A Brief History of Linear and Mixed-Integer Programming Computation

Robert E. Bixby

2010 Mathematics Subject Classification: 90C05, 90C10
Keywords and Phrases: Linear programming, mixed-integer programming, simplex algorithm, branch-and-bound, implementation, computer

The early years

For many of us, modern-day linear programming (LP) started with the work of George Dantzig in 1947. However, it must be said that many other scientists have also made seminal contributions to the subject, and some would argue that the origins of LP predate Dantzig's contribution. It is matter open to debate [36]. However, what is not open to debate is Dantzig's key contribution to LP computation. In contrast to the economists of his time, Dantzig viewed LP not just as a qualitative tool in the analysis of economic phenomena, but as a method that could be used to compute actual answers to specific real-world problems. Consistent with that view, he proposed an algorithm for solving LPs, the simplex algorithm [12]. To this day the simplex algorithm remains a primary computational tool in linear and mixed-integer programming (MIP).

In [11] it is reported that the first application of Dantzig's simplex algorithm to the solution of a non-trivial LP was Laderman's solution of a 21 constraint, 77 variable instance of the classical Stigler Diet Problem [41]. It is reported that the total computation time was 120 man-days!

The first computer implementation of an at-least modestly general version of the simplex algorithm is reported to have been on the SEAC computer at the then National Bureau of Standards [25]. (There were apparently some slightly earlier implementations for dealing with models that were "triangular", that is, where all the linear systems could be solved by simple addition and subtraction.) Orchard-Hays [35] reports that several small instances having as many as 10 constraints and 20 variables were solved with this implementation.

The first systematic development of computer codes for the simplex algorithm began very shortly thereafter at the RAND Corporation in Santa Monica, California. Dantzig's initial LP work occurred at the Air Force following the end of World War II, influenced in part by military logistics problems that arose during the war. In 1952 Dantzig moved from the Air Force to the RAND Corporation, apparently with the specific purpose of focusing on the further development of his fundamental new ideas. Part of the effort was to build computer implementations of the simplex algorithm, and Orchard-Hays was assigned the task of working with Dantzig. The result was a four-year collaboration at RAND that laid the foundation for the computational development of the subject.

The start did not go smoothly. The simplex algorithm was at that point far from a well-defined computational procedure, and the computers of the day were nothing like what we think of as a computer today. Their first implementation used a device known as a Card Programmable Calculator (CPC). As the name suggests, it wasn't really a computer, but as Orchard-Hays [35] described it "an ancient conglomeration of tabulating equipment, electro-mechanical storage devices, and an electronic calculator (with tubes and relays), long since forgotten. One did not program in a modern sense, but wired three patch-boards which became like masses of spaghetti". The first implementation computed an explicit inverse at each iteration, and Dantzig was appalled when he saw the result [35]; the future of the simplex algorithm didn't look promising. He then recalled an idea proposed to him by Alex Orden, the product-form of the inverse. This method, which remained a staple of simplex implementations for over twenty years, was the starting point for a second and more successful CPC implementation. It was reportedly capable of handling LPs with up to 45 constraints and 70 variables and was used to solve a 26 constraint, 71 variable instance of the Stigler model. Total computation time was reported to be about 8 hours, a good portion of that time being spent manually feeding cards into the CPC. That was 1953.

In 1954–55 the algorithms were improved and re-implemented on an IBM 701, IBM's first real "scientific computer". This implementation could handle LPs with 101 constraints, and was used in extensive computations on a model devised by the economist Alan Manne. This appears to have been the first real application of the simplex algorithm.

The 701 implementation was followed in 1955–56 by an implementation for the IBM 704. This code was capable of handling LPs with up to 255 constraints, including explicit constraints for any upper bounds. It became known as RSLP1, and seems to have been the first code to be distributed for use by a wider audience. It was later improved to handle 512 constraints, and released for use by CEIR, Inc. around 1958-59 under the name of SCROL. LP was coming of age and beginning to enjoy significant use in the oil industry.

Orchard-Hays moved from RAND to CEIR in Arlington, Va., in 1956 and began the development of the LP/90 code for the IBM 7090. It was capable of handling up to 1024 constraints. LP/90 was released in 1961-62, with improvements continuing into 1963. This code was followed by LP/90/94 for the IBM 7094, released in 1963/64. This code was then taken over by CEIR, Ltd. in the UK. The LP/90/94 code can fairly be characterized as the culmination of the first-generation of LP codes, and this seems to be the last really successful code over which Orchard-Hays had significant influence. With it a new generation of developers emerged to continue the computational development of LP and, in the not-to-distant future, MIP. A key figure in motivating these developments was E. M. L. (Martin) Beale in the UK. Among those who worked with Beale and were influenced by his vision of mathematical programming were R. E. Small and Max Shaw, followed by John Tomlin and John Forrest, both of whom continue to influence the field to this day.

LP/90/94 was also a milestone because it became, by all accounts, the first commercially used MIP code based upon branch-and-bound [9]. The conversion of this code to handle mixed-integer problems seems to have been initiated around 1964–65 by Beale and Small [4]. They used an approach suggested by Land and Doig [29] with dichotomous branching as proposed by Dakin [14]. This code was then taken over by Max Shaw in 1965 [39]:

> Back in the 60s the IBM 7094 was a 36 bit word machine with 32K words of storage. It was nevertheless a super computer of its time. A team in the USA at CEIR INC. lead by William Orchard-Hays wrote a standalone LP system (LP 90/94) that mixed linear programming with brilliant system design that could solve LP problems up to 1000 rows or so. This code was written exclusively in 7094 machine code and used all manner of advanced techniques to maximise computing efficiency. I never met Bill Orchard-Hays and his team but when I studied their code I was most impressed.
>
> The revised simplex method of George Dantzig was implemented such that the transformation vectors (we called them etas) were held on tape and were read forward to update vectors being expressed in terms of the basis, added to etas for vectors brought into the basis, and read backward to compute the price or feasibility advantage of vectors to be brought into the solution.

Shaw reports that this code was used in the first successful applications of MIP, which included:

- Re-location of factories in Europe by Philips Petroleum
- The selection of ships and transport aircraft to support deployment of UK military assets
- Refinery infrastructure investments by British Petroleum
- Selecting coal mines for closure by the UK National Coal Board

In his own words:

> There was some excitement for customers using the LP 90/94 system in 1967-8 as they had never been able earlier to get optimal results to their mixed-integer models.

This really demonstrated for the first time, contrary to common belief, that a search procedure based on branch-and-bound could be used to solve real-world MIPs to optimality. That was true in spite of the fact that the algorithmic opportunities on the machines of the day were severely limited. Again, quoting Shaw:

> The version of the 7094 used by CEIR only had tape storage. This caused us to search to the bottom of each branch of the tree of bounded solutions until we got an integer value; and then track back up the tree using the bound obtained from the best integer solution so far.

The 70s and 80s: The next generation

This brings us to the 1970s. The computational aspects of the subject were now close to twenty years old and both LP simplex codes and branch-and-bound codes for MIP, though primitive, were available. It was in a very real sense the end of the Orchard-Hays era, one strongly influenced by his pioneering implementations of the simplex algorithm. It also marked the introduction of the IBM 360 class of computers. The expanded capabilities of these machines meant not only that problems could be solved more quickly, but perhaps more importantly that new ideas and methods could be tried that would have been unworkable on the previous generation of computers. It was also the beginning of a period of great promise for linear and mixed-integer programming.

For LP, important ideas such as the implicit treatment of bounds within the simplex algorithm, which reduced the number of explicit constraints in the model, the use of LU-factorizations, the use of sophisticated LU-updates, based upon the Forrest-Tomlin [18] variant of the Bartels-Golub [3] update, and improved variable-selection paradigms such as devex pricing, as proposed by Paula Harris at British Petroleum [24]. The dual simplex algorithm, proposed by Lemke in 1954 [30] also became a fairly standard part of LP codes, though its use was restricted almost exclusively to re-optimization within MIP branch-and-bound trees (amazingly, the ability to explicitly deal with dual infeasibilities does not seem to have emerged until the mid-1990s). The basic form of these algorithms, developed in the early 70s, seems to have remained more-or-less constant into the mid-1980s. The implementations were almost exclusively written in assembler code and highly tuned to exploit the specific characteristics of the target machine.

On the integer programming side there was also major progress. A number of completely new codes were introduced. These new codes offered a tight integration between the underlying LP solver and MIP code. And the MIP codes themselves became much more sophisticated. Tree search moved beyond the very inefficient LIFO search dictated by earlier computer architectures. Sophisticated node and variable selection procedures were developed, including the important notion of pseudo-costs, still heavily in use today. Many of these developments are nicely documented in [19] and [28]. The net result was that MIP was developing into a more powerful tool beginning to see more extensive applications in practice. However, while these codes did continue to be refined and improved, at a certain fundamental level they also remained in a largely unchanged form. Indeed, they remained largely unchanged until the late-1990s! This is a remarkable testimony to their effectiveness. However, it was also a form of roadblock to further developments in the subject: they made MIP a viable practical tool for the first time, but they also helped create totally unrealistic expectations for what remained fundamentally a primitive technology.

The first generation of these new codes, developed and/or released around 1970, included FMPS [40], UMPIRE [17], MPSX [5], MPS III, and APEX. These were followed by the introduction of MPSX/370 (for the IBM 370) around 1974 [6], an improved version of MPSX, SCICONIC around 1976, an improved version of UMPIRE, and finally APEX III, the final version of the APEX codes, released around 1982. (See [19] and [28] for further details on these systems.) And in 1980 the Whizard extension of MPS III was developed at Ketron, which had earlier purchased MPS III from Management Science. Whizard was developed jointly by Eli Hellerman and Dennis Rarick, but also worked on extensively by John Tomlin and Jim Welch among others at Ketron [43]. It was a remarkable LP code for its time, including very efficient LU-factorization and LU-update capabilities, and among the first really successful presolve and postsolve capabilities for LP, based to some extent on ideas from the apparently quite advanced FMPS presolve implementation [43].

During this period, two additional important developments occurred. In 1977, the MINOS code, developed at Stanford primarily by Michael Saunders, was released. This was primarily a non-linear programming code, but included a very good, stable implementation of the primal simplex algorithm. Around the same time, in 1979, the XMP code developed by Roy Marsten, using the Harwell LA05 linear-algebra routines, was also released [32]. Both codes were written in portable FORTRAN, and were among the first portable codes in general use. (Some earlier versions of FMPS and UMPIRE were also written in FORTRAN.) Moreover, XMP had an additional, important property: it was written with the idea that it could be embedded in other codes, and thus used as a LP-solving-subroutine in "larger" LP-based algorithmic procedures. The most powerful solvers of the day, written largely as closed systems, were not easily used in this way and represented a serious hindrance most particularly to research in integer programming. This situation is well described by remarks of Grötschel and Holland [21], commenting on their use of MPSX/370 in work on the traveling salesman problem. They note that if the LP-package they were using had been "better suited for a row generation process than MPSX is, the total speed-up obtained by faster (cut) recognition procedures might be worth the higher programming effort".

Another key development during this period was the introduction around 1980 of the IBM personal computer (PC). Personal computers were not new at that time, but the release of the IBM PC marked the beginnings of the business applications of PCs, and it was the event that led to the realization that PCs could be used as platforms for the development of practical LP and MIP codes. It was several years before widely-available MIP codes for PCs were developed, but LP codes began to emerge rather quickly, probably as early as 1983. Sharda and Somarajan [38] report on several such codes, including early versions for the still commonly used LINDO code. The first versions of the XpressMP [15] code were also finding industry use [2] in 1983.

Of course the PCs available in those days were a mere shadow of the powerful desktop computers now available. In [38] computational results were reported for a number of PC codes, including LINDO, comparing these codes to MPSX/370 on a small set of LP test problems. The PC codes were run on an IBM PC with an 8087 math co-processor and 640K of RAM. MPSX was run on an IBM 3081D mainframe. LINDO was written in FORTRAN, as presumably were most of the PC codes of that time. Based upon the LINPACK benchmarks for those machines (http://www.netlib.org/benchmark/performance.pdf), one could estimate that the 3081D was roughly 15 times faster than the PC being used. The largest instances used in [38] had roughly 1000 constraints and 1000 variables. LINDO solved 14 of the 16 instances, the best of any of the PC codes tested, taking 5100 seconds in one case, while MPSX was never slower than 13 seconds on any of the models, and solved all 16. Based upon the geometric means of the ratios of the solution times for LINDO versus MPSX/370, LINDO was slightly more than 166 times slower! A fair conclusion from these numbers was that PC codes did, in some cases, provide a useful alternative to the powerful mainframe codes of the day, but were still far behind in overall performance, even taking into account the differences in machine speed. These results seem to confirm the general feeling at the time that LP codes had reached a final level of maturity. Machines would no doubt get faster, but after nearly 40 years of development, the simplex algorithm was viewed as not likely to see further significant improvements. Events were to prove this belief to be totally wrong.

Two additional developments occurred during this period that would have fundamental effects on the future of LP (and hence MIP). In 1979, L. Khachiyan [27] showed for the first time that LPs could be solved in polynomial time. This was not an unexpected result, given the fact that LP was known to be in NP and co-NP; nevertheless, it was a fundamental advance, not least of which because of its important theoretical implications in the theory of combinatorial optimization [22]. The applicability to LP computation was however limited and this use of Khachiyan's algorithm was quickly abandoned.

Modern LP codes

The work of Khachiyan was followed in 1984 by the paper of N. Karmarkar [26]. Karmarkar used projective transformations to demonstrate a polynomial-time bound for LP that was not only far better than the bounds for Khachiyan's method, it also corresponded to a computational approach that was applicable in practice. Karmarkar's paper led to a remarkable flurry of theoretical work in linear programming and related areas that, in many ways, continues to this day in convex programming and related subjects [37].

On the computational side, AT&T developed the KORBX system [8], in what turned out to be a largely unsuccessful attempt to commercially exploit Karmarkar's breakthrough. However, at the same time, researchers were quick to recognize the connections between Karmarkar's theoretical contribution and earlier work of Fiacco and McCormick on log-barrier methods. This realization eventually led to the development of a class of algorithms known as primal-dual log-barrier algorithms. These results are well documented on the computational side in the work of Lustig, Marsten, and Shanno [31], who developed the OB1 FORTAN code implementing early versions of this log-barrier algorithm. This code was generally available around 1991 and together with the improvements happening during that same period with simplex algorithms – in codes such as CPLEX and OSL – this spelled the end for the KORBX code. While OB1 itself also failed to be commercially successful, it nevertheless was the leading barrier code of its day and generated an enormous amount of interest and activity.

The period around 1990 was a remarkably active period in LP. The work of Karmarkar had stimulated a rebirth of interest in LP, both on the theoretical and computation sides. Not only did this lead to a better understanding and improved implementations of barrier algorithms, it also led to a rebirth of interest in simplex algorithms and is responsible to a degree for some of the early developments in the CPLEX LP code, first released in 1988. At about the same time, IBM also released its OSL code, the designated replacement for MPSX/370, developed primarily by John Forrest and John Tomlin. These two codes – CPLEX and OSL – were the dominant LP codes in the early 1990s, and included implementations of both primal and dual simplex algorithms as well as, eventually, barrier algorithms. For the CPLEX code, many of these developments are documented in [7]. Among the most important advances that occurred during this time were the following:

- The emergence of the dual simplex algorithm as a general purpose solver (not just restricted to use in branch-and-bound algorithms)
- The development of dual steepest-edge algorithms (using a variant proposed in [16])
- Improved Cholesky factorization methodology for barrier algorithms and the introduction of parallelism in these algorithms
- Vastly improved linear algebra in the application of simplex algorithms for large, sparse models [20].

In [7] I reported in detail on the overall improvements in the CPLEX LP code from 1988 through 2002, and subsequently updated these results in 2004.

The following is a summary of these results:

	Improvement factor
Algorithmic improvement (machine independent)	
Best of barrier, primal simplex, and dual simplex:	3300×
Machine improvement:	1600×
Total improvement $(3300 \cdot 2000)$:	5,280,000×

These results show that in a period of sixteen years, from 1988 to 2004, by at least some measure, the average speed of at least one LP code – independent of any machine effects – improved by a factor of roughly 3300, far in excess of the improvements in the speed of computing machines over that same period; moreover, combining the effects of the algorithms and the machines gives an improvement factor exceeding six orders of magnitude, nothing short of remarkable.

Note that we have used here as our algorithm the best of barrier, primal, and dual. One can argue whether this is a legitimate approach, but it is the one that I have used. It means that, for each model in the test set, each of the three algorithms was run, and the solution time of the fastest of the three was taken as the solution time for the model. It should also be noted that crossover to a basis was used in all cases when the barrier algorithm was applied. This was done in large part because, in all of the major commercial implementations of barrier algorithms, crossover is considered an integral part of the algorithm. It serves to compensate for the numerical difficulties often encountered by barrier algorithms. In addition, the vast majority of LPs that are solved from scratch in practice are the root solves of MIPs, and a basis is then essential to exploit the advanced-start capabilities of simplex algorithms in the branch-and-bound (or now more correctly, branch-and-cut) search tree. Using barrier algorithms within the tree is generally impractical.

The above results represent a fundamental change in a subject that twenty-five years ago was considered fully mature. It is interesting to also examine in more detail what is behind these numbers. One finds that of the three listed algorithms, primal simplex is now rarely the winner. Dual and barrier dominate; moreover, because of current trends in computing machinery, with individual processors making relatively little progress, and most increased power coming from increasing the number of cores per CPU chip, the ability to exploit parallelism is becoming more and more important. Barrier algorithms can and have been very effectively parallelized, while there has been essentially no success in parallelizing simplex algorithms. The result is that barrier algorithms are increasingly the winning algorithm when solving large linear programs from scratch. However, since crossover to a basis is an essential part of barrier algorithms, and this step is fundamentally a simplex computation and hence sequential, the fraction of time taken by crossover is ever increasing.

The improvements we have seen in LP computation are clearly good news for the application of these technologies. Indeed, this has led to the common view among practitioners that LP is a "solved problem": it is now common that LPs with several hundred thousand constraints and variables are solved without difficulty. However, there remains considerable room for improvement. The numerical difficulties that are often encountered with barrier algorithms and particularly in the subsequent crossover step represent a major hurdle; moreover, for integer programming (a subject we will return to shortly) computational tests show that, in current practice, roughly 2 % of real-world MIPs are blocked in their solution by the difficulty of the underlying LPs. This combined with the fact that since 2004 there have been essentially no improvements in the standard LP algorithms, means that LP is threatening in the future to again become a significant bottleneck in our ability to solve real-world problems of interest.

Modern MIP codes

Let me now return to the topic of computation in MIP. While LP is a fundamental technique in the modern application of quantitative techniques to the solution of real-world problems, in the context of optimization, it is MIP that dominates.

As previously noted in this paper, MIP codes passed an important milestone in the early 1970's with the introduction of several powerful new codes – notably SCICONIC, MPSX/370 and MPS III with Whizard – using what were then state-of-the art implementations of simplex algorithms tightly integrated with LP based branch-and-bound, and combined with a wide variety of generally simple, but very effective heuristic techniques to improve the overall search. That was an important step forward in the field. However, the dominance of these codes also led to stagnation in the field.

In the years between mid-60s and the late 90s, there was a steady stream of fundamental theoretical work in integer programming and related areas of combinatorial optimization. Important parts of this work were motived by the seminal paper of Dantzig, Fulkerson and Johnson in 1954 [13].

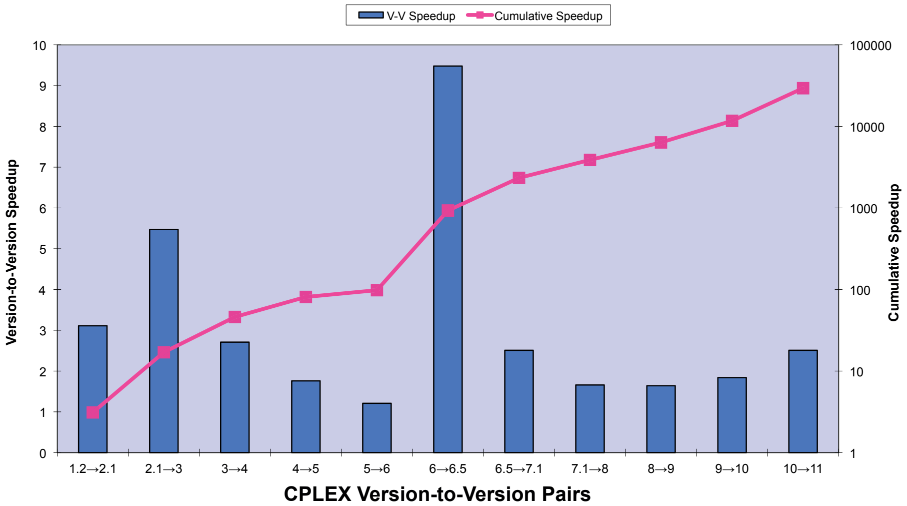
Other fundamental contributions in this period included the work of Gomory on pure integer programs, the work on Edmonds on matching and polyhedral combinatorics, subsequent work by Padberg, Grötschel, Wolsey and others developing and applying cutting-plane techniques (with roots in the paper of Dantzig, Fulkerson and Johnson [13] as well as the work of Edmonds), and a substantial body of theory of disjunctive programming developed primarily by Balas. In addition, there were very important papers by Crowder, Johnson, and Padberg for 0/1 pure integer programs [10] and Van Roy and Wolsey for general MIP [42] that demonstrated the practical effectiveness of cutting-plane techniques and MIP presolve reductions in solving collections of real-world MIPs, MIPs that appeared intractable using traditional branch-and-bound. Indeed, in both of these cases, existing commercial codes (in the first case MPSX/370 and in the second SCICONIC) were directly modified to demonstrate the efficacy of these ideas. In spite of that fact, there was no real change in the generally available commercial codes. They got faster, but only because machines got faster and LP algorithms improved. The basic MIP algorithms in use remained largely those developed in the 70s.

To complete our story of the evolution of MIP software to the present, let me now return to some of the important codes that were developed subsequent to the developments in the 70s. There is a long list, but foremost among these have been XpressMP, the first MIP version being released in 1989, CPLEX MIP, with the first release in 1991, and much more recently, the Gurobi [23] mixed-integer solver, first released in 2009. It should also be mentioned that during this period there was also another solver that was influential to the development of the subject, the MINTO code developed at Georgia Tech [34] and first released around 1991. This code was not intended to be a competitor to commercial solvers, and it was never really used widely in applications. However, it was a milestone in the sense that it was the first general purpose MIP code to make systematic use of cutting-plane techniques, a set of methods that have subsequently proved to be fundamental in the development of MIP. Moreover, though this code was a research code, it was clearly well implemented and provided an important test of the efficacy of these methods.

Another key software development in this period was the introduction of the concept of a callable library as first realized in the initial versions of CPLEX. The idea behind this structure, which involved an early example of what was effectively an object-oriented design, was to treat LP as a kind of black-box tool that could be used as an embedded solver in the development of other algorithmic techniques, most importantly in algorithms for solving integer programs. This callable library approach was enormously successful, and became the model for essentially all future codes in this domain.

Let me now turn to a discussion of the computational progress that has occurred since the early 90s. In late 2007, I undertook a massive computational test using the CPLEX codes that had been released over the years. This test made use of an extensive library of real-world problems that had been collected from academic and industry sources over a period of almost twenty years. From this extensive library, a test set of 1892 representative models was selected. Using these models, and using a bank of identical computing machines, I recompiled each of the corresponding twelve CPLEX released versions – from Version 1.2 (the first version having MIP) through CPLEX 11 – to run on the target machine. I then ran each of the 1892 models with each different CPLEX version, using a time limit of 30,000 seconds, roughly 1/3 of a day. I then compared consecutive versions by taking each model that was solved to optimality by at least one of the two versions, computing the ratios of the solve times (using the time limit for models that did not solve to optimality), and then computing the geometric means of these ratios. The results of these tests are summarized in the chart below:

This chart can be read as follows. The scale on the left refers to the bars in the chart and the scale on the right to the piecewise-linear line through the middle. First looking at the bars, we see, for example that in this test CPLEX 2.1 was approximately 3.1 times faster than CPLEX 1.2, and that each subsequent



version, with the arguable exception of CPLEX 6.0, represented a significant improvement over the previous version. Two particular bars in this chart stand out, the one comparing CPLEX 3.0 to 2.1 and the one comparing CPLEX 6.5 to 6.0. The first of these, representing an improvement factor of nearly 5.5, corresponds to the maturity of the dual simplex algorithm.

The second and by far the biggest improvement occurred in 1998, a speedup exceeding a factor of 10.0. How and why did this happen? The way I like to describe it is as follows. As noted above, the late 90s were preceded by a period of some thirty years of important theoretical and computational developments, many clearly relevant to MIP computation, but virtually none of which had been implemented in commercial codes. The conclusion was clear. It was time to change that. With CPLEX version 6.5 a systematic program was undertaken to include as many of these ideas as possible. You see the result in the chart. The net effect was that in 1998 there was a fundamental change in our ability to solve real-world MIPs. With these developments it was possible, arguably for the first time, to use an out-of-the box solver together with default settings to solve a significant fraction of non-trivial, real-world MIP instances. I would venture to say that if you had asked any of the top MIP researches in the field prior to that time if that would have been possible, they would have said no.

The subject had changed, and changed fundamentally. The piecewise-linear line through the graph is an attempt to capture the overall magnitude of that change. It was computed by multiplying the effects of the individual improvements, producing a projected, machine-independent improvement of a factor of over 29,000.

And, this trend has continued. Tests carried out in 2009 using public benchmarks maintained by Hans Mittelmann at the University of Arizona [33] indicated that Gurobi 1.0, the first release of the Gurobi solver, had performance that was roughly equivalent to that of CPLEX 11.0. Since the release of Gurobi 1.0, we have measured the improvements for subsequent releases, up through the current 5.0 release. Using the standard approach of taking ratios of solve times and computing geometric means, the total improvement was a factor of 16.2, and this on top of the factor of 29,000 in the period prior to 2009, yielding a combined machine-independent factor far exceeding that for LP; moreover, this phenomenon is not restricted to CPLEX and Gurobi. The recent Mittelmann benchmarks demonstrate equally impressive performance by other codes, notably XpressMP and the open-source solver SCIP [1]. It's a great story for the future of our subject, and it shows no signs of stopping.

Acknowledgment. The author would like to thank Robert Ashford, John Gregory, Ed Rothberg, Max Shaw and John Tomlin for several useful e-mail exchanges that contributed to this article.

References

[1] Achterberg, T. 2009. SCIP: solving constraint integer programs. *Math. Programming Computation*, 1 (1) 1–41.

[2] Ashford, R. 2012. Private communication.

[3] Bartels, R. H., G. H. Golub. 1969. The simplex method of linear programming using LU decomposition. *Communications of the Association for Computing Machinery* 12 266–268.

[4] Beale, E. M. L., R. E. Small. 1965. Mixed integer programming by a branch and bound technique, *Proc. IFIP Congress*, Vol. 2 (W. Kalench, Ed.), Macmillan, London (1965) 450-451.

[5] Benichou, M., J. M. Gauthier, P. Girodet, G. Hentges, G. Ribière, O. Vincent. 1971. Experiments in mixed-integer linear programming. *Math. Programming* 1 76–94.

[6] Benichou, M., J. M. Gauthier, G. Hentges, G. Ribière. 1977. The efficient solution of large scale linear programming problems. Some algorithmic techniques and computational results. *Math. Programming* 13 280–322.

[7] Bixby, R. E. 2002. Solving real-world linear programs: a decade and more of progress. *Operations Research* 50 (1) 1–13.

[8] Carolan, W. J., J. E. Hill, J. L. Kennington, S. Niemi, S. J. Wichmann. 1990. An empirical evaluation of the KORBX algorithms for military airlift applications. *Operations Research.* 38 (2) 240–248.

[9] Cook, W. 2012. Markowitz and Manne + Eastman + Land and Doig = Branch and Bound, *this volume.*

[10] Crowder, H., E. L. Johnson, M. Padberg. 1983. Solving large-scale zero-one linear programming problems. *Operations Research* 31 (5) 803–834.

[11] Dantzig, G. 1963. *Linear Programming and Extensions*. Princeton University Press, Princeton.

[12] Dantzig, G. 1948. Programming in a linear structure, U.S. Air Force Comptroller, USAF, Washington, D.C.

[13] Dantzig, G., D. R. Fulkerson, S. Johnson. 1954. Solution of a large scale traveling salesman problem. *Operations Research* 2 393–410.

[14] Dakin, R. J. 1965. A tree search algorithm for mixed integer programming problems, *Computer Journal* 8 250–255.

[15] Fair Isaac Corporation. 2012. *Xpress-Optimizer reference manual.* (`http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx`)

[16] Forrest, J. J., D. Goldfarb. 1992. Steepest-edge simplex algorithms for linear programming. *Math. Programming* 57 341–374.

[17] Forrest, J. J. H., J. P. H. Hirst, J. A. Tomlin. 1974. Practical solution of large mixed integer programming problems with UMPIRE. *Management Science* 20 (5) 736–773.

[18] Forrest, J. J. H., J. A. Tomlin. 1972. Updated triangular factors of the basis to maintain sparsity in the product form simplex method. *Math. Programming* 2 263–278.

[19] Geoffrion, A. M., R. E. Marsten. 1972. Integer programming algorithms: a framework and state-of-the-art survey. *Management Science* 18 465–491.

[20] Gilbert, J. R., T. Peierls. 1988. Sparse partial pivoting in time proportional to arithmetic operations. *SJSSC* 9 862–874.

[21] Grötschel, M., O. Holland. 1991. Solution of large-scale symmetric travelling salesman problems. *Math. Programming* 51 141–202.

[22] Grötschel, M., L. Lovász, A. Schrijver. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1 169–197.

[23] Gurobi Optimization, Inc. 2012. Gurobi optimizer reference manual. (`http://www.gurobi.com`)

[24] Harris, P. J. J. 1974. Pivot selection methods of the devex LP code. *Math. Programming* 5 1–28.

[25] Hoffman, A., A. Mannos, D. Sokolowsky, D. Wiegmann. 1953. Computational experience in solving linear programs. *SIAM J.* 1 1–33.

[26] Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming, *Combinatorica* 4 373–395.

[27] Khachiyan, L. G. 1979. A polynomial algorithm in linear programming (in Russian). *Doklady Akademii Nauk SSSR* 244 1094–1096.

[28] Land, A., S. Powell. 1979. Computer codes for problem of integer programming. *Annals of Discrete Mathematics* 5 221–269.

[29] Land, A., A. G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica* 28 (3) 597–520.

[30] Lemke, C. E. 1954. The dual method of solving the linear programming problem. *Naval Res. Logist. Quart.* 1 36–47.

[31] Lustig, I. J., R. Marsten, D. F. Shanno. 1994. Interior point methods for linear programming: Computational state of the art. *ORSA J. Comput.* 6(1) 1–14.

[32] Marsten, R. E. 1981. XMP: A structured library of subroutines for experimental mathematical programming. *ACM Trans. Math. Software* 7 481–497.

[33] Mittelmann, H. 2012. Benchmarks for Optimization Software (`http://plato.asu.edu/bench.html`).

[34] Nemhauser, G. L., M. W. P. Savelsbergh, G. C. Sigismondi. 1994. MINTO, A Mixed INTeger Optimizer. *Operations Research Letters* 15 47–58.

[35] Orchard-Hays, W. 1990. History of the development of LP solvers. *Interfaces* 20 (4) 61–73.

[36] Schrijver, A. 2012. *This volume.*

[37] Shanno, D. F. 2012. *This volume.*

[38] Sharda, R, C. Somarajan. 1986. Comparative performance of advanced microcomputer systems. *Comput. & Ops. Res.* 13 (2/3) 131–147.

[39] Shaw, M. 2012. Private communication.

[40] Sperry-Univac. 1975. Sperry-Univac 1100 Series Functional Mathematical Programming System (FMPS) Programming Reference UP-8198.

[41] Stigler, G. J. 1945. The cost of subsistence, *J. Farm Econom.* 27 (2) 303–314.

[42] Van Roy, T. J., L. A., Wolsey. 1987. Solving mixed integer programming problems with automatic reformulation. *Operations Research* 35 (1) 45–57.

[43] Tomlin, J. 2012. Private communication.

Robert E. Bixby
8 Briarwood Ct.
Houston, Texas, 77019
USA
bixby@gurobi.com