Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 8, n. 1, 2021.

Trabalho apresentado no XL CNMAC, Evento Virtual - Co-organizado pela Universidade do Mato Grosso do Sul (UFMS).

A simple vectorization algorithm to address the *d-MP* problem without generating duplicate candidates

Majid Forghani-elahabad [1]

Center of Mathematics, Computing and Cognition–CMCC, Federal University of ABC–UFABC, Santo André, SP, Brazil

Emilio Francesquini [2]

Center of Mathematics, Computing and Cognition–CMCC, Federal University of ABC–UFABC, Santo André, SP, Brazil

Wei-Chang Yeh [3]

Integration and Collaboration Laboratory, Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Abstract. The reliability of a multistate flow network (MFN) can be evaluated indirectly based on d-minimal paths (*d-MP*s). In fact, with all the *d-MP*s at hand, an MFN's reliability can be computed by calculating a union probability. Hence the determination of all *d-MP*s in an MFN has been a very attractive problem in the last decades. As a result, many algorithms have been proposed to solve the *d-MP* problem. However, as the number of *d-MP*s grows exponentially with the size of the network, the available algorithms in the literature are not so practical. Here, we propose a vectorization algorithm to address the problem and show its practical efficiency using three benchmarks. The numerical results show that the vectorization algorithm solves the problem more than three times faster than the non-vectorization one in some cases.

Keywords. System reliability, Vectorization algorithm, *d-MP* problem, Multistate flow network

1 Introduction

A multistate flow network (MFN), which is also called a stochastic-flow network, is a network flow whose components, including nodes and arcs, can have more than two different states or performance levels [9]. The reliability of a two-terminal MFN for demand level of d, denoted by R_d, is defined as the probability of transmitting at least d units of data (flow or goods) from a source node to a destination node in the network [8]. There are many real-world systems such as transportation networks, computer and communication networks, distribution systems, and power transmission systems that can be modeled as an MFN. This is why the problem of reliability evaluation of MFNs has turned to be of great importance [2]. The reliability of two-terminal MFN can be evaluated indirectly in terms of minimal paths (MPs) [2, 4, 6, 7, 13, 14, 16, 18] or minimal cuts (MCs) [3, 5, 8–11, 15] through a three-stage process; (1) determining all the MPs or MCs, (2) finding all the *d-MP*s or *d-MC*s, which are called the *d-MP* or *d-MC* problem, and (3) calculating a union probability as the network reliability. The focus of this work is on the second stage and the *d-MP* problem, and hence we assume that all the MPs are available in advance. A path in a

[1]m.forghani@ufabc.edu.br

[2]e.francesquini@ufabc.edu.br

[3]yeh@ieee.org

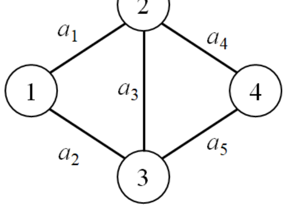
network is a sequence of adjacent arcs from a source node to a sink node, and an MP is a path so that none of its proper subsets is a path. Several authors have worked to improve the solution of the *d-MP* problem. Based on a breadth-first search technique, Chen et al. [2] proposed a recursive algorithm for the determination of all the *d-MP*s for all the demand values d. Yeh and Zuo [18] proposed a subtraction-based algorithm to find all the *d-MP*s for all the d values and showed its practical efficiency in comparison with some addition-based algorithms in the literature.

However, to the best of our knowledge, there are no vectorization algorithms proposed in the literature to address this problem. Hence, in this work, we propose a vectorization algorithm based on the available algorithms in the literature and show its practical efficiency by employing three benchmarks taken from the literature.

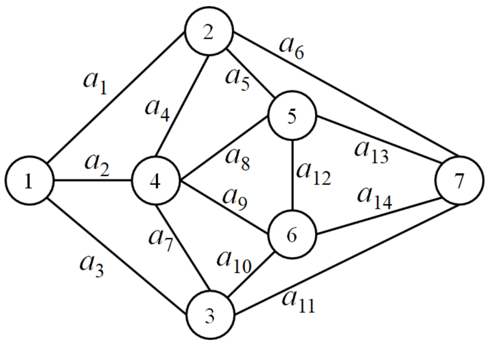
The remaining of the paper is organized as follows. Section 2 introduces the required notations and definitions along with some preliminaries and explanations. Then, in Section 3, a background on the vectorization process is presented. Next, experimental results are outlined in Section 4. Finally, we conclude in Section 5.

2 The *d-MP* problem

Let $G = G(N, A, M)$ be an MFN, where $N = \{1, 2, \cdots, n\}$ is the set of nodes with nodes 1 and n being the source and destination nodes, respectively, $A = \{a_i |\ 1 \leq i \leq m\}$ is the set of arcs, and $M = (M_1, M_2, \cdots, M_m)$ is a vector in which M_i gives the maximum capacity of arc a_i, for $i = 1, 2, \cdots, m$. The vector $X = (x_1, x_2, \cdots, x_m)$ is a state vector in which x_i represents the current capacity of a_i, for $i = 1, 2, \cdots, m$. Let $V(X)$ be the maximum flow of the network from the source node to the destination node under system vector X. $Z(X) = \{a_i \in A | x_i > 0\}$ and $e_i = 0(a_i)$ be a system vector in which the capacity level is 1 for a_i and 0 for other arcs. Let also $P_1,\ P_2,\ \cdots,\ P_h$ be all the MPs in the network (so h is the number of all the MPs) and $K_j = \min\{M_r | a_r \in P_j\}$ be the capacity of the jth MP, for $j = 1, 2, \cdots, h$. Finally, we note that a system state $X = (x_1, x_2, \cdots, x_m)$ is said less than or equal to system state $Y = (y_1, y_2, \cdots, y_m)$, i.e., $X \leq Y$, when $x_i \leq y_i$, for $i = 1, 2, \cdots, m$, and also $X < Y$, when $X \leq Y$ and there exists at least one j, $j = 1, 2, \cdots, m$, such that $x_j < y_j$. For instance, in the given network in Fig. 1a, we have $N = \{1, 2, 3, 4\}$ and $A = \{a_1, a_2, a_3, a_4, a_5\}$. Assuming $M = (3, 1, 1, 2, 2)$, any non-negative integer-valued vector $(x_1, x_2, x_3, x_4, x_5) \leq M$ can be considered as a system vector. We also have four MPs, $P_1 = \{a_1, a_4\}$, $P_2 = \{a_1, a_3, a_5\}$, $P_3 = \{a_2, a_5\}$, and $P_4 = \{a_2, a_3, a_4\}$ in this network. So, considering $M = (3, 1, 1, 2, 2)$, it is easy to see that $K_1 = 2$, $K_2 = 1$, $K_3 = 1$ and $K_4 = 1$. Next, we have three definitions from the literature [7, 16].



(a) A small size netwok.



(b) A medium size network.

Figure 1: Two known benchmark networks of small and medium size taken from the literature [4].

Definition 2.1. *Assuming f_j as the amount of flow on MP P_j, for $j = 1, \cdots, h$, the vector $F = (f_1, f_2, \cdots, f_h)$ is called a feasible flow vector (FFV) at demand level d, denoted by d-FFV here, when it satisfies the following system.*

$$\begin{array}{ll} (i)\ f_1 + f_2 + \cdots + f_h = d, & \\ (ii)\ 0 \leq f_j \leq \min\{K_j, d\}, & j = 1, 2, \cdots, h, \\ (iii)\ \sum_{j:\ a_i \in P_j} f_j \leq \min\{M_i, d\}, & i = 1, 2, \cdots, m, \end{array} \tag{1}$$

For instance, it can be calculated that $F = (1, 1, 1, 0)$ is a 3-*FFV* in the network given in Fig. 1a.

Definition 2.2. *A state vector $X = (x_1, x_2, \cdots, x_m)$ is d-MP if and only if the following system is satisfied.*

$$\begin{array}{l} (i)\ V(X) = d, \\ (ii)\ V(X - e_i) = d - 1, \quad \text{for each } i \text{ that } a_i \in Z(X). \end{array} \tag{2}$$

For instance, it can be seen that $X = (2, 1, 1, 1, 2)$ is a 3-*MP* in the network given in Fig. 1a.

Definition 2.3. *A system vector $X = (x_1, x_2, \cdots, x_m)$ is called a d-MP candidate when there is a d-FFV, say $F = (f_1, \cdots, f_h)$, that satisfies the following relation:*

$$x_i = \sum_{j:\ a_i \in P_j} f_j, \qquad \forall i = 1, 2, \cdots, m. \tag{3}$$

In fact, the obtained *d-MP* candidate from a *d-FFV* through Eq. (3) is called the associated *d-MP* candidate with that *d-FFV*. For instance, in Fig. 1a, $X = (2, 1, 1, 1, 2)$ is a 3-*MP* candidate which can be obtained from $F = (1, 1, 1, 0)$. It is notable that this vector is also a (real) 3-*MP*. The following result, which is proven in [14], shows that it always happens.

Theorem 2.1. *Every d-MP is a d-MP candidate.*

Although one can use the given condition in Definition 2.2 to check each candidate for being a *d-MP*, Lin and his coworkers [14] proposed the following theorem which can be employed to determine all the *d-MP*s among the candidates.

Theorem 2.2. *The set of all the minimal elements among all the d-MP candidates is equal to the set of all the d-MPs.*

We note that a vector $X \in \Psi$ is a minimal vector if there is no other vector $Y \in \Psi$ such that $Y < X$, namely, X is not necessarily less than all the vectors in Ψ; however, there is no another vector in Ψ less than it. Therefore, one can first find all the *d-MP* candidates by solving the Diophantine system (1) and using Eq. (3), then remove the possible duplicates, and finally remove all the non-minimal vectors to determine the set of all the *d-MP*s. Notice that one can remove all the duplicates and non-minimal vectors at once, but it is less efficient in practice, as our numerical results here show. Hence, the following algorithm can be considered as a slightly improved version of the proposed algorithm by Lin et al. [14].

Algorithm 1
Step 1. Find all the *d-FFV*s by solving the system (1).
Step 2. Compute the associate *d-MP* candidate, i.e., X, with each *d-FFV* obtained in Step 1 through (3).
Step 3. Remove the duplicate candidates obtained in Step 2.

Step 4. Remove the non-minimal candidates.

We note that several algorithms, which are more efficient than Algorithm 1, have been proposed in the literature. However, in this paper, we focus on the impact which vectorization techniques can have on the *d-MP* problem algorithms. Therefore, in this case, the choice of algorithm is arbitrary as any of them would be adequate to generate the experimental evaluation results shown in Section 4.

3 Vectorization

Vectorization can be described as the process of converting the application of an operation to a single value (a scalar) to the application of the same operation to multiple values (a vector) at once. It is, therefore, a parallelization technique that can improve the performance of many kinds of computer loads. It should not be confused with the application of standard parallelization techniques, which use multiple processors to achieve similar results since vector operations happen in the context of a single processor [17].

Vector processors have existed in one form or another since the '70s, and virtually all modern processors provide some form of SIMD (Single Instruction Multiple Data) capabilities. Intel x86, for example, offers multiple versions of their SSE and AVX extensions. Other processor vendors such as ARM also provide extensions such as Neon. More recently, GPUs have also begun to include vector instructions [12].

Invariably all these approaches work using long registers that are usually several times the size of regular registers used for scalar operations. While 128 and 256-bit registers are commonplace, 4096-bit registers are also available in some specialized architectures. Let us consider 256-bit vector registers. Typical integers are stored in 32 bits which means that the time it takes to, for instance, sum a pair of integers using scalar operations is the same as summing eight pairs of integers when using vector instructions.

Most optimizing compilers and numeric computing suites such as Mathematica, MATLAB, and NumPy offer some support for SIMD operations. However, effectively and automatically vectorizing code that was not explicitly written to profit from these SIMD instructions is still a very active research problem in Computer Science.

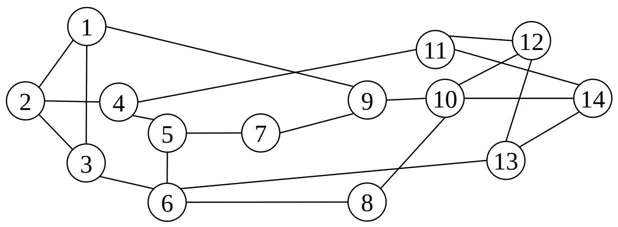
In this work, we leverage vectorization explicitly, using 128-bit vectors on a commodity processor to improve the algorithm's performance.

4 Numerical results

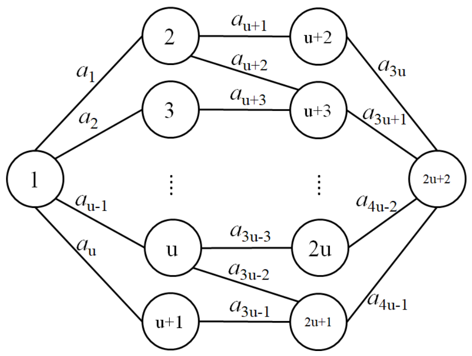
Here, we make several numerical comparisons between the three algorithms. The first one (*A1*) is Algorithm 1. The second one (*MA*) is an improved version of Algorithm 1 in which steps 2 and 3 are merged so that no duplicate candidate is generated. The third algorithm (*VMA*) is the vectorized version of the second one.

All the experiments were done on a computer Intel(R) Core(TM) i7-7500U CPU 2.7 GHz, with 16 GB of RAM. We employ three benchmark networks taken from the literature given in Figs. 1b, 2a and 2b. The maximum capacities of arcs in all of these benchmarks are set to be 3, i.e., $M_i = 3$.

All the generated results are summarized in Tables 1 and 2. The columns in the tables are d, the demand value, n_{can}, the number of candidates, n_{d-MP}, the number of *d-MP*s. Also, t_{A1}, t_{MA}, t_{VMA} are respectively the running times of A1, MA and VMA. We note that Table 2 provides the related results to the general network in Fig. 2b whose size varies with u, and the first column



(a) The NSF benchmark.



(b) The general network topology.

Figure 2: The NSF benchmark and a general network topology taken from the literature [4, 7].

Table 1: The final results on the benchmarks given in Figs. 1b and 2a

Benchmarks	d	n_{can}	n_{d-MP}	t_{A1} (s)	t_{MA} (s)	t_{VMA} (s)	t_{MA}/t_{VMA}
Fig. 1b	4	76530	8071	474,18 ± 0,25	3.52 ±0.02	1.63 ± 0.00	2.16
Fig. 1b	5	180266	12292	14,15 ± 0,07	16.31 ±0.15	5.84 ± 0.01	2.79
Fig. 1b	6	220762	12639	197,47 ± 0,07	21.54 ±0.13	7.99 ± 0.04	2.70
Fig. 1b	7	149408	7350	958,25 ± 0,38	9.76 ±0.08	4.95 ± 0.02	1.97
Fig. 1b	8	62748	3011	1354,26 ± 4,93	4.58 ±0.04	3.98 ± 0.04	1.15
						Geo. Mean	2.06
Fig. 2a	4	616152	18950	717,18 ± 1,13	132.95 ±0.33	38.80 ± 0.07	3.43
Fig. 2a	5	1069403	20153	3413,77 ± 1,24	229.83 ±0.34	70.49 ± 0.10	3.26
Fig. 2a	6	689297	12387	5006,83 ± 2,45	100.61 ±0.21	39.40 ± 0.06	2.55
Fig. 2a	7	193204	3109	1371,52 ± 0,27	25.69 ±0.05	21.85 ± 0.21	1.18
Fig. 2a	8	24170	438	78,41 ± 0,05	25.25 ±0.10	25.31 ± 0.20	1.00
						Geo. Mean	2.02
						Full Geo. Mean	2.04

in this table, u, lists the parameter used to generate the corresponding network. Comparing the running times of the three algorithms given in Table 1 clearly shows the practical efficiency of MA and VMA compared with A1. Hence, we did not run A1 on the general network topology resulting in the absence of column t_{A1} in Table 2. The presented results in the tables are an average of at least ten executions. The standard deviation of the samples is presented alongside the average. Notice that MA employs a radix-tree with a branching factor of M to merge steps 2 and 3 in A1. Therefore, in $O(log_M n_{can})$ steps, it can determine if a newly generated candidate was already known. Computationally, this is more efficient than generating all candidates and then removing duplicates, not due to lower algorithmic complexity, but because of the savings in memory usage for the temporary storage of duplicates, and thus a more efficient use of the processor caches. This approach's effect can be seen in the number of candidates, n_{can}, and execution times. For instance, in the network given in Figure 1b, considering $d = 4$, n_{can} is lower than the respective value when $d = 5$ (76530 *vs.* 180266). The results provided in tables 1 and 2 clearly show the practical efficiency of VMA in comparison with the other ones. Notice that, as the number of candidates or *d-MP*s increases, so does VMA's performance due to more efficient use of vectorization.

The performance bottleneck of the current implementation is in the fourth step of Algorithm 1, *i.e.*, the removal of the non-minimal candidates. More than 95% and 75% of the execution times

Table 2: The final results on the general network topology given in Fig. 2b.

u	d	n_{can}	n_{d-MP}	t_{MA} (s)	t_{VMA} (s)	t_{MA}/t_{VMA}
	10	93511	6485	4,42 ± 0.04	2,38 ± 0.02	1,86
	11	44292	2702	0,83 ± 0.01	0,61 ± 0.00	1,36
5	12	13495	805	0,19 ± 0.00	0,18 ± 0.00	1,05
	13	2834	155	0,13 ± 0.00	0,12 ± 0.00	1,03
	14	304	19	0,12 ± 0.00	0,12 ± 0.00	1,01
					Geo. Mean	1.22
	13	870069	25554	252,67 ± 0.75	70,97 ± 0.22	3,56
	14	304361	8170	32,13 ± 0.39	12,24 ± 0.07	2,63
6	15	72792	1876	5,80 ± 0.04	5,04 ± 0.05	1,15
	16	10648	281	4,56 ± 0.05	4,57 ± 0.04	1,00
	17	832	26	4,52 ± 0.04	4,56 ± 0.05	1,00
					Geo. Mean	1.60
	16	6439087	82614	6853.59 ± 10.77	1784.97 ± 2.96	3.84
	17	1749225	21198	669.61 ± 0.53	287.60 ± 1.13	2.33
7	18	324272	3906	186.45 ± 0.55	174.31 ± 0.95	1.07
	19	36512	469	170.31 ± 1.34	169.85 ± 0.36	1.00
	20	2176	34	171.71 ± 1.55	169.54 ± 0.40	1.01
					Geo. Mean	1.58
					Full Geo. Mean	1.45

for MA and VMA are spent in this step, respectively. Currently, the complexity of this step is quadratic. However, it may be improved using multidimensional divide-and-conquer approaches such as the one proposed by Bentley [1] which we aim to investigate in our future works.

5 Conclusion

The *d-MP* and *d-MC* have been very attractive problems in recent decades as the reliability of multistate flow networks can be computed indirectly by using the *d-MP*s or *d-MC*s. In this work, focusing on the *d-MP* problem, a vectorization algorithm was proposed based on the available regular (non-vectorization) algorithms in the literature. We first improved the regular version of the algorithm so that no duplicate candidate is produced and then enhanced its practical efficiency via vectorization. Three known benchmarks were employed to generate the numerical results in which our proposed vectorization algorithm outperformed the others considerably. For future works, we plan to improve candidates' checking process, use longer vector registers, and use parallelization techniques.

Acknowledgments

The authors thank CNPq (grant 306940/2020-5) for supporting this work.

References

[1] Bentley, J. L. Multidimensional divide-and-conquer. *Communications of the ACM 23*, 4 (1980), 214–229.

[2] Chen, X., Tao, J., Bai, G., and Zhang, Y. Search for d-mps without duplications in multistate two-terminal networks. In *2017 Second International Conference on Reliability Systems Engineering (ICRSE)* (2017), IEEE, pp. 1–7.

[3] Forghani-elahabad, M. 1 exact reliability evaluation of multistate flow networks. In *Systems Reliability Engineering*. De Gruyter, 2021, pp. 1–24.

[4] Forghani-elahabad, M., and Bonani, L. H. Finding all the lower boundary points in a multistate two-terminal network. *IEEE Transactions on Reliability 66*, 3 (2017), 677–688.

[5] Forghani-elahabad, M., and Kagan, N. Assessing reliability of multistate flow networks under cost constraint in terms of minimal cuts. *International Journal of Reliability, Quality and Safety Engineering 26*, 05 (2019), 1950025.

[6] Forghani-elahabad, M., and Kagan, N. Reliability evaluation of a stochastic-flow network in terms of minimal paths with budget constraint. *IISE Transactions 51*, 5 (2019), 547–558.

[7] Forghani-elahabad, M., Kagan, N., and Mahdavi-Amiri, N. An mp-based approximation algorithm on reliability evaluation of multistate flow networks. *Reliability Engineering and System Safety 191* (2019), 106566.

[8] Forghani-Elahabad, M., and Mahdavi-Amiri, N. On search for all d-mcs in a network flow. *Iranian Journal of Operations Research 4*, 2 (2013), 108–126.

[9] Forghani-elahabad, M., and Mahdavi-Amiri, N. A new efficient approach to search for all multi-state minimal cuts. *IEEE Transactions on Reliability 63*, 1 (2014), 154–166.

[10] Forghani-elahabad, M., and Mahdavi-Amiri, N. An improved algorithm for finding all upper boundary points in a stochastic-flow network. *Applied Mathematical Modelling 40*, 4 (2016), 3221–3229.

[11] Forghani-elahabad, M., and Mahdavi-Amiri, N. An algorithm to search for all minimal cuts in a flow network. *Advances in Systems Science and Applications 20*, 4 (2020), 1–10.

[12] Hennessy, J. L., and Patterson, D. A. *Computer architecture: a quantitative approach.* Elsevier, 2011.

[13] Lamalem, Y., Housni, K., and Mbarki, S. An efficient method to find all d-mps in multistate two-terminal networks. *IEEE Access 8* (2020), 205618–205624.

[14] Lin, J.-S., Jane, C.-C., and Yuan, J. On reliability evaluation of a capacitated-flow network in terms of minimal pathsets. *Networks 25*, 3 (1995), 131–138.

[15] Mansourzadeh, S. M., Nasseri, S. H., Forghani-elahabad, M., and Ebrahimnejad, A. A comparative study of different approaches for finding the upper boundary points in stochastic-flow networks. *International Journal of Enterprise Information Systems (IJEIS) 10*, 3 (2014), 13–23.

[16] Niu, Y.-F., Xu, X.-Z., He, C., Ding, D., and Liu, Z.-Z. Capacity reliability calculation and sensitivity analysis for a stochastic transport network. *IEEE Access 8* (2020), 133161–133169.

[17] Rauber, T., and Rünger, G. *Parallel programming: for multicore and cluster systems.* Springer-Verlag, 2010.

[18] Yeh, W.-C., and Zuo, M. J. A new subtraction-based algorithm for the d-mps for all d problem. *IEEE Transactions on Reliability 68*, 3 (2019), 999–1008.