Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

Thorsten Koch[1], Timo Berthold[2], Jaap Pedersen[3], Charlie Vanaret[4]

Progress in Mathematical Programming Solvers from 2001 to 2020

[1] 0000-0002-1967-0077
[2] 0000-0002-6320-8154
[3] 0000-0003-4047-0042
[4] 0000-0002-1131-7631

ZIB Report 21-20 (February 2021)

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

Progress in Mathematical Programming Solvers from 2001 to 2020

Thorsten Koch[a,b,1], Timo Berthold[c,2], Jaap Pedersen[b,3], Charlie Vanaret[a,4]

[a] *Technische Universität Berlin, Chair of Software and Algorithms for Discrete Optimization, Straße des 17. Juni 135, 10623 Berlin, Germany*
[b] *Zuse Institute Berlin, Takustraße 7, 14195 Berlin, Germany*
[c] *Fair Isaac Germany GmbH, Stubenwald-Allee 19, 64625 Bensheim, Germany*

Abstract

This study investigates the progress made in LP and MILP solver performance during the last two decades by comparing the solver software from the beginning of the millennium with the codes available today. On average, we found that for solving LP/MILP computer hardware got faster about 20 times, and the algorithms improved by a factor of about nine for LP and over 30 for MILP, giving a total speed-up of about 180 and 600 times, respectively. However, the above numbers have a very high variance and they considerably underestimate the progress made on the algorithmic side: many problem instances can nowadays be solved within seconds, which the old codes are not able to solve within any reasonable time.

Keywords: LP-solver, MILP-solver, Mathematical Programming Software, Benchmark

1. How much did the state-of-the-art in (Mixed Integer) Linear Programming solvers progress during the last two decades?

The present article aims at providing one possible answer to this question. We will argue how progress in LP and MILP solvers can be measured, how to evaluate this progress computationally, and how to interpret our results. Our findings are summarized in Figure 1 and Figure 2. Clearly, they need more context, which will provide in the main part of this article.

Without doubt, practical solving Linear Programs (LP) and Mixed Integer Linear Programs (MILP) has made tremendous progress during the last 40+ years. The question "how much?" naturally arises. And how much of this progress is due to algorithmic improvement compared to advances in hardware and compilers?

Email address: `koch@zib.de` (Thorsten Koch)
[1]ORCID: 0000-0002-1967-0077
[2]ORCID: 0000-0002-6320-8154
[3]ORCID: 0000-0003-4047-0042
[4]ORCID: 0000-0002-1131-7631

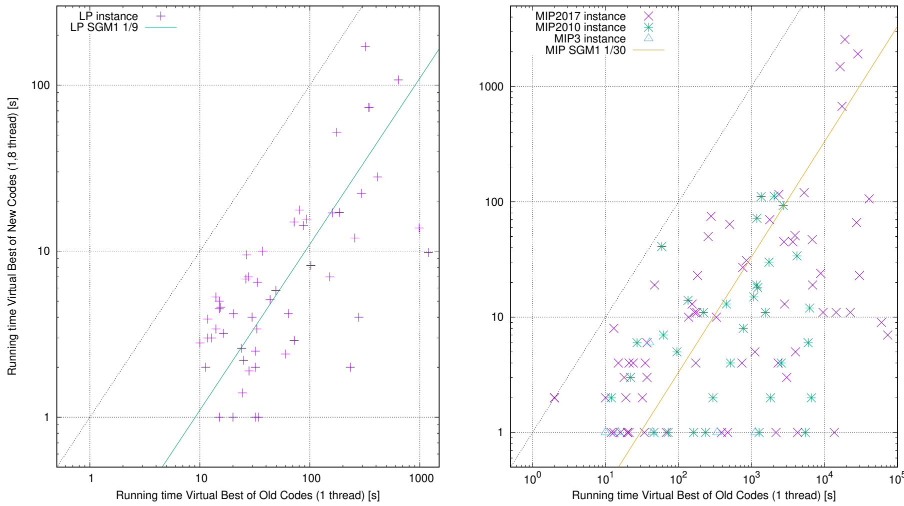

Figure 1: Comparison of the running times of various LP (left) and MILP (right) instances between the virtual best of CPLEX 7, XPRESS 14, and MOSEK 3, from around 2001 and the virtual best of CPLEX 12.10, GUROBI 9.0, XPRESS 8.11, MOSEK 8.1, and COPT 1.4 from 2020 running with either 1 or 8 threads on a log scale.

1.1. Previous studies

This question has been asked before. There are five studies that focus solely on the CPLEX solver and cover the 1990s and 2000s. The first two, from Bixby et al. [1, 2], investigate the progress from 1987 to 2001 regarding the solution of LPs; the latter concludes: *Three orders of magnitude in machine speed and three orders of magnitude in algorithmic speed add up to six orders of magnitude in solving power: A model that might have taken a year to solve 10 years ago can now solve in less than 30 seconds.* For the period from 1997 to 2001, the geometric mean speed-up computed over 677 instances was 2.3. However, it should be noted that the speed-up for large models with more than 500,000 rows was over 20.

Bixby et al. [3] examine MILP solving. The study considers 758 instances and compares CPLEX 5.0 (released in 1997) and CPLEX 8.0 (2002). The geometric mean of the speed-up is about 12. For models that took long to solve with version 5.0, the speed-up is considerably higher, reaching an average of 528 for those instances that required over 27 hours to solve with the older code.

Achterberg and Wunderling [4] continued the study up to CPLEX 12.5 in 2012. The overall geometric mean speed-up on 2,928 MILP models turned out to be 4.71. Again, for instances where version 8.0 took longer to solve, an average speed-up of up to 78.6 was observed. This is still an underestimation, as the old solver hit the time limit of 10,000 seconds for 732 of the instances, while the new one only had 87 time outs.

Lodi [5] compared CPLEX 1.2 (1991) with CPLEX 11.0 (2007) on 1,734 MILPs and reported a geometric mean speed-up of 67.9. Another revealing metric shown is the number of instances solved to optimality within the time limit of 30,000s. On 1,852 MILPs, CPLEX 1.2 was able to solve a mere 15.0%, while version 11.0 on the same hardware could solve 67.1%.

Koch et al. [7, 8] compared the performance of a wide variety of solvers on

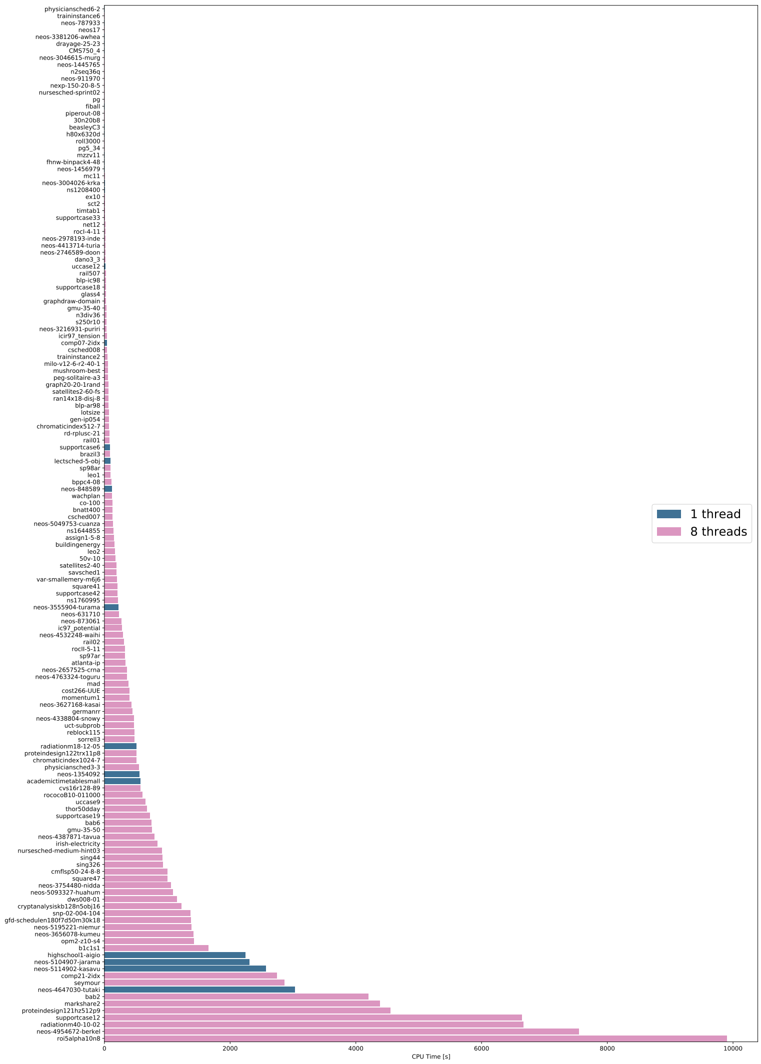

Figure 2: Runtime of the virtual best new solver for those 149 instances from the MIPLIB 2017 [6] benchmark set that could not be solved by any of the old solvers within 24 h. Number of threads indicate which was faster.

the MIPLIB 2010. The progress from 1996 to 2011 was investigated and the conclusion was unsurprisingly similar. On the one hand, instances that were already "fast" did not get much faster. On the other hand, many instances that used to be "difficult" got solved considerably faster, and thereby were the ones that contributed the most to the overall speed-up.

Since all of these studies are at least ten years old, it seems about time to give an update on whether LP and MILP development is still going strong.

1.2. Setup of this study

One could argue that all studies, including the present one, have intrinsic biases. The threshold for discarding problems as too easy influences the observed speed-up factors. The higher the threshold, the higher the speed-up. The same happens on the other end: the lower the time limit given to the solver, the lower the achievable speed-up.

Another bias comes from the selection of instances. Instances usually do not enter a collection because they are quickly solved on the first try. Therefore, there is a tendency to collect "difficult" instances. On the other hand, modeling practices rely on the efficiency of current solvers, which leads to a selection that under-represents modeling practices that cannot (at the time) be satisfyingly solved.

While the aforementioned issues are intrinsic to the concept of benchmarking, another limitation is not, and we would like to overcome it with our new study.

We refer to the fact that considering a single solver may not be sufficient. When performing the initial tests for the MIPLIB 2010 [7], all three main commercial solvers achieved roughly the same geometric average running time over the whole benchmark set. The speed difference for individual instances, however, was as large as a factor of 1,000 between the fastest and the slowest solver. It depends heavily on the instance which solver is the fastest. When MIPLIB 2010 was released, at least one of the three solvers was able to solve each instance within one hour, but it took years until one *single* solver was capable of solving each instance within an hour. To solve a particular instance, why not use the best solver available? Therefore, it seems natural to us that to discuss the overall performance gain, we use the virtual best solver available at the time, unless otherwise stated. The term virtual best refers to making a decision on an instance-per-instance base, akin to having a perfect oracle that could always make a correct choice on which solver to run.

In this article, all running times are given for the two virtual solvers OLD and NEW, where OLD is the best among the CPLEX Linear Optimizer 7.0.0 (2000), the Xpress-MP Hyper Integer Barrier Optimizer Release 14.10 (2002), and MOSEK Version 3.2.1.8 (2003) for solving LPs. These codes run single-threaded, with the exception of the barrier method LP solvers within XPRESS and MOSEK. The best achievable result was systematically kept. NEW is the best among the IBM ILOG CPLEX Interactive Optimizer 12.10 (2019), the GUROBI Optimizer 9.0 (2020) and the FICO XPRESS Solver 8.11 (2020), as well as MOSEK Version 8.1 (2017) and COPT Version 1.4 (2020) for solving LPs. All solvers were run both sequentially, i.e. single threaded, and in parallel, allowing the solver to use up to eight threads.

Our study focuses on the developments of the past twenty years, for three reasons. The first, festive reason is to focus on the period that EUROPT has been active, following the spirit of this special issue. The second, apparent reason is that this nicely covers the development of LP and MILP solving in the 21st century (so far). The third, most practical and constraining reason is that it was very hard to get old and still running solver binaries. As we experienced, a 20 year period is borderline and in some respect already a too extensive time span. There are no contemporary binaries able to run on the old 32-bit computers. And the old 32-bit binaries already failed to run on one of our newer systems. Furthermore, as can be seen in Table 1 the speed difference between the old code on the old system and the new code on the new computers is already so huge that only few instances are available for meaningful comparison with reasonable effort.

Name	B&B nodes OLD/P-III	B&B nodes NEW/i7	Time [s] OLD/P-III	Time [s] NEW/i7	Speed-up
nw04	131	24	54	1.62	33
mas76	467,454	192,719	198	3.50	57
neos-1122047	64	1	79	1.28	62
mod011	16,671	3,288	926	5.77	160
air05	1,961	2,189	440	2.11	209
qiu	36,452	4,434	1,393	1.49	935
cap6000	16,768	1	268	0.10	2,680
bell5	687,056	915	411	0.03	13,700
neos1171737	28,354	1	116,745	2.67	43,725

Table 1: Comparison of selected instances: OLD on 870 MHz Pentium-III vs NEW on 3.6 GHz i7-9700. Remarks: the slowest of NEW with one thread needs 45 s to solve mas76 on the i7. The speed-up is than exactly the clock ratio between the computers (the solver in OLD and NEW are different though). The biggest speed-up happens when the number of B&B nodes can be reduced. However, whenever there is only one node, no additional speed-up from parallelization occurs.

2. Progress in hardware, software and algorithms

There is a continuous evolution of the performance of LP and MILP solvers due to two main, intertwined drivers, namely the development of computers (Section 2.1) and the algorithmic advances (Section 2.2). These two sources of progress cannot be easily separated. In the following, we will provide experimental results and discuss which factors influenced the change in performance over time and in which direction.

Unless otherwise stated, all computations have been carried out on an 8-core, 8-thread Intel Core i7-9700K CPU @ 3.60 GHz with 64 GB of RAM. It should be noted that modern CPUs adjust their clock speed to the load. The used system might speed up to 4.7 GHz when running only a single thread. Unfortunately, there is no easy way to track which speed was actually used during a particular run. Therefore, 25% and more variation of computing time in the measurements are not uncommon. In an experiment the performance of a single thread halved as we kept the other seven threads busy. For the eight-core runs, this is less pronounced, as the machine is already under almost full load by the task to be performed.

2.1. Progress in hardware and computational environment

In 2001, two of the latest CPUs were the Intel 32-bit Pentium-III at around 1 GHz and the Pentium-4 at 1.5 GHz. IBM offered the 64-bit POWER7 at over

3 GHz. Although 64-bit systems were available, the first 64-bit PC processor was introduced in 2003 and it took quite some years until more than four gigabytes of memory became standard. One should keep in mind that there is a gap of several years between the availability of a new architecture and the common use of it by developers and companies. In the following, we list the major developments in hardware and compilers that came into widespread use during the past twenty years:

Higher clock speed and increased memory bandwidth: both developments also accelerate old code, even if not recompiled.

More efficient processing of instructions: superscalar processors, out-of-order execution, branch prediction, and instruction speed-up. As a consequence, code optimized for an older processor architecture might not perform optimally on a new one. Recompilation is required to fully exploit the improvements.

New instructions: for example, FMA and AVX. To exploit these extensions, the code needs at least to be recompiled. The use of highly optimized subroutines (e.g., ATLAS, OpenBLAS, or IMKL) can provide further speed-up. Barrier solvers often have specific subroutines implemented in instruction-set specific assembly code.

Parallel cores and simultaneous multi-threading (SMT): both have increased the maximal computational performance of a single CPU drastically, however a substantial redesign of the algorithms is needed to exploit them. There is almost no automatic benefit for existing codes. Additionally, SMT in particular makes it even harder to determine the best number of parallel threads to use on a given CPU. If memory accesses are the bottleneck, not using SMT can lead to better running times. This is aggravated by the power management of modern CPUs which can decrease clock frequency in case of an increased number of running threads.

Move from 32-bit to 64-bit addressing/processing: this allows to use more than 4 GB of RAM and to process 64-bit numbers faster. There is no benefit for existing 32-bit codes. Since more memory is used per integer, it possibly can even slow down computations. With some reimplementation, however, 64-bit addressing can contribute to performance gains, e.g., by making hash collisions less likely.

Improved optimizing compilers: from, for example, GCC version 2.95 (2001) to GCC version 10 (2020), compilers have improved a lot and generate better performing code. Recompilation is required to benefit from this.

For an overview of hardware and compiler impact on LP and MILP solver development in the 1980s and 1990s, see [9].

Comparing old and recent architectures is intricate. Old sequential 32-bit codes using the instruction set available in 2001 will not fully exploit modern architectures. Conversely, new parallel 64-bit codes based on recent instructions will not even run on old hardware.

We performed two small tests to estimate the pure hardware speed-up for solving mathematical optimization problems. First, we ran 33 LP instances using the single threaded CPLEX 7 barrier algorithm without crossover on an old 870 MHZ Pentium-III and the new i7-9700 system, and compared the running times. The speed-up is 21 on average, although it varies between 16 and nearly 47, depending on the particular instance.

Since the demands of barrier and simplex algorithms are quite diverse, we performed a second test: we solved min-cost-flow problems with the network simplex code in CPLEX. It can be assumed that this code did not change significantly between version 7.0 and 12.10, as the number of iterations on all instances is identical. We ran 16 publicly available instances in four different settings: CPLEX 7 on a 870 MHz Pentium-III and on an i7-9700, and we ran CPLEX 12 on an otherwise empty i7-9700 and on a fully loaded system. There is no measurable performance difference between the two CPLEX versions regarding the network simplex running on the same hardware. CPLEX 7 running on the 870 MHZ Pentium-III and an empty i7-9700 supposedly running at 4.7 GHz boost speed differ by a factor of 20 on average. However, if we fully load the system with a multi-core `stream` benchmark [10], the performance is halved. One should bear in mind that for each situation, it is not clear where the bottleneck is. The network simplex is known to be highly dependent on the performance of the memory subsystem. Overall, the hardware speed-up we experienced was not constant. The minimum factor is around 15 (i7 empty) and seven (i7 loaded), and the maximum we experienced was more than 45. We would like to point out that small differences in running times are not significant and that the overall impact of the compiler seems small.

The hardware landscape has been changing even more dramatically in the last 10 years, during which GPU accelerators have become widely available. However, as of 2020 (to the best of our knowledge), none of the state-of-the-art LP/MILP solvers exploits them. Indeed, GPUs are tailored much towards dense processing, while solvers rely heavily on super-sparse linear algebra.

2.2. Progress in algorithms

Two decades of research certainly led to significant algorithm improvements. The improvements for MILP include many new heuristic methods, such as RINS [11] and local branching [12], several classes of new or improved cutting planes, e.g., MCF cuts [13], and a large number of *tricks for the bag*, such as conflict analysis [14], symmetry detection [15], solution polishing and dynamic search. Most of them either exploit some special structure in the instances, or address a shortcoming in the algorithm for a particular class of problems. Furthermore, codes have been ported to 64-bit addressing and are therefore able to utilize larger amounts of memory. Moreover, many algorithms have been parallelized [16], in particular the MILP tree search and barrier methods for LP solving.

In the area of LP solving, theoretical progress has been quite limited. There were nonetheless numerous improvements to deal with difficult instances. In general, the linear algebra has been sped up by (better) exploiting hyper sparsity and using highly optimized subroutines. Preprocessing got better. The parallelization of the barrier methods improved, and there exists nowadays a parallel variant of the simplex algorithm, although its scalability is limited [17]. Nevertheless, with very few exceptions other than due to sheer size, LPs that can be solved today can also be solved with the old codes, provided one is willing to wait long enough. As Figure 1 shows, LP solvers have become approximately nine times faster since the beginning of the millennium. One should note that it is often not the same algorithm of the same solver that became faster, but the fastest choice now is faster than the fastest choice then. Additionally, the ability to solve huge instances has improved considerably.

In the computational study done in 1999, Bixby et al. [1] used a 400 MHz P-II with 512 MB of memory. The largest LP that this machine could handle had 1,000,000 rows, 1,685,236 columns, and 3,370,472 non-zeros. In a workshop in January 2008 on the *Perspectives in Interior Point Methods for Solving Linear Programs*, the instance `zib03` with 29,128,799 columns, 19,731,970 rows and 104,422,573 non-zeros was made public. As it turned out, the simplex algorithm was not suitable to solve it and barrier methods needed at least about 256 GB of memory, which was not easily available at that time. The first to solve it was Christian Bliek in April 2009, running CPLEX out-of-core with eight threads taking 12,035,375 seconds = 139 days to solve the LP without crossover. Each iteration took 56 hours! Using today's codes on a machine with 2 TB memory and 4 E7-8880v4 CPUs @ 2.20 GHz with a total of 88 cores, this instance can be solved in 59,432 seconds = 16.5 hours with just 10% of the available memory used. This is a speed-up of 200 within 10 years. However, when the instance was introduced in 2008, none of the codes was able to solve it. So there was infinite progress in the first year. This pattern is much more pronounced with MILP. First is the step from unsolvable to solved. This is almost always due to algorithmic improvements. Then there is a steady progress both due do algorithmic and hardware improvements until the instance is considered easy. From then on, if any at all, speed-ups are mostly by hardware only. The largest LP in XPRESS' instance collection of practically relevant models has more than 200,000,000 columns and more than 1,300,000,000 non-zeros. It can be solved in about one and a half hours. Solving an instance of this size was impossible with off-the-shelf hardware and software in 2001.

Before describing our computational experiments in more detail, note that there are a few caveats to bear in mind:

- Since we are interested in the performance of the overall domain, we will compare the virtual best solver OLD from around 2001 (consisting of XPRESS, CPLEX, and MOSEK) with the virtual best solver NEW from 2020 (consisting of XPRESS, CPLEX, GUROBI, MOSEK, and COPT).

- It could be argued that the parameter defaults are better tuned now and therefore the old codes would benefit more from hand-tuned parameters than the new ones. At the same time, the new codes have more parameters to tune and considerably more sub-algorithms that can be employed. We decided that it is out of the scope of this study to try to hand-tune every instance and therefore only the default values of the solvers will be used.

- Benchmarking got more prominent and fierce during the last decade, in particular until 2018 (see Mittelmann [18]). There has been considerable tuning on the MIPLIB instances, especially on MIPLIB 2010 and MIPLIB 2017. It is fair to say that this clearly benefits the newer solvers and might lead to an overestimation of the progress.

- It should also be noted that instances `dano3mip, liu, momentum3, protfold` and `t1717` from MIPLIB 2003 still remain unsolved, although substantial effort was put into solving them to proven optimality. Furthermore, there are several old instances which still cannot be solved in a reasonable amount of time without reformulation or special purpose codes..

3. Computations

As demonstrated numerous times, the test set and experimental setup have a crucial influence on the outcome of computational studies. The main question is how to handle instances that one actor of the comparison cannot solve. In our case – not too surprisingly – NEW is able to solve any instance that OLD can solve, but not vice versa. When comparing solvers, it is customary to set the maximum run time allowed as time for the comparison. This is reasonable if one compares two solvers on a pre-selected test set. In our case, the test set and the run time can be chosen; this means that any speed-up can be attained by increasing the number of instances that the old codes cannot solve and increasing the run time allowed. Therefore, we decided to split those questions.

3.1. Test set selection

As a test set for LPs, we chose the instances used by Hans Mittelmann for his LP benchmarks [18] which are listed in Table 2:

`buildingenergy`	`cont11`	`cont1`	`cont4`
`datt256`	`ds-big`	`ex10`	`fhnw-binschedule0`
`fome13`	`graph40-40`	`irish-e`	`L1_sixm250obs`
`Linf_520c`	`neos-3025225`	`neos-5052403-cy`	`neos-5251015`
`neos`	`nug08-3rd`	`pds-100`	`physiciansched3-3`
`qap15`	`rail02`	`rail4284`	`rmine15`
`s100`	`s250r10`	`s82`	`savsched1`
`scpm1`	`set-cover-model`	`shs1023`	`square41`
`stormG2_1000`	`stp3d`	`supportcase10`	`tpl-tub-ws1617`

Table 2: LP instances of Hans Mittelmann's *Benchmark of Simplex LP solvers (1-18-2021)* and *Benchmark of Barrier LP solvers (12-28-2020)*

The following instances were excluded from our tests because either OLD solved them in under 10 seconds or NEW solved them in less than one second: `chrom1024-7, neos3, ns1688926, self, stat96v1`.

Additionally, we included the following instances from previous Mittelmann LP benchmarks in the test set: `dbic1`, `ns1644855`, `nug15`, and we further added three larger models, named `engysys1, engysys2, engysys3`, which we currently use for real-world energy systems research to have some checks with instances that did not appear in any benchmark so far.

Finally, the LP relaxations of all MIPLIB-2017 benchmark instances are added to the set of LP. Again, ignoring all instances for which OLD solved them in under 10 seconds or NEW solved them in less than one second resulting in the instances shown in Table 3.

`bab2`	`ex9`	`highschool1-aigio`	`k1mushroom`
`neos-2075418-temuka`	`neos-3402454-bohle`	`neos-5104907-jarama`	`ns1760995`
`opm2-z10-s4`	`rail01`	`splice1k1`	`square47`
`supportcase19`	`triptim1`		

Table 3: MIPLIB 2017 as LP instances

For LPs, OLD eventually solved any of our test instances that NEW can solve given enough time, provided the available memory is sufficient. One can argue about this selection. Since we compare virtual best solvers, the performance (or lack thereof) of a particular solver does not matter. It is safe to say that these instances were selected because they pose some problems to existing solvers. Therefore, the speed-up experienced on these instances is likely in favor of NEW.

For the MILP comparison we used three versions of MIPLIB, namely, MIPLIB 2017, MIPLIB 2010, and MIPLIB 2003.

It should be noted that we excluded all instances that could not be solved by NEW within six hours as there is little chance that OLD will solve them. This affect the following instances `neos-3024952-loue`, `splice1k1`, `s100` (from MIPLIB 2017) and `ds`, `timtab2`, `swath`, `mkc`, `t1717`, `liu`, `stp3d`, `dano3mip`, `momentum3` (from MIPLIB 2003). Instances `mas74` (from MIPLIB 2017) and `m100n500k4r1`, `dfn-gwin-UUM` (from MIPLIB 2010) were omitted for numerical reasons. Furthermore, we excluded any instance for which a meaningful speed-up could not be computed due to the limited precision of our timing. This happens if OLD converges within less than ten seconds, and NEW under one second. This led to the exclusion of the following 19 instances: `modglob`, `gesa2`, `set1ch`, `irp`, `p2756`, `pp08aCUTS`, `fiber`, `sp150x300d`, `neos-827175`, `drayage-100-23`, `fixnet6`, `p200x1188c`, `gesa2-o`, `swath1`, `pp08a`, `vpm2`, `neos-1171448`, `tanglegram2`, `cbs-cta`. Note that `nw04` stayed in the test set, because, though OLD solved it in just two seconds, NEW needed more than one second, i.e., two seconds, to solve it. Therefore, a meaningful comparison was possible.

3.2. Explanation of Figure 1

Figure 1 aggregates the results for all instances, both LP (left) and MILP (right), that could be solved within 24 hours by both OLD and NEW. In total, those are 56 of 60 LP and 105 of 339 MILP instances. Each symbol represents a single instance, with the x coordinate corresponding to the running time of the virtual best old solver and the y coordinate corresponding to the running time of the virtual best new solver. Note that both axes are log-scaled and that we needed one order and two orders of magnitude more to represent the old running times for LP and MILP instances, respectively. The slowest instance that the old codes could solve in a day took less than three minutes for the new codes. We clipped the times to one second, as this is the precision we could measure.

The dotted grey diagonal is the break-even line. Any instance where OLD is faster than NEW would be above this line. As you can see, this is not the case for any instance. However, there is one MILP instance (`nw04`), for which both virtual solvers took roughly the same time. Several instances lie on the clipped one second line, with running times for OLD of up to 6,000 seconds. All of these have become trivial to solve for NEW. A reason might be the empirical observation that increasing the allowed running time has a diminishing effect. We will discuss this further in the conclusions related to Figure 3.

In each plot, a colored diagonal line represents the shifted geometric mean of the speed-up factor for LP and MILP instances, respectively. All instances (of the corresponding problem type) above the line show a speedup factor less than the mean speedup, all instances below show an even larger speedup. As you can see, there are some rather extreme cases, in particular for MILPs. While the LPs are more concentrated around the mean line, there is no significant difference between MIPLIB 2010 and MIPLIB 2017 instances.

These shifted geometric means are one of the main findings. For instances that could be solved by both, OLD and NEW, the pure algorithmic speedup of LP solving was about a factor of nine in the last twenty years and the speedup of MILP solving was about 30.

3.3. Explanation of Figure 2

Figure 2 considers instances that are missing in Figure 1: instances that could only be solved by new codes, here specifically the 149 (out of 240) instances from MIPLIB 2017 for which this holds. We see the instances sorted by the running time of the virtual best new solver. Note that this is a linear scale, not a log scale.

Except two instances, all instances were solved in less than one hour and the majority (123 of 149) in less than ten minutes. While for most instances, one of the eight-threaded solvers is fastest, for the hardest instances that require more than half an hour with NEW, a single-threaded solver wins in four out of 13 cases.

This seemingly counter-intuitive behavior is explained by the fact that these are instances that do not go into tree search, but consist of having to solve a hard root node. This explains both, that they do not benefit from tree search parallelization and that they still take considerable time with NEW, given that the overall solution time is dominated by a hard initial LP.

Of course, modern MILP solvers also use parallelization at the root node. However, this mostly happens in terms of concurrent algorithms: Different LP solvers run concurrently to solve the root LP relaxation; two alternative cut loops run concurrently, and the better one is chosen to continue with at the end of the root. Concurrent optimization is excellent for hedging against worst-case behavior, but it is inherently slower when the default option would have won in either case. In such a situation, the additional variants (like running primal simplex and barrier for LP solving or an alternative cut loop) compete for the same resources, and deterministic synchronization might lead to idle times. In our experiment, the CPU uses turbo boost for single-thread runs, even amplifying this situation. Thus, we would expect a single-threaded solver to win on an instance that can be solved without much branching and for which dual simplex is the fastest algorithm to solve the initial relaxation.

One main ideal in speeding up the MILP solution process is to reduce the number of branch-and-bound nodes needed. Nearly all new methods mentioned above, heuristics, cutting planes, conflict analysis, symmetry detection, dynamic search, aim at reducing the number of nodes. The final success is reached, once an instance can be solved in the root node. The progress in this regard is quite visible: From our 339 instances, OLD solved two in the root node, while NEW was able to achieve this with 25 instances. Unfortunately, the current main direction in hardware development is to increase the number of available threads and the main benefit from parallelization is the ability to process more nodes in parallel. Therefore, hardware and algorithmic development are to a certain extend now non-synergistic. This was different in the past.

3.4. Take the results with a grain of salt

We refrained from aggregating instances that could be solved by both, OLD and NEW, and instances that could only be solved by NEW, into a single score.

While this might be done by using a time limit for unsolved instances and possibly even a penalty, it can easily skew results. As a most extreme example, consider using a PAR10 measure (hence weighing all timeouts with a factor of ten times the time limit). Due to the high number of instances that cannot be solved by the old codes, we would obtain almost arbitrarily large speed-up numbers. Setting the time limit to 24 hours and using PAR10, using the average solving time of 85 seconds achieved by NEW for the instances unsolved by OLD, we would get a potential speed-up of more than 10,000. Without PAR10 it would still be over 1,000, and using a timeout of one hour we would get 44. Only the last one is close to the speed-up observed on instances that both versions could solve.

It seems much more sound to report the speed-up factor on solved instances, 36, and the large number of instances (156/240) that could be solved by NEW, but not by OLD. This also shows where the biggest progress in LP and MILP solving lies: *Making new problems and whole problem classes tractable.*

4. Conclusion

4.1. LP

For LPs, we computed an average speed-up factor of nine. Combining this with the hardware speed-up, we can conclude that solving LPs got about 180 times faster in the last two decades. However, the main difference comes from the switch to 64-bit computing, allowing to solve much bigger instances, in particular with parallelized barrier codes.

Furthermore, it is fair to say that the solver implementations became ever more refined, leading to extremely stable codes. At the same time, little progress has been made on the theoretical side.

4.2. MILP

In this case, the picture gets more diverse. For those instances solved with the old codes, the average speed-up due to improved algorithms (Figure 1) is about 30. Combining this with the hardware speed-up, we find an average total speed-up for solving MILPs of 600: five minutes then become less than a second now. Solving MILPs got faster more than 30% every year during the last 20 years.

The most impressive result is shown in Figure 2: the vastly increased ability to solve MILP instances at all. 149 of 240 instances (62%) from the MIPLIB 2017 benchmark cannot be solved by any of the old solvers within a day, even on a modern computer. In contrast, the geometric mean running time for solving these instances by NEW is just 104 seconds.

To summarize, in 2001, one would be pleasantly surprised if one of the solvers would readily solve an arbitrary MILP instance. Nowadays, one is unpleasantly surprised if none of the solvers can tackle it.

Figure 3 depicts the effect. The number of instances that are solvable right away is ever increasing, but the shape of the frontier stays identical; it is simply pushed to the right. However, it is important to note that the instances on the left are precisely the ones that we wish to solve.

If we just speed-up computations, the curve will become more L-shaped but not move. This is what happens, e.g., due to (better) parallelization, or if we use

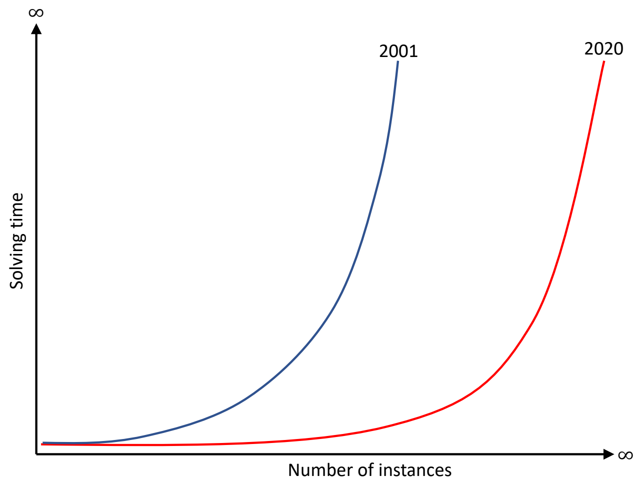

Figure 3: The number of instances that are (quickly) solvable is monotonically increasing over time and the frontier of "difficult" instances is pushed further to the right

the old codes on a new machine. However, it does not change much regarding the overall solvability of instances. To really move the curve to the right, algorithmic improvements beyond pure computational speed-ups are needed.

4.3. Outlook

A problem we foresee for the future is diminishing returns: As can be deduced from the results, having more and faster cores will not improve the solvability significantly. There is only one instance that OLD solves within 24 hours, but not within one hour. As [19] describes, there are individual instances that can be solved by massive amounts of computing power, however, there are few of them. A similar situation is true regarding memory. There are, without doubt, some extremely large instances. However, the number of instances that require terabytes of memory are few. And if they do, scaling to higher numbers of cores does not work particularly well, due to limited memory bandwidth. There are special algorithms for distributed systems like, e.g., [20], but these are still far from becoming usable by out-of-the-box solvers. Seeing the change in computer architecture, in particular GPUs, it becomes increasingly challenging for the solvers to fully exploit the available hardware. This opens interesting directions for research.

A similar observation can be made for the algorithmic side. Overall, it is experienced that every added algorithmic idea affects an increasingly smaller subset of instances. So as always, our hope has to be for breakthrough ideas to appear. However, so far, solvers still provide significant algorithmic improvements with every release. While additional speed-up by hardware has gone mostly stale, LP and MILP solvers are still going strong.

Acknowledgements

The work for this article has been conducted in the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 05M20ZBM).

This work has been supported by the German Federal Ministry of Economic Affairs and Energy (BMWi) through the project UNSEEN (fund no 03EI1004D): *Bewertung der Unsicherheit in linear optimierten Energiesystem-Modellen unter Zuhilfenahme Neuronaler Netze.*

We thank Carsten Dresske for providing us with access to a still nicely running Pentium-III powered computer.

We thank IBM for providing us with CPLEX Version 7, FICO for providing us with XPRESS Version 14, and MOSEK for providing us with MOSEK Version 3.

References

[1] E. R. Bixby, M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling, MIP: Theory and practice — closing the gap, in: M. J. D. Powell, S. Scholtes (Eds.), System Modelling and Optimization, Springer US, Boston, MA, 2000, pp. 19–49.

[2] R. E. Bixby, Solving real-world linear programs: A decade and more of progress, Operations Research 50 (2002) 3–15. doi:10.1287/opre.50.1.3.17780.

[3] R. E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling, Mixed-integer programming: a progress report, in: M. Grötschel (Ed.), The Sharpest Cut: The Impact of Manfred Padberg and His Work, MPS-SIAM Series on Optimization, SIAM, 2004, pp. 309—325. doi:10.1137/1.9780898718805.ch18.

[4] T. Achterberg, R. Wunderling, Mixed integer programming: Analyzing 12 years of progress, in: Facets of Combinatorial Optimization, Springer, 2013, pp. 449–481. doi:10.1007/978-3-642-38189-8_18.

[5] A. Lodi, Mixed integer programming computation, in: 50 years of Integer Programming 1958–2008, Springer, 2010, pp. 619–645. doi:10.1007/978-3-540-68279-0.

[6] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, Y. Shinano, MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library, Mathematical Programming Computation (2021). doi:10.1007/s12532-020-00194-3.

[7] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, K. Wolter, MIPLIB 2010, Mathematical Programming Computation 3 (2011) 103–163. doi:10.1007/s12532-011-0025-9.

[8] T. Koch, A. Martin, M. E. Pfetsch, Progress in academic computational integer programming, in: Facets of Combinatorial Optimization, Springer, 2013, pp. 483–506. doi:10.1007/978-3-642-38189-8_18.

[9] R. Ashford, Mixed integer programming: A historical perspective with Xpress-MP, Annals of Operations Research 149 (2007) 5.

[10] J. D. McCalpin, STREAM: Sustainable memory bandwidth in high performance computers, 2013. https://www.cs.virginia.edu/stream.

[11] E. Danna, E. Rothberg, C. L. Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming 102 (2004) 71–90. doi:10.1007/s10107-004-0518-7.

[12] M. Fischetti, A. Lodi, Local branching, Mathematical Programming 98 (2003) 23–47. doi:10.1007/s10107-003-0395-5.

[13] T. Achterberg, C. Raack, MCF-separator: detecting and exploiting multicommodity flow structures in MIPs, Mathematical Programming Computation 2 (2010) 125–165. doi:10.1007/s12532-010-0015-3.

[14] T. Achterberg, Conflict analysis in mixed integer programming, Discrete Optimization 4 (2007) 4–20. doi:10.1016/j.disopt.2006.10.006.

[15] F. Margot, Exploiting orbits in symmetric ILP, Mathematical Programming 98 (2003) 3–21. doi:10.1007/s10107-003-0394-6.

[16] T. Berthold, J. Farmer, S. Heinz, M. Perregaard, Parallelization of the FICO Xpress-Optimizer, Optimization Methods and Software 33 (2018) 518–529. doi:10.1080/10556788.2017.1333612.

[17] Q. Huangfu, J. A. J. Hall, Parallelizing the dual revised simplex method, Mathematical Programming Computation 10 (2018) 119–142. doi:10.1007/s12532-017-0130-5.

[18] H. Mittelmann, Benchmarks for Optimization Software, 2020. http://plato.asu.edu/bench.html.

[19] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, M. Winkler, Solving Previously Unsolved MIP Instances with ParaSCIP on Supercomputers by using up to 80,000 Cores, Technical Report 20-16, ZIB, Takustr. 7, 14195 Berlin, 2020.

[20] D. Rehfeldt, H. Hobbie, D. Schönheit, A. Gleixner, T. Koch, D. Möst, A massively parallel interior-point solver for linear energy system models with block structure, Technical Report 19-41, ZIB, Takustr. 7, 14195 Berlin, 2019.