An Efficient Limited Memory Interval Algorithm for Global Optimization

Min Sun

Abstract—**This article presents a global optimization algorithm of the interval type that requires only a limited amount of memory and treats standard constraints. It is shown to be able to find one globally optimal solution under certain conditions. It has been tested with many examples with various degrees of complexity and a large variety of dimensions ranging from 1 to 100 in a basic personal computer. The numerical experiments have indicated that the algorithm would have a better chance to successfully find a good approximation of a globally optimal solution than a recently proposed memoryless version. Yet, it still finds such a solution much more quickly and using much less memory space than a conventional interval method. The effects of the memory size on reliability and overall efficiency are investigated. A good compromised algorithm would require only a very limited memory size.**

Index Terms—**Constraints, Global optimization, Interval algorithm, Limited memory.**

I. Introduction

Many important operations research problems aim at solving this problem

minimize f(x),
subject to $h(x) = 0$, $g(x) \leq 0$, $x \in X$. (P)

There are two commonly used global optimization approaches: stochastic or deterministic. Stochastic algorithms search the whole domain only in a probabilistic fashion so that at most they can yield a good estimate of a globally optimal solution in a probabilistic sense. Stochastic search methods (such as the simulated annealing method and genetic algorithms) have been more popular choices than deterministic methods because of their simplicity of implementation, relative quickness for reaching an approximate solution, less memory demands, and a wider range of applicable problems. Deterministic algorithms offer attractive alternatives for solving problem (P). They are generally based on the idea of branch and bound [11]. Among them, interval methods offer both sound theoretical foundation and reliable numerical solutions [14]. Despite attractive features of the interval method, most published reports on their applications seem to be generally limited to optimization problems in low dimensions (say, much less than 100 according to our recent survey of literature). Obviously, there are three major concerns in solving large dimensional problems: large amount of memory space, slow speed of convergence, and requirement of acceptable bounds of the objective function over any interval subdomains. A memoryless interval algorithm has been recently proposed, aiming at easing the first two concerns [17]. Indeed, the reported results have indicated that the memoryless interval algorithm significantly improved memory requirement and convergence speed, while retaining a good degree of reliability. This article reports one new version of the interval algorithm that shows improvement in reliability while sacrificing little both in memory space usage and in overall speed of convergence.

Manuscript received December 5, 2008.

M. Sun is with Department of Mathematics, The University of Alabama,Tuscaloosa, AL 35487 USA, currently visiting Applied Math Dept of Hong Kong PolyU (present phone: 852-27665641; fax: 852-23629045; e-mail: msun@gp.as.ua.edu).

II. Interval Methods

The standard branch and bound method was originally introduceed in [5] and [10], and more recently presented in [11]. Its main idea is the recursive refinement of partition of the search domain and underestimation of f(x) over the partitioned subdomains. Interval methods (see [14] for earlier work) are in the general framework of branch and bound along with interval arithmetic. The interval arithmetic provides an effective means of underestimation of programmable functions, and offers an additional benefit of including roundoff errors. Following the initial works in late 1950s and early 1960s, research on interval methods became a more heated topic from late 1970s to early 1990s (cf. [1], [16], [8]) among many researchers in several fields. A solid foundation had been laid by the end of 1980s. Subsequent improvements were done since 1990s (e.g., [4], [3], [18], [15], [20]).

Let f^* be the global minimum value of the objective function f(x), and x^* a global minimizer in X. As in the interval analysis literature, we use boxes and intervals interchangeably. A typical interval method uses 2 major objects, a list L that holds all the subintervals of partitions that remain to be processed, and an inclusion function $F(Y) = [Lb(F(Y)), Ub(F(Y))]$ that offers a lower bound and an upper bound of f(x) over any box Y to be processed. The general procedure would consist of these major steps.

Algorithm 1. (Standard interval algorithm for global optimization)

1). Initialization. Set the list $L = \phi$. Set the working box $Y=X$.

2). Subdivision of Y. The algorithm splits up Y into subboxes. Add the resulting subboxes to L.

3). Deletion conditions: To increase efficiency of the method, unwanted boxes V (where no global minimizer can be located) need to be identified and then deleted.

4). Selection of a new working box Y from L.

5). Termination criterion. Obviously, any interval algorithm stops if there are no more boxes to be processed. But practically, it may stop earlier according to some other termination criteria.

Two well known early versions of interval methods (Ichida-Fujii [12] and Hansen [8]) fall into this framework. Ichida-Fujii's algorithm selects a new working box based on the smallest lower bound of inclusion function, while Hansen's algorithm selects a new working box based on the oldest age or on the largest size. By now, there are a large variety of implemented versions of the interval method (e.g., [3], [18]). There are also several accelerating devices reported in the literature. Interval methods have been used for solving many different kinds of mathematical problems arising from various fields of applications. But a quick survey of a large number of published reports on their applications seems to indicate that they are generally limited to problems in fairly low dimensions (say, much less than 100 in most cases). Obviously, there are indeed two major concerns in solving large dimensional problems: large amount of memory space required to hold boxes for further processing, and slow speed of convergence due to a large number of boxes to be processed.

Maintaining a memory structure is seen as a very common strategy in many global optimization methods. Genetic algorithms and its variations explicitly maintain a population of candidate solutions. Tabu search [7] maintains a tabu list that represents information about recently visited solutions. A standard interval algorithm keeps track of a list of all the subboxes that might contain some global solutions. In case unisolated global solutions exist, this list can grow very quickly without a finite bound. There is even a memory-based version of simulated annealing [2]. Even some local search methods also employ a memory structure. One typical example is the BFGS quasi-Newton method where an approximate inverse Hessian matrix has to be memorized between two consecutive iterations of update.

Memory structures are used in various algorithms for different purposes. In the case of standard interval algorithm, a list of boxes with unlimited length is used to ensure that no global solutions will be lost. But whether all the global solutions will be identified to any desired degree of accuracy depends on specific implementation of the algorithm. For example, only one global solution is guaranteed to be estimated accurately by the standard Ichida-Fujii algorithm. But the standard Hansen's algorithm is capable of identifying all the global solutions (possibly under expanses of a lot more CPU times). It is commonly believed that any optimization method that is capable of identifying one global solution or a good estimate of one global solution within a reasonable time frame would be of a good practical value. One memoryless interval algorithm was recently designed, which only targets one global solution in a way similar to Ichida-Fujii interval algorithm. But for the other global solutions, it no longer commits any computer memory and CPU time since they may not be extracted accurately anyway. It trades the loss of other global solutions with much improved memory requirement and convergence speed. It completely abandons the list, breaking away from the standard memory philosophy of the interval method and the branch-and-bound method in general.

Algorithm 2. (Memoryless interval algorithm for unconstrained global optimization)

1). Initialization. Set the working box Y=X.

2). Subdivision of Y. The algorithm splits up Y into subboxes.

3). Deletion conditions: Unwanted subboxes are identified and deleted.

4). Reset Y to the subbox V with the lowest Lb(F(V)).

5). Check termination criterion.

The 3 major steps can be implemented without using a list. Theoretically, the memoryless algorithm is guaranteed to capture one global solution under certain conditions. Numerically, it has a good chance to capture one global solution. It is an interval algorithm with the least amount of memory requirement. Thus it is likely the fastest interval algorithm. But several important issues remain to be investigated. One of them is the improvement of reliability. We address this issue by reintroducing the list. But unlike the standard interval method, the list is no longer unlimited. We add a small hard limit M on the length of list.

Algorithm 3. (Limited memory interval algorithm for unconstrained global optimization)

Given f(x), X, M, and F(.).

Step 1. Initialization:

Step 1a. Set a working interval Y=X. Set the list L = ϕ.

Step 1b. Get F(Y).

Save f_{best} = f(c), where c = Mid(Y).

Step 1c. Set y = Lb(F(Y)).

Step 2. Update:

Step 2a. Take any k in {i: Wid(Y) = Wid(Y_i)}, where Y = Y_1x Y_2… xY_d.

Step 2b. Bisect Y normal to the coordinate direction k, obtaining intervals V_1 and V_2.

Step 2c. Get F(V_1) and F(V_2).

Step 2d. Set y_1 = Lb(F(V_1)), y_2 = Lb(F(V_2)).

Step 2e. Deletion. Check deletion condition(s) to see if V_1 and V_2 can be deleted. For example,

$f_{best} < y_i \rightarrow$ Delete V_i, for i=1, 2.

Step 2f. Place surviving box(es) into list L. If the total number of boxes in L exceeds M, only the M boxes with the best y-values are kept.

Step 2g. Selection. Select a box from L that has the smallest y-value as Y, and the corresponding y-value becomes y.

Step 2h. Update f_{best} = min{f_{best}, f(c)}, where c = Mid(Y).

Step 3. If one of the prescribed termination criteria holds, then stop with output:

$f^* \cong y$, $f^* \in F(Y)$, $x^* \cong$ Mid(Y).

Step 4. Go to Step 2.

Theoretically, the new algorithm is guaranteed to capture one global solution under the same conditions used for the convergence of the memoryless algorithm. Numerically, it has a better chance to capture one global solution. A large amount of supporting numerical evidence will be presented in the next section. Some of our test examples contain additional constraints. When constraints are present in (P), the algorithm

can be adjusted to handle them (see [17] or [18]).

III. Numerical Results

In our implementation of interval algorithms 1-3, several other acceleration devices are incorporated whenever appropriate.

L_p = a primary list of boxes that represents the remaining region to be searched. This is used only in algorithm 1.

L_s = a saved list of boxes that are not deleted but do not need to be further processed (i.e. inactive) according to some prescribed tolerances (ε_{box}, ε_f) listed below. This is used mainly in algorithm 1.

ε_{box} = a small box size threshold. Any active box V with size Wid(V) less than ε_{box} will be moved from L_p to L_s.

ε_f = a small threshold of deviation of the objective function values. Any active box with the fluctuation of the objective function value less than ε_f will be stored into L_s as well.

nf_{max} = the maximum number of function (f(.) or F(.)) calls allowed. It is checked only once for every certain number of iterations. This limit is relaxed when an algorithm continues to improve its best solution.

cpu_{max} = the maximum CPU time allowed. It is also checked once for every certain number of iterations.

A bad initial solution is supplied to each algorithm for every test example. It is used to initialize f_{best}. However, when constraints are present, an infeasible initial solution is intentionally selected which increases degree of difficult and reduces success rate under the specified stopping conditions.

To test performance of the new algorithm, we have used a large number of examples with or without constraints. Most of these examples have been widely used by other people for testing their new optimization algorithms (e.g., [6], [9], [13], [18] , [19]). Among those are: Rastrigin function, Goldstein-Price function, piecewise function, Levy functions, Branin function, Shubert function, our linear complementarity problem, our discrete Halmilton-Jacobi-Bellman equation problem, De Jong function, Colville function, Griewank function, Rosenbrock function, Zakharov function, sphere function, Schwefel functions, step function, and Ackley function. Modified versions of some of those functions have been included as well. Among those examples, 16 of them are formulated with flexible dimensions. We vary those dimensions as 4, 10, 40, and 100. Different dimensions resulted in different test examples. The total number of examples we have tested is over 100. Their dimensions vary from 1 to 100. Many of those examples are often regarded as difficult benchmark examples by other people. Obviously it is not a good idea to explicitly state all those examples. Since all the test examples are taken from published papers, the currently best known objective function value of each example is generally available. If it is not available, our own best solution is adapted. Thus we have shifted each objective function so that the currently best known objective function value of any optimization problem becomes zero. Constrained and unconstrained problems are separately grouped so that we may get a better idea of effect of constraints on the performance of the algorithms. Similarly, problems are further divided into 4 groups in terms of the size of dimension (1-6 for small dimensions, 9-13 for medium dimensions, 40-100 for high dimensions). Limits on the number of function calls and CPU time consumption are set to different values for the different ranges of dimensions. The same examples are also included in [17]. But test results are not identical for the first 2 algorithms since different sets of algorithm parameters have been used so that new numerical results are generated instead of duplication of existing results.

We have used two ranking scores (originally introduced in [17]) to quantitatively measure the performance of each algorithm. One of them is a composite ranking score to quantitatively compare various results. A composite ranking score R_q reflects the quality of the final solution in terms of the objective function value as well as the maximum amount of constraint violation. More precisely, we first calculate a ranking score r_f based on the final best objective function value (called f_{best}).

Objective Function Value: Ranking Score r_f ($f^* = 0$)					
f_{best}	<0.001	<0.01	<1.0	< 10	≥ 10
r_f	1 (best)	2	3	4	5

Then we calculate a ranking score r_c based on the maximum amount of constraint violation of the final best solution $V_c = \max\{|h_i(x_{best})|, \max\{0, g_j(x_{best})\}: i=1,\ldots, m, j=1,\ldots, p\}$.

Constraint Violation Amount: Ranking Score r_c ($V^*_c = 0$)					
V_c	<0.01	<0.1	<1.0	< 10	≥ 10
r_c	1 (best)	2	3	4	5

The composite ranking score for solution quality is then defined as

$$R_q = \max\{r_f, r_c\}.$$

The other score is the total number of objective function (f(.) or F(.)) calls (n_f or n_F). Those two scores would reflect the effectiveness of global optimization algorithm. For constrained optimization problems, each algorithm would require a certain number of calls of the constraint functions. Those calls have been omitted when the number of function calls is calculated. We observe that the original objective function and its inclusion function would require significantly different computational efforts. So they are separately counted. Then additional numerical tests are performed to estimate how many f-calls (say, N_{Ff}) would be equivalent to a single F-call. This factor is used to determine a combined number of objective function calls.

$$R_{nf} = n_f + N_{Ff} * n_F.$$

The total number is a major effectiveness indicator. But CPU time would also include various CPU time overheads required by each algorithm. But due to page limitations, CPU results are omitted.

Numerical results of 6 different sets of examples are presented below. A separate table is displayed for each set of examples. We have implemented all the three interval algorithms. Their main difference is the limit on the length of list L. The limit is 1 for the memoryless interval algorithm (MLIA). The standard interval algorithm (SIA) uses a fairly large limit (say, 50,000) and it stops when that limit is reached. The new limited memory interval algorithm (LMIA(M)) is tested with limit values M=2, 3, 4, 5, 6, 7, 8, 9,

10, 20, 30, 40, and 50. These values are shown in the first row of each table given later.

The main body of each table contains two sets of data separated by /: quality ranking scores R_q and percentage of R_{nf} relative to the maximum value in each row. Several percentage figures show 00, indicating the fact that those R_{nf} values are of at least 2 orders of magnitudes smaller than their respective maximum values. The first column shows the example id. The second column contains |L|, the actual size of list L under SIA. Each of the remaining columns contains R_q over R_{nf}-percentage. The bottom two rows are averaged values of R_q and R_{nf}-percentage. All of the test results have been generated by an AMD Turion 64 X2 mobile technology TL-58 /1.9GHz laptop computer with 2GB of RAM under 32 bit Windows Vista environment.

Example set 1. This set contains 41 examples of small dimensions ranging from 1 to 6 that do not contain any constraints other than the bound constraints. Summary of the test results is shown in Table 1. Clearly, MLIA is consistently the fastest. Although LMIA(M) with small M values may take a little bit more time to converge, they usually improve the quality of final solution. SIA is most reliable. But it requires more computational efforts most of times.

Example set 2. This set contains 20 examples of small dimensions between 1 and 6 with additional equality and\or inequality constraints. Now the overall ranking R_q would reflect the quality of final solution in terms of its objective function value as well as the amount of constraint violation. Summary of its test results is in Table 2. Generally speaking, the performance of each algorithm is down a little compared with its performance on unconstrained problems. This is partially due to the fact that we used the worst case scenario in the initialization step as pointed out in Section II.

Example set 3. This set contains 15 unconstrained examples of medium dimension 10. Summary of the test results is in Table 3. The new algorithm is about 2 orders of magnitudes faster than its standard version in terms of the number of objective function calls, while it maintains a compatible degree of quality. For these 15 examples, performance of MLIA and LMIA(M) somehow exceed our normal expectations.

Example set 4. This set contains 5 constrained problems with dimensions ranging from 9 to 13. Summary of its test results is in Table 4. Again, the constraints very much affected every algorithm's performance. Constraints made MLIA and LMIA(M) to exit more quickly. The tough constraints made SIA worse than MLIA and LMIA(M) in quality ranks. In fact, the data show that SIA did not encounter any feasible solutions at all after processing so many boxes. This issue is to be examined further elsewhere.

Example set 5. This set contains 15 unconstrained problems with dimensions all equal to 40. Summary of its test results is in Table 5. Now MLIA becomes the best in all aspects of rankings. We did not test enough constrained problems of dimensions 40 or higher. So no results on constrained problems of higher dimensions will be reported below.

Example set 6. This set again contains 15 unconstrained problems with dimensions all equal to 100. Summary of its test results is in Table 6. In any event, MLIA is still a lone top performer in terms of the number of function calls.

In conclusion, LMIA(M) for relatively small values of M possess all the major observed advantages of the newly developed MLIA over SIA. They improved the reliability of MLIA with a little sacrifice of additional computational time (the memory increase is pretty much negligible). It is a good idea to use MLIA to quickly get trial solutions. In they are not satisfactory, LMIA(M) with a very limited M value (say 4) would be adopted to get improved results. In most cases, this strategy would yield compatible final solutions with less CPU time than SIA.

REFERENCES

[1]. Alefeld, G., Herzberger, J.: Introduction to Interval Computations, Academic Press, New York, NY (1983)
[2]. Ali, M.M., Torn, A., Viitanen, S.: A direct search variant of the simulated annealing algorithm for optimization involving continuous variables, Computers & Oper. Res. 29, 87-102(2002)
[3]. Clausen, J., Zilinskas, A.: Subdivision, sampling, and initialization strategies for simplical branch and bound in global optimization, Computers & Math. Applications 44, 943-955(2002)
[4]. Csallner, A.E.: Lipschitz continuity and the termination of interval methods for global optimization, Computers & Math. Applications 42,1035 -1042(2001)
[5]. Falk, J. E., Soland, R.M.: An algorithm for separable nonconvex programming problems, Management Science 15, 550-569(1969)
[6]. Floudas, C.A., Pardalos, P.M.: A Collection of Test Problems for Constrained Global Optimization Algorithms, Springer-Verlag, Berlin Heidelberg (1990)
[7]. Glover, F.: Tabu search — Part I, ORSA J. on Computing 1: 3, 190-206(1989)
[8]. Hansen, E.R.: Global Optimization Using Interval Analysis, Marcel Dekker, NY (1992)
[9]. Hedar, A.R., Fukushima, M.: Derivative-free filter simulated annealing method for constrained continuous global optimization, J. of Global Optimization 35, 521-549(2006)
[10]. Horst, R.: An algorithm for nonconvex programming problems, Mathematical Programming 10, 312-321(1976)
[11]. Horst R., Tuy H.: Global Optimization, Deterministic Approaches, Springer-Verlag, Berlin (1990)
[12]. Ichida, K., Fujii, Y.: An interval arithmetic method for global optimization, Computing 23, 85-97(1979)
[13]. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed., Springer-Verlag, Berlin (1996)
[14]. Moore, R.E.: Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ (1966)
[15]. Pedamallu, C. S., Ozdamar, L., Csendes, T., Vinko, T.: Efficient interval partitioning approach for global optimization, J. of Global Optimization 42, 369-384(2008)
[16]. Ratscheck, H., Rokne, J.: New Computer Methods for Global Optimization, Wiley, New York, NY (1988)
[17]. Sun, M., A Fast memoryless interval algorithm for global optimization, submitted for publication (2008).
[18]. Sun, M., Johnson, A. W.: Interval branch and bound with local sampling for constrained global optimization, J. of Global Optimization 33, 61-82(2005)
[19]. Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster, IEEE Trans. on Evolutionary Computation 3, 82-102(1999)
[20]. Zhang, X., Liu, S.: Interval algorithm for global numerical optimization, Engineering Optimization 40, 849 – 868(2008)

Table 1. Quality ranking scores R_q / percentage of R_{nf}: example set 1

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
1	58	1/98	1/98	1/98	1/98	1/99	1/100	1/100	1/98	1/89	1/79	1/70	2/16	2/15	2/11	2/09
2	62	1/97	1/97	1/97	1/98	1/98	1/100	1/100	1/100	1/91	1/76	1/67	1/57	1/47	1/35	3/07
3	51	1/99	1/99	1/99	1/99	1/99	1/100	1/100	1/100	1/98	1/73	1/65	3/28	3/25	3/23	3/20
4	65	1/98	1/98	1/98	1/98	1/99	1/100	1/100	1/100	1/94	1/84	1/71	1/61	1/49	1/36	1/21
5	5	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
6	1	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
7	37	1/97	1/97	1/97	1/97	1/97	1/99	1/99	1/99	1/100	1/98	1/84	1/65	4/53	4/39	4/22
8	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/67	1/67
9	230	1/100	1/68	1/69	1/52	1/52	3/19	3/17	4/12	4/11	4/10	4/09	4/08	4/20	4/14	4/07
10	3110	5/100	5/11	5/09	5/07	5/05	5/03	5/03	5/02	5/02	5/02	5/01	5/01	5/01	5/01	5/00
11	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/67	1/67
12	6	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/67	1/67
13	5	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/93	1/87	1/61
14	967	1/100	1/88	1/73	1/57	1/27	1/13	1/12	1/11	1/09	1/09	1/08	1/11	4/04	4/03	5/01
15	937	1/100	1/36	1/36	1/29	3/20	3/12	3/17	3/07	3/06	3/15	3/13	3/09	3/03	3/06	3/03
16	1892	5/100	5/14	5/12	5/09	5/07	5/04	5/04	5/03	5/03	5/02	5/02	5/02	5/01	5/01	/01
17	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/78	1/78
18	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	3/83	5/78
19	18	1/97	1/97	1/97	1/97	1/97	1/97	1/97	1/97	1/97	1/97	1/98	1/98	1/100	1/81	1/44
20	24	1/99	1/99	1/99	1/99	1/99	1/99	1/99	1/98	1/98	1/98	1/99	1/99	1/100	1/72	1/32
21	85	1/08	2/100	2/62	2/61	2/60	2/11	3/28	3/13	3/02	3/01	3/01	3/01	3/01	3/01	3/00
22	3856	1/100	1/18	1/14	1/11	1/07	2/04	2/03	1/04	1/04	5/02	5/01	5/03	5/02	5/01	5/01
23	293	1/100	1/86	1/72	1/72	1/44	4/11	4/10	4/09	4/09	4/06	4/06	4/05	3/17	3/11	3/06
24	17	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	3/94	3/56
25	49	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	3/99	3/85	3/81	3/59	4/48	4/32	5/04
26	13	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/36	1/36
27	51	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/67	1/67	1/62	4/34
28	12	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/97	1/86	1/62
29	9	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/72	1/66
30	11	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/99
31	113	1/92	1/92	1/92	1/92	1/77	1/100	1/91	1/76	3/53	3/59	4/51	4/44	4/35	4/23	4/13
32	8	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/96	1/96
33	22k	2/100	3/11	2/10	3/07	4/07	4/04	4/03	5/03	5/03	5/02	5/02	5/02	5/01	5/01	5/00
34	73	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/97	1/96
35	20	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/96	1/69	1/66
36	16	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/99	1/91
37	20	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
38	29	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/84	1/82
39	20	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/96
40	33	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/83
41	25k	2/100	2/11	2/09	2/07	2/05	2/02	2/02	2/02	2/02	2/02	2/01	2/01	2/01	2/01	2/00
av-q		1.24	1.29	1.27	1.29	1.37	1.51	1.54	1.56	1.66	1.76	1.78	1.85	2.00	2.10	2.32
av-n	1438	/100	/20	/16	/14	/11	/06	/07	/06	/05	/05	/04	/04	/03	/03	/02

Table 2. Quality ranking scores R_q / percentage of R_{nf}: example set 2

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
42	31	1/100	1/100	1/100	1/100	1/100	1/59	1/58	1/57	1/49	1/49	1/48	1/45	1/42	1/38	1/19
43	31	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/95	1/52	5/39
44	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/87	1/39
45	29	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/95	1/87	1/79	3/54
46	16	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/98	1/98	1/98	1/56
47	44	1/100	1/100	1/100	1/100	1/91	1/65	1/60	1/56	1/51	1/44	3/52	3/44	1/22	1/17	4/12
48	8892	1/100	1/09	1/08	1/06	1/04	1/03	1/02	1/02	1/02	1/02	1/02	1/01	1/01	1/01	1/00
49	7	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/88	1/58	1/44	1/38	1/37	1/36
50	752	1/100	1/43	1/33	1/28	1/21	1/12	1/11	2/11	2/10	2/08	3/06	3/05	5/05	5/04	3/02
51	58	1/100	1/100	1/100	1/98	1/93	1/42	1/41	1/40	1/38	1/37	1/32	1/27	1/22	1/17	1/11
52	21	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/98	2/77	2/70
53	22k	2/100	3/08	3/07	3/05	3/03	3/02	3/02	3/02	3/01	3/01	3/01	3/01	3/01	3/00	3/00
54	48	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/77	5/57
55	12k	3/100	3/08	2/06	3/05	3/03	5/01	4/02	5/01	5/01	5/00	5/01	5/01	5/01	5/00	5/00
56	261	1/100	1/100	1/100	1/83	1/68	3/75	3/69	3/64	3/57	3/51	3/49	3/39	4/27	4/21	4/07
57	642	1/48	4/100	4/77	4/57	5/38	5/14	5/14	5/12	5/14	5/13	5/04	5/02	5/01	5/01	5/00
58	4141	5/100	1/12	1/11	1/09	1/07	3/04	3/04	3/03	1/03	1/03	1/02	1/02	1/02	1/01	1/01
59	9544	1/100	1/24	1/20	1/17	1/12	1/05	1/04	1/06	3/06	3/05	3/04	3/04	3/02	3/01	3/01
60	13k	4/100	1/07	1/06	1/05	1/03	1/02	1/02	1/02	1/01	1/01	1/01	1/01	1/01	1/00	1/00
av-q		1.53	1.37	1.32	1.37	1.42	1.74	1.68	1.79	1.79	1.79	1.95	1.95	2.00	2.16	2.74
av-n	3776	/100	/16	/13	/11	/08	/04	/04	/04	/04	/03	/03	/02	/02	/01	/01

ISBN: 978-988-17012-7-5

Table 3. Quality ranking scores R_q / percentage of R_{nf}: example set 3

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
61	187	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/86	1/86	1/86	1/86	1/71	1/57	4/36
62	37	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/85
63	27	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/90	1/87
64	44	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
65	24k	5/100	5/26	5/28	5/20	5/14	5/07	5/06	5/05	5/05	5/04	5/03	5/03	5/02	5/02	5/01
66	20	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
67	25k	5/100	5/48	5/39	5/30	5/20	5/10	5/09	5/08	5/07	5/06	5/05	5/04	5/03	5/02	5/01
68	31	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
69	51	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/85	1/84
70	51	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/98	1/98	1/97	1/96
71	49	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/90	1/90
72	83	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/94	1/88
73	48	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/90	1/87
74	111	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
75	23k	3/100	3/24	3/19	3/24	3/16	2/08	3/07	2/07	2/06	2/05	2/04	3/03	3/03	3/02	3/01
av-q		1.67	1.67	1.60	1.67	1.67	1.60	1.67	1.60	1.60	1.60	1.60	1.67	1.67	1.67	1.87
av-n	4874	/100	/31	/26	/26	/18	/10	/10	/09	/08	/07	/06	/05	/05	/04	/03

Table 4. Quality ranking scores R_q / percentage of R_{nf}: example set 4

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
76	12k	5/100	5/19	5/16	5/11	5/07	5/01	5/01	5/02	5/01	5/01	5/01	5/00	5/00	5/00	5/00
77	15k	4/100	1/22	1/17	1/14	1/09	1/05	1/04	1/04	1/04	1/03	1/03	1/02	1/02	1/01	1/01
78	13k	4/100	4/26	5/23	5/16	5/14	4/03	4/03	4/02	4/01	4/02	4/01	4/01	4/00	4/00	4/00
79	7942	5/100	1/29	1/22	1/18	1/12	5/01	4/09	5/00	5/00	5/00	5/00	5/00	5/00	5/00	5/00
80	12k	4/100	3/37	3/30	3/23	3/15	3/08	3/07	3/06	3/05	3/05	3/04	3/03	3/02	3/02	4/00
av-q		4.40	2.80	3.00	3.00	3.00	2.15	3.40	3.60	3.60	3.60	3.60	3.60	3.60	3.60	3.80
av-n	12k	/100	/79	/57	/49	/40	/36	/44	/29	/27	/24	/21	/16	/12	/09	/04

Table 5. Quality ranking scores R_q / percentage of R_{nf}: example set 5

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
81	857	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/96	1/93	1/93	1/89	1/82	1/75	1/53	4/36
82	186	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
83	146	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
84	221	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/93
85	63k	5/100	5/67	5/51	5/37	5/13	5/12	5/12	5/11	5/10	5/08	5/06	5/05	5/04	5/03	5/01
86	80	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
87	61k	5/100	1/85	1/68	1/51	1/34	1/17	1/15	1/14	1/12	1/10	1/09	1/07	1/05	1/03	1/02
88	137	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
89	277	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
90	120	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
91	212	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
92	63k	5/100	5/86	5/69	5/52	5/34	5/17	5/15	5/14	5/12	5/10	5/09	5/07	5/05	5/03	5/02
93	209	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
94	540	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
95	45k	4/100	3/92	3/69	3/56	3/37	3/18	3/18	3/15	3/15	3/12	3/10	3/08	3/06	3/04	3/02
av-q		2.00	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	1.93	2.13
av-n	16k	/100	/87	/67	/53	/35	/20	/19	/17	/16	/14	/12	/10	/08	/06	/04

Table 6. Quality ranking scores R_q / percentage of R_{nf}: example set 6

ex	|L|	50k	50	40	30	20	10	9	8	7	6	5	4	3	2	1
96	2237	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/99	1/96	1/93	1/90	1/84	1/76	1/53	4/36
97	464	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
98	388	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
99	742	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
100	55k	5/50	5/100	5/78	5/59	5/38	5/18	5/16	5/15	5/13	5/11	5/09	5/07	5/05	5/04	5/02
101	200	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
102	52k	5/41	5/100	5/80	5/61	5/41	5/20	5/18	5/16	5/14	5/12	5/10	5/08	5/06	5/04	5/02
103	377	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
104	709	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
105	386	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
106	654	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
108	54k	5/40	5/100	5/80	5/60	5/40	5/20	5/18	5/16	5/14	5/12	5/10	5/08	5/06	5/04	5/02
109	579	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
110	1585	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100	1/100
111	101	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100	5/100
av-q		2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.07	2.27
av-n	11k	/48	/100	/80	/62	/43	/24	/22	/20	/18	/16	/15	/13	/11	/9	/7