New insights into the assessment of k-out-of-n and related systems

Y. Dutuit
LAP,
Université Bordeaux I,
33405 Talence Cedex, FRANCE
dutuit@hse.iuta.u-bordeaux.fr

A. Rauzy
LaBRI, CNRS,
351, cours de la Libération,
33405 Talence Cedex, FRANCE
rauzy@labri.u-bordeaux.fr

Abstract

k-out-of-n and related systems have received much attention in the recent past years. Hundreds of articles were devoted to various methods to assess them. In this article, we show that there exist very efficient algorithms to compute the reliability of k-out-of-n, l-to-h-out-of-n and consecutive k-out-of-n systems. k-within-r-out-of-n systems are intrinsically much harder. We study the performance of Binary Decision Diagrams on these systems. Then, we propose a new approximation scheme. This algorithm is much more efficient in practice than already proposed methods.

keywords: k-out-of-n and related systems, Binary Decision Diagrams

1 Introduction

A k-within-r-out-of-n system consists of n components linearly arranged. The system is failed if there is at least a window of r consecutive components with among them at least k failed components. k-within-r-out-of-n systems were introduced by Tong [Ton85] and Griffith [Gri86]. They generalize both k-out-of-n systems ($r = n$) and consecutive k-out-of-n systems ($r = k$). years. Hundreds of articles were devoted to various methods to assess them. The survey by Chao et al. [CFK95] and the book by Misra [Mis93] provide comprehensive surveys on this topic. The recent article by Habib and Szántai [HS00] deserves also a special mention for it compares experimentally several methods.

In this article, we show that there exist $\mathcal{O}(k.n)$ algorithms to compute the exact value of k-out-of-n and consecutive k-out-of-n system reliabilities (as a by-product we study also l-to-h-out-of-n-systems). The algorithm we propose for k-out-of-n systems looks like the method proposed by Rushdi in [Rus86, Rus91]. Those dealing with l-to-h-out-of-n and consecutive k-out-of-n systems are new, at least to a certain extent (since they are just variations on the former). Their conciseness is noticeable: they consist of about 10 lines of ANSI C code.

Basically, these algorithms consist in sliding a window of size k through the components and to consider the number of failed components inside the window. k-within-r-out-of-n systems are intrinsically much harder. In this general case, counting arguments do no suffice. One has to consider individually each component of the window, which leads to a combinatorial explosion. The various methods proposed in the literature to handle these systems are therefore designed to compute lower and upper bounds on reliability rather than its exact value (see [CFK95, HS00] for a review of recent developments). Moreover, most of them assume that all components have the same reliability.

In this article, we propose two approach to tackle k-within-r-out-of-n systems.

First, we study the performance of Bryant's Binary Decision Diagrams (BDDs) [Bry86, BRB90]. BDDs are the state-of-the-art data structure to encode and to manipulate Boolean functions. Since their introduction in the reliability analysis framework [Rau93, CM94], BDDs have proved to be the most efficient technique to assess fault trees. We report twofold experimental results. On the one hand, we show that BDDs are able to deal with rather large systems (with several dozens of components). By the way, we correct a claim by Habib and Szántai who wrote in [HS00] that only simulation is able to assess (almost) exact results for such systems. On the other hand, we exhibit a formula that gives the size of the BDD encoding a k-within-r-out-n system, for $n \geq 2r$. This formula makes clear that BDDs are subject to combinatorial explosion.

Second, we propose a new approximation scheme. Conversely to BDDs but according to most of proposed methods, this algorithm assumes that all components have the same reliability. Our algorithm computes a lower bound on system unreliability in $\mathcal{O}(2^h.k.n)$, where $0 \leq h \leq r$ is a parameter that fixes the accuracy of the lower bound. By comparing values obtained for h and $h-1$, it is possible to deduce an upper bound. We report experimental results that show that our algorithm outperforms the best methods already proposed. It gives much more accurate results, even for small h's. Moreover, it is much more efficient in terms of computational cost. It is actually able to give accurate results for systems far beyond the limits of any other known method.

The remaining of this article is organized as follows. k-out-of-n and consecutive k-out-of-n systems are studied section 3. Results with Binary Decision Diagrams are presented section 4. The new approximation scheme is described section 5. Finally, experimental results of this algorithm are presented section 6.

2 Nomenclature

In what follows, we consider n components c_1, c_2, ..., c_n linearly arranged and that can be in one of two states, either good or failed (not good). We assume that component reliabilities may vary and that component failures are s-independent. Moreover, we assume that component reliabilities P_i (probability of survival) and unreliabilities Q_i (probability of failure) are available at no cost ($P_i + Q_i = 1$).

A k-within-r-out-of-n system over components c_1 up to c_n ($k \leq r \leq n$) is failed iff there are r consecutive components which include among them, at least k failed components.

k-out-of-n and consecutive k-out-of-n systems correspond respectively to the cases $r = n$ and $r = k$. A l-to-h-out-of-n system over components c_1 up to c_n is failed iff there are at least l and at most h failed components among the n.

Throughout the article, we keep the same meaning for k, r and n. Events are denoted by caligraphic capital letters, e.g. $\mathcal{E}$. Since all algorithms presented here iterate over components, events are indiced with positions e.g. $\mathcal{E}_i$ $(1 \leq i \leq n)$. $\overline{\mathcal{E}}$ denotes the complementary of the event $\mathcal{E}$ in the considered set of outcomes. The probability $p(\mathcal{E})$ of an event $\mathcal{E}$ is denoted by the same capital letter as the event, here E.

We use the following notations.

- $\mathcal{Q}_i$ $(1 \leq i \leq n)$ denotes the event "the component c_i is failed".
- $\mathcal{L}_i^{l/r}$ $(0 \leq l \leq k,\ r \leq i \leq n)$ denotes the event "there are at least l failed components among c_{i-r+1}, c_{i-r+2}, ... , c_i". r is omitted when it is clear from the context, e.g. $\mathcal{L}_i^l$.
- $\mathcal{H}_i^{h/r}$ $(0 \leq h \leq k, r \leq i \leq n)$ denotes the event "at most h among components c_{i-r+1}, ... , c_i are failed". r is omitted when it is clear from the context, e.g. $\mathcal{H}_i^h$.
- $\mathcal{F}_i^{k/r}$ $(r \leq i \leq n)$ denotes the event "there is at least a window of r consecutive components in c_1, ... , c_i among which at least k are failed". In other words, $\mathcal{F}_i^{k/r} = \bigcup_{r \leq l \leq i} \mathcal{L}_l^{k/r}$. k and r are omitted when they are clear from the context, e.g. $\mathcal{F}_i$.
- $\phi[c/v]$ denotes the Boolean formula ϕ in which the constant $c \in \{0, 1\}$ has been substituted for the variable v.

Moreover, we use the following derived notations.

- $\mathcal{K}_i^l \stackrel{\text{def}}{=} \mathcal{L}_i^{l/i}$, i.e. at least l among components c_1, ... ,c_i are failed.
- $\mathcal{T}_i^{l,h} \stackrel{\text{def}}{=} \mathcal{L}_i^{l/i} \cap \mathcal{H}_i^{h/i}$ i.e. at least l and at most h among components c_1, ... , c_i are failed.
- $\mathcal{C}_i^l \stackrel{\text{def}}{=} \mathcal{L}_i^{l/l} \cup \mathcal{F}_{i-1}^{k/k}$, i.e. either components c_{i-l+1}, ... , c_i are failed or there is at least a window of r consecutive components in c_1, ... , c_{i-1} among which at least k are failed.
- $\mathcal{S}_i^l \stackrel{\text{def}}{=} \mathcal{L}_i^{l/r} \cap \mathcal{H}_i^{l/r} \cap \bigcap_{j=r}^{i-1} \mathcal{H}_j^{k-1/r}$ i.e. there are exactly l failed components among c_{i-r+1}, ... , c_i and no window of r consecutive components c_{j-r+1}, ... , c_j, $j < i$, contains more than $k-1$ failed components.

3 Easy k-out-of-n and related systems

3.1 Algorithms

K_n^k, $T_n^{l,h}$ and C_n^k denote respectively unreliabilities of k-out-of-n, l-to-h-out-of-n and consecutive k-out-of-n systems. The following equalities hold.

$$K_i^j = \begin{cases} 0 & \text{if } \ j > i \\ 1 & \text{if } \ j \leq 0 \\ Q_i.K_{i-1}^{j-1} + P_i.K_{i-1}^j & \text{otherwise} \end{cases} \quad (1)$$

$$T_i^{l,h} = \begin{cases} 0 & \text{if } \ (l > i) \ \vee \ (h < 0) \\ 1 & \text{if } \ (l \leq 0) \ \wedge \ (h \geq i) \\ Q_i.T_{i-1}^{l-1,h-1} + P_i.T_{i-1}^{l,h} & \text{otherwise} \end{cases} \quad (2)$$

$$C_i^j = \begin{cases} 0 & \text{if } \ j > i \\ 1 & \text{if } \ j \leq 0 \\ Q_i.C_{i-1}^{j-1} + P_i.C_{i-1}^k & \text{otherwise} \end{cases} \quad (3)$$

Equations 1, 2 and 3 can be seen as recursive definitions of k-out-of-n, l-to-h-out-of-n and consecutive k-out-of-n systems. They give also a mean to compute K_i^j, $T_i^{l-(h-j),h-(h-j)}$ and C_i^j ($0 \leq j \leq k$ or h) by strates: first for $i = 1$, then for $i = 2$, and so on up to $i = n$.

The corresponding ANSI C functions are given figures 1, 2 and 3. Since they consists of two nested loops over n and k (or h) they are respectively in $\mathcal{O}(k.n)$, $\mathcal{O}(h.n)$ and $\mathcal{O}(k.n)$. Arrays `K`, `T` and `C` are passed as parameters of these functions. They could be allocated and freed inside the functions as well.

The algorithm to assess k-out-of-n systems has strong similarities with the one proposed by Rushdi in [Rus86, Rus91]. The two other ones are new, even if the underlying recursive equations appeared under other forms in some articles (see for instance [CFK95, Mis93] for reviews).

It is possible to improve a bit the algorithms by restricting the inner loop in following way: one set the starting value for j at i (until $i < k$) and its final value at $k - i$ (when $i \geq n - k$).

3.2 Experimental Results

For verification purposes, table 1 presents reliabilities of some l-to-h-out-of-n systems with identical components. These results correct slightly those reported in [JG85]. They were obtained instantaneously.

Tables 2 and 3 give some running times obtained on a laptop computer with a pentium I processor cadenced at 166 MHz and running Linux. The computation of very large system reliabilities (with several thousands components) takes only few seconds. In order to appreciate these running times, one could compare them with those of the RAFT-GFP algorithm presented Fig. 2. of reference [II95], that are about 20 times greater. Indeed,

```
double k_out_of_n(int k, int n, double* Q, double* K)
{
  double Pi, Qi;
  int    i, j;
  for (j=k;j>=0;j--) K[j]=(j==0 ? 1.0 : 0.0);
  for (i=1;i<=n;i++) {
    Qi=Q[i]; Pi=1-Qi;
    for (j=k;j>0;j--) K[j]=Qi*K[j-1]+Pi*K[j];
    }
  return(K[k]);
}
```

Figure 1: The ANSI C function to assess k-out-of-n systems

```
double l_h_out_of_n(int l, int h, int n, double* Q, double* T)
{
  double Pi, Qi;
  int    i, j;
  for (j=h;j>=0;j++) T[j]=(j<=h-l ? 1.0 : 0.0);
  for (i=1;i<=n;i++) {
    Qi=Q[i]; Pi=1-Qi;
    for (j=h;j>0;j++) T[j]=Qi*T[j-1]+Pi*T[j];
    T[0]=Pi*T[0];
    }
  return(T[h]);
}
```

Figure 2: The ANSI C function to assess l-to-h-out-of-n systems

```
double consecutive_k_out_of_n(int k, int n, double* Q, double* C)
{
  double Pi, Qi, Ck;
  int    i, j;
  for (j=k;j>=0;j--) C[j]=(j==0 ? 1.0 : 0.0);
  for (i=1;i<=n;i++) {
    Qi=Q[i]; Pi=1-Qi; Ck=C[k];
    for (j=k;j>0;j--) C[j]=Qi*C[j-1]+Pi*Ck;
    }
  return(C[k]);
}
```

Figure 3: The ANSI C function to assess consecutive k-out-of-n systems

Table 1: Reliability of some l-to-h-out-of-n systems

System	p=0.5	p=0.6	p=0.7	p=0.8	p=0. 9
$l = 5, h = 8, n = 10$	0.612305	0.787404	0.803343	0.617821	0.263754
$l = 5, h = 9, n = 12$	0.786865	0.859247	0.737695	0.441073	0.110867
$l = 10, h = 12, n = 15$	0.147186	0.376102	0.594794	0.540925	0.181811

Table 2: Running times in seconds for (consecutive) k-out-of-10000 systems

k	2	10	100	1000	2500	5000	7500	9998
k-out-of-n	0.00	0.01	0.04	0.54	1.17	2.20	3.23	4.26
consecutive k-out-of-n	0.00	0.00	0.04	0.47	1.10	2.20	3.25	4.33

Table 3: Running times in seconds for l-to-h-out-of-10000 systems

$l \downarrow \ h \rightarrow$	2	10	100	1000	2500	5000	7500	9998
2	0.00	0.00	0.05	0.47	1.22	2.31	3.32	4.35
10	-	0.03	0.18	0.56	1.17	2.18	3.19	4.21
100	-	-	0.04	0.41	1.01	2.03	3.05	4.91
1000	-	-	-	0.54	1.17	2.16	3.18	4.20
2500	-	-	-	-	1.16	2.15	3.18	4.22
5000	-	-	-	-	-	2.17	3.20	4.20
7500	-	-	-	-	-	-	3.18	4.05
9998	-	-	-	-	-	-	-	4.20

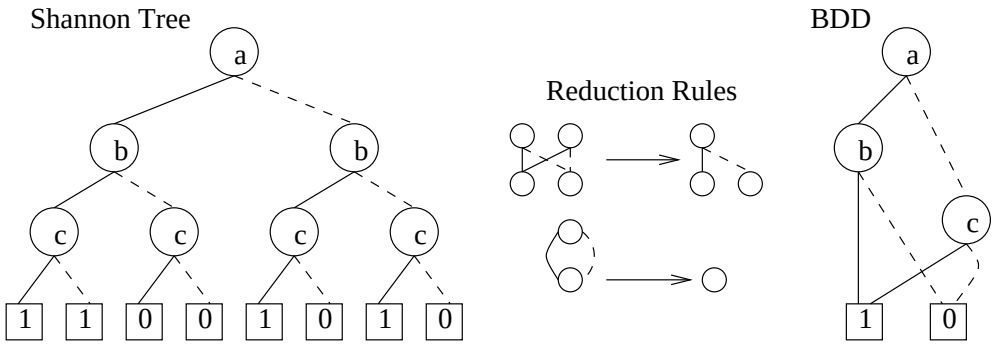

Figure 4: From the Shannon tree to the BDD encoding $ab + \overline{a}c$.

this comparison is not absolutely fair because processors were not the same. Nevertheless, this shows that our very simple procedure competes with the fastest known algorithms.

It is worth noticing that values computed for so large systems are subject to strong deviations, due to rounding errors.

4 Performance of Binary Decision Diagrams

We turn now to k-within-r-out-of-n systems and their assessment by means of Binary Decision Diagrams (BDDs).

4.1 Binary Decision Diagrams

The BDD associated with a formulae is a compact encoding of the truth table of this formula. This representation is based on the Shannon decomposition: let ϕ be a Boolean formula that depends on the variable v, then $\phi = v.\phi[1/v] + \overline{v}.\phi[0/v]$. By choosing a total order over the variables and applying recursively the Shannon decomposition, the truth table of any formula can be graphically represented as a binary tree. Each internal node encodes a formula $\psi = v.\psi[1/v] + \overline{v}.\psi[0/v]$. It is labeled with the variable v and it has two outedges (a *then*-outedge, pointing to the node that encodes $\psi[1/v]$, and a *else*-outedge, pointing to the node that encodes $\psi[0/v]$). Leaves are labeled with either 0 or 1. The value of the formula under a given variable assignment is obtained by descending along the corresponding branch. The Shannon tree for the formula $ab + \overline{a}c$ and the lexicographic order is pictured Fig. 4 (dashed lines represent *else*-outedges).

Indeed such a representation is very space consuming. It is however possible to shrink it by means of the two following reduction rules.

- Isomorphic subtrees merging. Since two isomorphic subtrees encode the same formula, at least one is useless.

- Useless nodes deletion. A node with two equal sons is useless since it is equivalent to its son ($v.\psi + \overline{v}.\psi = \psi$).

By applying these two rules as far as possible, one gets the BDD that encodes the formula. A BDD is therefore a directed acyclic graph. It is unique, up to an isomorphism [Bry86]. This process is illustrated Fig. 4.

Logical operations (and, or, xor, ...) can be directly performed on BDDs. Among other consequences, this means that the complete binary tree is never built, then shrunk: the BDD encoding a formula is obtained by composing the BDDs encoding its subformulae. A caching principle is used to store intermediate results of computations. This makes usual logical operations (conjunction, disjunction) polynomial in the size of their operands. A complete implementation of a BDD package is described in [BRB90]. The reader interested in more details should refer to this article.

By applying the Shannon decomposition to probabilities:

$$p(v.\psi_1 + \overline{v}.\psi_0) \quad = \quad p(v).p(\psi_1) + (1 - p(v)).p(\psi_0)$$

one gets an linear time algorithm to assess the probability of a formula from the BDD that encodes this formula [Rau93].

4.2 Results on Habib and Szántai examples

In order to assess the performance of BDDs on k-within-m-out-of-n systems, we described them by means of the following parametric formula.

$$\Phi_n^{k/r} \quad \stackrel{\text{def}}{=} \quad \sum_{i=r}^{n} k/r(c_{i-r+1}, \ldots, c_i)$$

The connective k/r (k-out-of-r) can be efficiently implemented on BDDs. The algorithm to do so is given appendix A.

Table 4 gives results we obtained on a desktop computer with a pentium III microprocessor cadenced at 733MHz and running Linux. These examples are those studied by Habib and Szántai in [HS00]. They were, up to this article, the biggest k-within-r-out-of-n (with $k < r < n$) considered in the literature. Their components have the same probability Q of failure.

In their article, Habib and Szántai report actually two kinds of results: some obtained by means of a Monte-Carlo simulation algorithm applying the variance reduction technique of Szántai [Szá86] and some other obtained with their method based on Boole-Bonferroni bounding technique. For the latters, they consider three levels of approximation. We give here only the most accurate results, obtained with the S_1-S_4 based method.

Table 4 presents, for each method, the obtained probability (or lower and upper bounds of the probability), the running time in seconds, and the number of nodes of BDDs. Running times for Habib and Szántai's results were obtained on a different computer, running

Table 4: Comparison of BDDs performance with results presented by Habib and Szántai

System	Method	$F_n^{k/r}$	Times	Sizes
$n = 15, r = 12, k = 8, Q = 0.75$	$S1$-$S4$ based	[0.916268,0.916268]	0.00	-
	BDD	0.916268	0.00	235
$n = 15, r = 10, k = 4, Q = 0.25$	$S1$-$S4$ based	[0.375167,0.407571]	0.00	-
	Simulation	0.395 ± 0.001257	2.85	-
	BDD	0.394538	0.00	289
$n = 15, r = 7, k = 5, Q = 0.25$	$S1$-$S4$ based	[0.053382,0.060869]	0.00	-
	Simulation	0.057 ± 0.000516	2.80	-
	BDD	0.0570453	0.00	246
$n = 30, r = 6, k = 3, Q = 0.10$	$S1$-$S4$ based	[0.136317,0.178668]	0.11	-
	Simulation	0.151 ± 0.000936	5.99	-
	BDD	0.1514353	0.00	363
$n = 40, r = 7, k = 4, Q = 0.10$	$S1$-$S4$ based	[0.038368,0.047818]	0.22	-
	Simulation	0.042 ± 0.000450	8.35	-
	BDD	0.0421106	0.00	1122
$n = 50, r = 40, k = 28, Q = 0.50$	$S1$-$S4$ based	[0.018356,0.024645]	2.09	-
	Simulation	0.0021 ± 0.000435	15.71	-
	BDD	0.0211604	1.11	211427
$n = 50, r = 35, k = 20, Q = 0.50$	$S1$-$S4$ based	[0.411635,0.583983]	38.56	-
	Simulation	0.463 ± 0.001920	50.48	-
	BDD	0.462869	147.28	3833835

a different operating system, and with programs written in a different language. Therefore, it makes a little sense to compare them directly with our own. However, orders of magnitude can be compared.

These results show that BDDs are able to deal with rather large systems. This corrects a claim by Habib and Szántai who wrote in [HS00] that only simulation is able to assess (almost) exact results for such systems. However, the two last examples enlight limits of BDDs. Their size grows quickly as k and r increase. A BDD node is encoded within 4 machine words. It means that nowadays computer memories are exhausted if BDDs with more than few millions nodes have to be built.

Table 4 shows that Monte-Carlo simulations give accurate results with a good efficiency. However, it would be interesting to study their accuracy with lower and more realistic component unreliabilities, for it is well known that rare events are difficult to capture with this kind of methods.

The Habib-Szántai's method gives good results as well. However, its performance should deteriorate as n increases (in reason of the increasing number of terms). Moreover, it requires that all components have the same reliability.

4.3 BDD Sizes

The second series of experiments aims to show how BDD sizes increase as k, r and n increase. We take the number of nodes as the size of a BDD.

It can be shown, but this requires a tedious case study that is outside the scope of this article, that if $n > 2r$, the size $\chi_n^{k/r}$ of the BDD that encodes a k-within-r-out-of-n is as follows.

$$\chi_n^{k/r} = 2 \times \sum_{i=1}^{r-1} \sum_{j=1}^{s} \binom{min(r-j, i-1)}{s-j} + (n-2r-1) \times \sum_{j=1}^{s} \binom{r-j}{s-j} + z \qquad (4)$$

where,

- $s = k$ if $k \leq r/2$ and $s = r - k + 2$ otherwise.
- $z = 1 - k$ if $k \leq r/2$ and $z = 1 - k + 2$ otherwise.

In order to illustrate this result, we fix first k and r and we let n increase. We repeat this experiment for various values of k, for $r = 16$ and for $n = 16, 24, 32, \ldots, 80$. Each row of the table 5 gives the growth of BDD sizes. For all values of k, after $n = 32$, the size grows linearly with n, which is indeed a very good new for BDDs.

Second, we fix r and n and we let k increase. Each column of the table 5 gives the growth of BDD sizes. A strong symmetry appears. The maximum size is obtained for $k = 9$, sizes for $9 + i$ and $9 - i$, $i = 1, 2, \ldots$ are approximately the same.

Third, we fix k and n and let r increase. Table 6 gives the size of the BDDs as well as the computation times (in seconds) for $n = 64$ and $k = 9$. We were unable to compute BDDs beyond $r = 20$. For r larger than 20, the given sizes were obtained by applying

Table 5: Sizes of BDDs encoding k-within-16-out-of-n systems

$k \downarrow n \rightarrow$	16	24	32	40	48	56	64	72	80
4	53	1465	5555	10035	14515	18995	23475	27955	32435
6	67	4183	32753	67697	102641	137585	172529	207473	242417
8	73	6135	76067	167587	259107	350627	442147	533667	625187
9	73	6390	84360	187320	290280	393240	496200	599160	702120
10	71	6133	76065	167585	259105	350625	442145	533665	625185
12	61	4177	32747	67691	102635	137579	172523	207467	242411
14	43	1455	5545	10025	14505	18985	23465	27945	32425

Table 6: Sizes of BDDs and running times for 9-within-r-out-of-64 systems.

r	10	12	14	16	18	20	22	24
Size	2365	23399	127975	496200	1517376	3892140	8690628	17281155
Time	0.01	0.05	0.35	3.12	46.91	431.36	?	?

equation 4. Table 6 illustrates the combinatorial explosion of BDD sizes when k and n are fixed and r increases (up to $n/2$).

Equation 4 makes explicit advantages and drawbacks of BDDs. On the one hand, when k and r are small, the BDD is small, even for large values of n. For instance the BDD that encodes a 5-within-10-out-of-1000 system is made only of $207,168$ nodes and is computed in 0.76s. On the other hand, when k and r are not too small, the BDD is huge even for small values of n. For instance the BDD that would encode a 15-within-30-out-of-60 system would be made of $986,022,749$ nodes.

5 Bounds on reliability of k-within-r-out-of-n systems

From now, we shall assume that all components have the same reliability P and unreliability Q. This section is organized as follows. First, we present a basic algorithm that runs in $\mathcal{O}(k.n)$. Then, we extend it into the full algorithm that runs in $\mathcal{O}(2^h.k.n)$. The basic algorithm corresponds to the case $h = 0$.

5.1 Principle

The basic algorithm consists of two nested loops. The outer loop iterates over components. The inner loop assesses S_i^j, for $0 \leq j \leq k$. The following equality holds. It justifies the algorithm.

$$\mathcal{F}_n \;=\; \mathcal{K}_r^k \;\cup\; \bigcup_{i=r+1}^{n} \mathcal{S}_i^k \tag{5}$$

Terms of the right member are pairwisely disjoint, therefore equality 5 can be lifted to probabilities. I.e.

$$F_n \;=\; K_r^k \;+\; \sum_{i=r+1}^{n} S_i^k \tag{6}$$

5.2 Decomposition of $\mathcal{S}_i^k$

$\mathcal{S}_i^l$ $(0 \leq l \leq k,\, r+1 \leq i \leq n)$ can be decomposed according to what enters and goes out a sliding window of r components. We distinguish four cases: the main case $0 < l < k-1$ and the three degenerated cases $l = 0$, $l = k-1$ and $l = k$.

case $0 < l < k-1$

$$\begin{aligned} \mathcal{S}_i^l \;=\;& (\mathcal{Q}_i \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{l-1}) \;\cup\; (\mathcal{Q}_i \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{l}) \\ \cup\; & (\overline{\mathcal{Q}_i} \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{l}) \;\cup\; (\overline{\mathcal{Q}_i} \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{l+1}) \end{aligned} \tag{7}$$

case $l = 0$

$$\mathcal{S}_i^0 \;=\; (\overline{\mathcal{Q}_i} \cap S_{i-1}^0) \;\cup\; (\overline{\mathcal{Q}_i} \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^1) \tag{8}$$

case $l = k-1$

$$\begin{aligned} \mathcal{S}_i^{k-1} \;=\;& (\mathcal{Q}_i \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{k-2}) \;\cup\; (\mathcal{Q}_i \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{k-1}) \\ \cup\; & (\overline{\mathcal{Q}_i} \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{k-1}) \end{aligned} \tag{9}$$

case $l = k$

$$\mathcal{S}_i^k \;=\; \mathcal{Q}_i \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{k-1} \tag{10}$$

Again, terms of members are pairwisely disjoint. Therefore, the above equalities can be lifted to probabilities.

By extending slightly the definition, we can set $S_r^l = T_r^{l,l}$. S_r^l as well as K_r^k can be computed in $\mathcal{O}(k.r)$ using functions given Fig. 1 and 2.

$\mathcal{Q}_i$ is independant of both $\mathcal{Q}_{i-r}$ and $\mathcal{S}_{i-1}^l$. Therefore, $p(\mathcal{Q}_i \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l) = Qz \times p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l)$. A similar remark holds for terms that involve $\overline{\mathcal{Q}_i}$ and $\overline{\mathcal{Q}_{i-r}}$.

The idea is to approximate $p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l)$ and $p(\overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^l)$ from $p(\mathcal{S}_{i-1}^l)$. We shall see that this can be done in constant time. Therefore, equations 7-10 give a mean to assess F_n in $\mathcal{O}(k.n)$.

The ANSI C code for this basic algorithm is given appendix B.

5.3 Approximation(s) of $p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l)$

Assume that we toss a coin r times with a probability P to draw a head. There are $\binom{r}{l}$ different drawings that give exactly l heads. The proportion of these drawings that start with a head is $\frac{\binom{r-1}{l-1}}{\binom{r}{l}} = \frac{l}{r}$. This proportion does not depend on P, because all drawings (that give l heads) have the same probability to occur, namely $P^l(1-P)^{r-l}$.

The idea is thus to approximate $p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l)$ as follows.

$$p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l) \approx \frac{l}{r} \times S_{i-1}^l \tag{11}$$

Indeed, $p(\overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^l) \approx \frac{r-l}{r} \times S_{i-1}^l$.

Let σ be a sequence of states of components c_1 up to c_{i-1} ending with l failed components among $c_{i-r}, \ldots, c_{i-1}$. Assume that $\sigma_{i-r} = 0$ (i.e. c_{i-r} is failed). Let ρ be any sequence such that σ and ρ differ on exactly two components: c_{i-r} and a component c_s such that $s > i-r$ and $\sigma_s = 1$ ($\rho_{i-r} = 1$, $\rho_s = 0$). Then, if σ belongs to $\mathcal{S}_{i-1}^l$, ρ belongs to $\mathcal{S}_{i-1}^l$ as well. The converse is not true. In other words, if the number x_i of sequences that belong to $\mathcal{S}_i^l$ is $a_i.\binom{l}{i}$ for a given positive number a_i, then the number y_i of sequences σ of $\mathcal{S}_i^l$ such that $\sigma(i-r) = 1$ is less than $a_i.\binom{l-1}{i-1}$. Therefore, $\frac{y_i}{x_i} < \frac{l}{i}$.

It follows that equation 11 gives an upper approximation of $p(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l)$.

5.4 Taking the recent past into account

To apply equation 11, one needs only to store the last value of S_i^l, for $0 \leq l < k$. It is possible to refine this approximation by taking into account states of components $c_{i-h}, \ldots, c_{i-1}$, for a given h such that $0 \leq h \leq r$. In other words, $\mathcal{S}_{i-1}^l$ can be decomposed as follows (extending equation 7).

$$\begin{array}{rcl|l}
\mathcal{S}_{i-1}^l & = & (\mathcal{S}_{i-1}^l \cap \mathcal{Q}_{i-1}) \cup (\mathcal{S}_{i-1}^l \cap \overline{\mathcal{Q}_{i-1}}) & h=1 \\
& = & (\mathcal{S}_{i-1}^l \cap \mathcal{Q}_{i-1} \cap \mathcal{Q}_{i-2}) \cup (\mathcal{S}_{i-1}^l \cap \mathcal{Q}_{i-1} \cap \overline{\mathcal{Q}_{i-2}}) & h=2 \\
& \cup & (\mathcal{S}_{i-1}^l \cap \overline{\mathcal{Q}_{i-1}} \cap \mathcal{Q}_{i-2}) \cup (\mathcal{S}_{i-1}^l \cap \overline{\mathcal{Q}_{i-1}} \cap \overline{\mathcal{Q}_{i-2}}) & \\
& = & \vdots & h=\vdots
\end{array}$$

The idea is to maintain 2^h vectors of probabilities, where h stands for the length of the history we want to consider. These vectors contain the probabilities $p(\mathcal{S}_{i-1}^l \cap \bigcap_{j=i-1}^{i-h} \mathcal{Q}_j^{a_j})$, where $a_j \in \{0,1\}$, $\mathcal{Q}_j^1$ denotes $\mathcal{Q}_j$ and $\mathcal{Q}_j^0$ denotes $\overline{\mathcal{Q}_j}$. The refinement of the approximation is as follows.

$$p\left(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^l \cap \bigcap_{j=i-h}^{i-1} \mathcal{Q}_j^{a_j}\right) \approx \frac{l - \sum_{j=i-h}^{i-1} a_i}{r-h} \times p\left(\mathcal{S}_{i-1}^l \cap \bigcap_{j=i-h}^{i-1} \mathcal{Q}_j^{a_j}\right)$$

$$p\left(\overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^l \cap \bigcap_{j=i-h}^{i-1} \mathcal{Q}_j^{a_j}\right) \approx \frac{(r-h) - (l - \sum_{j=i-h}^{i-1} a_i)}{r-h} \times p\left(\mathcal{S}_{i-1}^l \cap \bigcap_{j=i-h}^{i-1} \mathcal{Q}_j^{a_j}\right)$$

$$
\begin{array}{cccccccccc}
\ldots & a_{i-r} & a_{i-r+1} & \ldots & a_{i-h} & a_{i-h+1} & \ldots & a_{i-1} & a_i \\
 & & [\ldots\ldots\ldots\ldots\ldots\ldots & l & \ldots\ldots\ldots\ldots\ldots\ldots] & & & & \\
 & [\ldots\ldots\ldots\ldots\ldots & l + a_{i-r} - a_i & \ldots\ldots\ldots\ldots\ldots] & & & & & \\
 & [\ l + a_{i-r} - a_i - \sum_{j=i-h}^{i-1} a_j\ ] & & & & & & &
\end{array}
$$

Figure 5: A pictural view of the number of failed components in a sliding r-window

The application of the above approximations requires careful case studies in order to make sure that $\sum_{j=i-1}^{i-h} a_j \leq l$.

The above left members can be computed incrementally as in the basic algorithm. The main case is as follows.

$$
\begin{array}{rcl}
p\left(\mathcal{S}_i^l \cap \bigcap_{j=i-h+1}^{i} \mathcal{Q}_j^{a_j}\right) & & \\
= & & p\left(\overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{l-a_i} \cap \bigcap_{j=i-h+1}^{i} \mathcal{Q}_j^{a_j}\right) \\
& + & p\left(\mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{l-a_i+1} \cap \bigcap_{j=i-h+1}^{i} \mathcal{Q}_j^{a_j}\right) \\
= & & p\left(\mathcal{Q}_i^{a_i} \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{l-a_1} \cap \mathcal{Q}_{i-h} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p\left(\mathcal{Q}_i^{a_i} \cap \overline{\mathcal{Q}_{i-r}} \cap \mathcal{S}_{i-1}^{l-a_1} \cap \overline{\mathcal{Q}_{i-h}} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p\left(\mathcal{Q}_i^{a_i} \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{l-a_i+1} \cap \mathcal{Q}_{i-h} \cap \bigcap_{j=i-h+11}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p\left(\mathcal{Q}_i^{a_i} \cap \mathcal{Q}_{i-r} \cap \mathcal{S}_{i-1}^{l-a_i+1} \cap \overline{\mathcal{Q}_{i-h}} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
\approx & & p(\mathcal{Q}_i^{a_i}) \times \frac{(r-h)-(l-a_i-\sum_{j=i-h+1}^{i-1} a_j-1)}{r-h} \times p\left(\mathcal{S}_{i-1}^{l} \cap \mathcal{Q}_{i-h} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p(\mathcal{Q}_i^{a_i}) \times \frac{(r-h)-(l-a_i-\sum_{j=i-h+1}^{i-1} a_j)}{r-h} \times p\left(\mathcal{S}_{i-1}^{l} \cap \overline{\mathcal{Q}_{i-h}} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p(\mathcal{Q}_i^{a_i}) \times \frac{l-a_i+1-\sum_{j=i-h+1}^{i-1} a_j-1}{r-h} \times p\left(\mathcal{S}_{i-1}^{l-1} \cap \mathcal{Q}_{i-h} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right) \\
& + & p(\mathcal{Q}_i^{a_i}) \times \frac{l-a_i+1-\sum_{j=i-h+1}^{i-1} a_j}{r-h} \times p\left(\mathcal{S}_{i-1}^{l-1} \cap \overline{\mathcal{Q}_{i-h}} \cap \bigcap_{j=i-h+1}^{i-1} \mathcal{Q}_j^{a_j}\right)
\end{array}
$$

Figure 5 may help the reader to figure out how above equations work.

Let $\tilde{F}_n^h$ denote the approximation of F_n obtained by applying the algorithm with a history of length h. Each iteration of the outer loop of the algorithm consists in updating the 2^h values for each l, $0 \leq l \leq k$. Therefore, the algorithm runs in $\mathcal{O}(2^h.k.n)$.

Now, there are two important facts.

Fact 1 *The following inequalities hold.*

$$\tilde{F}_n^0 \leq \tilde{F}_n^1 \leq \ldots \leq \tilde{F}_n^r = F_n$$

Fact 2 *The following inequality holds (for $h = 0, \ldots, r-2$).*

$$\tilde{F}_n^{h+1} - \tilde{F}_n^h \geq \tilde{F}_n^{h+2} - \tilde{F}_n^{h+1}$$

At the moment, we have no short proof for these two facts. The proof of fact 1 is by means of double induction on n and k to show that the sum of the computed values for S_i^l ($0 \leq l \leq k-1$, $r+1 \leq i \leq n$) is greater than the sum of their actual values. We have no simple proof of fact 2. Anyway, both facts are supported by strong experimental evidences.

Fact 1 asserts that the algorithm computes a lower bound and converges to the exact values. Fact 2 asserts that its convergence is fast. Moreover, it can be used to compute upper bounds. The following inequality holds for $h = 1, \ldots, r$.

$$F_n \quad \leq \quad \tilde{F}_n^h + (r-h).[\tilde{F}_n^h - \tilde{F}_n^{h-1}] \tag{12}$$

Proof. $\tilde{F}_n^r = \tilde{F}_n^h + \sum_{i=h+1}^{r} \tilde{F}_n^i - \tilde{F}_n^{i-1}$. By fact 2, we have $\tilde{F}_n^i - \tilde{F}_n^{i-1} \leq \tilde{F}_n^h - \tilde{F}_n^{h-1}$ for $h < i \leq r$. The inequality 12 follows just by summing the terms.

The idea is therefore to compute successively the following ranges.

h	lower bound	upper bound
0	$\tilde{F}_n^0$	$min(1, r.\tilde{F}_n^0)$
1	$\tilde{F}_n^1$	$\tilde{F}_n^1 + (r-1).[\tilde{F}_n^1 - \tilde{F}_n^0]$
$\vdots$		
h	$\tilde{F}_n^h$	$\tilde{F}_n^h + (r-h).[\tilde{F}_n^h - \tilde{F}_n^{h-1}]$
$\vdots$		

The process is iterated until either h reaches a given value or the maximum relative uncertainty goes below a predefined threshold. The maximum relative uncertainty is defined as follows.

$$\frac{(r-h).[\tilde{F}_n^h - \tilde{F}_n^{h-1}]}{\tilde{F}_n^h}$$

It is worth noticing that this algorithm is still in $\mathcal{O}(2^h.k.n)$.

6 Experimental results

6.1 Comparison with results Habib and Szántai

Tables 7, 8, 9 and 10 report results of the algorithm described in the previous section on the four biggest examples studied by Habib and Szántai. We set the maximum value for h to 15 and the threshold for the maximum relative uncertainty to 1%. In all of the

Table 7: $n = 30, r = 6, k = 3, Q = 0.10, F_{30}^{3/6} = 0.1514353$

h	lower bound	upper bound	%error	%uncert.	time
$S1$-$S4$ based	0.136317	0.178668	11	31	0.11
0	0.140738	0.985163	7.6	600	0.00
1	0.144795	0.165079	4.6	14	0.00
2	0.147685	0.159249	2.5	7.8	0.00
3	0.149676	0.155649	1.2	4	0.00
4	0.150902	0.153355	0.35	1.6	0.00
5	0.151435	0.151968	0.0002	0.35	0.00

Table 8: $n = 40, r = 7, k = 4, Q = 0.10, F_{40}^{4/7} = 0.0421106$

h	lower bound	upper bound	%error	%uncert.	time
$S1$-$S4$ based	0.038368	0.047818	9.8	25	0.22
0	0.0395055	0.316044	6.6	700	0.00
1	0.0405621	0.0469018	3.8	16	0.00
2	0.0412406	0.0446329	2.1	8.2	0.00
3	0.0416707	0.043391	1.1	4.1	0.00
4	0.0419282	0.0427006	0.44	1.8	0.00
5	0.0420626	0.0423316	0.11	0.64	0.00

Table 9: $n = 50, r = 40, k = 28, Q = 0.50, F_{50}^{28/40} = 0.0211604$

h	lower bound	upper bound	%error	%uncert.	time
$S1$-$S4$ based	0.018356	0.024645	15	34	2.09
0	0.0208034	0.852939	1.7	4000	0
1	0.0209393	0.026241	1.1	25	0.00
2	0.021023	0.0242029	0.65	15	0.00
3	0.0210771	0.0230771	0.4	9.5	0.00
4	0.0211122	0.0223766	0.23	6	0.00
5	0.0211346	0.0219171	0.12	3.7	0.00
6	0.0211481	0.0216095	0.058	2.2	0.01
7	0.0211557	0.0214048	0.022	1.2	0.02
8	0.0211592	0.0212735	0.0057	0.54	0.04

Table 10: $n = 50, r = 35, k = 20, Q = 0.50, F_{50}^{20/35} = 0.462869$

h	lower bound	upper bound	%error	%uncert.	time
$S1$-$S4$ based	0.411635	0.583983	12	42	38.56
0	0.453422	1	2.1	120	0.00
1	0.455741	0.534583	1.6	17	0.00
2	0.45744	0.513492	1.2	12	0.00
3	0.458748	0.500632	0.9	9.1	0.00
4	0.459776	0.491635	0.67	6.9	0.00
5	0.460588	0.48493	0.5	5.3	0.01
6	0.461226	0.479755	0.36	4	0.01
7	0.461725	0.475673	0.25	3	0.02
8	0.462106	0.472408	0.17	2.2	0.04
9	0.46239	0.469778	0.1	1.6	0.07
10	0.462593	0.467667	0.06	1.1	0.15
11	0.462729	0.465999	0.03	0.71	0.33

subsequent tables, we give the percentage of relative error, i.e. $\frac{F_n^{k/r} - LB}{LB}$, as well as the percentage of relative uncertainty, i.e. $\frac{UB - LB}{LB}$. Both values are of interest to give a picture of the accuracy of methods.

The following observations can be made on these results.

- Running times are excellent.
- The algorithm converges in few steps even on strong requirements on the maximum relative uncertainty.
- Approximations are much more accurate than those obtained with the algorithm by Habib and Szántai, even for $h = 1$.
- The lower bound is much closer to the exact result than the upper bound. It could be the case that with more mathematical efforts the latter can be improved.

6.2 Further experiments

We shall now report further experiments to show the advantages, but also the limits, of our method.

Table 11 reports results on the last example of [HS00] with various values of Q. As previously, we set the threshold for the maximum relative uncertainty to 1%. Each row of the table corresponds to a value of Q. The given results are those obtained for the smallest value of h such that the relative error is below the threshold. These results show that the algorithm is rather sensitive to the reliability of components. Its accuracy decreases at Q tends to 0.5. The same picture is obtained for all values of k, r and n.

Table 11: $n = 50, r = 35, k = 20$

Q	$F_{50}^{20/35}$	h	LB	UB	%error	%uncert.	time
0.75	0.999519	0	0.999381	1	0.014	0.062	0.00
0.6	0.882321	9	0.881804	0.889007	0.059	0.82	0.07
0.5	0.462869	11	0.462729	0.465999	0.03	0.71	0.33
0.4	0.0851995	10	0.0851587	0.0859966	0.048	0.98	0.13
0.25	0.000253384	8	0.000253262	0.000255672	0.048	0.95	0.03
0.1	5.51169e-11	5	5.51017e-11	5.5526e-11	0.028	0.77	0.01
0.01	2.63586e-30	2	2.63558e-30	2.64961e-30	0.011	0.53	0.00

One could expect that the accuracy of our algorithm decreases as n increases, because it is the case for most of other approximation schemes. In order to study how much its degrades, we consider 15-within-20-out-of-n systems. We study these systems for two reasons: first, their exact reliabilities can be assessed by means of BDDs. Second, real-life problems such as those mentioned by Papastavridis and Koutras in [PK93], namely quality control procedures and radar detection, should be such that k is close to r and n is large w.r.t. r. Table 12 reports results we obtained for several values of n. These results show that the accuracy of our method actually decreases as n increases, but very slowly. Moreover, it seems that the accuracy decreases linearly with n. This gives a rule of thumb to correct the lower bound: compute the exact value for k, r and some low values of n, deduce the decrease of accuracy as a function of n, compute the lower bound for the actual value of n and finally correct it.

To end this presentation, table 13 reports results we obtained on 224-within-256-out-of-n systems. In order to make the reliability of such systems realistic, we took $Q = 0.75$. These results show that our method is still efficient and accurate on very large systems (much larger than any system already studied in the literature).

7 Conclusion

In this article, we showed that there exist very simple yet very efficient algorithms to assess k-out-of-n, l-to-h-out-of-n and consecutive k-out-of-n systems. Namely, these algorithms run respectively in $\mathcal{O}(k.n)$, $\mathcal{O}(h.n)$ and $\mathcal{O}(k.n)$. In our humble point of view, they make of a little interest further improvements in the assessment of systems of the above classes.

Then, we considered k-within-r-out-of-n systems that are instrinsically much harder.

First, we studied the performance of Binary Decision Diagrams on k-within-r-out-of-n systems. We explicited the number of nodes of the BDD that encodes such a system for the case where $n \geq 2r$. We show by means of an experimental study that BDD makes it possible to compute the exact value of the reliability of k-within-r-out-of-n systems for small values of r (say $r \approx 10$) but large values of n (say $n \approx 1000$). It is worth noticing that BDD can be used even in the case where components have different reliabilities.

Table 12: $r = 20, k = 15$

	n	$F_{50}^{15/20}$	LB	UB	%error	%uncert.	time
$Q = 0.1, h = 4$	40	1.28822e-10	1.28799e-10	1.29422e-10	0.018	0.48	0.00
	50	1.88329e-10	1.88289e-10	1.89335e-10	0.021	0.56	0.00
	60	2.47837e-10	2.47778e-10	2.49249e-10	0.024	0.59	0.00
	70	3.07344e-10	3.07268e-10	3.09162e-10	0.025	0.62	0.00
	80	3.66851e-10	3.66757e-10	3.69075e-10	0.026	0.63	0.00
	90	4.26358e-10	4.26247e-10	4.28988e-10	0.026	0.64	0.01
	100	4.85865e-10	4.85737e-10	4.88902e-10	0.026	0.65	0.01
$Q = 0.5, h = 10$	40	0.10052	0.0999556	0.101953	0.56	2	0.13
	50	0.13667	0.134815	0.139778	1.4	3.7	0.20
	60	0.171319	0.168218	0.17631	1.8	4.8	0.26
	70	0.204582	0.200302	0.21143	2.1	5.6	0.33
	80	0.236509	0.23114	0.245116	2.3	6	0.38
	90	0.267154	0.260787	0.277397	2.4	6.4	0.44
	100	0.29657	0.28929	0.308319	2.5	6.6	0.51

Table 13: $r = 256, k = 224, Q = 0.75$

n	h	LB	UB	%uncert.	time
512	1	1.07986e-05	1.92281e-05	78	0.05
	5	1.08976e-05	1.61837e-05	49	0.89
	10	1.098e-05	1.44353e-05	31	37.24
1024	1	3.06454e-05	5.61601e-05	83	0.12
	5	3.09509e-05	4.74539e-05	53	2.61
	10	3.12129e-05	4.23426e-05	36	111.26
2048	1	0.703347e-04	1.30026e-04	85	0.29
	5	0.710531e-04	1.09998e-04	55	6.11
	10	0.716746e-04	0.98162e-04	37	259.9
4096	1	1.49708e-04	2.77744e-04	86	0.62
	5	1.51253e-04	2.35077e-04	55	13.11
	10	1.52593e-04	2.09793e-04	37	556.83

Then, we proposed an algorithm to compute lower and upper bounds of k-within-r-out-of-n system reliabitilities in the case where all components have the same reliability. This algorithm runs in $\mathcal{O}(2^h.k.n)$, where $0 \leq h \leq r$ is parameter that fixes the allowed amount of computations. We show by means of experiments that it outperforms the best algorithms proposed up to now: it is much faster and it gives much more accurate results.

References

[BRB90] K. Brace, R. Rudell, and R. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45. IEEE 0738, 1990.

[Bry86] R. Bryant. Graph Based Algorithms for Boolean Fonction Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[CFK95] M.T. Chao, J.C. Fu, and M.V. Koutras. Survey on Reliability Studies of Consecutive-k-out-of-n:f & related systems. *IEEE Transactions on Reliability*, 44(1):120–127, 1995.

[CM94] O. Coudert and J.-C. Madre. MetaPrime: an Iteractive Fault Tree Analyser. *IEEE Transactions on Reliability*, 43(1):121–127, March 1994.

[Gri86] W.S. Griffith. On consecutive k-out-of-n failure systems and their generalizations. *Reliability and Quality Control*, pages 157–165, 1986.

[HS00] A. Habib and T. Szántai. New bounds on the reliability of consecutive k-out-of-r-from-n:f system. *Reliability Engineering and System Safety*, 68:97–104, 2000.

[II95] L.A. Belfore II. A $\mathcal{O}(n.(log_2(n)^2))$ Algorithm for Computing the Reliability of k-out-of-n:G & k-to-l-out-of-n:G systems. *IEEE Transactions on Reliability*, 44(1), March 1995.

[JG85] S.P. Jain and K. Gopal. Reliability of k-to-l-out-of-n Systems. *Reliability Engineering*, 12:175–179, 1985.

[Mis93] K.R. Misra. *New Trends in System Reliability Evaluation.* Fundamental Studies in Engineering, 16. Elsevier, 1993. ISBN 0-444-816607.

[PK93] S.G. Papastravidis and M.V. Koutras. Bounds for Reliability of Consecutive k-within-m-out-of-n:f Systems. *IEEE Transactions on Reliability*, 42(1):156–160, March 1993.

[Rau93] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 05(59):203–211, 1993.

[Rus86] A.M. Rushdi. Utilization of Symmetric Switching Functions in the Computations k-out-of-n Systems Reliability. *Microelectronics and Reliability*, 26(5):973–987, 1986.

[Rus91] A.M. Rushdi. Comment on: An Efficient Non-recursive Algorithm for Computing the Reliability of k-out-of-n Systems. *IEEE Transactions on Reliability*, 40(1):60–61, April 1991.

[Szá86] T. Szántai. Evaluation of a special multivariate gamma distribution. *Math. Prog. Study.*, 27:1–16, 1986.

[Ton85] Y.L. Tong. Rearrangement inequality for the longuest run, with application to network reliability. *Journal of Applied Probability*, 22:286–393, 1985.

A Computation of BDDs for k-out-of-n systems

The algorithm that computes the BDD of a k-out-of-n system is essentially the logical counter-part of the algorithm of Fig. 1. It is based on the following recursive equations.

$$\begin{aligned} 0/n(\phi_1, \ldots, \phi_n) &= 1 \\ k/n(\phi_1, \ldots, \phi_n) &= 0 \text{ if } k > n \\ k/n(\phi_1, \ldots, \phi_n) &= \phi_1.k-1/n-1(\phi_2, \ldots, \phi_n) + \overline{\phi_1}.k/n-1(\phi_2, \ldots, \phi_n) \end{aligned}$$

This recursive principle can be implemented efficiently in an imperative style by means of two nested loops. The outer loop just iterates n times the inner loop. The inner loop goes through the $k+1$ cells of a vector $\vec{V}$ of BDDs. At the ith iteration of the outer loop, the BDD of the jth cell V_j encodes the formula $j/i(\phi_1, \ldots, \phi_i)$.

The pseudo code for this algorithm is as follows.

$V_0 \leftarrow$ `BddOne`
for $j = 1, \ldots, k$ do $V_j \leftarrow$ `BddZero` done
for $i = 1, \ldots, n$ do
    for $j = k, \ldots, 1$ do
        $V_j \leftarrow$ `BddOr`(`BddAnd`(ϕ_i, V_{j-1}), `BddAnd`(`BddNot`(ϕ_i), V_j))
        done
    $V_0 \leftarrow$ `BddAnd`(`BddNot`(ϕ_i), V_0)
    done
return V_k

B The basic algorithm

The ANSI C function that implements the basic algorithm is given Fig. 6. In order to simplify instructions, we use two arrays `M` and `N` in which the quotients $\frac{l}{r}$'s and $\frac{r-l}{r}$'s are stored. The array `T` is an auxiliary data structure that is used to store the S_{i-1}^l's.

```
double lower_bound_k_within_r_out_of_n(int k, int r, int n, double Q)
{
  double *S, *T, *M, *N, F, P;
  int    i, l;

  M=(double*) calloc(sizeof(double),k);
  N=(double*) calloc(sizeof(double),k);
  for (l=0;l<k;l++) { M[l]=l/(double)r; N[l]=1-M[l]; }

  S=(double*) calloc(sizeof(double),k);
  T=(double*) calloc(sizeof(double),k);

  F=0; P=1-Q;
  S[0]=1.0; for (l=1;l<k;l++) S[j]=0.0;
  for (i=1;i<=r;i++) {
    F=F+q*S[k-1];
    for (l=k-1;l>0;l--) S[l]=Q*S[l-1]+P*S[l];
    S[0]=P*S[0];
    }

  for (i=r+1;i<=n;i++) {
    for (l=0;l<k;l++) T[l]=S[l];
    S[0]=P*(T[0]+M[1]*T[1]);
    for (l=1;l<k-1;l++)
      S[l]=Q*(N[l-1]*T[l-1]+M[l]*T[l]) + P*(N[l]*T[l]+M[l+1]*T[l+1]);
    S[k-1]=Q*(N[k-2]*T[k-2]+M[k-1]*T[k-1]) + P*N[k-1]*T[k-1];
    F=F+Q*N[k-1]*T[k-1];
    }

  free(S); free(T); free(M); free(N);
  return(F);
}
```

Figure 6: The ANSI C function to compute a lower bound of the reliability of k-within-r-out-of-n systems.