computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

A PRACTICAL PERFORMANCE INDEX FOR COMPARING OF OPTIMIZATION SOFTWARE

Andrea Attanasio[1], Patrizia Beraldi[2], Francesca Guerriero[3]

1) Center of Excellence for High Performance Computing, Università della Calabria, 87030 Rende (CS) - Italy, attanasio@unical.it
2) Dipartimento di Elettronica, Informatica e Sistemistica, Università degli Studi della Calabria, 87030 Rende (CS) - Italy, beraldi@deis.unical.it
3) Dipartimento di Elettronica, Informatica e Sistemistica, Università degli Studi della Calabria, 87030 Rende (CS) - Italy, guerrier@deis.unical.it

Abstract: *In this paper we propose a new practical performance index for ranking of numerical methods. This index may be very helpful especially when several methods are tested on a large number of instances, since it provides a concise and precise idea of the relative efficiency of a method with the respect to the others. In order to evaluate the efficiency of the proposed rule, we have applied it to the numerical results presented on previously published papers.*

Keywords: *Performance index, Optimization software performance.*

1. INTRODUCTION

The increasing emphasis on the computational aspects of optimization methods and their impact on the solution real-world applications have prompted the need to design meaningful indices for performance evaluation. However, many difficulties arise in evaluating and interpreting the results in a fair and balanced way.

First of all, we have to establish the object of evaluation. Actually, for any method, it is possible to define different algorithms and software implementations whose efficiency strongly depends on the compiler and on the hardware platform used.

Furthermore, we have to choose the criteria on which we shall carry out the evaluating process. The traditionally used performance measures are: the *computational performance* (speed and memory, robustness), the *solution quality* (accuracy), and the *scope* (problem type and size).

A frequently used notion in the comparison of numerical methods is the one of "the best method". The "best" is a relative concept that depends on subjective criteria, whose choice depends on the goal of the comparison: the best may mean the fastest, or the easiest to apply, or the most reliable.

Another important issue to address in the evaluating process is the choice of the test problems. The testing phase of optimization software could be inadequate for several reasons: the number of test problems could be too small, the instances could have small size and present some regularity. We observe that the testing is a very crucial phase since if any error occurs in the manner of carrying out the experiments, the same error may affect any performance index used to compare the results.

Solving the controversy surrounding the reporting of results from scientific experiments is out the scope of this paper. Relevant papers try to give instructions on the manner of carrying out numerical experiments in a correct way. Interested readers are referred, for example, to [1], [2] and [3], where the authors examined the issues involved in reporting on the empirical testing of parallel mathematical programming algorithms.

The main contribution of this paper is to define a new performance index, *once* the user has stated clearly what is tested, which performance criteria are considered, and which performance measure is used. The proposed index gives a concise idea of the relative efficiency of a method with respect to the others, when it is applied to solve a given set of test problems.

The rest of the paper is organized as follows. In the next section we introduce a simple example showing the weaknesses of certain known rules. In Section 3, we define our performance index. Finally, in Section 4, we illustrate the efficiency of our rule by considering the computational results of some numerical methods proposed to solve two classes of optimisation problems.

2. AN EXAMPLE

Let p be the number of test problems, m the number of methods that we want to rank, and let $C_{i,j}$ be the cost (for example the execution time) required to solve test j (j = 1, 2,..., p) by the method i (i = 1, 2,..., m).

Let us consider the simple case with $m = 2$ and $p = 3$ reported in Table 1.

Table 1 - Prediction results

Method	Test 1	Test 2	Test 3
1	60	10	5
2	30	20	10

In order to compare the results of Method 2 with respect to those of Method 1, a straightforward rule is to consider the ratio of the sum of the costs:

$$R^{(1)} = \frac{\sum_{J=1}^{p} C_{1,j}}{\sum_{J=1}^{p} C_{2,j}} = \frac{75}{60} = 1.25 .$$

Another simple rule is defined by the following index:

$$R^{(2)} = \frac{\sum_{J=1}^{p} (C_{1,j} / C_{2,j})}{p} = \frac{3}{3} = 1,$$

corresponding to the average speed-up of Method 1 over Method 2. The value of $R^{(1)}$ shows that Method 2 works better than Method 1, whereas $R^{(2)}$ indicates that there is no difference in the performance of the two methods. However, the use of both indices may be misleading since Method 2 improves 50% over Method 1 on the first test (the improvement of Method 2 over Method 1 for solving test j is defined by the ratio $(C_{1,j}/C_{2,j})/C_{1,j}$), whereas Method 1 improves 50% over Method 2 for the other two tests. This means that Method 1 is globally more efficient than Method 2.

This simple example suggests that the above exposed rules are not reliable. Actually, $R^{(1)}$ does not consider the difference among the costs required by each method to solve all the test problems. On the other hand, $R^{(2)}$ is useful only if the ratio $C_{1,j}/C_{2,j}$ is greater than or equal to one, for all the test.

Other difficulties arise when, for some tests, one method either fails to solve the test problem or it finds a different solution with respect to the one obtained by the other methods. In this situation, one possibility is to avoid to include in the comparison the results of this particular test problem.

3. A NEW PRACTICAL PERFORMANCE INDEX

Much work has been devoted in defining performance measures for comparing of optimization software developed for specific problems. The interested readers are referred, for example, to [4], [5], [6] and [7]. Our index is more general and can be used to compare a large number of methods tested on a specified set of instances. More specifically, for each method i, we define the total quality index R_i by the following pair of values:

$$R_i = (R_i^{(SQ)}, R_i^{(CP)}) .$$

Here $R_i^{(SQ)}$ is a measure of the solution quality (i.e., robustness) of the method i. In particular, we have chosen to define $R_i^{(SQ)}$ as the percentage of successes, i.e. the ratio between the number of successful exits over the number p of test problems; as such $R_i^{(SQ)} \in [0,1]$. We observe that this index may be omitted when the value is equal for all the methods or meaningless for some particular application.

$R_i^{(CP)}$ is the index of the *computational performance* defined as follows:

$$R_i^{(CP)} = \frac{\sum_{j=1}^{p} r_i^{(j)}}{p}, \; i = 1, 2, ..., m, \qquad (1)$$

where:

$$r_i^{(j)} = \frac{C_{i,j}}{\min_k \{C_{k,j}\}}.$$

More specifically, for each method i and for each test problem j, we define a score $r_i^{(j)}$ which is equal to the ratio of the cost of this method over the cost of the best-considered method. Thus, $R_i^{(CP)}$ is an average score that indicates how much a particular method has been less efficient, on average, than the most successful method (defined from now on as the *ideal* method). This value takes into account not only if a method works better than the others, but also how much the method outperforms the others.

For the example reported in Table 2, we have:

Table 2 - Our performance index

Method	Test 1	Test 2	Test 3
1	60	10	5
2	30	20	10
r_1	2	1	1
r_2	1	2	2

and we obtain $R_1^{(CP)} = 1.\bar{3}$ and $R_2^{(CP)} = 1.\bar{6}$

We observe that our definition of $R_i^{(CP)}$ can be viewed as a generalization of the rule used by Brown and Bartholomew-Biggs in [8] to rank some methods for solving unconstrained optimization problems. Their idea consists in assigning 1 point to the most successful code, 2 points to the second, and so on. In this way, the total score obtained for each method reflects the frequency of outperforming. The main drawback of the Brown-Biggs' rule is that it provides a qualitative ranking rather than a quantitative one.

It is worth mentioning that the index of computational performance $R^{(CP)}$ is particularly useful also to establish the speed-up of parallel methods and to compare their efficiency with the sequential counterparts [9]. Obviously, this is possible only when the costs $C_{i,j}$ used to rank the methods correspond to the execution times.

In the case of failure for some test problem j, we suggest to replace the failure with the maximum cost computed over all the methods used for solving test j. This solution seems to be reasonable because the main aim of the rule is to rank the methods using a relative index (i.e., we can only establish if a method works better than the others considered for solving the limited set of selected test problems). In this case, the index $R^{(SQ)}$ will take into account the percentage of the failure of the method.

4. NUMERICAL ILLUSTRATION

In order to evaluate the efficiency of our rule, we have considered methods proposed in the literature to solve two classes of classical optimization problems, i.e. the shortest path problem in a directed graph, and the problem of finding the stationary points of a nonlinear and unconstrained function.

In the former case, we use the results collected by Bertsekas [10] on 16 network problems solved by using the Bellman-Ford (B-F) method, the D'Esopo-Pape (D-P) method, the Small Label First (SLF) method, the Threshold (THR) method and the combination of the last two methods (SLF-THR). The cost chosen by Bertsekas to evaluate the performance of the method is the execution time (in secs).

On the basis of the results reported in table 3, we obtain the ranking of Fig. 1. Note that, we have reported only the values of the index $R^{(CP)}$ of the computational performance, since all the methods terminate with the optimal solution.

Our index reveals that the D-P method is about 11 times slower on the average than the SLF-THR, whereas the performance of SLF-THR is very close to that of the ideal method, which solves all the test problems in the minimum time over all the considered methods (this means that none of the methods is ideal). The analysis of the results presented by Bertsekas in [10] completely matches our conclusions: "... the SLF method can also be combined with the threshold algorithm thereby considerably improving its practical performance", or: "... for some test problems the D'Esopo-Pape algorithm performs very poorly; we have not seen in the literature any report of a class of randomly generated sparse problems where this algorithm exhibits such poor behaviour", and so on.

As another example we have considered the numerical results reported in [11] to solve unconstrained non linear optimisation problems by limited memory Quasi-Newton methods. In particular, comparative tests have been conducted on a set of 18 well-known test problems (three of these have two different dimensions). Among the noticeable variety of collected results, we have considered (according to the choice of the authors of [10]) only the CPU time for nine different codes, whose names are reported in Table 4.

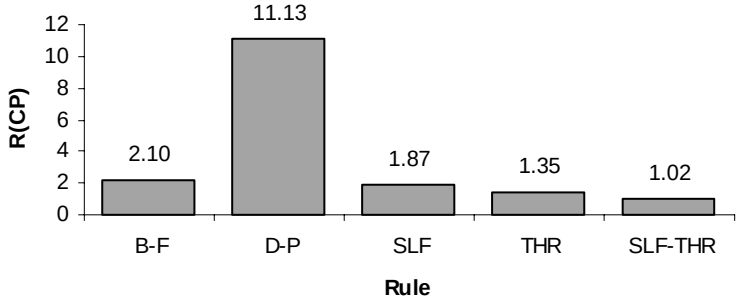

Figure 1 - Ranking based on our rule

Table 3 - Time (in seconds) to solve 16 network problems by 5 shortest paths methods

Test Problem	B-F	D-P	SLF	THR	SLF-THR
1	0.117	0.1	0.083	0.066	0.05
2	0.467	0.583	0.383	0.2	0.2
3	1,25	1.82	1.13	0.533	0.433
4	1.983	2.683	2.017	0.867	0.717
5	0.417	0.4	0.333	0.217	0.233
6	0.993	0.917	0.717	0.5	0.533
7	1.933	1.767	1.483	0.95	1.017
8	3.333	2.75	2.1	1.5	1.62
9	1.5	6.65	1.28	1.47	1.15
10	7.28	332.2	5.4	6.42	4.43
11	14.6	279.2	10.3	12.73	8.677
12	19.97	326	13.88	18	11.95
13	0.483	1	0.55	0.3	0.2
14	0.883	1.783	1.033	0.733	0383
15	1.233	2.333	1.56	0.95	0.65
16	1.75	3.567	2.033	1.85	0.817

Table 4 - Nonlinear optmization codes used for computational experiments

Code	Reference	Code	Reference
C1	CONMIN-CG	C2	CONMIN-BFGS
C3	E04DGF	C4	L-BFGS (m=3)
C5	L-BFGS (m=5)	C6	L-BFGS (m=7)
C7	BBVSCG (m=3)	C8	BBVSCG (m=5)
C9	BBVSCG (m=7)		

The results are summarized in Table 5.

Table 5 - Time (in seconds) to solve 18 unconstrained optimization problems by the 9 codes

P	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	0.0233	0.0238	0.0211	0.0278	0.0316	0.0383	0.0229	0.0326	0.0315
2	0.026	0.0557	0.0301	0.0556	0.0523	0.0513	0.0456	0.0619	0.0516
3	0.0028	0.0058	0.007	0.0059	0.0073	0.0074	0.0032	0.0042	0.0031
4	0.0912	0.1042	0.077	0.1752	0.2045	0.2461	0.1066	0.1359	0.1723
5	0.0109	0.0285	0.0278	0.031	0.0361	0.0383	0.0175	0.023	0.0279
6	0.0052	0.0373	0.0148	0.0187	0.0224	0.0256	0.0129	0.0142	0.0218
	0.0224	1.7774	0.0464	0.0733	0.0936	0.1125	0.0288	0.0586	0.0707
7	0.4707	0.287	F	3.5291	0.7419	0.4794	0.4597	0.4616	0.4111
	2.1548	1.1544	F	F	5.7458	3.2839	1.8518	1.4923	0.7219
8	0.0604	3.5806	F	0.1266	0.159	0.1824	0.1276	0.1066	0.1271
	0.1437	0.0355	0.1997	0.0182	0.0261	0.0236	0.0186	0.019	0.0228
9	0.2719	F	0.1596	0.4425	0.5301	0.5577	0.2381	0.2883	0.3168
10	0.0063	0.0073	0.0097	0.0142	0.0162	0.0177	0.0098	0.0108	0.0191
11	0.4817	0.0356	0.0231	0.0432	0.0305	0.0387	0.0344	0.0262	0.0279
12	0.0207	0.0183	0.033	0.0337	0.0355	0.0402	0.0187	0.0199	0.0222
13	0.1457	2.1017	0.0749	0.1094	0.1345	0.1574	0.0968	0.1431	0.1422
14	0.0457	1.7946	0.0378	0.0706	0.0941	0.1227	0.0576	0.0645	0.0854
15	0.1247	2.0963	0.0536	0.1367	0.1512	0.1243	0.0854	0.1087	0.1494
16	0.0078	0.0103	0.0114	0.015	0.0151	0.0169	0.013	0.0161	0.0163
17	0.0497	0.0362	0.0871	0.1073	0.1152	0.126	0.0272	0.0253	0.0262
18	8.8249	26.57	9.0296	8.2528	6.6732	6.6005	8.5565	7.229	9.2365

By ranking the codes according to our rule, we obtain the scores reported in Fig. 2.

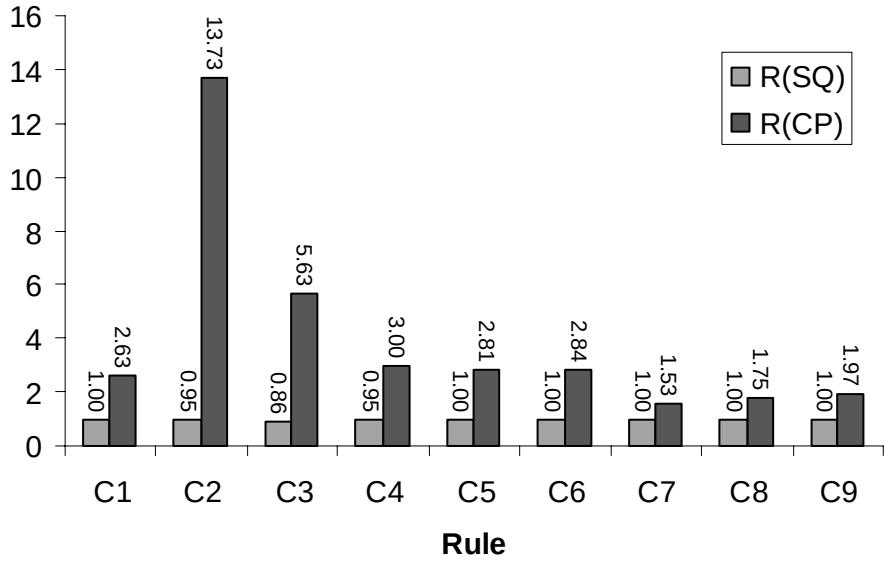

Figure 2 - Ranking based on our rule

In [11] it is pointed out that all methods have a practical appeal. E04DGF appears to be the least efficient method for the library test problems, and when the number m is increased from 3 to 7, there is no significant improvement in performance. Similar conclusions (and many others) can be derived by examining the details of the results of Figure 2. This confirms that our rule is sound and reliable.

5. CONCLUSION

In this paper we have proposed a new cumulative index for ranking numerical methods used to solve optimization problems. The results have confirmed that our index is sound and reliable and, thus, it promises to be a useful tool to measure the performance of any optimization method, even implemented in parallel, especially when several methods are used for the comparison and the number of test problems is so large to make difficult the analysis of the numerical results using other approaches.

ACKNOWLEDGEMENT

This research was partially supported by the Center of Excellence for High Performance Computing, University of Calabria, Italy. This support is gratefully acknowledged.

REFERENCES

1. R.H.F. Jackson, P.T. Boggs, S.G. Nash and S. Powell, "Guidelines for Reporting Results of Computational Experiments: Report of the ad hoc Committee", *Mathematical Programming*, Vol. 49, pp. 413-425, 1991.
2. F.A. Lootsma, "Comparative Performance Evaluation, Experimental Design, and Generation of Test Problems in Non-Linear Optimization", In K. Schittkowski (editor), *Computational Mathematical Programming*, NATO ASI Series, Springer-Verlag, Berlin, pp. 249-260, 1985.
3. R.S. Barr, and B.L. Hickman, "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Options", *ORSA Journal on Computing*, Vol. 1, pp. 2-32, 1993.
4. K.L. Hiebert, "An Evaluation of Mathematical Software that solves Nonlinear Least Square Problems", *ACM Transaction of Mathematical Software*, Vol. 7, pp. 1-16, 1981.
5. J.J. More, B.S. Garbov, and K.E. Hillstrom, "Testing Unconstrained Optimization Software", *ACM Transaction of Mathematical Software*, Vol. 7, pp. 17-41, 1981.
6. K.L. Hiebert, "An Evaluation of Mathematical Software that solves Systems of Nonlinear Equations", *ACM Transaction of Mathematical Software*, Vol. 8, pp. 5-20, 1982.
7. M. Al-Baali, "A Rule for Comparing Two Methods in Practical Optimization", *Technical Report No. 119, Department of Systems*, University of Calabria, Rende, Italy, 1992.
8. A.A. Brown and M.C. Bartholomew-Biggs, "Some Effective Methods for Unconstrained Optimization

Based on the Solution of Systems of Ordinary Differential Equations", *Technical Report No. 178*, *Numerical Optimization Centre, The Hatfield Polytechnic*, Hatfield, England, 1987.

9. D.P. Bertsekas, F. Guerriero and R. Musmanno, "Parallel Shortest Paths Methods for Globally Optimal Trajectories", In J. Dongarra, L. Grandinetti, J. Kowalik, G. Joubert (editors), *High Performance Computing: Technology and Applications*, Elsevier, Amsterdam, pp. 303-315, 1995.
10. D.P. Bertsekas, "A Simple and Fast Label Correcting Algorithm for Shortest Paths", *Networks*, Vol. 23, pp. 703-709, 1993.
11. X. Zou, M. Navon, M. Berger, K.H. Phua, T. Schlick and F.X. Le Dimet, "Numerical Experience with Limited-Memory Quasi-Newton and Truncated Newton Methods", *SIAM Journal on Optimization*, Vol. 3, pp. 582-608, 1993.

Andrea Attanasio *is graduated in Electrical Engineering (1989) at University of Calabria; he obtained a master degree in Information Technology (1990) at CEFRIEL Center in Milan. He worked several years in multinational information technology companies. Actually he works as Technical Director of the Center of Excellence for High Performance Computing at University of Calabria, a multi disciplinary center which addresses many topics, including grid computing, optimization and simulation of large scale systems. His areas of interest are: high performance computing; grid computing; optimization of logistics and transportation systems; information & communication technology.*

Patrizia Beraldi *is an Assistant Professor of Operations Research at the Faculty of Engineering, University of Calabria. She received in 1999 the Ph.D in Computer Science and System Engineering from the University of Calabria. Her major research interests: network optimisation, stochastic programming, theory and applications, and parallel computing.*

Her publications have appeared in a variety of journals, including Computational Optimization and Applications, Optimization Methods & Software, Journal of Optimization Theory and Applications, Parallel Computing, Operations Research, Computers and Operations Research, European Journal of Operational Research.

Francesca Guerriero *is an Associate Professor of Operations Research at the University of Calabria, Italy, Faculty of Engineering. She received a Ph.D in Computer Science and System Engineering from the University of Calabria. Her major research interests are in optimisation theory, logistics, network optimisation and parallel computing.*

Her publications have appeared in a variety of journals, including Computational Optimization and Applications, Optimization Methods & Software, Journal of Optimization Theory and Applications, Parallel Computing, Operations Research, Computers and Operations Research, European Journal of Operational Research.