Simplification of inclusion-exclusion on intersections of unions with application to network systems reliability

Citation for published version:
Schäfer, L, Garcia, S & Srithammavanh, V 2018, 'Simplification of inclusion-exclusion on intersections of unions with application to network systems reliability', Reliability Engineering & System Safety, vol. 173, pp. 23-33. https://doi.org/10.1016/j.ress.2018.01.003

Digital Object Identifier (DOI):
10.1016/j.ress.2018.01.003

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Reliability Engineering & System Safety

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Simplification of Inclusion-Exclusion on Intersections of Unions
with Application to Network Systems Reliability

Lukas Schäfer1,∗, Sergio García1, Vassili Srithammavanh2

1School of Mathematics, University of Edinburgh, United Kingdom
2AIRBUS Group Innovations, France

Abstract

Reliability of safety-critical systems is a paramount issue in system engineering because in most practical situations the reliability of a non series-parallel network system has to be calculated. Some methods for calculating reliability use the probability principle of inclusion-exclusion. When dealing with complex networks, this leads to very long mathematical expressions which are usually computationally very expensive to calculate. In this paper, we provide a new expression to simplify the probability principle of inclusion-exclusion formula for intersections of unions which appear when calculating reliability on non series-parallel network systems. This new expression exploits the presence of many repeated events and has many fewer terms, which significantly reduces the computational cost. We also show that the general form of the probability principle of inclusion-exclusion formula has a double exponential complexity, whereas the simplified form has only an exponential complexity with a linear exponent. Finally, we compare its computational efficiency against the sum of disjoint products method KDH88 for a simple artificial example and for a door management system, which is a safety-critical system in aircraft engineering.

Keywords: Structural reliability; combinatorics; non series-parallel systems; inclusion-exclusion

1 Introduction

Reliability of a network system is the probability of the system not failing. It is a critical issue in different fields such as computer networks, information networks or gas networks. In particular, reliability of safety-critical network systems ([21], [19]) is an important topic in system engineering. For example, aircraft architecture has safety-critical network systems such as fly-by-wire, actuation, fire warning and door management systems. In most practical situations, the reliability of a complex network system (e.g. a system that is not series-parallel) has to be calculated exactly [8]. There are several methods to calculate or simulate the reliability of a complex system which have been developed in recent decades. Some classical static modelling techniques, including reliability block diagram models [10], fault tree models, and binary decision diagram models, have been widely used to model static systems. A general introduction to these methods can be found in [21]. For time-dependent systems, modeling techniques such as Markov models [12], dynamic fault tree models [3] and Petri net models [30] have been used. Reliability system calculation can also be divided into systems or multi-state components, where the components of a system operate in any of several intermediate states with various effects on the entire system performance ([17], [10], [15], [20], [25] and [14]), or with binary-state components, where either a component works perfectly or not at all. In this paper, we consider binary-state components. Furthermore, reliability of complex systems with specific graph structures, like systems with a hypercube structure ([11], [13]), got attention over the last years. But whereas most of these methods consider systems with two-terminal nodes or k-terminal nodes where all k-nodes have to be connected, we consider a different type of complex systems and a specific structure that has multiple functions with multiple start and end nodes.

∗Corresponding author at: University of Edinburgh, School of Mathematics, James Clerk Maxwell Building, The King’s Buildings, Peter Guthrie Tait Road, Edinburgh, EH9 3FD, United Kingdom. E-mail address: lukas.schaefer@ed.ac.uk
In this paper, we propose a new method to calculate the reliability of such a complex system with a new way of writing the classic probability principle of inclusion-exclusion formula. The classical probability principle of inclusion-exclusion formula is

\[ P\left( \bigcup_{i=1}^{n} A_i \right) = \sum_{i=1}^{n} \left( (-1)^{i+1} \sum_{J \subseteq \{1, \ldots, n\}, \mid J \mid = i} P\left( \bigcap_{j \in J} A_j \right) \right). \]  

(1)

The new method detects which combination of events leads to the same event when simplified and has, therefore, many fewer summands than the classical formula for intersections of unions.

Practical reliability calculations often involve very long expressions when the probability principle of inclusion-exclusion formula \([1]\) is used. Therefore, there are many approaches in the literature on general network reliability calculations to simplify the probability principle of inclusion-exclusion such as, for example, partitioning techniques \([6]\) and the sum of disjoint products method \((11, 7, 2, 18, 22, 29)\). The sum of disjoint products method is the most often used approach, with recent results in \((28, 27, 20, 5)\) and \([23]\).

All these methods need the exact system structure to simplify the reliability calculation. Therefore, in this paper, we propose a new approach to simplify the probability principle of inclusion-exclusion without needing the exact system structure, and to apply it to the calculation of the reliability of complex network systems in system engineering. In the following, we introduce what kind of complex network systems we consider and why we consider a method that does not need the exact system structure.

In system engineering, most network systems have multiple functions that have to be performed and these are not always independent (e.g., they share components). Reliability can be increased if different sets of components in the network can perform the same function. Therefore, these functions are implemented multiple times in the network system through different sets of components, and calculation of the reliability of the network system becomes a very complex task.

In our paper, we assume that all failure probabilities of the components are known exactly. We do not consider the case when these probabilities are known only approximately (e.g., either by estimation or a confidence interval). If the different components of the network are independent of each other, then we can easily calculate the reliability of a set of components. Through this we can calculate the reliability of one implementation of a function, which is defined as the probability of the event that one implementation of the function does not fail. Finally, the probability of an intersection of such events can be calculated easily. However, if full independence cannot be assumed, then the calculation becomes very expensive, usually prohibitively so.

Our main motivation lies in dealing with optimization problems with reliability constraints where this calculation means that costs for models even for very small networks make the problems intractable. The main reason for this is the large number of variables and non-linear constraints involved in the reliability calculation within the optimization model. Furthermore, the approaches to simplify the probability principle of inclusion-exclusion formula mentioned before (e.g., sum of disjoint products) are not suitable for use within an optimization formulation, because the exact structure of the system has to be known before constructing the reliability constraints. Also, approximation and the use of lower or upper bounds for the principal of inclusion-exclusion are not suitable for exact optimization because they are not in general monotone increasing or decreasing with regard to the reliability. In practice, optimization models that involve reliability are usually either solved through heuristics or by assuming series-parallel systems \((11, 9, 24)\).

In this paper, we show how to calculate the reliability of a network system in which components are not necessarily independent in a way that requires considerably fewer operations (and, thus, it is much cheaper computationally) than the direct use of the probability inclusion-exclusion principle. The key is to exploit the fact that, when dealing with a network system, the probability inclusion-exclusion principle has many repeated terms when applied to intersections of unions. Furthermore, we compare the computational efficiency and number of summands of our new method with the sum of disjoint products method KDH88 from \([7]\). KDH88 is a sum of disjoint products method with multiple-variable inversion that can be easily applied to a network system with subsystems that are subgraphs with multiple start and end nodes and not just paths with one start and end node. Therefore, it is applicable for the complex systems we consider and is suitable to make comparisons against.

The rest of the paper is organized as follows. In Section 2 we provide our motivation by showing why it can be expensive to calculate the reliability of a non series-parallel system. We also state the main result (Proposition 1) which provides a formula to calculate the reliability for a network system in an exact way with a much lower number of operations. To be able to use the formula from Proposition 1 we provide

\[ P\left( \bigcup_{i=1}^{n} A_i \right) = \sum_{i=1}^{n} \left( (-1)^{i+1} \sum_{J \subseteq \{1, \ldots, n\}, \mid J \mid = i} P\left( \bigcap_{j \in J} A_j \right) \right). \]  

(1)
an algorithm in Appendix C which can be easily implemented. In Section 3 we compare our method with
KDH88 and the classic probability principle of inclusion-exclusion for simple artificial examples and a
door management system application. Finally, we provide some conclusions and discuss future perspectives
in Section 4.

2 Motivation and Main Result

We start by showing that, if independence cannot be assumed, then it can be very expensive to calculate
the probability of a non series-parallel network system with multiple functions and implementations. Af-


-\begin{align*}
\text{Let } n = \text{number of functions in the system and } t_i = \text{number of implementations of function } i \\
\text{in the system. Let } F_i, i \in \{1, \ldots, n\}, \text{ be the event that function } i \text{ of a system does not fail in a specific period of time} \\
\text{and } F_{ij}, j \in \{1, \ldots, t_i\}, \text{ be the event that implementation } j \text{ of function } i \text{ does not fail in a specific period of time. Let } F = \{F_1, \ldots, F_n\} \text{ be the set of all functions and } F_i = \{F_{i1}, \ldots, F_{it_i}\} \text{ be the set of} \\
\text{all implementations of function } i. \text{ Furthermore, let } R \text{ be the event that the system does not fail. The} \\
\text{reliability of the system, } P(R), \text{ is the probability that no function in } F \text{ fails. A function } F \in F \text{ does not} \\
\text{fail if at least one of its implementations does not fail. Therefore,} \\

P(R) = P \left( \bigcap_{i=1}^{n} F_i \right) = P \left( \bigcap_{i=1}^{n} \bigcup_{j=1}^{t_i} F_{ij} \right) .
\end{align*}

Because the different functions and implementations may not be independent, } P(R) \text{ is not easily calculable.}
In order to work on this expression, first we need to establish some notations. Let

\[ W = \{1, \ldots, t_1\} \times \ldots \times \{1, \ldots, t_n\}, \text{ and} \]

\[ B_w = \bigcap_{i=1}^{n} F_{iw_i} \text{ for } w = (w_1, \ldots, w_n) \in W, \]

where } w_i \in \{1, \ldots, t_i\} \text{ and represents the implementation index of function } i. \text{ We then have that}

\[ P(R) = P \left( \bigcap_{i=1}^{n} \bigcup_{j=1}^{t_i} F_{ij} \right) = P \left( \bigcup_{w \in W} \left( \bigcap_{i=1}^{n} F_{iw_i} \right) \right) = P \left( \bigcup_{w \in W} B_w \right) . \tag{2} \]

Now the probability principle of inclusion-exclusion can be used and it follows that

\[ P(R) = \frac{|W|}{t=1} \left( (-1)^{t+1} \sum_{I \subseteq W, \left| I \right| = t} P \left( \bigcap_{j \in I} B_j \right) \right) . \tag{3} \]

The number of summands in } (3), \text{ which is equal to the number of possible intersections of } B_w \text{’s, is}

\[ \sum_{t=1}^{|W|} \left( \begin{array}{c} |W| \\ t \end{array} \right) = 2^{|W|} - 1 \] with } |W| = \prod_{i=1}^{n} |F_i|. \text{ We therefore have a doubly exponential computational} 

complexity. Table 1 shows the number of summands for different values of the number of functions and
implementations, with the assumption that every function has the same number of implementations.
Proposition 1. Let simplify the formula. The result we are looking for is the following: Therefore, it seems natural to determine which combinations lead to the same intersection set, and then becomes very expensive. However, note that there are many (a priori different) terms that, when the intersection of the sets is calculated, lead to the same intersection set, that is,\[
\exists I, J \subseteq W : I \neq J \land \bigcap_{i \in I} B_i = \bigcap_{j \in J} B_j.
\] (4)
For example, let \( F = \{F_1, F_2\}, F_1 = \{F_{11}, F_{12}\} \) and \( F_2 = \{F_{21}, F_{22}\} \). It follows that \[
P(R) = P((F_{11} \cap F_{21}) \cup (F_{11} \cap F_{22}) \cup (F_{12} \cap F_{21}) \cup (F_{12} \cap F_{22})).
\]
It can be seen, for example, that \[
B_{(1,1)} \cap B_{(2,2)} = (F_{11} \cap F_{21}) \cap (F_{12} \cap F_{22})
= F_{11} \cap F_{12} \cap F_{21} \cap F_{22}
= (F_{11} \cap F_{21}) \cap (F_{11} \cap F_{22}) \cap (F_{12} \cap F_{21})
= B_{(1,1)} \cap B_{(1,2)} \cap B_{(2,1)}.
\]
Therefore, it seems natural to determine which combinations lead to the same intersection set and then simplify the formula. The result we are looking for is the following:

**Proposition 1.** Let \( F = \{F_1, \ldots, F_n\} \) and let \( F_i = \{F_{i1}, \ldots, F_{it_i}\}, i \in \{1, \ldots, n\} \), be sets of events such that \( F_i = \bigcup_{j=1}^{t_i} F_{ij} \). Let \( R = \bigcap_{i=1}^{n} F_i \), \( m = \sum_{i=1}^{n} t_i \) and, given \( k \in \{n, n+1, \ldots, m\} \), let

\[
C_k = \{E = \{E_1, \ldots, E_k\} : E_u = F_{i(u)j(u)} \text{ for some } i(u) \in \{1, \ldots, n\}, j(u) \in \{1, \ldots, t_{i(u)}\}, u \in \{1, \ldots, k\},
\{i(1), i(2), \ldots, i(k)\} = \{1, \ldots, n\} \text{ and } E_p \neq E_q \text{ for } p, q \in \{1, \ldots, k\} \text{ and } p \neq q \}.
\]
That is, \( C_k \) is the family of sets \( E = \{E_1, \ldots, E_k\} \) of implementations not failing where every function \( i, i \in \{1, \ldots, n\} \), is implemented at least once. Then

\[
P(R) = \sum_{k=n}^{m} (-1)^{k-n} \sum_{E \in C_k} P \left( \bigcap_{j=1}^{k} E_j \right).
\] (5)

The proof for Proposition 1 can be found in Appendix A. A generating algorithm for sets \( C_k, k \in \{n, n+1, \ldots, m\} \) is given in Appendix C.

Table 1 shows the total number of summands that we obtain when we use the result stated in Proposition 1 for different numbers of functions and implementations. We assume that every function has the same number of implementations. In this case, the number of summands can be calculated by summing over \( k \in \{n, \ldots, m\} \) the cardinalities \( |C_k| \). We also have that

| \( |\mathcal{F}| \) | \( |\mathcal{F}_i| \) | Summands |
|---|---|---|
| 2 | 2 | 15 |
| 2 | 3 | 5.11 \times 10^2 |
| 2 | 4 | 6.55 \times 10^4 |
| 3 | 2 | 2.55 \times 10^2 |
| 3 | 3 | 1.34 \times 10^8 |
| 3 | 4 | 1.84 \times 10^{17} |
| 4 | 2 | 6.55 \times 10^4 |
| 4 | 3 | 2.41 \times 10^{24} |
| 5 | 2 | 4.29 \times 10^9 |
| 5 | 3 | 1.41 \times 10^{73} |

Table 1: Number of summands in the probability principle of inclusion-exclusion formula.

As can be seen, even for a small number of functions and implementations, the calculation of \( P(R) \) becomes very expensive. However, note that there are many (a priori different) terms that, when the intersection of the sets is calculated, lead to the same intersection set, that is,
\[
\sum_{k=n}^{m} |C_k| = \prod_{i=1}^{n} \left(2^{|F_i|} - 1\right),
\]

which is proven in Lemma 3. Therefore, the expression given by \[5\] has an exponential computational complexity with a linear exponent.

Table 2: Number of summands in formula \[5\].

| | \(\frac{|F|}{2}\) | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 5 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Summands | 9 | 49 | 225 | 27 | 343 | 3375 | 81 | 2401 | 243 | 16807 | 16807 |

When comparing Tables 1 and 2, we can see an enormous reduction in the number of terms involved to calculate the same value of \(P(R)\).

3 Computational Tests

In this section, we run computational tests on simple artificial system examples and a door management system application from aircraft architecture engineering. For our computational tests, we compare the number of summands and the computational time of the sum of disjoint product method KDH88 from \[7\], with the classic probability principle of inclusion-exclusion \[3\] and our proposed method. We are using KDH88 by applying it on \[\text{(2)}\]. All methods are implemented in Python 2.7 and run on an Intel Core i7-6600U with CPU 2.60GHz and 32GB memory.

3.1 Simple Artificial System

First we compare the methods for some examples of a simple artificial system. We set all \(t_i = t\), and assume that each implementation \(j\) of \(i\), \(i \in \{1, \ldots, n\}\) and \(j \in \{1, \ldots, t\}\), corresponds to an elementary component \(a_{ij}\) which has a behaviour independent from all other components and has the common elementary reliability \(r\). We then have a system of \(t \times n\) components. We run computational tests for \(n \in \{2, \ldots, 6\}\), \(t \in \{2, \ldots, 4\}\) and different \(r\). Because all components are independent, we know that the reliability \(R\) can be calculated with \(R(t, n, r) = ((1 - (1 - r)^t)^n)\). To check the validity of the methods and their implementation, we compared the calculated reliability against \(R(t, n, r)\). Table 3 shows the number of summands and computational time for all three methods.

| \(n\) | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 6 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|
| t | 9 | 49 | 225 | 27 | 343 | 3375 | 81 | 2401 | 243 | 16807 |
| Summands | 0.00018 | 0.00071 | 0.00266 | 0.00052 | 0.000672 | 0.000952 | 0.000120 | 0.0002959 | 0.0002959 | 1.65340 |
| Time (s) | 0.00023 | 0.00121 | 0.00269 | 0.00075 | 0.000933 | 0.001025 | 0.000199 | 0.0002959 | 0.0002959 | 1.65340 |

Table 3: Computational results for examples of simple artificial systems.

Table 3 shows that the number of summands for our method is far greater compared to KDH88, but
our method is computationally more efficient. This is because it is much more computationally expensive to calculate the few disjoint products/summands with KDH88 than all summands that are needed in our method. Furthermore the summands that are created by KDH88 are not simple products like the summands in our method. However, both methods are far more efficient and have fewer summands than the classic probability principle of inclusion-exclusion.

3.2 Door Management System

Our next computational test is based on an aircraft architecture system, in which there are many different types of safety-critical systems such as networked aircraft systems, fire warning systems or stall recovery systems. In this example we consider the door management system (DMS) of an aircraft which falls into the category of networked aircraft systems. DMS is a safety-critical system which checks the status of doors, regulates the locks and relays information to on-board computers and pressurization regulators. See Figure 1 for an example of a DMS at one door.

Figure 1: Door management system for one door.

Figure 1 shows only a part of a DMS in an aircraft. The DMS is responsible for all outside doors of the aircraft, as can be seen in Figure 2.

Figure 2: Door management system for multiple doors.

As a DMS is a safety-critical system which cannot be repaired during use, it must meet some reliability threshold. As the DMS is responsible for all outside doors of the aircraft and also allows for the possibility that one component (e.g., switches, actuators, etc.) can be used by different doors, it is a complex network system for which the calculation of reliability is computationally very expensive as shown in Section 2. However, the computational effort required in calculating the reliability of the DMS is reduced considerably by using Proposition 1.

The functionality of the DMS for each door can be seen as a function of the systems. Hence, suppose the aircraft has \( n \) doors and consider the event set \( \mathcal{F} = \{F_1, \ldots, F_n\} \) where \( F_i \) is the event “The functionality
of the DMS for door $i$ in the system does not fail”. Let an implementation of the functionality of a door be a set of components and the corresponding connections which can check the status of a door, regulate the locks and relay information to on-board computers and pressurization regulators. Moreover, suppose that no subset of these components and connections can be removed without losing functionality. Because the DMS is a safety-critical system that has to be redundant, there are at least two implementations for every door. Let $\mathcal{F}_i = \{F_{i1}, \ldots, F_{it}\}$ and $t_i \geq 2$, $i \in \{1, \ldots, n\}$, be event sets where $F_{ij}$ is the event “The functionality of implementation $j$ of door $i$ does not fail”. Lastly, let $R$ be the event “The DMS system fails for no door”. With these event sets, we can calculate the reliability $P(R)$ of the DMS with the formula from Proposition 4

$$P(R) = \sum_{k=n}^{m} \left( -1 \right)^{k-n} \sum_{E \in \mathcal{C}_k} P \left( \bigcap_{j=1}^{k} E_j \right) .$$

(7)

The calculation of $P(\bigcap_{j=1}^{k} E_j)$ for $E = \{E_1, \ldots, E_k\} \in \mathcal{C}_k$, $k \in \{n, n+1, \ldots, m\}$ is simple. Let $T_c$ be the event that component $c$ of the DMS system does not fail. Since we are considering a static system, we know the probability $P(T_c) = a_c$ with $a_c \in (0, 1)$. Let $\mathcal{T}_{F_{ij}}$ be the set of components of implementation $j$ for door $i$. Since we assume that all components have independent failures for all $i \in \{1, \ldots, n\}$ and for all $j \in \{1, \ldots, t_i\}$, it follows that

$$P (F_{ij}) = P \left( \bigcap_{c \in \mathcal{T}_{F_{ij}}} T_c \right) = \prod_{c \in \mathcal{T}_{F_{ij}}} P(T_c) .$$

Therefore, for all $k \in \{n, n+1, \ldots, m\}$ and for all $E \in \mathcal{C}_k$, we have that

$$P \left( \bigcap_{j=1}^{k} E_j \right) = P \left( \bigcap_{j=1}^{k} \left( \bigcap_{c \in \mathcal{T}_{E_j}} T_c \right) \right) = P \left( \bigcap_{c \in \bigcup_{j=1}^{k} \mathcal{T}_{E_j}} T_c \right) = \prod_{c \in \bigcup_{j=1}^{k} \mathcal{T}_{E_j}} P(T_c) .$$

If $z$ is the number of components of the system, we know that each summand of (7) is a product of at most $z$ factors.

We ran computational tests for different numbers of doors $n$, implementations $t$ and number of components/connections. We tested them for our proposed method and the KDH88 method, but we did not run them for the classic probability principle of inclusion-exclusion. For every combination of $n$ and $t$, we ran 100 different systems. All systems have different number of components and different reliabilities. An example system can be found in Appendix E as well as a link to the data of all other DMS systems. Table 4 shows the minimum, maximum and median number of components/connections, the minimum, maximum and median number of summands and minimum, maximum and median execution times for both methods and every combination of $n$ and $t$.

| $(n,t)$ | Components | Proposed | KDH88 |
|--------|------------|----------|--------|
|       | Summands   | Time(s)  | Summands | Time(s)  |
|       | Min     | Max    | Med    | Min  | Max    | Med  | Min | Max | Med |
| (2,2) | 42 65 56.0 | 9 0.00017 | 0.000037 | 0.00020 | 4 5 | 5 | 0.00028 | 0.00023 | 0.00036 |
| (2,3) | 57 81 72.0 | 49 0.00087 | 0.00128 | 0.00103 | 13 20 | 17 | 0.00252 | 0.00421 | 0.00332 |
| (3,2) | 57 76 69.0 | 27 0.00056 | 0.00070 | 0.00061 | 7 13 | 11 | 0.00102 | 0.00215 | 0.00174 |
| (3,3) | 68 96 84.0 | 81 0.00662 | 0.00857 | 0.00760 | 34 87 | 59 | 0.02927 | 0.09249 | 0.06568 |
| (4,2) | 76 101 90.0 | 81 0.00187 | 0.00231 | 0.00208 | 10 28 | 21 | 0.00744 | 0.01381 | 0.01074 |
| (4,3) | 83 119 102.5 | 2401 0.05573 | 0.07409 | 0.06377 | 81 339 | 187 | 0.42229 | 3.23434 | 1.88462 |
| (5,2) | 88 128 111.0 | 243 0.00599 | 0.00807 | 0.00704 | 19 67 | 45 | 0.02448 | 0.11609 | 0.08271 |
| (5,3) | 96 146 125.0 | 16807 0.45604 | 0.60943 | 0.54729 | 110 1067 | 619 | 12.21276 | 95.74859 | 59.34907 |

Table 4: Computational results for door management system.

We obtained similar results as in the other computational tests. With our proposed method there is a constant number of summands for the 100 different systems, but this value is much larger than the number of summands when using KDH88. Also, the minimum and maximum number of summands for the different systems when using KDH88 is very different. This shows that the efficiency of KDH88 is highly dependent on the exact structure of the system. Furthermore, we can see that our method is computationally much more efficient, especially for larger systems.
4 Conclusions

In this paper we have introduced a new expression (Proposition 1) which considerably reduces the computational effort needed to calculate the probability principle of inclusion-exclusion when applied to intersections of unions of events. It has been shown that the formula obtained can be applied to the reliability calculation of a certain kind of complex network systems and it decreases the computational time significantly. It has also been shown that the computational complexity is reduced from doubly exponential to exponential with linear exponent. Moreover, we have provided a comparison with the general sum of disjoint products method KDH88 from [7] and showed that our method is more computationally efficient as KDH88 for our examples.

The complex network systems we considered in this paper are systems with multiple functions, which means they have multiple start and end nodes in the system. If a system with only one function is considered, the simplification does not take effect and the number of terms of the probability principle of inclusion-exclusion does not decrease. We also considered that we have multiple implementations for each function. For a system with only one implementation per function, our proposed calculation method does not have any advantages. In all the other cases, that is, multiple functions and multiple implementations per function, the expression proposed in this paper can be applied very effectively.

Another limitation of our method is the same that KDH88 and other SDP methods have. The different paths in the system that represent the implementations for each function and the failure probability of all components in the system have to be known.

It must be noted that the result introduced in this paper does not only give the option to calculate reliability more efficiently. It also allows to formulate optimization problems of complex network systems that include the exact reliability of the system without depending solely on heuristics to solve it.

A potential extension for future research is to generalize Proposition 1 and Lemma 2. For example, the implementation for a function \( i \) can also be used for function \( j \) with \( i \neq j \) which results in \( F_i \cap F_j \neq \emptyset \). This may result in a simplification of the probability principle of inclusion-exclusion with summand coefficients that are not \(-1\) or \(1\), and still give a decrease on the number of summands compared to Proposition 1.

Another potential extension for future research is to see if this simplification can be used with a multi-state system. In this case, the components of a system operate in any of several intermediate states with various effects on the entire system reliability.

Acknowledgements

The research of Sergio García has been funded by Fundación Séneca (project 19320/PI/14). Lukas Schäfer is funded by an EPSRC Industrial CASE studentship (EP/L509420/1) in partnership with Airbus Group.

References

[1] J. Abraham. An improved algorithm for network reliability. *IEEE Transactions on Reliability*, R-28(1):58 – 61, 1979.
[2] A. Balan and L. Traldi. Preprocessing minpaths for sum of disjoint products. *IEEE Transactions on Reliability*, 52(3):289 – 295, 2003.
[3] M. Boyd. *Dynamic fault tree models: Techniques for analysis of advanced fault tolerant computer systems*. PhD thesis, Durham, NC, USA, 1992. UMI Order No. GAX92-02503.
[4] D. Coit and A. Smith. Solving the redundancy allocation problem using a combined neural network/genetic algorithm approach. *Computers and Operations Research*, 23(6):515 – 526, 1996.
[5] E. Datta and N. K. Goyal. Sum of disjoint product approach for reliability evaluation of stochastic flow networks. *International Journal of System Assurance Engineering and Management*, 8(2):1734–1749, 2017.
[6] W. Dotson and J. Gobien. A new analysis technique for probabilistic graphs. *IEEE Transactions on Circuits and Systems*, 26(10):855 – 865, 1979.
[7] K. Heidtmann. Smaller sums of disjoint products by sub product inversion. *IEEE Transactions on Reliability*, 38:305 – 311, 1989.
[8] C. Hwang and F. Tiliman. System-reliability evaluation techniques for complex/large systems-A review. *IEEE Transactions on Reliability*, 30(5):416 – 423, 1981.
[9] W. Kuo, V. Prasad, F. Tillman, and C. Hawang. Optimal reliability design fundamental and application. Cambridge University Press, 2001.

[10] G. Levitin. Block diagram method for analyzing multi-state systems with uncovered failures. Reliability Engineering and System Safety, 92(6):727 – 734, 2007.

[11] X. Li, S. Zhou, X. Xu, L. Lin, and D. Wang. The reliability analysis based on subsystems of $(n,k)$-star graph. IEEE Transactions on Reliability, 65(4):1700–1709, 2016.

[12] Y. Li, H. Huang, Y. Liu, N. Xiao, and H. Li. A new fault tree analysis method: Fuzzy dynamic fault tree analysis. Eksploatacja i Niezawodnosć-Maintenance and Reliability, 14(3):208 – 214, 2012.

[13] L. Lin, L. Xu, S. Zhou, and D. Wang. The reliability of subgraphs in the arrangement graph. IEEE Transactions on Reliability, 64(2):807–818, 2015.

[14] Y.-K. Lin and C.-F. Huang. Reliability of a multi-state computer network through $k$ minimal paths within tolerable error rate and time threshold. Quality and Reliability Engineering International, 32:1393–1405, 2016.

[15] C. Liu, N. Chen, and J. Yang. New method for multi-state system reliability analysis based on linear algebraic representation. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 229(5):469–482, 2015.

[16] Y. Mo, Y. Liu, and L. Cui. Performability analysis of multi-state series-parallel systems with heterogeneous components. Reliability Engineering and System Safety, 2017.

[17] Y.-F. Niu, Z.-Y. Gao, and W. H. Lam. A new efficient algorithm for finding all $d$-minimal cuts in multi-state networks. Reliability Engineering and System Safety, 166(Supplement C):151 – 163, 2017.

[18] S. Rai, M. Veeraraghavan, and K. Trivedi. A survey of efficient reliability computation using disjoint products approach. Networks, 25(3):147 – 163, 1995.

[19] M. Ram and J. P. Davim. Advances in Reliability and System Engineering. Springer, 2017.

[20] J. E. Ramirez-Marquez and D. W. Coit. A Monte-Carlo simulation approach for approximating multi-state two-terminal reliability. Reliability Engineering and System Safety, 87(2):253 – 264, 2005.

[21] M. Rausand. Reliability of safety-critical systems: Theory and applications. Wiley-IEEE, New York, 3rd edition, 2014.

[22] A. Rauzy, E. Chtelet, Y. Dutuit, and C. Brenguer. A practical comparison of methods to assess sum-of-products. Reliability Engineering and System Safety, 79(1):33 – 42, 2003.

[23] S. Singh, A. Verma, S. Chatterjee, and V. Ramana. An efficient methodology to solve the $k$-terminal network reliability problem. In 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), pages 25 – 28, 2016.

[24] S. Twum and E. Aspinwall. Models in design for reliability optimisation. American Journal of Scientific and Industrial Research, 4:95 – 110, 2013.

[25] N.-C. Xiao, L. Duan, and Z. Tang. Surrogate-model-based reliability method for structural systems with dependent truncated random variables. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 231(3):265–274, 2017.

[26] J. Xing, C. Feng, X. Qian, and P. Dai. A simple algorithm for sum of disjoint products. In 2012 Proceedings Annual Reliability and Maintainability Symposium, pages 1 – 5, 2012.

[27] W.-C. Yeh. An improved sum-of-disjoint-products technique for the symbolic network reliability analysis with known minimal paths. Reliability Engineering and System Safety, 92(2):260 – 268, 2007.

[28] W.-C. Yeh. An improved sum-of-disjoint-products technique for symbolic multi-state flow network reliability. IEEE Transactions on Reliability, 64(4):1185 – 1193, 2015.

[29] J. Yuan and K. Ko. A factoring method to calculate reliability for systems of dependent components. Reliability Engineering and System Safety, 21(2):107 – 118, 1988.

[30] D. Zhong and Z. Qi. On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops: OTM Confederated International Workshops and Posters, AWsOME, CAMS, COMINF, IS, KSiNBIT, MIOS-CIAO, MONET, OnToContent, ORM, PerSys, OTM Academy Doctoral Consortium, RDDS, SWIS, and ScBGIS 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I, chapter A Petri Net Based Approach for Reliability Prediction of Web Services, pages 116 – 125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
A Proof of Proposition 1

In this section, we prove Proposition 1 after stating and proving an auxiliary result (Lemma 2). As we mentioned earlier (see (1)), there are different subsets of $W$ in (3) that lead to the same intersection set and, therefore, the same probability. The first result of the following lemma enables us to count how many different subsets of $W$ with the same cardinality $t \in \{1, \ldots, |W|\}$ lead to the same intersection set. The second result of Lemma 2 gives us the coefficient of an intersection set $E$ in formula (5).

Lemma 2.

Let $t$, $n \in \mathbb{N}^+$ and let $A_1, \ldots, A_n$ be non-empty sets with $A_i \cap A_j = \emptyset \forall i \neq j$, $A = \bigcup_{i=1}^n A_i$, $k = |A|$ and $D = A_1 \times \ldots \times A_n$. Furthermore, let us define $s(e) = \{e_1, \ldots, e_n\}$ for $e = (e_1, \ldots, e_n) \in D$ and, given $I \subseteq A$ and $A = (A_1, \ldots, A_n)$, let $p(I, A) = \prod_{i=1}^n |A_i \cap I| = |(A_1 \cap I) \times \ldots \times (A_n \cap I)|$.

Let $c(A, t)$ be the total number of non-empty subsets $\{e^1, \ldots, e^t\} \subseteq D$ of cardinality $t$ such that $\bigcup_{i=1}^t s(e^i) = A$. Then

1. $c(A, t) = \sum_{i=0}^{k-n} (-1)^i \sum_{I \subseteq A} \binom{|p(I, A)|}{t}$, \hfill (8)

2. $\sum_{t=1}^{|D|} (-1)^{t-1} c(A, t) = (-1)^{k-n}$. \hfill (9)

Proof.

1. First we prove (8) by induction over $k \geq n$. We use $\binom{i}{j} = 0$ if $j > i$ several times throughout the proof. The first time it is needed is to see that for all $t$ with $t > |D|$, we have that $c(A, t) = 0$ because there exist no subsets of $D$ with cardinality strictly greater than $|D|$. Furthermore, if $t > |D|$ and because $p(I, A) \leq |D|$, the right-hand side of (8) is 0. Therefore, (8) holds for $t > |D|$. In the following, we assume that $t \leq |D|$.

If $k = n$, then $|A_1| = \ldots = |A_n| = 1$ and $|D| = 1$ and we can assume $t = 1$. Thus

$$\sum_{i=0}^{k-n} (-1)^i \sum_{I \subseteq A} \binom{|p(I, A)|}{1} = (-1)^0 \binom{1}{1} = 1.$$ 

Moreover, $|D| = 1$ means that there exists exactly one vector $e \in D$ and $s(e) = A$. Therefore, $c(A, 1) = 1$ and the formula is correct for $k = n$.

Let us assume that (8) holds for $n, k, t \in \mathbb{N}^+ \text{ with } k \geq n$. We will show that it also holds for $k+1$. For this, let $A_1, \ldots, A_n$ be non-empty sets with $A_i \cap A_j = \emptyset$ for all $i \neq j$, $A = \bigcup_{i=1}^n A_i$ with $|A| = k+1$ and $A = (A_1, \ldots, A_n)$. We can write the number of non-empty subsets $\{e^1, \ldots, e^t\} \subseteq D$ of cardinality $t$ from $D$ with $\bigcup_{i=1}^t s(e^i) = A$ as the number of subsets $\{e^1, \ldots, e^t\} \subseteq D$ of cardinality $t$ minus the number of subsets $\{e^1, \ldots, e^t\} \subseteq D$ of cardinality $t$ minus the number of subsets $\{e^1, \ldots, e^t\} \subseteq D$ of cardinality $t$ for $s(e) = A$, $J \subseteq A \text{ and } |J| \leq k+1 - n$ (because $|A\backslash J| \geq n$) and $p(A \backslash J, A) = \prod_{i=1}^n |(A_i \cap \{A_i \backslash J\})| = \prod_{i=1}^n |(A_i \backslash J)| \neq 0$. Since for these sets $\{A_i \backslash J\}$ we have that $|A_i \backslash J| \leq k$, we can use the induction hypothesis to obtain that the number of such sets, for each $J$, is $c(A \backslash J, t)$, where we define $A \backslash J = (A_1 \backslash J, \ldots, A_n \backslash J)$. Therefore, using that $|D| = p(A, A)$ we have that

$$c(A, t) = \binom{|D|}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{J \subseteq A, |J| = j, p(A \backslash J, A) \neq 0} c(A \backslash J, t) \right)$$

$$= \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{J \subseteq A, |J| = j, p(A \backslash J, A) \neq 0} \binom{k+1-j-n}{t} \sum_{i=0}^{k+1-j-1} (-1)^i \sum_{|I| = k+1-j-i} \binom{|p(I, A \backslash J)|}{t} \right).$$

We can drop the condition $p(A \backslash J, A) \neq 0$, because if $p(A \backslash J, A) = 0$ then there exists $i \in \{1, \ldots, n\}$ such that $A_i \backslash J = \emptyset$ and therefore we have for all $I \subseteq A \backslash J$ that $p(I, A \backslash J) = 0$. Thus, by dropping the condition only zeros are added to the sum. Furthermore based on the definition of function $p$, we know that for all $I, J \subseteq A$:

$$p(I, A \backslash J) = \prod_{i=1}^n |(A_i \backslash J) \cap I| = \prod_{i=1}^n |A_i \cap (I \backslash J)| = p(I, J, A).$$

10
Therefore
\[
c(A, t) = \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{J \subseteq A, |J|=j} \binom{k+1-j-n}{t} \left( \sum_{i=0}^{k+1-j-n} (-1)^{i} \binom{p(I \setminus J, A)}{t} \right) \right)
\]
\[
= \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{J \subseteq A, |J|=j} \binom{k+1-j-n}{t} \left( \sum_{i=0}^{k+1-j-n} (-1)^{i} \binom{p(I \setminus J, A)}{t} \right) \right)
\]
\[
= \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{J \subseteq A, |J|=j} \binom{k+1-j-n}{t} \left( \sum_{i=0}^{k+1-j-n} (-1)^{i} \binom{p(I \setminus J, A)}{t} \right) \right). \tag{10}
\]

Next, we define \(A[\ell] = \{ I \subseteq A : |I| = \ell \}, \ell \in \mathbb{N}^+, \) and we have that
\[
\forall \ell \in \{1, \ldots, |A| - 1\} \forall L \in A[\ell] \forall j \in \{1, \ldots, |A| - \ell\}:
\]
\[
|\{ (I, J) \in A[\ell + j] \times A[j] : J \subseteq I, I \setminus J = L \}| = |\{ (I, J) \in A[\ell + j] \times A[j] : L \cap J = \emptyset, I = L \cup J \}| = |\{ J \in A[j] : J \subseteq A \setminus L \}|
\]
\[
= \binom{|A| - \ell}{j}. \tag{11}
\]

The last step is known from the general result for unordered sampling without replacement in combinatorics. \[\text{(11)}\] can now be used to rewrite \[\text{(10)}\] by summing over sets \(L = I \setminus J\) with coefficients \((|A| - \ell)\) instead of summing over \(J\) and \(I\) separately. We can write now that
\[
\text{(10)} = \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{\ell=0}^{k+1-j-n} (-1)^{\ell} \sum_{L \subseteq A, |L| = \ell + j - \ell} \binom{|A| - (k + 1 - j - \ell)}{j} \binom{p(L, A)}{t} \right)
\]
\[
= \binom{p(A, A)}{t} - \sum_{j=1}^{k+1-n} \left( \sum_{\ell=0}^{k+1-j-n} (-1)^{\ell} \sum_{L \subseteq A, |L| = \ell + j - \ell} \binom{j + \ell}{j} \binom{p(L, A)}{t} \right). \tag{12}
\]

Let now \(\ell \in \{1, \ldots, k+1-n\}\) and \(L \in A[k+1-\ell] \). It holds that
\[
\forall j \in \{1, \ldots, \ell\} \exists \ell \in \{0, \ldots, k+1-n-j\} : j + \ell = \ell
\]
and that \(L\) appears exactly once for all possible combinations \(j + \ell = \ell\) with the coefficients \((-1)^{\ell-j} \binom{\ell}{j}\) \((-1)^{\ell+j} \binom{\ell}{j}\) in \[\text{(12)}\]. Therefore, we can take the sum over the sets \(L \in A[k+1-\ell]\) for \(\ell \in \{1, \ldots, k+1-n\}\) and it holds that
\[
\text{(12)} = \binom{p(A, A)}{t} - \sum_{\ell=1}^{k+1-n} \left( \sum_{L \subseteq A, |L| = \ell} \binom{\ell}{j} \left( \sum_{j=1}^{\ell} (-1)^{\ell-j} \binom{\ell}{j} \binom{p(L, A)}{t} \right) \right)
\]
\[
= \binom{p(A, A)}{t} - \sum_{\ell=1}^{k+1-n} \left( \sum_{L \subseteq A, |L| \leq \ell} (-1)^{\ell} \left( \sum_{j=1}^{\ell} (-1)^{j} \binom{\ell}{j} \binom{p(L, A)}{t} \right) \right). \tag{13}
\]

Finally, by using the following known result from combinatorics
\[
\sum_{i=0}^{n} (-1)^{i} \binom{n}{i} = \sum_{i=1}^{n} (-1)^{i} \binom{n}{i} + 1 = 0, \tag{14}
\]
we have that

\[ (13) = \binom{p(A, A)}{t} - \sum_{\ell=1}^{k+1-n} \left( \sum_{L \subseteq A, |L|=k+1-\ell} (-1)^\ell \binom{p(L, A)}{t} \right) \]

\[ = \binom{p(A, A)}{t} + \sum_{\ell=1}^{k+1-n} \left( \sum_{L \subseteq A, |L|=k+1-\ell} (-1)^\ell \binom{p(L, A)}{t} \right) \]

\[ = \sum_{\ell=0}^{k+1-n} \left( \sum_{L \subseteq A, |L|=k+1-\ell} (-1)^\ell \binom{p(L, A)}{t} \right). \]

This completes the proof for (8).

2.) Now we will prove the second result, (9), for \( n \geq 1 \) and \( k \geq 1 \) with \( n \leq k \) by induction over \( m = k - n \). Let \( n \geq 1, k \geq 1 \) and let \( A_1, \ldots, A_n \) be non-empty sets with \( A_i \cap A_j = \emptyset \) for all \( i \neq j \), \( A = \bigcup_{i=1}^n A_i \) with \( |A| = k \), \( A = (A_1, \ldots, A_n) \) and \( D = A_1 \times \ldots \times A_n \). We can rewrite (9) by using (8) as follows:

\[
\sum_{t=1}^{|D|} (-1)^{t-1} c(A, t) = \sum_{t=1}^{p(A, A)} \left( (-1)^{t-1} \sum_{i=0}^{k-n} \left( \sum_{L \subseteq A, |L|=k-i} (-1)^i \binom{p(L, A)}{t} \right) \right) 
\]

\[
= \sum_{i=0}^{k-n} \left( (-1)^i \sum_{L \subseteq A, p(L, A) \neq 0, |L|=k-i} \binom{p(L, A)}{t} \right). \quad (15)
\]

Next we use (14) again and it holds that

\[
(15) = \sum_{i=0}^{k-n} (-1)^i \left( \sum_{L \subseteq A, p(L, A) \neq 0, |L|=k-i} \binom{p(L, A)}{t} \right) (-1)^i \left( \sum_{t=0}^{p(L, A)} (-1)^i \binom{p(L, A)}{t} - 1 \right) 
\]

\[
= \sum_{i=0}^{k-n} (-1)^i \left( \sum_{L \subseteq A, p(L, A) \neq 0, |L|=k-i} (-1)^2 \right) 
\]

\[
= \sum_{i=0}^{k-n} (-1)^i \left( \sum_{L \subseteq A, p(L, A) \neq 0, |L|=k-i} 1 \right) 
\]

If we now modify the external sum on the previous expression to start with \( i = n \), it follows that

\[
\sum_{t=1}^{|D|} (-1)^{t-1} c(A, t) = (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{L \subseteq A, p(L, A) \neq 0, |L|=k-i} 1 \right). \quad (16)
\]

If \( m = 0 \) and therefore \( n = k \), then
\[
\sum_{t=1}^{[D]} (-1)^{t-1} c(A, t) = (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A, |I|=i, \delta \not\in I} 1 \right) = (-1)^n \cdot (-1)^n \cdot 1 = (-1)^{k-n}.
\]

Therefore, equation [9] holds for \( m = 0 \).

We assume now that it holds for \( n \geq 1 \) and \( k \geq 1 \) with \( m = k - n \) and \( m \geq 0 \). Without loss of generality, we show that it also holds for \( m + 1 \) with \( m + 1 = (k + 1) - n \) by fixing \( n \). For this, let \( A_1, \ldots, A_n \) be non-empty sets with \( \delta \not\in I \) and \( A = \bigcup_{i=1}^{n} A_i \) with \( |A| = \sum_{i=1}^{n} |A_i| = k \). Furthermore, let \( A_1^*, A_2^*, \ldots, A_n^* \) be non-empty sets with \( A^* = \bigcup_{i=1}^{n} A_i^* \) and \( |A^*| = k + 1 \). Without loss of generality, let \( A_2 = A_2^*, \ldots, A_n = A_n^* \) and \( A_1 = A_1^* \setminus \{\delta\} \) with \( \delta \not\in \bigcup_{i=2}^{n} A_i \). Also, let \( \bar{A}_1, \ldots, \bar{A}_{n-1} \) be non-empty sets with \( \bar{A}_1 = A_2, \ldots, \bar{A}_{n-1} = A_n \) and \( \bar{A} = \bigcup_{i=1}^{n-1} \bar{A}_i \). By using [16], it holds that

\[
p(A^*, A^*) = \sum_{t=1}^{[D]} (-1)^{t-1} c(A^*, t)
= (-1)^{k+1} \sum_{i=n}^{k+1} (-1)^i \left( \sum_{I \subseteq A^*, \ p(t,A^*) \not= 0, \ |I|=i} 1 \right)
= (-1) \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(t,A^*) \not= 0, \ |I|=i} 1 \right) \right).
\]

The sum over the subsets \( I \) can be split by considering whether the subsets contain \( \delta \) or not:

\[
= (-1) \left( \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^* \setminus \{\delta\}, \ p(t,A^* \setminus \{\delta\}) \not= 0, \ |I|=i} 1 \right) \right) + \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^* \setminus \{\delta\}, \ p(t,A^* \setminus \{\delta\}) \not= 0, \ |I|=i} 1 \right) \right) \right).
\]

Using that \( A^* \setminus \{\delta\} = A \) and that no subset of \( A \) can be of cardinality \( k + 1 \), we can rewrite the first sum:

\[
= (-1) \left( \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A, \ p(t,A) \not= 0, \ |I|=i} 1 \right) \right) + \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A, \ p(t,A) \not= 0, \ |I|=i} 1 \right) \right) \right).
\]

where we use [10] to obtain this last inequality. Finally, by using the induction hypothesis for \( m = k - n \):

\[
= (-1) \left( \left( (-1)^{k-n} + \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(t,A^*) \not= 0, \ |I|=i} 1 \right) \right) \right) \right) + \left( (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(t,A^*) \not= 0, \ |I|=i} 1 \right) \right).
\]

To prove that [9] holds for \( m + 1 \), we only have to show that

\[
(-1)^k \sum_{i=n}^{k+1} (-1)^i \left( \sum_{I \subseteq A^*, \ p(t,A^*) \not= 0, \ |I|=i} 1 \right) = 0. \tag{17}
\]
First, we rewrite the left-hand side of (17) as follows:

\[
(-1)^k \sum_{i=n}^{k+1} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right)
= (-1)^k \sum_{i=n}^{k+1} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right)
= (-1)^k \sum_{i=n-1}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right).
\] (18)

The following observations are needed to rewrite (18) further.

1. First note that \( A^* \setminus A^*_1 = A \), and that for all \( I \subseteq \hat{A} \):

\[
p(I \cup \{\delta\}, A^*) = \prod_{i=1}^{n} |A^*_i \cap (I \cup \{\delta\})| = |\{\delta\}| \prod_{i=2}^{n} |A^*_i \cap I| = \prod_{i=1}^{n-1} |\hat{A}_i \cap I| = p(I, \hat{A}).
\] (19)

2. In addition, \( A^* \setminus \{\delta\} = A \) and for all \( I \subseteq A \) it holds that

\[
p(I \cup \{\delta\}, A^*) = \prod_{i=1}^{n} |A^*_i \cap (I \cup \{\delta\})| = (|A_1 \cap I| + 1) \prod_{i=2}^{n} |A^*_i \cap I|.
\]

Therefore,

\[
p(I \cup \{\delta\}, A^*) \neq 0 \Leftrightarrow \prod_{i=2}^{n} |A^*_i \cap I| = \prod_{i=1}^{n-1} |\hat{A}_i \cap I| = p(I, \hat{A}) \neq 0.
\]

3. Moreover, it holds that

\[
\forall I \subseteq A : p(I, \hat{A}) \neq 0 \land |I \cap A_1| \neq 0 \Leftrightarrow p(I, \hat{A}) \cdot |I \cap A_1| = p(I, A) \neq 0.
\] (20)

Now we can rewrite (18) by splitting the sum over subsets \( I \) by considering whether or not the subset is disjoint with \( A_1 \).

\[
(-1)^k \sum_{i=n-1}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right)
= (-1)^k \sum_{i=n-1}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right) + (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A^*, \ p(I,A^*) \neq 0, \ |I|=i} 1 \right).
\] (21)

By using (19), we can rewrite the sum over \( I \subseteq A^* \setminus A^*_1 = \hat{A} \) and, by using (20), the sum over \( I \subseteq A^* \setminus \{\delta\} = A \). Furthermore, we can change the upper limit of the first sum in (21) to \( |\hat{A}| = k - |A_1| \). Therefore, we can write that (21) is

\[
= (-1)^k \sum_{i=|A_1|}^{k-|A_1|} (-1)^i \left( \sum_{I \subseteq A, \ p(I,A) \neq 0, \ |I|=i} 1 \right) + (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A, \ p(I,A) \neq 0, \ |I|=i} 1 \right)
= (-1)^{|A_1|} (-1)^{k-|A_1|} \sum_{i=n-1}^{k-|A_1|} (-1)^i \left( \sum_{I \subseteq \hat{A}, \ p(I,\hat{A}) \neq 0, \ |I|=i} 1 \right) + (-1)^k \sum_{i=n}^{k} (-1)^i \left( \sum_{I \subseteq A, \ p(I,A) \neq 0, \ |I|=i} 1 \right).
\] (22)
And because \((k - |A_1|) - (n - 1) < m + 1\) and \(k - n < m + 1\), we can use the induction hypothesis on both sums and we use (16) to obtain that

\[
22 = \left( (-1)^{|A_1|} (-1)^{k-|A_1|-(n-1)} + (-1)^{k-n} \right) \\
= \left( (-1)^{k-(n-1)} + (-1)^{k-n} \right) = 0.
\]

Hence, we have proved that

\[
\sum_{t=1}^{|D|} (-1)^{t-1}c(A, t) = (-1)^{k-n}
\]
holds for any \(n \geq 1\) and \(k \geq 1\) with \(k - n \geq 0\).

We can now prove our main result Proposition 1.

Proof of Proposition 1.

Since \(R = \bigcap_{F \in \mathcal{F}} F\), it follows that

\[
P(R) = P\left( \bigcap_{i=1}^n F_i \right) = P\left( \bigcap_{i=1}^n \left( \bigcup_{j=1}^{t_i} F_{ij} \right) \right),
\]

(23)

Let

\[
W = \{1, \ldots, t_1\} \times \ldots \times \{1, \ldots, t_n\},
\]

\[
B_w = \bigcap_{i=1}^n F_{iw_i} \quad \text{and} \quad \mathfrak{B}_w = \{F_{1w_1}, \ldots, F_{nw_n}\} \quad \text{for} \quad w = (w_1, \ldots, w_n) \in W.
\]

We can rewrite (23) as

\[
P(R) = P\left( \bigcup_{w \in W} \left( \bigcap_{i=1}^n F_{iw_i} \right) \right) = P\left( \bigcup_{w \in W} B_w \right).
\]

Using the probability principle of inclusion-exclusion (1), it holds that

\[
P(R) = \sum_{t=1}^{|W|} \left( (-1)^t + \sum_{I \subseteq W, |I| = t} P \left( \bigcap_{j \in I} B_j \right) \right).
\]

(24)

Based on the definition of \(B_w, w \in W\), we know that

\[
\forall I \subseteq W \exists k \in \{n, n+1, \ldots, m\} \quad \text{and} \quad \exists \mathbf{E} = \{E_1, \ldots, E_k\} \in C_k : \bigcap_{j \in I} B_j = \bigcap_{i=1}^k E_i.
\]

(25)

We define for all \(k \in \{n, n+1, \ldots, m\}\) and \(\mathbf{E} \in C_k \) : \(D_{\mathbf{E}} = \{B_w : \mathfrak{B}_w \cap \mathbf{E} = \mathfrak{B}_w \quad \text{and} \quad w \in W\}\).

Furthermore, for all \(k \in \{n, \ldots, m\}\), \(\mathbf{E} \in C_k \) and \(\ell \in \{1, \ldots, |D_{\mathbf{E}}|\}\) let us define
With loss of generality, we show it also holds for $\ell$. Proof. We prove (27) by induction. Let $C_k = \{ F \in C_k : |F| = k \}$ and $T(E, \ell) = \{ I \subseteq W : |I| = \ell \text{ and } \bigcup_{i \in I} \mathcal{M}_i = E \}$, and $t(E, \ell) = |T(E, \ell)|$.

If we use (25), we can rewrite (24) to a sum over $E \in C_k$, $k = n, \ldots, m$, where the coefficients are $\sum_{i=1}^{|W|} (-1)^{i+1} t(E, i)$. Furthermore, we can rewrite the coefficient to $\sum_{i=1}^{|D_E|} (-1)^{i+1} t(E, i)$, because $t(E, \ell)$ is zero for $E \in C_k, k = n, \ldots, m$ and $\ell \geq 1$ if $|D_E| < \ell$. Hence, we have that (24) is

$$
T(E, \ell) = \sum_{k=n}^m \left( \sum_{E \in C_k} \left( \sum_{i=1}^{|D_E|} (-1)^{i+1} t(E, i) \right) P \left( \bigcap_{j=1}^k E_j \right) \right)
$$

and using (9) from Lemma 2, it holds that

$$
= \sum_{k=n}^m \left( \sum_{E \in C_k} \left( \sum_{i=1}^{|D_E|} (-1)^{i+1} c(E, i) \right) P \left( \bigcap_{j=1}^k E_j \right) \right)
$$

In addition, let $E = (F_1 \cap \ldots \cap F_m \cap E)$. Based on the first result (3) from Lemma 2 we know that $t(E, \ell) = c(E, \ell)$ for all $k \in \{n, n+1, \ldots, m\}$, $E \in C_k$ and $\ell \in \{1, \ldots, |D_E|\}$. This gives us that (25) is

$$
= \sum_{k=n}^m \left( \sum_{E \in C_k} \left( \sum_{i=1}^{|D_E|} (-1)^{i+1} c(E, i) \right) P \left( \bigcap_{j=1}^k E_j \right) \right)
$$

and using (9) from Lemma 2 it holds that

$$
= \sum_{k=n}^m \left( \sum_{E \in C_k} (-1)^{k-n} P \left( \bigcap_{j=1}^k E_j \right) \right)
$$

$$
= \sum_{k=n}^m \left( \sum_{E \in C_k} (-1)^{k-n} \sum_{E \in C_k} P \left( \bigcap_{j=1}^k E_j \right) \right)
$$

This completes the proof and we have shown that (5) holds.

\[ \square \]

B  Time Complexity of Main Result

In this section, we will prove the time complexity of the new formula in Proposition 1 via Lemma 3

**Lemma 3.** Let $n \geq 1$, $F_i = \{ F_{i1}, \ldots, F_{it_i} \}, i \in \{1, \ldots, n\}$, be sets of cardinality $t_i$, $t_i \geq 1$ and $F = (F_1, \ldots, F_n)$. Define $m := \sum_{i=1}^n t_i$ and, given $k \in \{n, n+1, \ldots, m\}$,

$C_k^F := \{ E \in \{E_1, \ldots, E_k\} : E_u = F_{(i(u))j(u)} \text{ for some } i(u) \in \{1, \ldots, n\}, j(u) \in \{1, \ldots, t_i(u)\}, u \in \{1, \ldots, k\}, \{i(1), i(2), \ldots, i(k)\} = \{1, \ldots, n\} \text{ and } E_u \neq E_v \text{ for } u \neq v \in \{1, \ldots, k\} \}$

Lastly, define $c(F) = \sum_{k=n}^m |C_k^F|$. It holds that

$$
c(F) = \sum_{k=n}^m |C_k^F| = \prod_{i=1}^n (2^{|F_i|} - 1).
$$

**Proof.** We prove (27) by induction. Let $n \geq 1$ and $t_i \geq 1$, $i \in \{1, \ldots, n\}$. We use induction on $\ell = m - n$. Because all $t_i$, $i \in \{1, \ldots, n\}$, are greater than 1, $m \geq n$ and we can first assume $\ell = 0$ with $m = n$. With $m = n$, we have that $t_1 = \ldots = t_n = 1$. Therefore $C_n = \{ \{F_{i1}, \ldots, F_{in}\} \}$ and $c(F) = \sum_{k=n}^m |C_k^F| = |C_n| = 1$. Also $\prod_{i=1}^n (2^{|F_i|} - 1) = \prod_{i=1}^n 1 = 1$ and (27) holds for $\ell = 0$.

We assume now that (27) holds for $n \geq 1$ and $t_i \geq 1$, $i \in \{1, \ldots, n\}$, with $\ell = m - n$ and $\ell \geq 0$. Without loss of generality, we show it also holds for $\ell + 1$ with $\ell + 1 = (m + 1) - n$ by fixing $n$. Let $n \geq 1$, $F_i = \{ F_{i1}, \ldots, F_{it_i} \}, i \in \{1, \ldots, n\}$, be sets of cardinality $t_i$, $t_i \geq 1$ and $m = \sum_{i=1}^n t_i$. We then know that $c(F) = \prod_{i=1}^n (2^{|F_i|} - 1)$. Without loss of generality, let $F_i = F_{i1}$, $i \in \{1, \ldots, n-1\}$, $F_n^* = F_n \cup \delta$, $\delta \notin F_n$, and $F^* = (F^*_1, \ldots, F^*_n)$. Furthermore, let $F_i = F_{i1}$, $i \in \{1, \ldots, n-1\}$ and $F = (F_1, \ldots, F_{n-1})$. For $k \in \{n, \ldots, m+1\}$, we can split the set $C_k^F$ by considering whether or not $\delta$ is contained in $E$. We have that
\[ C_k^F^* = \{ E \in C_k^F^* : \delta \in E \} \cup \{ E \in C_k^F^* : \delta \notin E \} \text{ and} \]
\[ |C_k^F^*| = |\{ E \in C_k^F^* : \delta \in E \}| + |\{ E \in C_k^F^* : \delta \notin E \}|. \]

For \( k \in \{ n, \ldots, m \} \), \( \{ E \in C_k^F^* : \delta \notin E \} = C_k^F \) and \( \{ E \in C_{m+1}^F : \delta \notin E \} = \emptyset \). By the induction hypothesis,
\[ \sum_{k=n}^{m+1} |\{ E \in C_k^F^* : \delta \notin E \}| = \sum_{k=n}^{m} |C_k^F| = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) = c(F). \tag{28} \]

The set \( \{ E \in C_k^F^* : \delta \in E \}, k \in \{ n, \ldots, m+1 \} \), can be further split into two sets by considering if, for \( E \in C_k^F^* \), \( F_n \cap E = \emptyset \) or \( F_n \cap E \neq \emptyset \) and have
\[ \{ E \in C_k^F^* : \delta \in E \} = \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \} \cup \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \} \text{ and} \]
\[ |\{ E \in C_k^F^* : \delta \in E \}| = |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \}| + |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \}|. \]

Let \( m = \sum_{i=1}^{n-1} t_i \). The set \( \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \}, k \in \{ n, \ldots, m+1 \} \), can also be rewritten as
\[ \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \} = \{ E \cup \{ \delta \} : E \in C_{k-1}^F \}. \]

Therefore it holds that
\[ |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \}| = |\{ E \cup \{ \delta \} : E \in C_{k-1}^F \}| = |C_{k-1}^F|. \]

Next, since \( |\bigcup_{i=1}^{n} F_i| = m + 1 \), we have that \( \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \} = \emptyset \) for \( k \in \{ n+2, \ldots, m+1 \} \). Because \( m + 1 - (n - 1) < (m + 1) - n = \ell + 1 \), we can use the induction hypothesis and it holds that
\[ \sum_{k=n}^{m+1} |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \}| = \sum_{k=n}^{m} |C_k^F^*| = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) = c(F)(29) \]

Now we consider the sets \( \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \}, k \in \{ n, \ldots, m \} \). For \( k = n \), let \( E \in C_k^F^* \) and we have that \( |E \cap F_i^*| = 1 \) for \( i \in \{ 1, \ldots, n \} \). Therefore, \( \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \} = \emptyset \).

For \( k \in \{ n+1, \ldots, m+1 \} \) it holds that
\[ \{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \} = \{ E \cup \{ \delta \} : E \in C_{k-1}^F \} \]
and it follows that
\[ |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \}| = |\{ E \cup \{ \delta \} : E \in C_{k-1}^F \}| = |C_{k-1}^F|. \tag{30} \]

By using the induction hypothesis again,
\[ \sum_{k=n}^{m+1} |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \}| = \sum_{k=n}^{m+1} |C_k^F^*| = \sum_{k=n}^{m} |C_k^F| = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) = c(F). \]

We can now write
\[ c(F^*) = \sum_{k=n}^{m} |C_k^F^*| \]
\[ = \sum_{k=n}^{m} |\{ E \in C_k^F^* : \delta \notin E \}| + \sum_{k=n}^{m} |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E = \emptyset \}| \]
\[ + \sum_{k=n}^{m} |\{ E \in C_k^F^* : \delta \in E \text{ and } F_n \cap E \neq \emptyset \}|. \]
and using (28), (29) and (30), we obtain that it is
\[ = 2c(F) + c(\hat{F}) \]
\[ = 2 \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) + \prod_{i=1}^{n-1} \left( 2^{|F_i|} - 1 \right) \]
\[ = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) + \prod_{i=1}^{n-1} \left( 2^{|F_i|} - 1 \right) \]
\[ = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right) * \left( 2^{|F_n|+1} - 1 \right) \]
\[ = \prod_{i=1}^{n} \left( 2^{|F_i|} - 1 \right). \]

This shows that (27) holds for \( \ell + 1 \) and it completes the proof.

\[ \square \]

C Generating Algorithm for Sets \( C_k \)

In this section, we provide a generating algorithm for the elements of sets \( C_k \) in Proposition 1. Furthermore, we include an example of how the algorithm works for \( n = 2 \) and \( t_1 = t_2 = 3 \).

The algorithm only needs the input \( n \) and \( t_i, i \in \{1, \ldots, n\} \) and is given in Algorithm 1. The correctness of Algorithm 1 is given in Appendix D.

**Algorithm 1 Generating Algorithm**

**Require:** \( n \) and \( t = \{t_1, \ldots, t_n\} \).

1. Create sets \( H_i = \{F_{i,p} \mid p \in \{1, \ldots, t_i\} \} \), \( i \in \{1, \ldots, n\} \) and let \( m = \sum_{i=1}^{n} t_n \).
2. for \( k \in \{n, \ldots, m\} \)
3. Set \( E = \{\} \).
4. Recursion\((k, n, E)\).
5. end for
6. procedure Recursion\((l, s, E)\).
7. Set \( ub_s = \min\{l - (s - 1), t_s\} \) and \( lb_s = \max\{1, l - \sum_{s=1}^{s-1} t_b\} \).
8. for \( i \in \{lb_s, \ldots, ub_s\} \)
9. Create all combinations \( Comb_{si} \) of \( H_s \) of length \( i \).
10. for \( J \in Comb_{si} \)
11. Set \( \hat{E} = E \cup J \).
12. if \( s > 1 \)
13. Recursion\((l - i, s - 1, \hat{E})\).
14. else
15. Output \( \hat{E} \) as element of \( C_k \).
16. end if
17. end else
18. end for
19. end for
20. end procedure

We now give an example of how to generate elements of \( C_k \) sets and the formula from Proposition 1 for \( n = 2 \) and \( t_1 = t_2 = 3 \). Furthermore, we show how the algorithm works by generating elements of \( C_3 \).

1. First, we define the two sets \( H_1 = \{F_{(1,1)}, F_{(1,2)}, F_{(1,3)}\} \) and \( H_2 = \{F_{(2,1)}, F_{(2,2)}, F_{(2,3)}\} \).
2. Next, we set \( E = \{\} \) and start the Recursion procedure with Recursion\((3, 2, E)\).
3. The next step is to calculate \( ub_2 \) and \( lb_2 \). In this case we have \( ub_2 = \min\{3 - (2 - 1), 3\} = 2 \) and \( lb_2 = \max\{1, 3 - (2 - 1) \times 3\} = 1 \).
4. For \( i = 1 \), we create all possible combinations \( Comb_{21} \) of \( H_2 \) of length 1 and obtain \( Comb_{21} = H_2 \).
5. We choose \( J = \{ F_{(2,1)} \} \) and we have that \( \hat{E} = E \cup J = \{ F_{(2,1)} \} \).

6. As \( s = 2 > 1 \), we start the recursion procedure with RECURSION\((2,1,\hat{E})\).

7. We have to calculate \( l_1 \) and \( u_2 \). We have that \( lb = ub = 2 \).

8. Therefore, \( i = 2 \) and we create \( Comb_{12} = \{ \{ F_{(1,1)}, F_{(1,2)} \}, \{ F_{(1,1)}, F_{(1,3)} \}, \{ F_{(1,2)}, F_{(1,3)} \} \} \).

9. First, we choose \( J = \{ F_{(1,1)}, F_{(1,2)} \} \) and we have that \( \hat{E} = F_{(2,1)}, F_{(1,1)}, F_{(1,2)} \).

10. As \( s = 1 \), \( \hat{E} \) is an element of \( C_3 \) and we iterate over the elements of \( Comb_{12} \). We also obtain \( \{ F_{(2,1)}, F_{(1,1)}, F_{(1,3)} \} \) and \( \{ F_{(2,1)}, F_{(1,2)}, F_{(1,3)} \} \) which are elements of \( C_3 \).

After running the algorithm for all \( k \in \{ 2, \ldots, 6 \} \), we obtain the following sets \( C_k, k \in \{ 2, \ldots, 6 \} \).

\[
\begin{align*}
C_2 & = \{ \{ F_{(1,1)}, F_{(2,1)} \}, \{ F_{(1,1)}, F_{(2,2)} \}, \{ F_{(1,1)}, F_{(2,3)} \}, \{ F_{(1,2)}, F_{(2,1)} \}, \{ F_{(1,2)}, F_{(2,2)} \}, \{ F_{(1,2)}, F_{(2,3)} \} \}, \\
C_3 & = \{ \{ F_{(2,1)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(1,1)}, F_{(1,3)} \}, \{ F_{(2,1)}, F_{(1,2)}, F_{(1,3)} \}, \{ F_{(2,2)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,2)}, F_{(1,1)}, F_{(1,3)} \}, \{ F_{(2,2)}, F_{(1,2)}, F_{(1,3)} \}, \\
& \quad \{ F_{(2,3)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,3)}, F_{(1,1)}, F_{(1,3)} \}, \{ F_{(2,3)}, F_{(1,2)}, F_{(1,3)} \}, \{ F_{(2,1)}, F_{(2,2)}, F_{(1,1)} \}, \{ F_{(2,1)}, F_{(2,2)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,2)}, F_{(1,3)} \}, \\
& \quad \{ F_{(2,1)}, F_{(2,3)}, F_{(1,1)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,3)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,1)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,3)} \} \}, \\
C_4 & = \{ \{ F_{(2,1)}, F_{(2,2)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,2)}, F_{(1,3)} \}, \{ F_{(2,1), F(2,3)}, F_{(1,1)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,3)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,1)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,3)} \} \}, \\
C_5 & = \{ \{ F_{(2,2)}, F_{(2,3)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,2)}, F_{(2,3)}, F_{(1,3)} \}, \{ F_{(1,1)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(1,1)}, F_{(2,3)}, F_{(1,3)} \} \}, \\
C_6 & = \{ \{ F_{(2,1)}, F_{(2,2)}, F_{(1,1)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,2)}, F_{(1,3)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,2)} \}, \{ F_{(2,1)}, F_{(2,3)}, F_{(1,3)} \} \}, \\
\end{align*}
\]

To see an example sum of \( S \) for \( n = 2 \) and \( t_1 = t_2 = 3 \), we use a simple artificial example that is also used later in the computational tests. We assume that each implementation of \( i \), \( i \in \{ 1, 2 \} \) and \( j \in \{ 1, \ldots, 3 \} \), corresponds to an elementary component \( a_{ij} \) which is independent of all other components and has reliability \( r_{ij} \). Therefore, for \( E \in C_k, k \in \{ 2, \ldots, 6 \} \), we find that \( P(E) = \prod_{(i,j) \in E} r_{ij} \) and for \( R \) we have that

\[
P(R) = r_{21}r_{11} + r_{21}r_{12} + r_{21}r_{13} + r_{22}r_{11} + r_{22}r_{12} + r_{22}r_{13} + r_{23}r_{11} + r_{23}r_{12} + r_{23}r_{13} \\
+ r_{21}r_{11}r_{12} + r_{21}r_{11}r_{13} + r_{21}r_{12}r_{13} + r_{22}r_{11}r_{12} + r_{22}r_{11}r_{13} + r_{22}r_{12}r_{13} + r_{23}r_{11}r_{12} + r_{23}r_{11}r_{13} \\
+ r_{23}r_{12}r_{13} + r_{22}r_{22}r_{11} + r_{22}r_{22}r_{12} + r_{22}r_{22}r_{13} + r_{21}r_{23}r_{11} + r_{21}r_{23}r_{12} + r_{21}r_{23}r_{13} + r_{22}r_{23}r_{11} + r_{22}r_{23}r_{12} \\
+ r_{22}r_{23}r_{13} + r_{22}r_{23}r_{13} + r_{21}r_{22}r_{22}r_{11} + r_{21}r_{22}r_{22}r_{12} + r_{21}r_{22}r_{22}r_{13} + r_{21}r_{22}r_{23}r_{11} + r_{21}r_{22}r_{23}r_{12} + r_{21}r_{22}r_{23}r_{13} \\
+ r_{21}r_{22}r_{23}r_{12} + r_{21}r_{22}r_{23}r_{13} + r_{21}r_{22}r_{23}r_{13} + r_{21}r_{22}r_{23}r_{12} + r_{21}r_{22}r_{23}r_{13} + r_{21}r_{22}r_{23}r_{12} \\
+ r_{22}r_{23}r_{11}r_{12} + r_{22}r_{23}r_{11}r_{13} + r_{22}r_{23}r_{12}r_{13} + r_{22}r_{23}r_{13}r_{12} + r_{22}r_{23}r_{13}r_{13}. \\
\]

D Proof of Correctness of Algorithm [1]

In this section, we prove in Lemma [3] that the sets \( C_k \) created by Algorithm [1] are equal to the sets \( C_k \) from Proposition [4].
Lemma 4. Let $n \in \mathbb{N}^+$, $t = \{t_1, \ldots, t_n\} \in \mathbb{N}^++$ and $m = \sum_{i=1}^n t_i$. Furthermore, let $\tilde{C}_k$, $k \in \{n, \ldots, m\}$, be the sets created with Algorithm 1 and for all $k \in \{n, \ldots, m\}$

\[C_k = \{E = \{E_1, \ldots, E_k\} : E_u = F_{i(u)}j(u) \text{ for some } i(u) \in \{1, \ldots, n\}, j(u) \in \{1, \ldots, t_i(u)\}, u \in \{1, \ldots, k\}, \{i(1), i(2), \ldots, i(k)\} = \{1, \ldots, n\} \text{ and } E_p \neq E_q \text{ for } p, q \in \{1, \ldots, k\} \text{ and } p \neq q\}.

We then have that $\tilde{C}_k = C_k$ for all $k \in \{n, \ldots, m\}$.

Proof. Without loss of generality, let $k \in \{n, \ldots, m\}$ be fixed. First, we prove that $\tilde{C}_k \subseteq C_k$.

We begin by showing that $l_b \leq u_b$ for all $s \in \{1, \ldots, n\}$ in the algorithm. We first choose $s = n$ and, hence, $l = k$. As seen before, $u_b = \min\{k - n + 1, t_n\}$ and $l_b = \max\{1, k - \sum_{b=1}^{n-1} t_b\}$. We know that $1 \leq t_n$, $1 \leq k - n + 1$, and $k - \sum_{i=1}^{n-1} t_i \leq k - (n - 1)$. Also,

\[k \leq m = \sum_{i=1}^n t_i \Leftrightarrow k - \sum_{i=1}^{n-1} t_i \leq t_n.

Hence, $l_b \leq u_b$. Without loss of generality, let $s \in \{1, \ldots, n - 1\}$ and let $\hat{t}_n, \ldots, \hat{t}_{s+1}$ be the value chosen with $l_b \leq \hat{t}_i \leq u_b$, $i \in \{s + 1, \ldots, n\}$. Therefore $l = k - \sum_{b=s+1}^n \hat{t}_b$. We know that $1 \leq t_s$ and $k - \sum_{b=s+1}^n \hat{t}_b - \sum_{b=1}^{n-1} t_b \leq k - \sum_{b=s+1}^n \hat{t}_b - (s - 1)$. Furthermore, we know that

\[k - \sum_{b=s+2}^n \hat{t}_b - \sum_{b=1}^n t_b \leq l_{s+1} \leq \hat{t}_{s+1} \Rightarrow k - \sum_{b=s+2}^n \hat{t}_b - \sum_{b=1}^{s+1} t_b - t_s \leq 0 \Rightarrow k - \sum_{b=s+2}^n \hat{t}_b - \sum_{b=1}^{s+1} t_b \leq t_s \Rightarrow \hat{t}_{s+1} \leq u_{b_{s+1}} \leq k - \sum_{b=s+1}^n \hat{t}_b - ((s + 1) - 1) \Rightarrow 0 \leq k - \sum_{b=s+1}^n \hat{t}_b - (s - 1) - 1 \Rightarrow 1 \leq k - \sum_{b=s+1}^n \hat{t}_b - (s - 1).

Hence, $l_b \leq u_b$. This gives us that $l_b \leq u_b$ for all $i \in \{1, \ldots, n\}$.

Let $\hat{E} \in \tilde{C}_k$, $\hat{H}_i = \hat{E} \cap H_i$, and $\hat{t}_i = |\hat{H}_i|$, $i \in \{1, \ldots, n\}$. We know that $1 \leq \hat{t}_i \leq t_i$, $i \in \{1, \ldots, n\}$. To prove that $\hat{E} \in C_k$, we just have to show $|\hat{E}| = k$. We have proven that $l_b \leq u_b$, $s \in \{1, \ldots, n\}$, and we know that $l_b \leq u_b$. Hence,

\[l_1 = \max\{1, k - \sum_{b=2}^n t_b\} \leq \hat{t}_1 \leq u_1 \min\{k - \sum_{b=2}^n \hat{t}_b, t_1\} \Rightarrow \hat{t}_1 = k - \sum_{b=2}^n \hat{t}_b \Rightarrow k = \sum_{b=1}^n \hat{t}_b = |\hat{E}|.

This gives us that $\tilde{C}_k \subseteq C_k$. Now we prove that $C_k \subseteq \tilde{C}_k$.

Let $E = \{E_1, \ldots, E_k\} \in C_k$. We know that $\forall l \in \{1, \ldots, k\} \exists i \in \{1, \ldots, n\} \exists j \in \{1, \ldots, t_l\} : E_l = F_{ij}$. Furthermore, let $\tilde{H}_i = \{F_{ij} : \exists i \in \{1, \ldots, n\}, j \in \{1, \ldots, t_i\} : E_i = F_{ij}\}$ and $\tilde{t}_i = |\tilde{H}_i|$ for all $i \in \{1, \ldots, n\}$. We know that $\sum_{i=1}^n \tilde{t}_i = k$ and $\tilde{t}_i \in \{1, \ldots, t_i\}$. In the Recursion procedure of the algorithm with $l = k$, $s = n$, and $E = \{\}$, we have that $u_b = \min\{k - n + 1, t_n\}$ and $l_b = \max\{1, k - \sum_{b=1}^{n-1} t_b\}$. As $1 \leq \tilde{t}_i$, $k - \sum_{i=1}^{n-1} \tilde{t}_i \leq k - \sum_{i=1}^{n-1} t_i = \tilde{t}_n$, $\tilde{t}_n \leq t_n$ and $\tilde{t}_n = k - \sum_{b=1}^{n-1} \tilde{t}_b \leq k - (n - 1)$, we have that $l_b \leq \tilde{t}_i \leq u_b$ and, therefore, $\tilde{t}_n$ can be chosen in the algorithm and $\tilde{H}_n$ is a combination of elements in $H_n$ of length $t_n$. 

20
Without loss of generality, let $s \in \{1, \ldots, n - 1\}$. We then have that $ub_s = \min\{k - \sum_{b=s+1}^{n} \hat{t}_n - (s - 1), t_s\}$ and $lb_s = \max\{1, k - \sum_{b=s+1}^{n} \hat{t}_n - \sum_{b=1}^{s-1} t_b\}$. We know that $1 \leq \hat{t}_s$, $k - \sum_{b=s+1}^{n} \hat{t}_n - \sum_{b=1}^{s-1} t_b \leq k - \sum_{b=s+1}^{n} \hat{t}_n - \sum_{b=1}^{s-1} t_b$, and $\hat{t}_s \leq t_s$ and $t_s = k - \sum_{b=s+1}^{n} \hat{t}_n - \sum_{b=1}^{s-1} t_b \leq k - \sum_{b=s+1}^{n} \hat{t}_n - \sum_{b=1}^{s-1} t_b$ and, therefore, $lb_s \leq t_s \leq ub_s$. Hence $t_1, \ldots, t_n$ can be chosen in the recursion of the algorithm and as $\hat{H}_i \subseteq H_i$ with cardinality $\hat{t}_i$ for all $i \in \{1, \ldots, n\}$, we know that $E \in \hat{C}_k$ and $C_k \subseteq \hat{C}_k$.

This proves that $C_k = \hat{C}_k$ for all $k \in \{n, \ldots, m\}$.

## E Examples of Door Management Systems

In this section, we show an example input for DMS with two doors and three paths. Data for all other systems can be found at [https://sites.google.com/site/sergiogarciaquiles/](https://sites.google.com/site/sergiogarciaquiles/) They are given as csv files where each row represents a path and each column represents a component or connection. The rows represent the paths are ordered by doors. For every row $i$, we have at column $j$ a 1 if the component/connection represented by column $j$ is used for the path that is represented by row $i$ and 0 otherwise. The Python code we used to run the computational tests has also been uploaded to this repository.

The example system has 70 elements of which 27 are components and 43 are connections. Table 5 shows what positions $t$ and connections $x$ are used in which path for every door. Failure probabilities of the components and connections are also provided.

| $t$  | $x$  | $[1, 1]$ | $[1, 2]$ | $[2, 3]$ | $[2, 2]$ | $[2, 1]$ | $[3, 3]$ | $[3, 1]$ | $[3, 2]$ | $[3, 3]$ | Prob. |
|------|------|---------|---------|---------|---------|---------|---------|---------|---------|---------|-------|
| $t_1$| 1    | 1       | 0       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_2$| 0    | 0       | 0       | 1       | 1       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_3$| 0    | 0       | 0       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_4$| 0    | 0       | 1       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_5$| 0    | 0       | 0       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_6$| 0    | 0       | 0       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_7$| 0    | 1       | 0       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_8$| 0    | 1       | 1       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_9$| 0    | 0       | 1       | 0       | 0       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |
| $t_{10}$| 0 | 0       | 0       | 1       | 1       | 0.020   | 0.020   | 0.020   | 0.020   | 0.020   | 0.020 |

Table 5: Example systems data