid as well as the two learned scaling selection models. We use the measurement $\sqrt{\alpha}$ to stretch attention levels close to 0. As a base line serves the best possible attention level, i.e., the minimum of $\sqrt{\alpha_{\text{st}}}$ and $\sqrt{\alpha_{\text{cr}}}$. All points on the main diagonal are therefore optimal for the considered instance. On the contrary, instances in the top-left corner belong to the undesirable group of cases where the numerics can be improved most drastically by simply switching the scaling method.

The figure confirms that both individual scaling methods, standard and Curtis-Reid, have a substantial number of suboptimal records. Using a trained model, however, changes the picture. Both the linear and the random forest based classifiers are able to classify most of the test instances correctly, with only a few visible deviations from the best possible main diagonal. In particular, there are no remaining cases in the top left corner using either of the learned models, i.e., all of the most extreme cases are classified correctly.

While the random forest regression gives a clearly better result in terms of the mean squared error, its additional benefits as a classifier that minimizes attention level over the linear model are diminishingly small on the test set.

Both the linear and the random forest classifiers achieve an attention level that is very close to optimal. In the light of these results, we selected the linear model for the use inside FICO Xpress because the loss is marginal compared to a random forest model. From a solver development point of view this comes with the added advantages that a linear model requires less memory than a random forest, and is interpretable. Our final implementation uses as coefficients

$$\tilde{\Delta}(f) = 0.014 f^{\text{coef}} + 0.07 f^{\text{obj}} + 0.008 f^{\text{rhs}}, \qquad \tau = -0.054.$$

It confirmed our expectation to see that the difference in the orders of magnitude spanned by the matrix coefficients is by far the most important feature. This coincides with the scaling algorithms mainly focussing on the matrix values themselves and not on the right-hand sides and objective coefficients. Curiously, the impact of the right-hand side values was larger than those of the objective. This might be explained by two aspects. Firstly, some of the MIP instances are either pure feasibility problems with no objective at all or models in which a single artificial variable is minimized. In both cases, the objective feature is zero, hence it has no predictive power. Secondly, in all MIPs, a certain percentage of the variables are integer and can therefore not be scaled. This restricts the degrees of freedom for column scaling compared to row scaling, and hence the degrees of freedom for objective vs. right-hand side scaling.

Computational Results

The use of the linear classification previously described is available as a new "autoscaling" functionality in Xpress 8.9 and enabled by default. In this section, we compare the LP and MIP stability and performance between autoscaling and standard scaling, which used to be the previous default scaling of FICO Xpress. As for the training, our test sets are comprised of a mixture of publicly available LP and MIP problems and proprietary data, usually from customers. We consider three test sets, one of which is a dedicated LP performance test set, and two MIP test sets, one of which we use for benchmarking solver run time, and the other consisting of numerically challenging instances.

LP The first test set consists of hard linear programming problems. It is comprised of a mixture of LPs that do not have integer variables by design as well as hard LP relaxations of actual MIP problems. This test set consists of 348 problems, which we solve with 3 permutation seeds, yielding a total of 1044 test instances. We use a 1 hour time limit and solve all instances with the parallel dual simplex method [19] with up to 8 threads. We present an overview of the obtained results in Table 1, where we aggregate all discussed numerical properties of a solution process and the required solving time. More precisely, we summarize all measures (except the number of inconsistencies) using a shifted geometric mean [1] with a shift of 1 as follows. A vector of measurements $X = (X_1, \ldots, X_n)$

Table 1: Summary of performance and numerical results on three test sets LP, MIP Benchmark, and MIP Numerics. In parentheses, the number of instances (matrix-seed combinations) is shown. Numbers are shifted geometric means with a shift of 1, while the column "rel." shows percentage deviations after enabling autoscaling.

	LP (1044)			MIP Benchmark (3300)			MIP Numerics (1857)		
	standard	auto	rel.	standard	auto	rel.	standard	auto	rel.
Numerics									
Dual									
Solves	1	1	±0 %	367	372	+1.2 %	5016	6527	+30.1 %
Fail.	0.01	0.01	±0 %	0.72	0.55	-30.0 %	16.70	10.74	-55.4 %
Primal									
Solves	1	1	±0 %	222	226	+1.8 %	2366	2827	+19.5 %
Fail.	0	0	±0 %	0.012	0.007	-79.4 %	0.108	0.067	-59.7 %
Inverts									
#	99	95	-4.7 %	441	439	-0.5 %	7002	8069	+15.2 %
Singular	0.02	0.01	-53.0 %	0.14	0.12	-17.7 %	1.25	0.94	-33.2 %
MIP Nodes									
#	–	–	–	109	112	+2.8 %	1575	1974	+25.4 %
Failures	–	–	–	0.012	0.007	-78.1 %	0.125	0.077	-61.7 %
Inconsistencies									
#	1	0	–	6	4	-50.0 %	57	40	-42.5 %
Condition									
Stable	–	–	–	–	–	–	55	184	+237.4 %
Ill-posed	–	–	–	–	–	–	1.10	0.82	-34.2 %
Att. Level	–	–	–	–	–	–	0.101	0.078	-28.6 %
Performance									
Time	157.9	151.3	-4.4 %	28.7	28.5	-0.6 %	–	–	–

is aggregated via

$$\mathrm{sgm}(X) = -1 + \prod_{i=1}^{n} (X_i + 1)^{\frac{1}{n}}.$$

The use of the shifted geometric mean is the de-facto standard method to aggregate performance measures, in particular running time, of MIP algorithms. Due to the exponential nature of the Branch-and-bound method, one is mostly interested in relative improvements when comparing different methods. For many test sets, the actual measurements might differ by orders of magnitude. While the geometric mean is known to reduce the weight of the extreme measurements at the high end of the scale compared to the arithmetic mean, the shift value has been introduced to diminish the influence of tiny measurements onto the result. When we count occurences of numerical problems such as the number of primal failures, we also circumvent the presence of zeros in the data, which could not be summarized by a geometric mean without a shift.

In Table 1, we report the same measurements for each of the three test sets LP, MIP Benchmark, and MIP Numerics separately, if a measurement is applicable. Each measurement is summarized for standard scaling and for autoscaling. We also show the relative percentage deviation.

For the LP test sets, not all measurements are applicable. As expected, the mean number of dual solves is 1. Perhaps surprisingly, the same holds for the number of primal solves. The latter is usually called as a final cleanup step

to polish the solution obtained by the dual simplex algorithm. We see that the number of inverts is slightly reduced by autoscaling, and the number of singular inverts is reduced significantly, albeit on a small scale. Finally, the Time row shows that autoscaling achieves a remarkable speedup of 4.4 % compared to standard scaling. If we restrict ourselves to the set of 358 instances that are affected by this change, which is not shown in the table, we see a time improvement of 11 %.

MIP Benchmark & Numerics We consider a benchmark set of 1100 MIP problems as well as a (disjoint) numerically challenging problem set consisting of 619 MIPs. Together with the aforementioned three permutation seeds used, we obtain 3300 benchmark and 1857 numerically challenging instances, respectively. As for the LP test set, we use a time limit of 1h. The branch-and-bound search is conducted in parallel using up to 20 threads. Only for the numerically challenging test set, we activate the collection of MIP kappa statistics after every node LP, which slows down the solution process by approximately 10 % on average.

We present the obtained numerical results in Table 1. All measurements except for the total number of inconsistencies were aggregated using a shifted geometric mean and a shift of 1.

We see a reduction in the dual failures by 30.0 % and 55.4 % on the benchmark and numerically challenging sets, respectively. Primal failures and singular inverts are reduced by 79 % and 17 % on the benchmark set and 59 % and 33 % on the numerically challenging test set.

We also show the number of problems that have been solved inconsistently across seeds. We call a problem inconsistently solved if the outcomes of running the same model with different seeds contradict each other regarding the computed primal and dual bounds. An extreme example of such an inconsistency occurs if a problem is declared infeasible for one seed, whereas using a different seed results in a feasible solution. Although both situations may mathematically be correct within the used floating point tolerances, a solution is usually preferable from a user perspective.

In contrast to the shifted geometric mean reduction elsewhere, we report inconsistencies in the corresponding row of Table 1 as the number of problems for which inconsistencies were encountered across seeds. Note that this time, the sum corresponds to the number of problems, not instances. In order to refer to the test set instance counters, the numbers in this row have to be multiplied by three, the number of permutation seeds. As one might expect, inconsistencies occur much more frequently on the numerically challenging test set, whereas they are negligible on the LP and MIP Benchmark sets. For all three instance sets, autoscaling reduces the number of inconsistent results. On the numerically challenging test set, autoscaling achieves a notable reduction from 57 to 40 problems with inconsistent results.

Kappa statistics are only enabled for the numerically challenging set. We report the Kappa statistics for the two ends of the Kappa classes, namely the

stable and the ill-posed LPs according to the value of the condition number κ. The use of autoscaling increases the shifted geometric mean of the number of stable conditioned LPs by more than a factor of three from 55 to 184, whereas the number of ill-posed LPs is substantially reduced by 34 %. The overall reduction of the condition numbers finally yields a reduction in the attention level by almost 30 %.

Because of the many encountered inconsistencies, we do not report time results for the challenging test set. For the MIP benchmark set, runtime performance is almost unchanged between standard and autoscaling. This might seem surprising in the light of the previously observed speedup of the dual simplex algorithm on the LP test set. However, together with a faster LP, we also observe an increase in the number of nodes for both MIP test sets, which we see as the main reason for the non-transfer of the LP performance improvements onto the MIP case. The increased number of nodes seems to be the price for an increased numerical robustness of the solution process. It may often happen that a less stable method cuts off a local subproblem prematurely as infeasible, whereas a more robust method has to spend a few extra nodes to prune that subtree.

Conclusion

In this paper, we demonstrated that a machine learning model can be used to robustify the solution process of LP and MIP solvers. More precisely, we presented a way to derive a linear model of three features which automatically decides between two alternative scaling options. In a detailed computational study we showed that the resulting model improves numerical stability of the MIP solver's solution process in many different aspects. It outperforms both individual methods and comes close to a virtual best selection w.r.t. the mean observed attention level. As an added benefit, the procedure significantly improved the mean runtime of the parallel dual simplex algorithm. The autoscaling procedure introduced in this paper is implemented within Xpress 8.9 and used by default. To the best of our knowledge, the present research is the first to address numerical issues in MIP solvers by ML methods and one of only a few examples where an ML model is used by default within a MIP solver (for a notable exception, see [3]).

The idea of using machine learning in the context of numerics of mathematical optimization problems can be extended in many directions. One could, for example, try to directly predict the attention level of an LP or MIP solve. Another possible extension of the methodology could aim at quadratic optimization problems by also considering the scaling of the involved quadratic matrices, whose coefficients on the one hand often operate on a different scale than the linear part of the problem and for which error propagation can be easily amplified (or diminished) by the quadratic nature of the problem. Also, using a convex combination of two (or more) scalings and learning the combination parameter(s) could be promising.

References

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[2] E. D. Andersen, J. Gondzio, C. Mészáros, X. Xu, et al. *Implementation of interior point methods for large scale linear programming*. HEC/Université de Geneve, 1996.

[3] D. Anderson, G. Hendel, P. Le Bodic, and M. Viernickel. Clairvoyant restarts in branch-and-bound search using online tree-size estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1427–1434, 2019.

[4] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[5] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik. Learning to branch. *arXiv preprint arXiv:1803.10150*, 2018.

[6] F. L. Bauer. Optimally scaled matrices. *Numerische Mathematik*, 5(1): 73–87, 1963.

[7] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

[8] J. M. Chambers. *Statistical Models in S*, chapter Linear Models. Wadsworth & Brooks/Cole, 1992.

[9] K. K. Cheung, A. Gleixner, and D. E. Steffy. Verifying integer programming results. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 148–160. Springer, 2017.

[10] Cplex Documentation. MIP kappa: detecting and coping with ill-conditioned MIP models. https://www.ibm.com/support/knowledgecenter/SS9UKU_12.10.0/com.ibm.cplex.zos.help/CPLEX/UsrMan/topics/discr_optim/mip/troubleshoot/60_ill_conditioned_MIP_kappa.html, 2020.

[11] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for gaussian elimination. *IMA Journal of Applied Mathematics*, 10(1):118–124, 1972.

[12] D. G. Espinoza. *On linear programming, integer programming and cutting planes*. PhD thesis, Georgia Institute of Technology, 2006.

[13] FICO. Xpress Optimizer, 2020. https://www.fico.com/en/products/fico-xpress-solver.

[14] J. J. Forrest and J. A. Tomlin. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical programming*, 2(1):263–278, 1972.

[15] D. Fulkerson and P. Wolfe. An algorithm for scaling matrices. *Siam Review*, 4(2):142–146, 1962.

[16] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al. MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. Technical report, Technical report, Optimization Online, 2019.

[17] A. M. Gleixner, D. E. Steffy, and K. Wolter. Iterative refinement for linear programming. *INFORMS Journal on Computing*, 28(3):449–464, 2016.

[18] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[19] Q. Huangfu and J. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

[20] E. B. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[21] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao. Learning to run heuristics in tree search. In *IJCAI*, pages 659–666, 2017.

[22] T. Larsson. On scaling linear programs—some experimental results. *Optimization*, 27(4):355–373, 1993.

[23] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[24] A. Lodi and A. Tramontani. Performance variability in mixed-integer programming. In *Theory Driven by Influential Applications*, pages 1–12. INFORMS, 2013.

[25] M. Miltenberger, T. Ralphs, and D. E. Steffy. Exploring the numerics of branch-and-cut for mixed integer linear optimization. In *Operations Research Proceedings 2017*, pages 151–157. Springer, 2018.

[26] N. Ploskas and N. Samaras. The impact of scaling on simplex type algorithms. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 17–22, 2013.

[27] Y. Tang, S. Agrawal, and Y. Faenza. Reinforcement learning for integer programming: Learning to cut. *arXiv preprint arXiv:1906.04859*, 2019.

[28] J. A. Tomlin. An improved branch-and-bound method for integer programming. *Operations Research*, 19(4):1070–1075, 1971.

[29] J. A. Tomlin. On scaling linear programming problems. In *Computational practice in mathematical programming*, pages 146–166. Springer, 1975.

[30] J. H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, Inc., 1988.

[31] Xpress Documentation. Analyzing and Handling Numerical Issues. https://www.fico.com/fico-xpress-optimization/docs/latest/solver/optimizer/HTML/chapter5_sec_numerics.html, 2020.