ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

Benchmarking Optimization Software with Performance Profiles

Elizabeth D. Dolan and Jorge J. Moré

Mathematics and Computer Science Division

Preprint ANL/MCS-P861-1200

January 2001

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation (Challenges in Computational Science) grant CDA-9726385 and (Information Technology Research) grant CCR-0082807.

Contents

1 Introduction 1

2 Performance Evaluation 3

3 Benchmarking Data 4

4 Case Study: Optimal Control and Parameter Estimation Problems 6

5 Case Study: The Full COPS 9

6 Case Study: Linear Programming 10

7 Conclusions 11

Acknowledgments 12

References 12

Benchmarking Optimization Software with Performance Profiles*

Elizabeth D. Dolan† and Jorge J. Moré‡

Abstract

We propose performance profiles — distribution functions for a performance metric — as a tool for benchmarking and comparing optimization software. We show that performance profiles combine the best features of other tools for performance evaluation.

1 Introduction

The benchmarking of optimization software has recently gained considerable visibility. Hans Mittlemann's [13] work on a variety of optimization software has frequently uncovered deficiencies in the software and has generally led to software improvements. Although Mittelmann's efforts have gained the most notice, other researchers have been concerned with the evaluation and performance of optimization codes. As recent examples, we cite [1, 2, 3, 4, 6, 12, 17].

The interpretation and analysis of the data generated by the benchmarking process are the main technical issues addressed in this paper. Most benchmarking efforts involve tables displaying the performance of each solver on each problem for a set of metrics such as CPU time, number of function evaluations, or iteration counts for algorithms where an iteration implies a comparable amount of work. Failure to display such tables for a small test set would be a gross omission, but they tend to be overwhelming for large test sets. In all cases, the interpretation of the results from these tables is often a source of disagreement.

The quantities of data that result from benchmarking with large test sets have spurred researchers to try various tools for analyzing the data. The solver's average or cumulative total for each performance metric over all problems is sometimes used to evaluate performance [1, 4, 6]. As a result, a small number of the most difficult problems can tend to dominate these results, and researchers must take pains to give additional information. Another drawback is that computing averages or totals for a performance metric necessitates discarding problems for which any solver failed, effectively biasing the results against the most robust solvers. As an alternative to disregarding some of the problems, a penalty value can be assigned for failed solver attempts, but this requires a subjective choice for the penalty. Most researchers choose to report the number of failures only in a footnote or separate table.

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation (Challenges in Computational Science) grant CDA-9726385 and (Information Technology Research) grant CCR-0082807.

†Department of Electrical and Computer Engineering, Northwestern University, and Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439 (dolan@mcs.anl.gov).

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439 (more@mcs.anl.gov).

To address the shortcomings of the previous approach, some researchers rank the solvers [4, 6, 15, 17]. In other words, they count the number of times that a solver comes in kth place, usually for $k = 1, 2, 3$. Ranking the solvers' performance for each problem helps prevent a minority of the problems from unduly influencing the results. Information on the size of the improvement, however, is lost.

Comparing the medians and quartiles of some performance metric (for example, the difference between solver times [4]) appears to be a viable way of ensuring that a minority of the problems do not dominate the results, but in our testing we have witnessed large leaps in quartile values of a performance metric, rather than gradual trends. If only quartile data is used, then information on trends occurring between one quartile and the next is lost; and we must assume that the journey from one point to another proceeds at a moderate pace. Also, in the specific case of contrasting the differences between solver times, the comparison fails to provide any information on the relative size of the improvement. A final drawback is that if results are mixed, interpreting quartile data may be no easier than using the raw data; and dealing with comparisons of more than two solvers might become unwieldy.

The idea of comparing solvers by the ratio of one solver's runtime to the best runtime appears in [2], with solvers rated by the percentage of problems for which a solver's time is termed *very competitive* or *competitive*. The ratio approach avoids most of the difficulties that we have discussed, providing information on the percent improvement and eliminating the negative effects of allowing a small portion of the problems to dominate the conclusions. The main disadvantage of this approach lies in the author's arbitrary choice of limits to define the borders of *very competitive* and *competitive*.

In Section 2, we introduce *performance profiles* as a tool for evaluating and comparing the performance of optimization software. The performance profile for a solver is the (cumulative) distribution function for a performance metric. In this paper we use the ratio of the computing time of the solver versus the best time of all of the solvers as the performance metric. Section 3 provides an analysis of the test set and solvers used in the benchmark results of Sections 4 and 5. This analysis is necessary to understand the limitations of the benchmarking process.

Sections 4 and 5 demonstrate the use of performance profiles with results [8] obtained with version 2.0 of the COPS [7] test set. We show that performance profiles eliminate the influence of a small number of problems on the benchmarking process and the sensitivity of results associated with the ranking of solvers. Performance profiles provide a means of visualizing the expected performance difference among many solvers, while avoiding arbitrary parameter choices and the need to discard solver failures from the performance data.

We conclude in Section 6 by showing how performance profiles apply to the data [13] of Mittelmann for linear programming solvers. This section provides another case study of the use of performance profiles and also shows that performance profiles can be applied to a wide range of performance data.

2 Performance Evaluation

Benchmark results are generated by running a solver on a set $\mathcal{P}$ of problems and recording information of interest such as the number of function evaluations and the computing time. In this section we introduce the notion of a performance profile as a means to evaluate and compare the performance of the solvers on a test set $\mathcal{P}$.

We assume that we have n_s solvers and n_p problems. We are interested in using computing time as a performance measure; although, the ideas below can be used with other measures. For each problem p and solver s, we define

$$t_{p,s} = \text{computing time required to solve problem } p \text{ by solver } s.$$

If, for example, the number of function evaluations is the performance measure of interest, set $t_{p,s}$ accordingly.

We require a baseline for comparisons. We compare the performance on problem p by solver s with the best performance by any solver on this problem; that is, we use the *performance ratio*

$$\rho_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : 1 \le s \le n_s\}}.$$

We assume that a parameter $\rho_M \ge \rho_{p,s}$ for all p, s is chosen, and $\rho_{p,s} = \rho_M$ if and only if solver s does not solve problem p. The performance of solver s on any given problem may be of interest, but we would like to obtain an overall assessment of the performance of the solver. If we define

$$p_s(\tau) = \frac{1}{n_p}\text{size}\Big\{p \in \mathcal{P} : \rho_{p,s} \le \tau\Big\},$$

then $p_s(\tau)$ is the probability that a performance ratio $\rho_{p,s}$ is within a factor τ of the best possible ratio. The function p_s is the (cumulative) distribution function for the performance ratio.

We use the term *performance profile* for the distribution function of a performance metric. Our claim is that a plot of the performance profile reveals all of the major performance characteristics. In particular, if the set of problems $\mathcal{P}$ is suitably large and representative of problems that are likely to occur in applications, then solvers with large probability $p_s(\tau)$ are to be preferred.

The term performance profile has also been used for a plot of some performance metric versus a problem parameter. For example, Higham [11, pages 296–297] plots the ratio $\gamma/\|A\|_1$, where γ is the estimate for the l_1 norm of a matrix A produced by the LAPACK condition number estimator. Note that in Higham's use of the term performance profile there is no attempt at determining a distribution function.

The performance profile $p_s : \mathbb{R} \mapsto [0, 1]$ for a solver is a nondecreasing, piecewise constant function, continuous from the right at each breakpoint. The value of $p_s(1)$ is the probability that the solver will win over the rest of the solvers. Thus, if we are interested only in the number of wins, we need only to compare the values of $p_s(1)$ for all of the solvers.

The definition of the performance profile for large values requires some care. We assume that $\rho_{p,s} \in [1, \rho_M]$ and that $\rho_{p,s} = \rho_M$ only when problem p is not solved by solver s. As a result of this convention, $p_s(\rho_M) = 1$, and

$$p_s(\rho_M^-) \equiv \lim_{\tau \to \rho_M^-} p_s(\tau)$$

is the probability that the solver solves a problem. Thus, if we are interested only in solvers with a high probability of success, then we need to compare the values of $p_s(\rho_M^-)$ for all solvers and choose the solvers with the largest value. The value of $p_s(\rho_M^-)$ can be readily seen in a performance profile because p_s flatlines for large values of τ; that is, $p_s(\tau) = p_s(\rho_M^-)$ for $\tau \in [\rho_S, \rho_M)$ for some $\rho_S < \rho_M$.

3 Benchmarking Data

The timing data used to compute the performance profiles in Sections 4 and 5 is generated with the COPS test set, which currently consists of seventeen different applications, all models in the AMPL [9] modeling language. The choice of the test problem set $\mathcal{P}$ is always a source of disagreement because there is no consensus on how to choose problems. The COPS problems are selected to be interesting and difficult, but this criteria is subjective. Because the main purpose of our testing is to uncover the best general solver on a wide range of problems, we include problems with multiple minima and problems with a large number of degrees of freedom. For each of the applications in the COPS set we use four instances of the application obtained by varying a parameter in the application, for example, the number of grid points in a discretization.

Section 4 focuses on only the subset of the eleven optimal control and parameter estimation applications in the COPS set, while the discussion in Section 5 covers the complete performance results. Accordingly, we provide here information specific to this subset of the COPS problems as well as an analysis of the test set as a whole. Table 3.1 gives the quartiles for three problem parameters: the number of variables n, the number of constraints, and the ratio $(n - n_e)/n$, where n_e is the number of equality constraints. In the optimization literature, $n - n_e$ is often called the degrees of freedom of the problem, since it is an upper bound on the number of variables that are free at the solution.

The data in Table 3.1 is fairly representative of the distribution of these parameters throughout the test set and shows that at least three-fourths of the problems have the number of variables n in the interval $[400, 5000]$. Our aim was to avoid problems where n was in the range $[1, 50]$ because other benchmarking problem sets tend to have a preponderance of problems with n in this range. The main difference between the full COPS set and the COPS subset is that the COPS subset is more constrained with $n_e \geq n/2$ for all the problems. Another feature of the COPS subset is that the equality constraints are the result of either difference or collocation approximations to differential equations.

We ran our final complete runs with the same options for all models. The options involve setting the output level so that we can gather the data we need, increasing the iteration

limits as much as allowed, and increasing the super-basics limits for MINOS and SNOPT to 5000. None of the failures we record in the final trials include any solver error messages about having violated these limits.

Table 3.1: Problem data for COPS test set

	Full COPS					COPS subset				
	min	q_1	q_2	q_3	max	min	q_1	q_2	q_3	max
Number of variables	48	400	1000	2402	5000	100	449	899	2000	4815
Number of constraints	0	150	498	1598	5048	51	400	800	1601	4797
Degrees of freedom	0	23	148	401	5000	0	5	99	201	1198
Degrees of freedom (%)	0.0	1.0	33.2	100.0	100.0	0.0	0.4	19.8	33.1	49.9

The script for generating the timing data sends a problem to each solver successively, so as to minimize the effect of fluctuation in the machine load. The script tracks the wall-clock time from the start of the solve, killing any process that runs 3,600 seconds, which we declare unsuccessful, and beginning the next solve. We cycle through all the problems, recording the wall-clock time as well as the combination of AMPL system time (to interpret the model and compute varying amounts of derivative information required by each solver) and solver time for each model variation. We repeat the cycle for any model for which one of the solver's AMPL times and the wall-clock times differ by more than 10 percent. To further ensure consistency, we have verified that the AMPL time results we present could be reproduced to within 10 percent accuracy. All computations were done on a SparcULTRA2 running Solaris 7.

We have ignored the effects of the stopping criteria and the memory requirements of the solvers. Ideally we would have used the same stopping criteria, but this is not possible in the AMPL environment. In any case, differences in computing time due to the stopping criteria are not likely to change computing times by more than a factor of two. Memory requirements can also play an important role. In particular, solvers that use direct linear equation solvers are often more efficient in terms of computing time provided there is enough memory.

The solvers that we benchmark here have different requirements. MINOS and SNOPT use only first-order information, while LANCELOT and LOQO need second-order information. The use of second-order information can reduce the number of iterations, but the cost per iteration usually increases. In addition, obtaining second-order information is more costly and may not even be possible. MINOS and SNOPT are specifically designed for problems with a modest number of degrees of freedom, while this is not the case for LANCELOT and LOQO. As an example of comparing solvers with similar requirements, Section 6 shows the performance of linear programming solvers.

4 Case Study: Optimal Control and Parameter Estimation Problems

We now examine the performance profiles of LANCELOT [5], MINOS [14], SNOPT [10], and LOQO [16] on the subset of the optimal control and parameter estimation problems in the COPS [7] test set. Figures 4.1 and 4.2 show the performance profiles in different ranges to reflect various areas of interest. Our purpose is to show how the performance profiles provide the necessary objective information for reasonable subjective analysis of a large test set.

Figure 4.1 shows the performance profiles of the four solvers for small values of τ. By showing the ratios of solver times, we eliminate any weight of importance that taking straight time differences might give to the problems that require a long run time of every solver. We find no need to eliminate any test problems from discussion. For this reason, solvers receive their due credit for completing problems for which one or more of the other solvers fails. In particular, $1 - p_s(\tau)$ is the fraction of problems that the solver cannot solve within a factor τ of the best solver, including problems for which the solver in question fails.

From this figure it is clear that LOQO has the most wins (has the highest probability of being the optimal solver) and that the probability that LOQO is the winner on a given problem is about .61. If we choose being within a factor of 4 of the best solver as the scope of our interest, then either LOQO or MINOS would suffice; but the performance profile shows that the probability that these two solvers can solve a job within a factor 4 of the best solver is only about 70%. SNOPT has a lower number of wins than either LOQO or MINOS, but its performance becomes much more competitive if we extend our τ of interest to 7.

Figure 4.2 shows the performance profiles for all the solvers in the interval $[1, 100]$. If we are interested in the solver that can solve 75% of the problems with the greatest efficiency, then MINOS stands out. If we hold to more stringent probabilities of completing a solve successfully, then SNOPT captures our attention with its ability to solve over 90% of this COPS subset, as displayed by the height of its performance profile for $\tau > 40$. This graph displays the potential for large discrepancies in the performance ratios on a substantial percentage of the problems. Another point of interest is that LOQO, MINOS, and SNOPT each have the best probability $p_s(\tau)$ for τ in some interval, with similar performance in the interval $[15, 40]$.

An observation that emerges from these figures is the lack of consistency in quartile values of time ratios. The top three solvers share a minimum ratio of 1, and LOQO and MINOS also share first quartile values of 1. In other words, these two solvers are the best solvers on at least 25% of the problems. LOQO bests MINOS's median value with 1 compared with 2.4, but MINOS comes back with a third quartile ratio of 4.3 versus 12.6 for SNOPT and 13.9 for LOQO. By looking at Figures 4.1 and 4.2, we see that progress between quartiles does not necessarily proceed linearly; hence, we really lose information if we do not provide the full data. Also, the maximum ratio would be ρ_M for our testing, and no obvious alternative value exists. As an alternative to providing only quartile values,

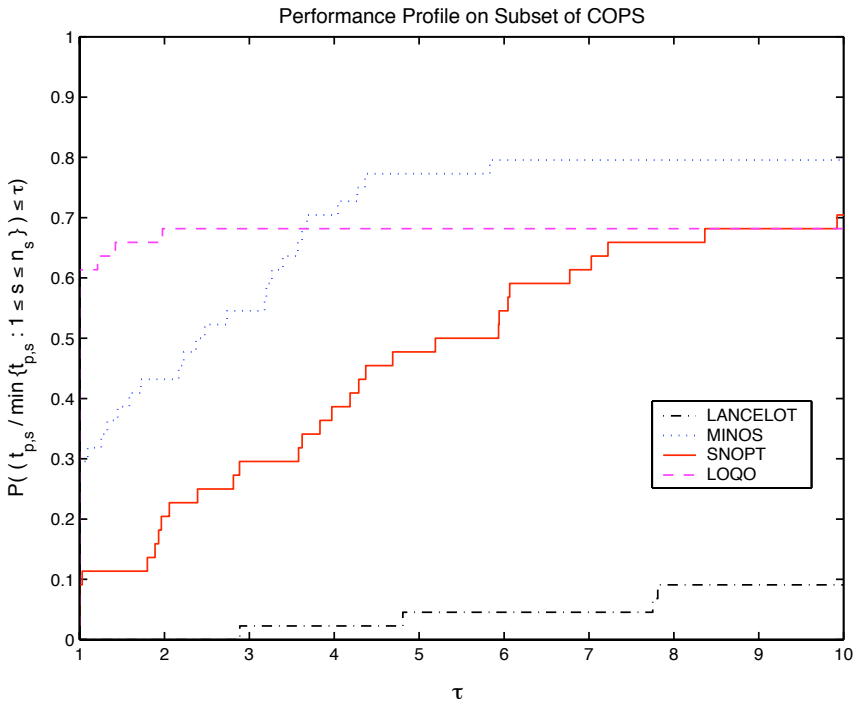

Figure 4.1: Performance profile on $[0, 10]$

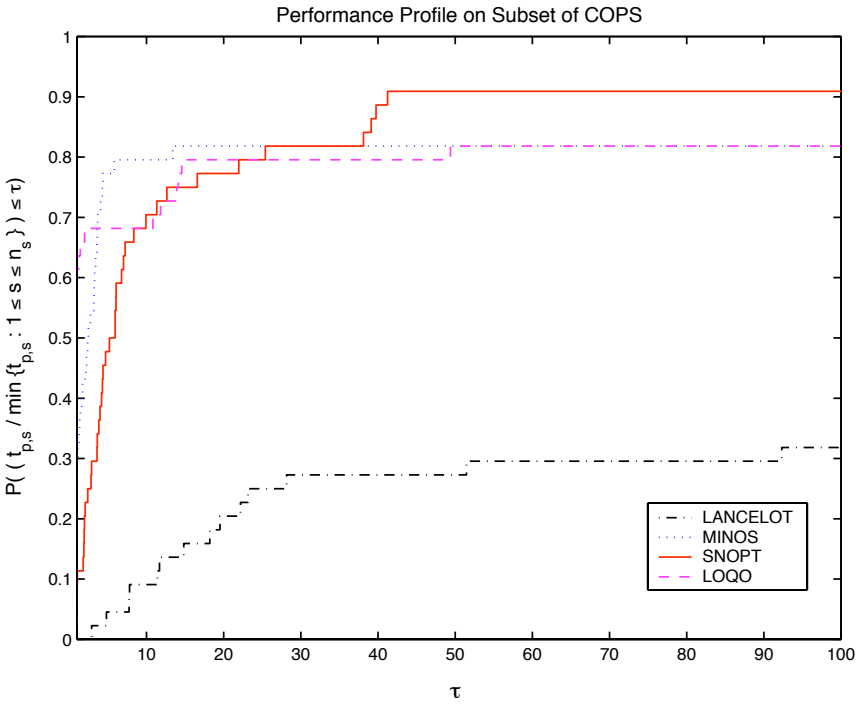

Figure 4.2: Performance profile on $[0, 100]$

however, the performance profile yields much more information about a solver's strengths and weaknesses.

We have seen that at least two graphs may be needed to examine the performance of the solvers. Even extending τ to 100, we fail to capture the complete performance data for LANCELOT and LOQO. As a final option, we display a log scale of the performance profiles. In this way, we can show all activity that takes place with $\tau < \rho_M$ and grasp the full implications of our test data regarding the solvers' probability of successfully handling a problem. Since we are also interested in the behavior for τ close to unity, we use a base of 2 for the scale. In other words, we plot

$$\tau \mapsto \frac{1}{n_p} \text{size} \{p \in \mathcal{P} : \log_2 (\rho_{p,s}) \leq \tau\}$$

in Figure 4.3. This graph reveals all the features of the previous two graphs and thus captures the performance of all the solvers. The disadvantage is that the interpretation of the graph is not as intuitive, since we are using a log scale.

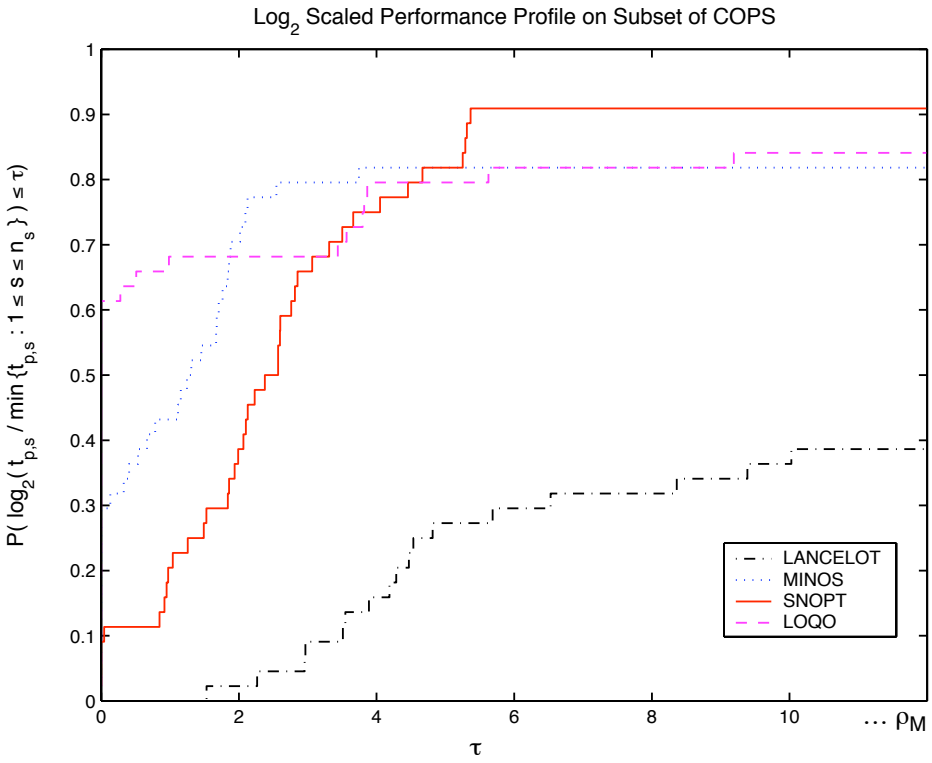

Figure 4.3: Performance profile in a $\log_2$ scale

Figures 4.1 and 4.2 are mapped into a new scale to reflect all data, requiring at least the interval $[0, \log_2(1043)]$ in Figure 4.3 to include the largest $\rho_{p,s} < \rho_M$. We extend the range slightly to show the flatlining of all solvers. The new figure contains all the information of the other two figures and, in addition, shows that each of the solvers fails on at least 8% of

the problems. This is not an unreasonable performance for the COPS test set because these problems were generally chosen to be difficult.

5 Case Study: The Full COPS

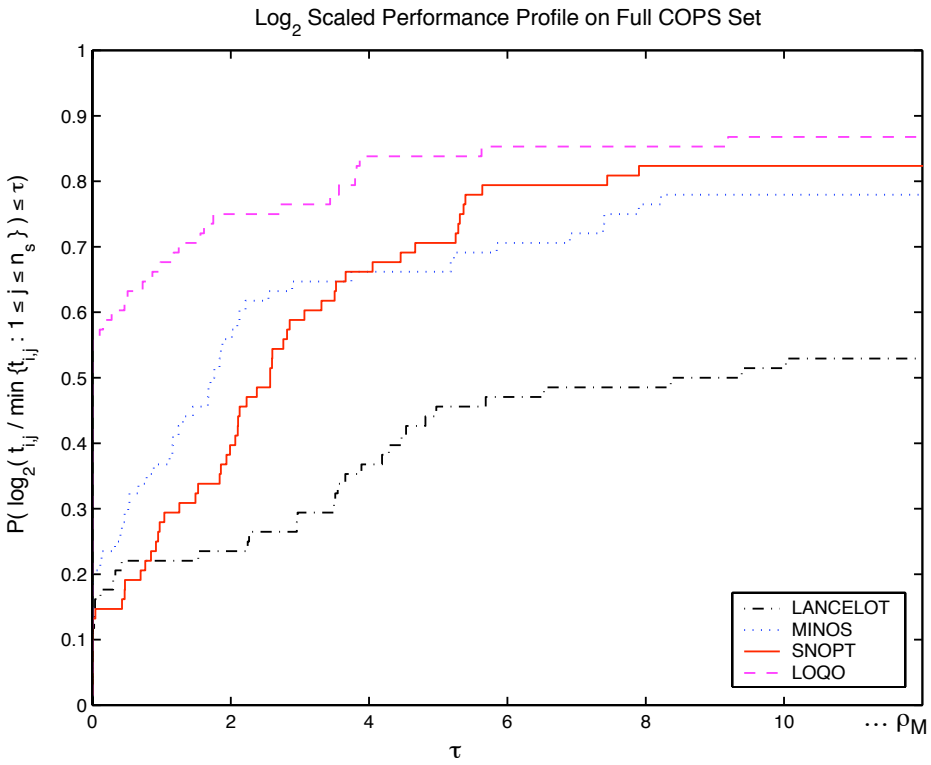

Figure 5.1: Performance profile for full COPS set

We now expand our analysis of the data to include all the problems in version 2.0 of the COPS [7] test set. We present in Figure 5.1 a $\log_2$ scaled view of the performance profiles for the solvers on that test set.

Figure 5.1 gives a clear indication of the relative performance of each solver. As in the performance profiles in Section 4, this figure shows that performance profiles eliminate the undue influence of a small number of problems on the benchmarking process and the sensitivity of the results associated with the ranking of solvers. In addition, performance profiles provide an estimate of the expected performance difference between solvers.

The most significant aspect of Figure 5.1, as compared with Figure 4.3, is that on this test set LOQO dominates all other solvers: the performance profile for LOQO lies above all others for all performance ratios τ. The interpretation of the results in Figure 5.1 is important. In particular, these results do not imply that LOQO is faster on every problem. They indicate only that, for any $\tau \geq 1$, LOQO solves more problems within a factor of τ of any other solver time. Moreover, by examining $p_s(1)$ and $p_s(\rho_M)$, we can also say that

LOQO is the fastest solver on approximately 58% of the problems, and that LOQO solves the most problems (about 87%) to optimality.

The difference between the results in Section 4 and these results is due to a number of factors. First of all, as can be seen in Table 3.1, the degrees of freedom for the full COPS test set is much larger than for the restricted subset of optimal control and parameter estimation problems. Since, as noted in Section 3, MINOS and SNOPT are designed for problems with a modest number of degrees of freedom, we should expect the performance of MINOS and SNOPT to deteriorate on the full COPS set. This deterioration can be seen by comparing Figure 5.1 with Figure 4.3 and noting that the performance profiles of MINOS and SNOPT are similar but generally lower in Figure 5.1.

Another reason for the difference between the results in Section 4 and these results is that MINOS and SNOPT use only first-order information, while LOQO uses second-order information. The benefit of using second-order information usually increases as the number of variables increases, so this is another factor that benefits LOQO.

The results in this section underscore our observation that performance profiles provide a convenient tool for comparing and evaluating the performance of optimization solvers, but, like all tools, performance profiles must be used with care. A performance profile reflects the performance only on the data being used, and thus it is important to understand the test set and the solvers used in the benchmark.

6 Case Study: Linear Programming

Performance profiles can be used to compare the performance of two solvers, but performance profiles are most useful in comparing several solvers. Because large amounts of data are generated in these situations, trends in performance are often difficult to see. As a case study, we use data obtained by Mittelmann [13]. Figure 6.1 shows a plot of the performance profile for the time ratios in the data *Benchmark of LP solvers on a Linux-PC (5-25-2000)*, which includes results for COPL_LP-1.0, PCx-1.1, SOPLEX-1.1, LPABO-5.6, MOSEK-1.0b, BPMPD-2.11, and BPMPD-2.14.

In keeping with our graphing practices with the COPS set, we designate as failures those solves that are marked in the original table as stopping close to the final solution without convergence under the solver's stopping criteria. One feature we see in the graph of Mittelmann's results that does not appear in the COPS graphs is the visual display of solvers that never flatline. In other words, the solvers that climb off the graph are those that solve all of the test problems successfully. As with Figure 4.3, all of the events in the data fit into this log-scaled representation. While this data set cannot be universally representative of benchmarking results by any means, it does show that our reporting technique is applicable beyond our own results.

As in the case studies in Sections 4 and 5, the results in Figure 6.1 give an indication of the performance of LP solvers only on the data set used to generate these results. In particular, the test set used to generate Figure 6.1 includes only problems selected by

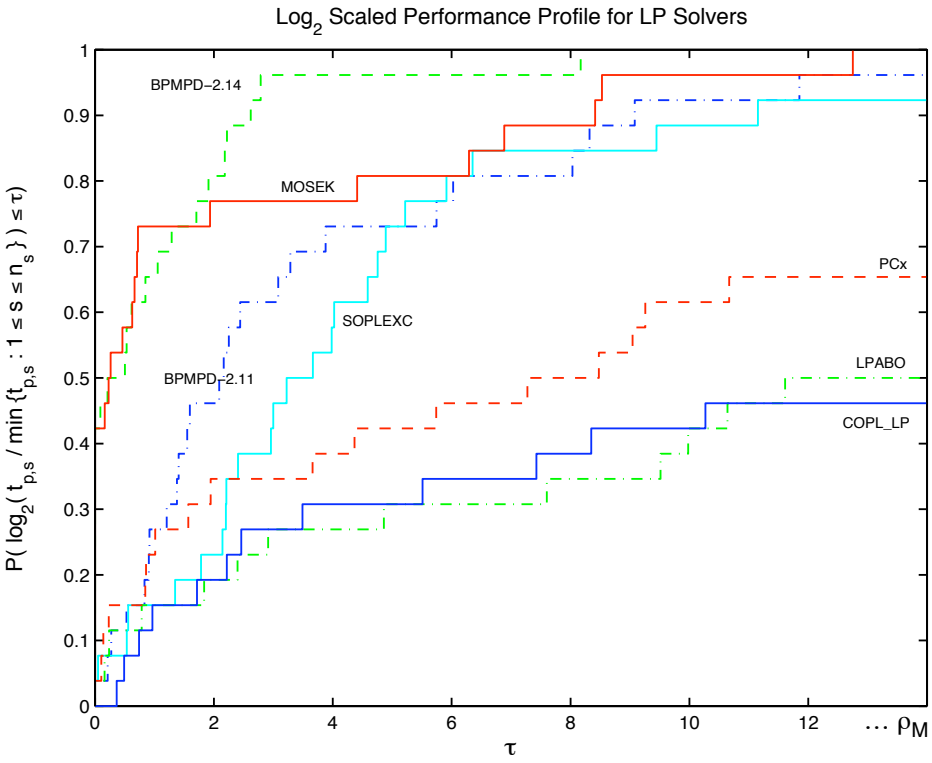

Figure 6.1: Performance profile for linear programming solvers

Mittelmann for his benchmark. The advantage of these results is that, unlike the solvers in Sections 4 and 5, all solvers in Figure 6.1 have the same requirements.

7 Conclusions

We have shown that performance profiles combine the best features of other tools for benchmarking and comparing optimization solvers. Clearly, the use of performance profiles is not restricted to optimization solvers and can be used to compare solvers in other areas.

The Perl script `perf.pl` on the COPS site [7] generates performance profiles formatted as Matlab commands to produce a composite graph as in Figures 4.1 and 4.2. The script accepts data for several solvers and plots the performance profile on an interval calculated to show the full area of activity. The area displayed and scale of the graph can then be adjusted within Matlab to reflect particular benchmarking interests.

Acknowledgments

Version 1.0 of the COPS problems was developed by Alexander Bondarenko, David Bortz, Liz Dolan, and Michael Merritt. Their contributions were essential because, in many cases, version 2.0 of the problems are closely related to the original version.

Alex Bondarenko, Nick Gould, Sven Leyffer and Bob Vanderbei contributed interesting and spirited discussions on problem formulation, while Bob Fourer and David Gay generously shared their AMPL expertise with us. Finally, we thank Hans Mittelmann for paving the way for COPS with his benchmarking work.

References

[1] H. Y. Benson, D. F. Shanno, and R. J. Vanderbei, *Interior-point methods for nonconvex nonlinear programming: Jamming and comparative numerical testing*, Technical Report ORFE-00-02, Princeton University, Princeton, New Jersey, 2000.

[2] S. C. Billups, S. P. Dirkse, and M. C. Ferris, *A comparison of algorithms for large-scale mixed complementarity problems*, Comp. Optim. Appl., 7 (1997), pp. 3–25.

[3] A. S. Bondarenko, D. M. Bortz, and J. J. Moré, *COPS: Large-scale nonlinearly constrained optimization problems*, Technical Memorandum ANL/MCS-TM-237, Argonne National Laboratory, Argonne, Illinois, 1998 (Revised October 1999).

[4] I. Bongartz, A. R. Conn, N. I. M. Gould, M. A. Saunders, and P. L. Toint, *A numerical comparison between the LANCELOT and MINOS packages for large-scale numerical optimization*, Report 97/13, Namur University, 1997.

[5] A. R. Conn, N. I. M. Gould, and P. L. Toint, LANCELOT, no. 17 in Springer Series in Computational Mathematics, Springer-Verlag, 1992.

[6] ——, *Numerical experiments with the* LANCELOT *package (Release A) for large-scale nonlinear optimization*, Math. Programming, 73 (1996), pp. 73–110.

[7] *COPS*. See `http://www.cops.com/`.

[8] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with COPS*, Technical Memorandum ANL/MCS-TM-246, Argonne National Laboratory, Argonne, Illinois, 2000.

[9] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, 1993.

[10] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An algorithm for large-scale constrained optimization*, Report NA97-2, University of California, San Diego, 1997.

[11] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, Pennsylvania, 1996.

[12] H. Mittelmann, *Benchmarking interior point LP/QP solvers*, Opt. Meth. Software, 12 (1999), pp. 655–670.

[13] ——, *Benchmarks for optimization software*, 2000. See `http://plato.la.asu.edu/bench.html`.

[14] B. A. Murtagh and M. A. Saunders, *MINOS 5.4 user's guide*, Report SOL 83-20R, Stanford University, 1995.

[15] S. G. Nash and J. Nocedal, *A numerical study of the limited memory BFGS method and the truncated Newton method for large scale optimization*, SIAM J. Optim., 1 (1991), pp. 358–372.

[16] R. J. Vanderbei, *LOQO user's manual – Version 3.10*, Opt. Meth. Software, 12 (1999), pp. 485–514.

[17] R. J. Vanderbei and D. F. Shanno, *An interior-point algorithm for nonconvex nonlinear programming*, Comp. Optim. Appl., 13 (1999), pp. 231–252.