An Iterative Technique for Improved Two-level Logic Minimization

Kunal R Shenoy Nikhil S Saluja Sunil P Khatri
Dept of Electrical and Computer Engineering,
University of Colorado at Boulder
{shenoyk, saluja, spkhatri}@colorado.edu

Abstract

In this paper, we describe an iterative heuristic technique to improve the quality of results obtained during two-level logic minimization using ESPRESSO. Although ESPRESSO minimizes the number of cubes in the solution effectively, there are several problem instances where its results are worse than the Quine-McCluskey based exact minimization technique. Our technique is designed to improve the results of ESPRESSO while utilizing ESPRESSO's Unate Recursive Paradigm based optimization heuristics, on account of their simplicity and power. Our technique is based on performing a series of iterations of ESPRESSO, in each of which we extract a number of cubes and append them into a HYPER-COVER. For the given (and subsequent) iterations of ESPRESSO, these cubes are considered as don't care cubes. A greater number of don't cares results in more and/or different reductions being explored by ESPRESSO, allowing possibly new expansions subsequently. By effectively selecting the number of iterations performed by our heuristic, we can trade off the improvement in solution quality against the run-time of our algorithm. We have implemented several variants of our iterative algorithm, and have compared their effectiveness. We show that with a small number of iterations, our technique is able to improve on the number of cubes in the solution, with an acceptable run-time overhead. The best variant is able to improve the ESPRESSO cube count by up to 18%, with an acceptable increase in run-time. In 58 examples where the ESPRESSO results can be potentially improved, one of the variants of our algorithm demonstrated better results than ESPRESSO for 27 cases.

1 Introduction

The objective of a two level minimizer is to find a sum-of-products representation with a minimal number of implicants while preserving functionality. Quine and McCluskey [1, 2] provided an algorithm to find the minimum cube cover for a given logic function. However, this algorithm requires the solution of the unate covering problem on a matrix which can have an exponential number of rows and columns. In practice, this method often times out on reasonable sized covers. Newer exact methods [3] improve performance, but are still fundamentally limited by the problem complexity. As a result, heuristic algorithms are typically utilized for two level minimization. ESPRESSO [4] is a heuristic two level logic minimization tool based on the Unate Recursive Paradigm. The minimization results of ESPRESSO are typically near optimal, with dramatically reduced run-times compared to the exact technique. In spite of the fact that ESPRESSO was developed about 20 years ago, this technique has been hard to improve on in terms of speed and quality of results.

In this paper, we describe our technique to bridge the gap between the results of an exact minimization tool and ESPRESSO, with the motivation that there may be situations where a designer would be prepared to spend additional computational resources to obtain a better cover. Our approach is iterative in nature, and based on the ESPRESSO algorithm at the core. The ESPRESSO routine is augmented by iteratively extracting a selected subset of primes of the cover, removing them from the on-set and adding them to the don't care set of the function. This allows the REDUCE algorithm of ESPRESSO to find better reductions of the cover, resulting in better overall results. We demonstrate that by performing a few iterations in this manner, we can obtain better results than ESPRESSO, with an acceptable run-time increase.

The remainder of this paper is organized as follows. Section 2 gives the motivation behind our algorithm. Section 3 gives a brief description of the Espresso algorithm while Section 4 describes the our algorithm. Section 5 describes 2 variants of our basic algorithm whereas Section 6 presents the results of our algorithms as compared to both Espresso and the exact methods. Finally we conclude with Section 7.

2 Motivation

Exact two-level logic minimization produces an minimum cover for a given logic function. This is done by reducing the original cover to a set of prime implicants, and then solving a unate covering problem where the columns are the $O(\frac{3^n}{n})$ primes and the rows are $O(2^n)$ minterms, resulting in a problem that is potentially doubly exponential in size. ESPRESSO is a heuristic algorithm which produces a near optimum solution in a fraction of the time, using an efficient Unate Recursive Paradigm. For small functions, ESPRESSO usually produces the minimum cover. However, for large functions, its solution may not be optimum. The motivation behind our algorithm is to obtain a solution closer to the minimum solution, with an acceptably higher run-time. Such an algorithm would be desirable when a designer may have additional computational resources at hand, and require a solution which is as close to optimum as possible. Our technique allows the user to explore this tradeoff.

Table 1 provides a concrete validation of these observations,

by comparing the exact and ESPRESSO algorithms for a few benchmark circuits from the MCNC91 benchmark suite. We observe that for the 7 examples in which both the methods complete, ESPRESSO typically produces results with a comparable number of cubes, while requiring dramatically lower run-times. The exact technique times out for the larger designs, and requires up to two orders of magnitude higher run-times than ESPRESSO for the others.

Based on Table 1, we motivate our work by observing that it would be desirable to come up with a technique which would improve the results of ESPRESSO, with an acceptably higher run-time than ESPRESSO. This would help bridge the gap between ESPRESSO and the exact technique. Note that for the *pdc* example, ESPRESSO produces a solution with 145 cubes while the exact method produces a solution with 96 cubes. Our goal is to bridge this gap in solution quality. Such a large difference in results is rare, however.

3 Basic ESPRESSO Algorithm

The ESPRESSO [4, 5] procedure for two level logic minimization is shown in Algorithm 1.

Algorithm 1 ESPRESSO(ϵ)

$(F, D, R) \leftarrow DECODE(\epsilon)$
$F \leftarrow EXPAND(F, R)$
$F \leftarrow IRREDUNDANT(F, D)$
$E \leftarrow ESSENTIAL\_PRIMES(F, D)$
$F \leftarrow F - E; D \leftarrow D + E$
repeat
  repeat
    $F \leftarrow REDUCE(F, D)$
    $F \leftarrow EXPAND(F, R)$
    $F \leftarrow IRREDUNDANT(F, D)$
  until not fewer terms in F
  $//LASTGASP$
  $G \leftarrow REDUCE\_GASP(F, D)$
  $G \leftarrow EXPAND(G, R)$
  $F \leftarrow IRREDUNDANT(F + G, D)$
  $//LASTGASP$
until $G = 0$
$F \leftarrow F + E; D \leftarrow D - E$
$LOWER\_OUTPUT(F, D)$
$RAISE\_INPUTS(F, R)$
$error \leftarrow (F_{old} \not\subset F) or (F \not\subset F_{old} + D)$
$return(F, error)$

ESPRESSO is based on the unate recursive paradigm [4]. Each of the core routines (REDUCE, EXPAND and IRREDUNDANT) in ESPRESSO are based on recursively splitting a function until unate leaves are obtained. The operation in question is then performed on the unate leaves, and the results recursively merged upwards to obtain the final result.

The core routines used in Espresso are briefly described below:

- EXPAND : Cubes are expanded, one at a time, by blocking them against the offset. All possible expansions of the cube are implicitly computed in this manner, and represented as a unate function. The expansion that covers most other cubes is selected.
- IRREDUNDANT : The main purpose of irredundant is to eliminate redundant cubes. IRREDUNDANT finds a minimal subset of cubes of a cover, such that this subset is also a valid cover. We construct a unate function g with each variable representing a cube of F, and whose value is a logic 1 iff the set of cubes represented by g is a cover of (F, D, R). By selecting the largest cube of g, we find the minimal irredundant cover of (F, D, R).
- REDUCE: To avoid the possibility of getting stuck in a locally minimum solution, we reduce the irredundant cover, and iterate on the EXPAND / IRREDUNDANT routines. This allows EXPAND, which is capable of finding good expansions, to operate on a different cover, allowing us to exit from locally minimum solutions.

  Reduction is order dependent, and maximally reduces cubes in some order so as to maintain a cover of (F, D, R). Among the three core routines of ESPRESSO, REDUCE is arguably the weakest. As a result, methods that allow us to obtain very different reductions would be helpful in the overall ESPRESSO algorithm.

The three routines above are run in an iterative manner until there is no further improvement in solution cost. In that case, we run a LASTGASP routine, which performs maximal reduction of each cube followed by an expansion which attempts to add a few more primes to the cover. If this yields an improvement, then we once again iterate through the REDUCE / EXPAND / IRREDUNDANT steps as before.

4 Our Approach

Our approach uses the ESPRESSO algorithm as the backbone. We run multiple iterations on ESPRESSO, with changing D sets, with the intuition that this will allow us to explore more reductions in the REDUCE step of ESPRESSO. Our method, which we call HYPER, is described in Algorithm 2.

Algorithm 2 HYPER(ϵ)

$(F, D, R) \leftarrow DECODE(\epsilon)$
$(F, D) \leftarrow ESPRESSO(F, D)$
$HYPER\_COVER \leftarrow 0$
repeat
  $D^* = EXTRACT\_CUBES(F)$
  $HYPER\_COVER \leftarrow HYPER\_COVER \cup D^*$
  $(F, D) = ESPRESSO(F \setminus D^*, D \cup D^*)$
until $F = 0$
$return(HYPER\_COVER)$

Initially, the $HYPER\_COVER$ is initialized to nil. Next, a set of cubes $D^* = \{c_i\}$ is removed from the cover F and are added to both the don't care set D as well as the $HYPER\_COVER$. We then call ESPRESSO on the cover $F \leftarrow F \setminus D^*$, with the don't care $D \leftarrow D \cup D^*$ iteratively until F becomes empty.

Example	# inputs	# outputs	P	Exact		ESPRESSO	
				Cost (cubes)	Time (sec)	Cost (cubes)	Time (sec)
pdc.pla	16	40	2406	96	330.05	145	1.05
spla.pla	16	6	2296	248	4.59	260	0.65
seq.pla	41	35	1459	334	10.74	336	0.49
cps.pla	24	109	654	157	1.71	163	0.32
misex3.pla	14	14	1848	-	> 2 Hrs	690	0.95
apex2.pla	39	3	1035	1035	50.50	1035	1.3
cordic.pla	23	2	1206	914	1.57	914	3.53
apex3.pla	54	50	280	280	1.34	280	0.24
ex4.pla	128	28	620	-	> 2Hrs	279	0.36
apex5.pla	117	88	1227	-	> 2Hrs	1088	3.07

Table 1: Comparison of the Computation Time and Solution Optimality between ESPRESSO and the exact algorithm.

More cubes in D results in more and new reductions being explored by ESPRESSO. This allows expand to perform different expansions, yielding better results.

Competing approaches maximally reduce each cube in the *gasp* step, to be expanded subsequently. The hope is that this exercise yields new primes for ESPRESSO to utilize. In contrast, our approach removes some cubes D^* and never allows them to be reduced in the future. The advantage of this is that the remaining cubes use the cubes D^* in their don't care, yielding new and more maximal reductions in the *gasp* step.

Note that our method is orthogonal to any other *gasp* approaches, and can therefore be used in conjunction with these approaches.

We can vary the number of iterations p of the $HYPER$ algorithm, and also the method of choosing D^*. The number of iterations is a user-specified parameter. We have implemented two methods to obtain D^* in each iteration of the $HYPER$ algorithm.

- The first method sorts the n cubes of F in decreasing order of size, and chooses the k largest cubes such that $\lceil \frac{n}{k} \rceil = p + 1 - i$, where i is the iteration number. The intuition behind this is that the largest cubes, when inserted in D^*, are likely to maximize the chances of other cubes being reduced in different ways.

  We refer to this as the SIZE heuristic.

- Another heuristic computes the distance of each cube of F from the remaining cubes using the *cdist()* routine in ESPRESSO. The cubes of F are sorted in ascending order of the sum of their distances to the remaining cubes. We then select the first k cubes from this sorted list, just as we did for the SIZE heuristic. The intuition behind this heuristic is that a cube which has the lowest total distance to other cubes in the cover is likely to be one which allows the most other cubes to be reduced in different manners than before.

  We refer to this heuristic as the DISTANCE heuristic.

It is easy to see that for any iteration of our algorithm, $HYPER\_COVER \cup (F \setminus D^*)$ is equivalent to the original function being minimized.

5 Variants of HYPER

We have implemented three variants of the HYPER algorithm, all of which utilize the idea described in Section 4.

Algorithm 3 HYPER-RED(ϵ)

```
(F, D, R) ← DECODE(ϵ)
F ← EXPAND(F, R)
F ← IRREDUNDANT(F, D)
HYPER_COVER = 0
repeat
  repeat
    repeat
      F ← REDUCE(F, D)
      F ← EXPAND(F, R)
      F ← IRREDUNDANT(F, D)
    until not fewer terms in F
    //LASTGASP
    G ← REDUCE_GASP(F, D)
    G ← EXPAND(G, R)
    F ← IRREDUNDANT(F + G, D)
    //LASTGASP
  until G = 0
  D* = EXTRACT_CUBES(F)
  HYPER_COVER ← HYPER_COVER ∪ D*
  F ← F \ D*
  D ← D ∪ D*
until F = 0
F ← HYPER_COVER
LOWER_OUTPUT(F, D)
RAISE_INPUTS(F, R)
error ← (F_old ⊄ F)or(F ⊄ F_old + D)
return(F, error)
```

5.1 HYPER-BLACK

In this variant, our implementation most resembles Algorithm 2. The code to compute D^* and make the various ESPRESSO calls is implemented in the ESPRESSO main.c file, outside the main ESPRESSO routine.

5.2 HYPER-RED

The HYPER-RED algorithm implements the computation of D^* within the ESPRESSO routine, as shown in Algorithm 3. The motivating idea behind this variant of our technique is to reduce the run-time overhead by including the HYPER computations within the main ESPRESSO routine. Also, we remove the ESSENTIAL_PRIMES computation in this case. The algorithm does reduce run-time compared to HYPER-BLACK, but the results are inferior to HYPER-BLACK, as described later.

Algorithm 4 HYPER-BLUE(ϵ)

$(F, D, R) \leftarrow DECODE(\epsilon)$
$F \leftarrow EXPAND(F, R)$
$F \leftarrow IRREDUNDANT(F, D)$
$HYPER\_COVER = 0$
$E \leftarrow 0$
repeat
  $E \leftarrow E \cup ESSENTIAL\_PRIMES(F, D)$
  $F \leftarrow F - E; D \leftarrow D + E$
  repeat
    repeat
      $F \leftarrow REDUCE(F, D)$
      $F \leftarrow EXPAND(F, R)$
      $F \leftarrow IRREDUNDANT(F, D)$
    until not fewer terms in F
    $//LASTGASP$
    $G \leftarrow REDUCE\_GASP(F, D)$
    $G \leftarrow EXPAND(G, R)$
    $F \leftarrow IRREDUNDANT(F + G, D)$
    $//LASTGASP$
  until $G = 0$
  $D^* = EXTRACT\_CUBES(F)$
  $HYPER\_COVER \leftarrow HYPER\_COVER \cup D^*$
  $F \leftarrow F \setminus D^*$
  $D \leftarrow D \cup D^*$
until $F = 0$
$F \leftarrow HYPER\_COVER$
$F \leftarrow F + E; D \leftarrow D - E$
$LOWER\_OUTPUT(F, D)$
$RAISE\_INPUTS(F, R)$
$error \leftarrow (F_{old} \not\subset F) or (F \not\subset F_{old} + D)$
$return(F, error)$

5.3 HYPER-BLUE

The HYPER-BLUE algorithm also implements the computation of D^* within the ESPRESSO routine. It is described in Algorithm 4. The major difference between this algorithm and the HYPER-RED algorithm is that unlike HYPER-RED, this algorithm extracts essential primes from F during each iteration. This improves run-time over HYPER-BLACK (since the HYPER computations are in the main ESPRESSO routine) and HYPER-BLUE as well (since each iteration requires the manipulation of fewer cubes after essential primes are removed). The results are still inferior to those of HYPER-BLACK, as we will observe in Section 6.

6 Results

We performed several experiments to compare our technique with ESPRESSO. All our code was written in C, within the ESPRESSO source tree. The experiments were run on an IBM IntelliStation running Linux with a 1.7 GHz Pentium-4 CPU and 1 GB of RAM. All run-times include the time required to perform book-keeping of the HYPER-COVER and the computation of D^* in each iteration of our algorithms.

We first ran ESPRESSO and its exact version on a set of 154 examples from the benchmark suite distributed with the ESPRESSO package [6]. For these experiments, we used a timeout of 40 minutes for the exact experiments. From the resulting examples, we removed all examples in which the number of cubes obtained by running ESPRESSO and the exact method were identical. Our algorithms were applied to the remaining 58 examples, to see if the ESPRESSO result could be improved upon, with reasonable run-time penalties.

Iterations	SIZE						DISTANCE					
	BLACK		RED		BLUE		BLACK		RED		BLUE	
	Win	Tie	Win	Tie	Win	Tie	Win	Tie	Win	Tie	Win	Tie
2	1	7	1	8	1	8	3	3	1	9	2	6
3	5	8	1	9	0	9	3	7	1	9	3	7
5	6	8	0	14	0	10	3	13	2	9	2	10
10	8	9	1	15	1	14	5	11	2	9	0	14
max	8	9	0	18	0	17	4	13	0	11	0	12

Table 2: Comparison of the 6 variants of our approach

In our experiments, we used the SIZE and DISTANCE heuristic for each of the HYPER-BLACK, HYPER-BLUE and HYPER-RED variants of our approach. Table 2 represents the comparison of the six methods, for varying number of iterations. In the row labeled *max* iterations, we extract a single cube into D^* for each iteration. The first column represents the number of iterations, while the next two columns list the number of examples in which the BLACK method (with the SIZE heuristic) obtained the best results (column 2) or was tied for the best results (column 3) among all six variants of our algorithm. The next four columns list similar information for the RED and BLUE methods. The last six columns list win and tie information for the DISTANCE heuristic.

Note that the BLACK method typically performs the best, with the most number of wins compared to the RED or BLUE. Also, the SIZE heuristic typically outperforms the DISTANCE heuristic for the BLACK method. The BLACK method with the SIZE heuristic was the best choice (or was tied for best place) in 17 cases for 10 iterations. The RED method is faster than the BLACK method since the HYPER computations are in the main ESPRESSO routine. However RED performs worse than BLACK, since there is a possibility of inserting non-essential cubes in the HYPER_COVER in the RED method (for either the SIZE or DISTANCE heuristic), reducing the quality of results. BLUE is typically faster than RED, since it extracts essential primes in every iteration, reducing the size of the problem solved in each iteration. However, its results are comparable with RED, and inferior to BLACK since the essential primes in the k^{th} iteration are not necessarily essential primes of the original problem.

Table 3 describes the detailed results for the BLACK method, with varying number of iterations and cube selection heuristics. The first column is the circuit being minimized, while the next 4 columns report the run-times (T) and final number of cubes (C) for the ESPRESSO-exact and ESPRESSO respectively. A *time* in columns 2 and 3 indicates that the exact method did not complete in 40 minutes. Note that the results in columns 2 through 5 may be slightly different in run-times from the similar results in Table 1, since the results of the two tables were generated on different machines. The remaining columns represent the run-times and final number of cubes with varying number of iterations (for the SIZE and DISTANCE heuristics). *A "−" in columns 6 through 25 indicates that the corresponding algorithm did not improve on*

the results of ESPRESSO, hence results are suppressed for readability. Out of a total of 58 examples, the BLACK algorithm (with either the DISTANCE or SIZE heuristic) showed an improvement over the results of ESPRESSO in 27 cases. 6 of these were instances where ESPRESSO-exact timed out. With 10 iterations, the BLACK method with SIZE heuristic was the best of all 6 methods in 8 instances, with a run-time increase of less than 2× (for examples with reasonable run-times). With the DISTANCE heuristic and 10 iterations, the BLACK method was the leader in 5 instances, with a run-time increase of up to 8×. This is attributed to the computation of distance in each iteration, which is quadratic in the size of the cover. Note that the run-times for examples with a "–" in columns 6 through 25 followed followed a similar trend.

Tables 5 and 4 are similarly organized tables for the RED and BLUE methods respectively. We note that the results reported in these tables are comparable in quality, but both typically inferior to the results of the BLACK method. Run-times of the BLUE method are the least, followed by the RED method. The BLACK method has the largest run-times (and best quality of results).

The BLACK method with the SIZE heuristic and 10 iterations demonstrated a improvement in cover size of between 1 and 26 cubes over ESPRESSO. Significant improvement was demonstrated for *ex1010* (8 cubes), *misex3* (16 cubes), *pdc* (26 cubes) and *test3* (18 cubes). With *max* iterations, these numbers increase to 14, 24, 36 and 40 cubes respectively, but the run-times are quite high in this case.

Figure 1 describes the run-time of the BLACK method (with the SIZE heuristic and 10 iterations) as a function of the iteration number. Results are shown for 5 examples. Note that the run-times drop off sharply with increasing iteration number. This is because the don't cares increase with iteration, allowing the iterations to complete much faster. Additionally, the number of REDUCE / EXPAND / IRREDUNDANT calls decreases with the iteration number as well. The total run-time for 10 iterations is often about 2× the regular ESPRESSO run-time, and occasionally as high as 3× (for medium and larger sized examples).

Given the reasonable run-time increase of the BLACK method with 10 iterations and the SIZE heuristic, we suggest these values for practical use. The application of our technique is in a situation where there is a limit on the number of cubes that we can implement. ESPRESSO minimizes a design, resulting in a higher cube count. ESPRESSO-exact times out on the design. In this case, we invoke our algorithm, with increasing p, with the hope that the minimized design results in fewer cubes than the limit.

7 Conclusion

In this paper, we have demonstrated an algorithm which is designed to bridge the gap in the quality of results obtained by ESPRESSO versus the exact technique. Although ESPRESSO obtains good quality results with very reasonable run-times, it may often be the case that a designer would like to improve the cover size further, with a reasonable increase in run-time.

Our algorithm is designed to iterate on ESPRESSO, with each iteration consisting of the removal of a set of cubes from the on-set. These cubes are appended to a result cover (HYPER_COVER) as well as the don't care set for subsequent iterations. This allows the REDUCE routine of ESPRESSO to explore more and different cube reductions, resulting in new expansions which can yield improved results.

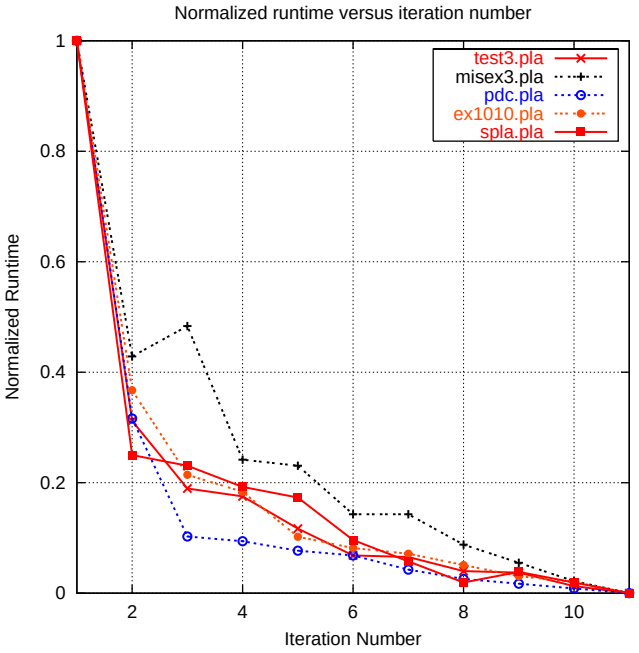

Figure 1: Normalized runtime of BLACK versus iteration number

We have implemented 3 flavors of our algorithm (BLACK, BLUE and RED), each with two different heuristics (SIZE and DISTANCE) for cube extraction. We showed that the BLACK algorithm, with the SIZE heuristic and 10 iterations performed best, obtaining the best (or joint best) solution among all 6 variants in 17 out of 27 examples. The cover size reduction obtained by our technique is as high as 18%, with a typical run-time penalty of about 2× compared to ESPRESSO. At least one of our examples yielded better results than ESPRESSO in 27 out of 58 examples.

References

[1] W. Quine, "The problem of simplifying truth functions," *American Mathematical Monthly*, vol. 59, pp. 521–531, 1952.

[2] E. McCluskey, "Algebraic minimization and the design of two-terminal contact networks," *Bell System Technical Journal*, vol. 35, pp. 1417–1444, 1956.

[3] P. C. McGeer, J. V. Sanghavi, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Espresso-signature: A new exact minimizer for logic functions," in *Design Automation Conference*, pp. 618–624, 1993.

[4] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[5] R. Rudell and A. Sangiovanni-Vincentelli, "Espresso-mv: Algorithms for multiple-valued logic minimization," in *Proceedings of the IEEE 1985 Custom Integrated Circuits Conference*, pp. 230–4, May 1985.

[6] "Espresso software distribution site, ftp://ic.eecs.berkeley.edu/pub/espresso/espresso-book-examples.tar.gz."

Ckt	EXACT		ESPRESSO		SIZE										DISTANCE									
					n=2		n=3		n=5		n=10		n=max		n=2		n=3		n=5		n=10		n=max	
	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C
9sym	0.02	63	0.01	65	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Zal2	0.04	63	0.03	65	–	–	0.02	64	0.03	64	0.04	63	0.21	63	–	–	–	–	–	–	–	–	0.20	64
alu3	0.06	64	0.01	66	–	–	–	–	0.02	65	0.05	65	–	–	–	–	–	–	–	–	–	–	–	–
apex4	0.51	427	0.53	436	–	–	–	–	–	–	1.18	432	24.10	430	–	–	–	–	1.94	435	2.23	435	28.75	433
b12	0.62	41	0.01	43	–	–	–	–	–	–	–	–	0.17	42	–	–	0.03	42	0.04	42	0.06	42	0.20	42
b2	0.26	104	0.04	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
b3	2.65	210	0.11	211	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bc0	9.18	177	0.18	179	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
clip	0.08	117	0.04	120	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
cps	1.68	157	0.31	163	–	–	0.48	162	–	–	0.90	162	–	–	–	–	–	–	0.68	162	0.98	162	10.84	162
dist	0.03	120	0.03	123	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
dk48	0.10	21	0.02	22	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex1010	*time*	*time*	1.03	284	1.17	282	1.28	281	1.52	280	2.16	276	25.69	270	1.52	282	1.72	282	1.93	280	2.44	279	27.69	273
ex4	*time*	*time*	0.42	279	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex5	*time*	*time*	0.07	74	–	–	–	–	0.29	71	0.53	73	2.51	71	–	–	–	–	0.57	73	0.77	70	3.92	69
exep	0.16	108	0.05	110	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
exp	0.03	56	0.02	59	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
exps	0.21	132	0.11	136	0.13	134	0.16	132	0.21	132	0.33	134	2.66	132	0.19	135	0.22	135	0.28	134	0.39	135	3.03	133
f51m	0.04	76	0.03	77	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ibm	*time*	*time*	0.03	173	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in1	0.26	104	0.05	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in2	0.08	134	0.02	136	–	–	0.04	135	0.07	135	0.10	135	1.13	135	0.09	135	0.10	135	0.12	135	0.18	135	1.54	135
in4	2.16	211	0.09	212	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
inc	0.00	29	0.01	30	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
intb	5.94	629	0.50	631	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
jbp	*time*	*time*	0.11	122	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
luc	0.03	26	0.00	27	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.08	26
m3	0.03	62	0.04	66	0.04	65	0.05	65	0.07	64	0.10	64	0.50	65	0.05	65	0.05	65	0.07	65	0.11	65	0.58	65
m4	0.15	101	0.13	105	–	–	–	–	–	–	0.29	103	1.97	102	–	–	–	–	–	–	–	–	–	–
mainpla	*time*	*time*	0.30	172	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
max1024	*time*	*time*	0.47	274	0.54	273	0.57	273	0.63	273	0.79	272	8.63	270	–	–	0.85	273	0.92	273	1.11	271	9.60	272
max128	0.11	78	0.07	83	–	–	0.11	80	0.13	81	0.19	81	0.81	79	–	–	0.11	82	0.14	82	0.24	81	0.79	79
max512	0.13	133	0.11	145	0.12	144	0.15	143	0.14	143	0.22	143	1.68	142	–	–	–	–	0.21	143	0.26	143	1.97	143
misex3	*time*	*time*	0.97	690	–	–	1.44	685	1.83	680	2.72	674	91.27	666	6.85	686	7.04	687	7.46	682	8.36	675	100.80	663
misex3c	*time*	*time*	0.33	197	–	–	–	–	–	–	–	–	21.90	196	–	–	–	–	1.13	196	1.93	196	30.19	196
misg	*time*	*time*	0.01	69	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mish	*time*	*time*	0.02	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
misj	*time*	*time*	0.00	35	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mlp4	0.42	121	0.05	128	–	–	–	–	–	–	–	–	1.03	127	–	–	0.10	127	0.12	127	0.16	127	1.12	127
mp2d	0.37	30	0.02	31	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
opa	0.13	77	0.08	79	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0.16	78	0.23	78	1.19	78
pdc	305.50	96	1.39	145	1.50	133	1.65	127	1.79	124	2.27	119	7.69	109	1.62	127	1.69	123	1.95	124	2.29	117	10.04	122
pope	0.31	59	0.08	62	0.10	61	0.10	61	0.16	61	0.22	61	0.85	61	–	–	–	–	–	–	–	–	0.92	61
risc	0.00	28	0.00	29	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
seq	11.26	334	0.50	336	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
soar	*time*	*time*	0.88	353	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
spla	3.84	248	0.66	260	–	–	–	–	0.93	257	1.24	257	15.98	250	1.03	256	1.10	253	1.21	248	1.53	248	17.42	249
sqr6	0.05	47	0.01	49	–	–	–	–	–	–	–	–	0.11	48	–	–	–	–	–	–	–	–	–	–
t1	32.88	100	0.06	102	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
t2	0.03	52	0.01	53	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
test3	*time*	*time*	7.15	541	8.14	539	9.14	532	11.01	528	14.36	523	233.20	501	10.97	540	11.44	540	12.73	537	16.67	531	287.00	499
ti	*time*	*time*	0.35	213	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x2dn	*time*	*time*	0.05	104	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x6dn	0.17	81	0.01	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x7dn	*time*	*time*	0.49	538	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 3: Detailed results for BLACK algorithm ("–" indicates results not an improvement over ESPRESSO)

Ckt	EXACT		ESPRESSO		SIZE										DISTANCE									
					n=2		n=3		n=5		n=10		n=max		n=2		n=3		n=5		n=10		n=max	
	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C
9sym	0.02	63	0.01	65	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Zal2	0.04	63	0.03	65	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
alu3	0.06	64	0.01	66	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
apex4	0.51	427	0.53	436	–	–	–	–	–	–	0.84	434	10.74	434	–	–	–	–	–	–	–	–	–	–
b12	0.62	41	0.01	43	–	–	–	–	0.03	42	0.05	42	0.12	42	–	–	–	–	0.04	42	0.06	42	0.16	42
b2	0.26	104	0.04	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
b3	2.65	210	0.11	211	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bc0	9.18	177	0.18	179	0.20	178	–	–	–	–	0.30	178	1.92	178	–	–	0.28	178	0.31	178	0.38	178	2.47	178
clip	0.08	117	0.04	120	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
cps	1.68	157	0.31	163	0.33	162	0.35	162	0.40	162	0.50	162	2.42	162	–	–	–	–	0.48	162	0.61	162	3.11	162
dist	0.03	120	0.03	123	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
dk48	0.10	21	0.02	22	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex1010	*time*	*time*	1.03	284	–	–	1.18	282	1.35	282	1.59	282	14.24	276	–	–	–	–	–	–	1.82	282	16.37	279
ex4	*time*	*time*	0.42	279	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex5	*time*	*time*	0.07	74	–	–	–	–	–	–	–	–	–	–	0.09	73	0.10	73	0.11	73	0.17	73	0.46	73
exep	0.16	108	0.05	110	–	–	–	–	–	–	–	–	0.11	109	–	–	–	–	–	–	–	–	–	–
exp	0.03	56	0.02	59	0.03	58	0.02	58	0.03	58	0.03	57	0.05	57	0.02	57	0.03	57	0.03	57	0.04	57	0.07	57
exps	0.21	132	0.11	136	0.14	135	0.15	134	0.15	132	0.19	133	0.55	132	0.14	134	0.15	134	0.17	133	0.20	134	0.60	133
f51m	0.04	76	0.03	77	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ibm	*time*	*time*	0.03	173	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in1	0.26	104	0.05	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in2	0.08	134	0.02	136	0.03	135	0.04	135	0.04	135	0.07	135	0.20	135	–	–	0.04	135	0.05	135	0.08	135	0.25	135
in4	2.16	211	0.09	212	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
inc	0.00	29	0.01	30	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
intb	5.94	629	0.50	631	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
jbp	*time*	*time*	0.11	122	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
luc	0.03	26	0.00	27	–	–	0.01	26	0.01	26	0.01	26	0.01	26	–	–	0.01	26	0.01	26	0.02	26	0.02	26
m3	0.03	62	0.04	66	–	–	–	–	–	–	0.06	65	0.14	65	–	–	–	–	–	–	–	–	–	–
m4	0.15	101	0.13	105	–	–	–	–	–	–	0.17	104	0.52	103	0.14	104	0.15	104	0.15	104	0.18	104	0.58	104
mainpla	*time*	*time*	0.30	172	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
max1024	*time*	*time*	0.47	274	0.50	273	–	–	–	–	0.63	273	4.45	272	–	–	–	–	–	–	–	–	–	–
max128	0.11	78	0.07	83	0.07	82	0.08	82	0.08	82	0.10	82	0.29	82	0.08	82	0.09	82	0.11	82	0.12	82	0.37	82
max512	0.13	133	0.11	145	0.10	143	0.13	143	0.12	141	0.16	144	0.76	143	0.14	144	–	–	0.16	144	0.20	144	0.99	144
misex3	*time*	*time*	0.97	690	–	–	1.17	688	1.39	688	2.02	683	37.78	685	4.16	688	4.32	688	4.53	688	4.91	688	45.64	687
misex3c	*time*	*time*	0.33	197	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	1.24	196	12.28	196
misg	*time*	*time*	0.01	69	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mish	*time*	*time*	0.02	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
misj	*time*	*time*	0.00	35	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mlp4	0.42	121	0.05	128	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mp2d	0.37	30	0.02	31	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
opa	0.13	77	0.08	79	–	–	–	–	–	–	–	–	0.30	78	–	–	–	–	0.11	78	0.13	78	0.35	78
pdc	305.50	96	1.39	145	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
pope	0.31	59	0.08	62	0.07	60	–	–	0.08	60	0.09	59	0.20	59	–	–	0.08	59	0.11	59	0.11	59	0.29	59
risc	0.00	28	0.00	29	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
seq	11.26	334	0.50	336	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
soar	*time*	*time*	0.88	353	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
spla	3.84	248	0.66	260	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
sqr6	0.05	47	0.01	49	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
t1	32.88	100	0.06	102	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
t2	0.03	52	0.01	53	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
test3	*time*	*time*	7.15	541	7.83	536	8.65	534	9.35	532	11.17	530	128.00	522	10.02	538	10.46	536	11.34	532	12.30	534	138.90	525
ti	*time*	*time*	0.35	213	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x2dn	*time*	*time*	0.05	104	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x6dn	0.17	81	0.01	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x7dn	*time*	*time*	0.49	538	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 4: Detailed results for BLUE algorithm ("−" indicates results not an improvement over ESPRESSO)

Ckt	EXACT		ESPRESSO		SIZE										DISTANCE									
					n=2		n=3		n=5		n=10		n=max		n=2		n=3		n=5		n=10		n=max	
	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C
9sym	0.02	63	0.01	65	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Zal2	0.04	63	0.03	65	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
alu3	0.06	64	0.01	66	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
apex4	0.51	427	0.53	436	–	–	–	–	0.67	435	0.89	434	15.90	433	–	–	–	–	–	–	–	–	–	–
b12	0.62	41	0.01	43	0.03	42	0.03	42	0.04	42	0.05	42	0.16	42	0.03	42	0.04	42	0.04	42	0.07	42	0.22	42
b2	0.26	104	0.04	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
b3	2.65	210	0.11	211	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
bc0	9.18	177	0.18	179	0.18	178	–	–	–	–	0.31	178	3.29	178	0.26	178	0.29	178	0.34	178	0.45	178	3.97	178
clip	0.08	117	0.04	120	0.05	119	0.05	119	0.06	119	0.09	119	0.72	119	0.07	119	0.08	119	0.09	119	0.12	119	0.85	119
cps	1.68	157	0.31	163	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
dist	0.03	120	0.03	123	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
dk48	0.10	21	0.02	22	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex1010	*time*	*time*	1.03	284	–	–	1.11	283	1.25	283	1.65	281	15.32	277	1.37	283	–	–	–	–	1.83	283	17.20	281
ex4	*time*	*time*	0.42	279	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ex5	*time*	*time*	0.07	74	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
exep	0.16	108	0.05	110	–	–	–	–	–	–	–	–	1.07	109	–	–	–	–	0.14	109	0.20	109	1.37	109
exp	0.03	56	0.02	59	–	–	–	–	0.04	58	0.05	58	0.16	56	0.03	57	0.04	57	0.05	56	0.06	56	0.21	56
exps	0.21	132	0.11	136	0.15	135	0.17	135	0.21	132	0.28	134	1.60	132	0.19	135	0.21	134	0.25	135	0.32	134	1.96	133
f51m	0.04	76	0.03	77	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ibm	*time*	*time*	0.03	173	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in1	0.26	104	0.05	106	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
in2	0.08	134	0.02	136	–	–	–	–	0.06	134	0.10	134	0.97	134	–	–	–	–	0.13	134	0.18	134	1.37	134
in4	2.16	211	0.09	212	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
inc	0.00	29	0.01	30	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
intb	5.94	629	0.50	631	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
jbp	*time*	*time*	0.11	122	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
luc	0.03	26	0.00	27	–	–	–	–	–	–	–	–	0.05	26	–	–	–	–	–	–	–	–	–	–
m3	0.03	62	0.04	66	–	–	0.04	65	0.06	65	0.07	65	0.22	65	–	–	0.05	65	0.04	65	0.06	65	0.21	65
m4	0.15	101	0.13	105	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mainpla	*time*	*time*	0.30	172	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
max1024	*time*	*time*	0.47	274	–	–	–	–	–	–	0.63	273	6.69	272	–	–	–	–	–	–	–	–	7.59	272
max128	0.11	78	0.07	83	0.09	82	0.09	81	0.08	81	0.11	81	0.35	81	–	–	–	–	–	–	–	–	–	–
max512	0.13	133	0.11	145	0.10	144	0.13	143	0.13	141	0.16	141	1.25	143	0.15	143	0.16	144	0.18	144	0.25	144	1.71	143
misex3	*time*	*time*	0.97	690	1.20	682	1.32	682	1.49	681	2.11	677	54.70	672	6.55	683	6.74	683	6.84	683	7.43	682	73.08	680
misex3c	*time*	*time*	0.33	197	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
misg	*time*	*time*	0.01	69	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mish	*time*	*time*	0.02	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
misj	*time*	*time*	0.00	35	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mlp4	0.42	121	0.05	128	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
mp2d	0.37	30	0.02	31	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
opa	0.13	77	0.08	79	–	–	–	–	–	–	–	–	0.48	78	0.10	78	0.11	78	0.12	78	0.16	78	0.67	78
pdc	305.50	96	1.39	145	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
pope	0.31	59	0.08	62	0.08	60	0.09	61	0.09	60	0.10	60	0.23	59	0.11	60	–	–	0.13	60	0.17	59	0.46	59
risc	0.00	28	0.00	29	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
seq	11.26	334	0.50	336	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
soar	*time*	*time*	0.88	353	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
spla	3.84	248	0.66	260	0.77	257	0.81	257	0.86	257	1.02	257	9.29	257	–	–	–	–	–	–	–	–	–	–
sqr6	0.05	47	0.01	49	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
t1	32.88	100	0.06	102	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
t2	0.03	52	0.01	53	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
test3	*time*	*time*	7.15	541	7.67	536	8.10	536	8.98	535	11.04	528	137.10	521	9.85	538	10.29	536	10.97	531	12.50	535	146.30	529
ti	*time*	*time*	0.35	213	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x2dn	*time*	*time*	0.05	104	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x6dn	0.17	81	0.01	82	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
x7dn	*time*	*time*	0.49	538	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 5: Detailed results for RED algorithm ("–" indicates results not an improvement over ESPRESSO)