Topology Design with Minimal Cost Subject to Network Reliability Constraint

Basima Elshqeirat, Sieteng Soh, *Member, IEEE*, Suresh Rai, *Senior Member, IEEE*, and Mihai Lazarescu, *Member, IEEE*

Abstract – This paper addresses an NP-hard problem, refered to as Network Topology Design with minimum Cost subject to a Reliability constraint (NTD-CR), to design a minimal-cost communication network topology that satisfies a pre-defined reliability constraint. The paper describes a *dynamic programming* (DP) scheme to solve the NTD-CR problem, and proposes a DP approach, called Dynamic Programming Algorithm to solve NTD-CR (DPCR-ST), to generate the topology using a selected sequence of spanning trees of the network, STX_{min}. The paper shows that our DPCR-ST approach always provides a feasible solution, and produces an optimal topology given an optimal order of spanning trees. The paper proves that the problem of optimally ordering the spanning trees is NP-complete, and proposes three greedy heuristics to generate and order only k spanning trees of the network. Each heuristic allows the DPCR-ST approach to generate STX_{min} using only k spanning trees, which improves the time complexity while producing a near optimal topology. Simulations based on fully connected networks that contain up to 2.3×10^9 spanning trees show the merits of using the ordering methods and the effectiveness of our algorithm vis-à-vis to four existing state-of-the-art techniques. Our DPCR-ST approach is able to generate 81.5% optimal results, while using only 0.77% of the spanning trees contained in networks. Further, for a typical 2×100 grid network that contains up to 1.899^{102} spanning trees, DPCR-ST approach requires only k=1214 spanning trees to generate a topology with a reliability no larger than 5.05% off from optimal.

Index Terms – Dynamic programming, network optimization, network reliability, network topology design.

ACRONYMS AND ABBREVIATIONS

NTD-CR	Network Topology Design with minimum Cost subject to a Reliability constraint
B&B	Branch & Bound
GA	Genetic Algorithm
NN	Neural Network

Manuscript received April 25, 2013; revised August 26, 2013; accepted December 11, 2013.

B. Elshqeirat, Department of Computing, Curtin University, Perth, Western Australia; (e-mail: Basima.elshoqeirat@postgrad.curtin.edu.au). S. Soh, Department of Computing, Curtin University, Perth, Western Australia, phone: +61 8 9266 2984; fax: +61 8 9266 2819 (e-mail: S.Soh@curtin.edu.au).
S. Rai, Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, USA. (e-mail: srai@lsu.edu).
M. Lazarescu, Department of Computing, Curtin University, Perth, Western Australia (e-mail: M.Lazarescu@curtin.edu.au).

SP	Swarm Particle
SA	Simulated Annealing
TS	Tabu Search
ACO	Ant Colony Optimization
DP	Dynamic Programming
DPCR-ST	Dynamic Programming Algorithm to solve NTD-CR
CN	Communication Network
BDD	Binary Decision Diagram
ACO-SA	Ant Colony Optimization and Simulated Annealing
LS-NGA	Local Search Genetic Algorithm
MC	Monte Carlo Simulation
DPA	Dynamic Programming Algorithm

NOTATION

G=(V, E)	A graph/network with |V| nodes and |E| links
v_i (e_j)	Node *i* (Link *j*) in graph G, where $v_i \in V$ ($e_j \in E$)
c_j (r_j)	Cost (Reliability) of e_j, $c_j>0$ ($0 \leq r_j \leq 1.0$)
R_{min}	Reliability constraint used in NTD-CR
ST_G	A set containing all spanning trees in G
n	Total number of spanning trees in G, $n=\|ST_G\|$
k	The total number of spanning trees in graph G that are used by DPCR-ST to produce its result; $k \leq n$
ST_i	A spanning tree *i*, for *i*=1, 2, …, *n*; $ST_i \in ST_G$
L_i	A set of links in $ST_i \in ST_G$
STX_i	A sequence of spanning trees $STX_i \subseteq (ST_1, ST_2, \ldots, ST_i)$
$G_i=(V, E_i \subseteq E)$	A subgraph of G=(V, E) constructed from all links of spanning trees in STX_i. This paper uses STX_i and G_i interchangeably
Link(•), Cost(•), Rel(•)	Functions that return all links in •, total cost of all links in •, and reliability of network that contains all links in •, respectively, where • can be presented as a graph G, a spanning tree ST_i, or as sequence of spanning trees STX_i, or the union of $\{ST[i-1, c]\cup\{ST_i\}$
G_{min}	The topology that has minimum cost among all possible topologies with $Rel(G_{min}) \geq R_{min}$. Note that $Cost(G_n) \geq Cost(G_{min})$
$\check{R}_{min}$	$\check{R}_{min}=round(\delta \times R_{min})$, for a positive integer multiplier δ and a function round(•) that returns the closest integer value of •. In this paper, we set δ=100, and thus $\check{R}_{min} \leq 100$
$DP[i, \check{r}]$	Element in row *i*=1, 2, …, *n*, and column $\check{r}$=0, 1, …, $\check{R}_{min}$, of Dynamic Programming (DP) table that stores five pieces of information: $C[i, \check{r}]$, $R[i, \check{r}]$, $STX[i, \check{r}]$, $L[i, \check{r}]$, and $J[i, \check{r}]$
r	The reliability constraint for each column $\check{r}$, calculated as $r=\check{r}/\delta$
$C[i, \check{r}]$	The cost of G_i subject to $Rel(G_i) \geq r$; $C[i, \check{r}]=\infty$ if $Rel(G_i)<r$. We aim to find a topology with the minimum $C[n, \check{R}_{min}]$, called G_{min} or STX_{min}
$R[i, \check{r}]$	The reliability of a selected STX_i with $Rel(G_i) \geq r$. $R[i, \check{r}]=0$ if $Rel(G_i)<r$
$STX[i, \check{r}]$	A data structure that stores the spanning trees in STX_i with $Rel(G_i) \geq r$. $STX[i, \check{r}]=\{\}$ if $Rel(G_i)<r$
$L[i, \check{r}]$	The set of links in E_i with $Rel(G_i) \geq r$. $L[i, \check{r}]=\{\}$ if $Rel(G_i)<r$

$J[i, \check{r}]$	An integer index that marks the ending column $\check{r}$ of a range of columns that have the same reliability of r
C_{min}	The minimum cost of each topology with reliability at least R_{min}
C_{total}	Total link cost which is calculated for each network after assigning each link cost using its cost matrix
C_{best}	The *minimum* among the costs of topologies generated using DPCR-ST with three heuristic order techniques

I. INTRODUCTION

Some critical applications (*e.g.*, emergency system, rescue, and military operations) must run on a network topology with a guaranteed minimum reliability so that they can operate without interruption, even in the presence of component failures [1]. Constructing a more reliable topology, however, requires higher cost, and thus its design aims to minimize the network installation *cost* (C) subject to the required *reliability* (R) level; we call this situation the *network topology design with cost objective and reliability constraint* (NTD-CR) problem. Specifically, given (a) locations of the various computer centers (nodes), (b) their connecting links, (c) each link's reliability and cost, and (d) the required operational reliability for the network, NTD-CR selects the most suitable set of links such that the resulting model meets its required all-terminal reliability [2] while minimizing its installation cost.

The NTD-CR problem has been shown to be NP-hard [3]; thus, one must often use heuristics or approximation solutions to design large sized topologies. There are many proposed techniques [4]-[22] that find optimal or approximately optimal solutions for the NTD-CR problem. Branch & Bound (B&B) techniques [5], [6], used to generate optimal solutions, are applicable only for designing small topologies due to the difficult nature of the problem. The existing algorithms that generate approximation solutions are mainly based on meta heuristic techniques, such as Genetic Algorithm (GA) [8]-[10], [12], Neural Network (NN) [4], Swarm Particle (SP) [15], Simulated Annealing (SA) [7], [16], Tabu Search (TS) [17], and Ant Colony

Optimization (ACO) [13]. While the meta heuristic-based algorithms may significantly reduce time complexity, they still require numerous iterations to converge, and thus use a considerable computational effort while producing only up to 48 out of 76, or 63.1%, optimal solutions [13]. Therefore, a more time efficient heuristic approach that can produce better results is still needed, especially for use in large scale networks.

There are two main contributions in this work. First, it proposes a *dynamic programming* (DP) formulation, and its algorithm DPCR-ST, to solve the NTD-CR problem. The algorithm is shown to always produce a feasible solution for the problem. Further, DPCR-ST will produce an optimal topology given a sequence of optimally ordered spanning trees, defined in Section IV.D.2 and IV.D.3. However, this paper shows that generating the optimal sequence is an NP-complete problem. Second, this paper proposes three different heuristics to compute only *k* spanning trees, which can be used by DPCR-ST to significantly reduce its time complexity. Extensive simulations using various fully connected networks containing up to 2.3×10^9 spanning trees show that this efficient approach produced 81.5% optimal results, which is significantly better than four existing state-of-the-art approaches [8], [9], [13], [14] while using only 0.77% of the spanning trees contained in networks. Furthermore, simulations on various grid networks with up to 200 nodes, 298 links, and 1.899^{102} spanning trees show the practicality of our techniques in designing larger network topologies. Note that a preliminary version of this paper appeared in [18].

The layout of this paper is as follows. Section II discusses the network model and notations. Section III formulates the NTD-CR problem, and describes its related problems and existing solutions. Section IV presents our proposed solution, and its theoretical analysis, while Section V provides simulation results to show its effectiveness and efficiency. Finally, Section VI concludes the paper with a discussion on the future work.

II. NETWORK MODEL

A *communication network* (CN) can be modeled by a probabilistic bidirectional simple graph G=(V, E), in which each vertex (node) $v_i \in V$ represents a computer center and each link $e_j \in E$ represents the connecting media (*e.g.*, cable, communication link) between the computer centers. All nodes' location and connecting links are known when the centers are established.

Further, we consider each center has sufficient fault-tolerance and backup components, which allow the center to continue its operation when there is any component failure. Thus, the paper assumes all nodes are always functioning. While we realize there is much complexity ignored here, we are focusing on the external network problem, not on the very complex nodes.

Each link e_j has a cost $c_j > 0$ that represents the cost to install e_j, and reliability $0 \leq r_j \leq 1.0$ that represents the probability that e_j is functioning. Link failures are assumed to be statistically independent and without repair. This assumption is used to reduce the combinatorial size of the already difficult problem [13]. Fig. 1 shows an example of the graph model of a network with four statically positioned nodes, and five links; Table I provides c_j and r_j values for each link e_j.

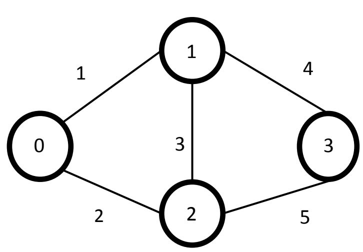


Fig. 1. An example network.

TABLE I
LINK WEIGHT AND SPANNING TREE SET FOR NETWORK IN FIG. 1

	ST_G			Link Weight		
i	ST_i	$Rel(ST_i)$	$Cost(ST_i)$	e_j	c_j	r_j
1	{2, 4, 5}	0.486	13	1	5	0.9

2	{1, 4, 5}	0.729	15	2	3	0.6
3	{1, 2, 4}	0.486	12	3	2	0.7
4	{2, 3, 5}	0.378	11	4	4	0.9
5	{1, 2, 5}	0.486	14	5	6	0.9
6	{2, 3, 4}	0.378	9			
7	{1, 3, 4}	0.567	11			
8	{1, 3, 5}	0.567	13			

A spanning tree *i*, ST_i, is a subgraph of G which is a tree, and composed all the vertices in G. Each spanning tree in a network with |V| nodes contains |V|-1 links. Let ST_G be a set of all spanning trees in G, $n=|ST_G|$, and L_i be the set of links in $ST_i \in ST_G$. Table I shows ST_G of the network in Fig. 1. Let Cost(ST_i) denote the cost of installing all links in spanning tree ST_i, calculated by taking the sum of c_j of each $e_j \in ST_i$. The cost of a network topology G, Cost(G), is calculated by taking the sum of all c_j for each $e_j \in G$. Let Rel(ST_i) denote the reliability of spanning tree ST_i, calculated by multiplying all r_j of each $e_j \in ST_i$. The network reliability of a topology G, Rel(G), is the probability that at least one ST_i in G is functional. In other words, it is the probability that a set of operational links provides a communication path between every pair of nodes. Calculating Rel(G) in general is an NP-hard problem [4], [23]; Section III.B provides details about calculating Rel(G). Notice that G can be constructed using nodes in V of G=(V, E), and all links in ST_G, and thus the paper uses Cost(ST_G)=Cost(G)=Cost(E), and Rel(ST_G)=Rel(G)=Rel(E). Similarly, Cost(ST_i)=Cost(L_i), and Rel(ST_i)=Rel(L_i).

III. Network Topology Design Problem and Solution

A. Network Topology Design (NTD-CR) Problem

Let Y_j be a decision variable {0, 1} that indicates if link e_j in G=(V, E) is selected (Y_j=1), or not selected (Y_j=0). The following two equations describe the NTD-CR problem.

$$\text{Minimize} \sum_{j=1}^{|E|} c_j \, Y_j \tag{1}$$

$$\text{Subject to Rel } (G_i=(V, E_i)) \geq R_{min} \tag{2}$$

Equation (1) calculates the minimum cost of a network topology G_i=(V, E_i) that contains links E_i=E–{e_j | Y_j=0}; *i.e.*, E_i is a set of selected links in (2) that form G_i that has a reliability of at least R_{min}. One may solve the NTD-CR problem by generating each possible set of links in (2) that form G_i. Then, calculate Cost(G_i) for each G_i that has reliability Rel(G_i)≥R_{min}, and select a G_i with the minimum cost as G_{min}. Unfortunately, this brute force solution, called BF-1 requires generating $2^{|E|}$ possible link selections, *i.e.*, the G_i. Further, the reliability calculation in (2) for each G_i requires exponential time; thus BF-1 is feasible only for designing small topologies.

As an alternative, let X_i be a decision variable {0, 1} that indicates if spanning tree ST_i in G=(V, E) is selected (X_i=1), or not selected (X_i=0). The following equations describe the NTD-CR problem.

$$\text{Minimize Cost}\left(\bigcup_{i=1}^{|ST_G|} ST_i\, X_i\right) \tag{3}$$

$$\text{Subject to Rel}\left(\bigcup_{i=1}^{|ST_G|} ST_i\, X_i\right) \geq R_{min} \tag{4}$$

Equation (3) calculates the minimum cost of the network containing only the selected spanning trees ST_i from (4). One may generate all 2^n possible combinations of spanning trees that meet the constraint in (4). Then, for each combination that has a reliability of at least R_{min}, use (3) to calculate its cost, and select a combination with the minimum cost as its G_{min}. This solution, henceforth called BF-2, is also prohibitive for use in large networks because a general network contains $n=O(|V|^{|V|})$ spanning trees [24]. In Section IV.A, we propose a DP approach to solve (3) and (4).

To illustrate the NTD-CR problem, consider the network in Fig. 1. For R_{min}=0.87, Fig. 2 shows the optimal network, G_{min}, whose links form a set of spanning trees {{2, 4, 5}, {1, 4, 5}, {1, 2, 4}, {1, 2, 5}} with Rel(G_{min})=0.88, and Cost(G_{min})=18; G_{min} does not contain spanning trees {1, 3, 4}, {1, 3, 5}, {2, 3, 5}, and {2, 3, 4} because link 3 is not selected.

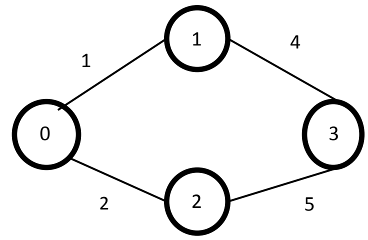


Fig. 2. Optimal solution for network in Fig. 1.

Notice that Rel(G)=0.927; and therefore, if we set R_{min}=0.927, (1) would have Y_j=1 for each e_j, and (3) considers all spanning trees in G. Thus, for this case, (1) and (3) produce G_{min}=G with a reliability of 0.927. In contrast, (2) produces a reliability of 0 if the selected links in (1) do not form any spanning tree.

B. Related Works

The existing solutions in the literature for the NTD-CR problem aim to generate either an optimal topology or an approximated topology from a given network G. The existing approximation methods mainly use Meta-heuristic techniques, *e.g.*, GA [8]-[10], [12], NN [4], SA [7], [16], TS [17], SP [15], and ACO [13]. Each NTD-CR solution requires calculating all-terminal reliability to be compared with the required reliability constraint R_{min}. The calculation can use either a simulation or an analytic method. The analytic method [25] produces an exact reliability result. However, its time complexity grows exponentially in the order of network size, and thus approaches that use the method, *e.g.*, [5], [6], are suitable only for use in small sized networks. The simulation methods for reliability calculation [23], [26]-[28] reduce the time complexity, but produce only estimated reliability values, acceptable for the heuristic solutions to NTD-CR because the values are used only to test for the feasibility of the resulting topology.

Jan *et al.* [5] considered a network G whose links have the same reliability values, and developed an algorithm that combines decomposition, B&B techniques to find an optimal solution, *i.e.*, a topology G_{min} with a minimum Cost(G_{min}), and a Rel(G_{min})≥R_{min}. Later, Koide *et* *al.* [6] generalized the problem in [5] for graph G with non-homogeneous link reliabilities, and developed another B&B algorithm to solve the problem. The B&B approaches [5], [6] are computationally expensive, and thus are suitable only for small sized networks with up to nine nodes.

Kumar *et al.* [19] have developed a GA-based approach to solve NTD-CR that includes two additional constraints, *i.e.*, diameter and average distance, and applied it to four test networks with up to nine nodes. Their approach [19] calculates the network reliability exactly. Although the problem in [19] is a superset of NTD-CR, its solution cannot be used to solve the NTD-CR problem because the problem considers only links with identical reliability and cost. Deeter & Smith [10] presented a GA approach to solve the NTD-CR problem. However, their solution considers alternative link reliabilities, and thus cannot be used to solve our NTD-CR problem in which each link may have different reliability values.

Dengiz *et al*. [8] proposed a heuristic GA approach, called NGA, to solve the NTD-CR problem. In [9], the same authors have developed another GA-based solution, called Local search GA (LS-NGA), using a special encoding structure, crossover, and mutation operators. Both techniques have been shown effective in generating near optimal solutions [8], [9]. These GA methods yield poor quality solutions for networks with more than 10 nodes [12]. Later, Gen [20] proposed a self-controlled GA to solve the NTD-CR problem. However, these GA methods [8], [9], [20] require the development, coding, and testing of a problem-specific GA, complicating the solution process. Mutawa *et al.* [29] proposed a steady-state GA, and Shao *et al.* [30] proposed an algorithm, called a shrinking and searching algorithm, to maximize network reliability under a cost constraint, which is a related NTD-CR problem, discussed later in this Section.

Marquez and Rocco [12] have presented a population-based heuristic approach called the probabilistic solution discovery algorithm. However, their approach is shown less effective compared to the more recent approach in [13].

Abo Elfotoh and Al-Sumait [4] developed a NN heuristic algorithm, and the authors in [21] used an artificial NN for the NTD-CR problem. As stated in [12], while the NN and artificial NN algorithms produce good results, they use a long procedure that needs extensive time and significant parameter tuning [12]. A deterministic version of SA was used by Atiqullah and Rao [7] to find the optimal design of small networks, *i.e.*, five nodes or less. Pierre *et al.* [16] also used SA to find optimal designs for packet switch networks where delay and capacity were considered, but reliability was not. Recently, a new metaheuristic called Cross-Entropy method [11] was developed for the NTD-CR problem. In addition, the Multiple TS algorithm [22] was used to solve the NTD-CR problem with 19 nodes; however, the algorithm may not reach the global optimum solution in a reasonable computation time when the initial solution is far away from the region where the optimum solution exists. In [14], a Binary Decision Diagram (BDD) is used to solve the same design problem for networks containing up to 81 nodes. This approach is based on a decomposition of Boolean functions called *Shannon decomposition*. BDD structure is a compact, implicit representation of the entire set of the functioning and failing network states. Our simulations in Section V.B show that DPCR-ST produce better results as compared to the BDD approach in [14].

Altiparmak *et al.* [13] proposed a hybrid approach based on Ant Colony Optimization and Simulated Annealing, called ACO-SA, for the NTD-CR problem for networks with up to 50 nodes. ACO-SA first generates a ring network, onto which it adds some links to produce a seed network topology that satisfies the reliability constraint; initially, the seed is considered the best global topology. Then, it uses ACO and SA to reduce the cost of the global topology. If a population in ACO contains better networks than the global topology, then these topologies are put in a set, and SA is used as a local search procedure to further improve those networks in the set to generate the best possible topology. ACO-SA repeats these procedures until a given stopping criterion is met. Our simulations, described in Section V.B, show that DPCR-ST outperforms ACO-SA.

In general, population based metaheuristics such as GA require numerous iterations before converging. Note that, for the NTD-CR problem, each iteration includes reliability computation of the approximated topology to be tested against the required constraint. Because reliability evaluation, using both exact or approximation methods, is computationally expensive (a typical Monte Carlo (MC) simulation iterates 10^6 times), the approach in general uses a considerable computational effort. For example, the GA based approach in [31] uses MC simulation to calculate the reliability of each candidate solution, and thus, for a typical GA solution with a population size of 6000 and 40 generations, it needs to run MC 240000 times. Therefore, the approaches are computationally expensive for use in large networks. As described in Section IV, our proposed approach uses a MC simulation [26] to estimate the reliability value of each topology, and runs the simulation only up to $\delta \times k$ times while producing 81.5% optimal results, for δ=100, and $k \leq 1214$.

The authors of [32], [33] have proposed a *dynamic programming algorithm* (DPA) to solve a related NTD problem, called NTD-RC, to construct a topology that maximized 2-terminal (all-terminal) reliability subject to a cost budget constraint. However, DPA cannot be used directly to solve the NTD-CR problem because the two related problems require two different dynamic programming formulations, and thus need different solutions. Further, to maximize the reliability, for each entry of a dynamic programming table, the NTD-RC problem needs to compute an exact reliability value, which is a very time consuming step because computing the reliability, in general, is known to be NP-hard [25]. On the other hand, to minimize network cost, the NTC-CR problem needs to generate only an approximated reliability, which can be solved using a significantly faster heuristic technique such as Monte Carlo Simulation [26].

IV. PROPOSED DYNAMIC PROGRAMMING-BASED SOLUTION

A. Dynamic Programming Formulation for NTD-CR

Let STX_i, for i=1, 2, …, n-1, n, be a sequence of spanning trees selected from i spanning trees in (ST_1, ST_2, …, ST_i), and $G_i=(V, E_i \subseteq E)$ be an induced graph whose links comprise of all links in STX_i. In this paper, we use STX_i and its G_i interchangeably because one can generate G_i from STX_i, and vice versa. Note that $0 \leq |STX_i| \leq i$, and there are 2^n different STX_i; we aim to select STX_n with a reliability of at least R_{min}, and the minimum cost, *i.e.*, $Rel(G_n) \geq R_{min}$ and minimum $Cost(G_n)$. An STX_i forms a *feasible* solution or topology if its reliability $Rel(STX_i) \geq R_{min}$; otherwise, it is a *non-feasible* solution. For Fig. 1 with R_{min}=0.82, STX_8=(ST_2, ST_7, ST_8) is a feasible solution because $Rel(STX_8)=0.841 \geq R_{min}$. Two sets of spanning trees STX_i and STX_j are *equivalent* if they contain the same links that form the same topology, and thus they contain the same set of spanning trees, and have the same reliability and cost. For Fig. 1, STX_2=(ST_1, ST_2) and STX_3=(ST_1, ST_3) are equivalent because both form the same topology.

Let DP[1 .. n, 0 .. $\check{R}_{min}$] be a 2-dimensional DP table, where $\check{R}_{min}=round(\delta \times R_{min})$, for a positive integer multiplier δ, and a function $round(\alpha)$ that returns the closest integer value of α. For example, the function returns $\check{R}_{min}$=92 ($\check{R}_{min}$=93) when we set δ=100, and R_{min}=0.9216 (R_{min}=0.9261).

Each element DP[i, $\check{r}$], for i=1, 2, …, n, $\check{r}$=0, 1, 2, …, $\check{R}_{min}$, stores five pieces of information: a cost $C[i, \check{r}]>0$, a reliability $0 \leq R[i, \check{r}] \leq 1.0$, $STX[i, \check{r}] \subseteq ST_G$, a set of links $L[i, \check{r}] \subseteq E$, and an integer index $0 \leq J[i, \check{r}] \leq \delta$. In essence, the columns of the DP table partition the reliability constraint R_{min} into δ consecutive reliability contraints, *i.e.*, R_{min}/δ, $(2 \times R_{min})/\delta$, …,

$(\delta\times R_{min})/\delta=R_{min}$. Specifically, each column index $\check{r}=0, 1, \ldots, \check{R}_{min}$, corresponds to a reliability constraint $r=0, 1/\delta, \ldots, (\check{R}_{min}/\delta)\approx R_{min}$, *i.e.*, $r=\check{r}/\delta$ and $\check{r}=round(\delta\times r)$, and each DP[$i$, $\check{r}$] is used to store four pieces of information for each selected topology G_i that has $Rel(G_i)\geq r$. Specifically, for each $Rel(G_i)\geq r$, we set $C[i, \check{r}]=Cost(G_i)$, $R[i, \check{r}]=Rel(G_i)$, $STX[i, \check{r}]=STX_i$, and $L[i, \check{r}]=E_i$. For $Rel(G_i)<r$, we set $C[i, \check{r}]=\infty$, $R[i, \check{r}]=0$, $STX[i, \check{r}]=\{\}$, and $L[i, \check{r}]=\{\}$. Note that $C[i, \check{r}]=0$ is not possible because each link is assumed to have a non-zero cost. As $C[n, \check{R}_{min}]$ is the cost of $G_n=(V, E_n\subseteq E)$ with $Rel(G_n)\geq R_{min}$, NTD-CR aims to generate DP[n, $\check{R}_{min}$] that contains the minimum $C[n, \check{R}_{min}]$, which represent the G_{min}.

For each range of columns $\check{r}_1\leq\check{r}\leq\check{r}_2$ in row i that contain the same reliability value, we set each $\check{r}_\Delta=J[i, \check{r}]=\check{r}_2$. Thus, index $\check{r}_\Delta=J[i, \check{r}]=0, 1, 2, \ldots, \check{R}_{min}$ marks the ending column of a range of columns that have the same reliability. For example, as later shown in Table II, we store $\check{r}_\Delta=J[1, \check{r}]=49$ at columns $\check{r}=0$ to $\check{r}=49$ because $R[1, 0]=R[1, 1]= \ldots =R[1, 49]$. Note that we set $\check{r}_\Delta=J[i, \check{r}]=\check{r}$ when $\check{r}_1=\check{r}_2$, *i.e.*, when the length of the range is one. Our DP approach computes each $C[i, \check{r}]$ using the following four equations.

$$C[i, \check{r}]=Cost(ST_i) \text{ for } i=1 \text{ with } Rel(ST_i)\geq r \tag{5}$$

$$C[i, \check{r}]=\infty \text{ for } i=1 \text{ with } Rel(ST_i)<r \tag{6}$$

$$C[i, \check{r}]=Min(C[i-1, \check{r}], Cost(ST_i)) \text{ for } i>1, \text{ and } Rel(ST_i)\geq r \tag{7}$$

$$C[i, \check{r}]=Min(C[i-1, \check{r}], Cost(L[i-1, \check{r}_\Delta] \cup L_i)) \text{ for } i>1,\ \check{r}_\Delta\leq\check{r}, \text{ and } Rel(L[i-1, \check{r}_\Delta] \cup L_i)\geq r \tag{8}$$

We explain the DP formulation in (5)-(8) as follows. In (5), when the first spanning tree has a reliability of at least r, it should be selected, giving $C[1, \check{r}]=Cost(ST_1)$. In contrast, when $Rel(ST_1)<r$, ST_1 is not selected because it does not meet the reliability constraint r; thus (6) sets $C[1, \check{r}]=\infty$ to denote that no spanning tree is selected.

Equations (7) and (8) are used for each remaining ST_i, for $i=2, 3, \ldots, n$. Equation (7) considers two options, selecting or not selecting ST_i, when $Rel(ST_i)\geq r$, and selects the option that produces the minimum cost. Specifically, when ST_i is selected (not selected), its cost is Cost(ST_i) (C[i-1, $\check{r}$]), and the equation selects the minimum between the two because both options satisfy the reliability requirement r. Note that the reliability value in the element would be changed to Rel(ST_i) if ST_i is selected. Further, (7) considers a situation when no trees have been selected for column $\check{r}$, *i.e.*, C[i-1, $\check{r}$]=∞, and R[i-1, $\check{r}$]=0, in which case it will select ST_i.

Equation (8) considers the case when selecting ST_i together with some previous sequence of selected trees that satisfies the required reliability r, *i.e.*, Rel(L[i-1, $\check{r}_\Delta$] ∪ L_i)≥r, for each possible $\check{r}_\Delta$=J[i-1, $\check{r}$]=0, 1, …, $\check{R}_{min}$. Like (7), (8) also considers the minimum cost between either selecting or not selecting ST_i; the former produces Cost(L[i-1, $\check{r}_\Delta$] ∪ L_i), and the latter produces C[i-1, $\check{r}$]. Specifically, when ST_i is selected (not selected), the cost is calculated from the selected spanning trees STX_i, (STX_{i-1}). Note that the reliability value in the column would be changed to Rel(L[i-1, $\check{r}_\Delta$] ∪ L_i) if ST_i is selected. Further, (8) also considers a situation when no trees have been selected for column $\check{r}$, *i.e.*, C[i-1, $\check{r}$]=∞, and R[i-1, $\check{r}$]=0, in which case it will select ST_i.

The DP formulation in (5)-(8) is similar to the DP solution for the well-known NP-complete 0-1 knapsack problem [34]. In the 0-1 knapsack problem [34], there are n items (spanning trees in NTD-CR), where each item i has weight x_i (Rel(ST_i) in NTD-CR) and value v_i (Cost(ST_i) in NTD-CR), and its goal is to select a set of items that have the *maximum* total value (*minimum* cost in NTD-CR as stated in (3)) while having a total weight of no more than a given weight constraint X_{max} (R_{min} in NTD-CR as stated in (4)). However, unlike for the 0-1 knapsack problem where the total cost of two items is the sum of each item's cost, in NTD-CR, Cost(ST_i)+Cost(ST_p)≥Cost(ST_i ∪ ST_p) because ST_i and ST_p may contain common links. Therefore (8) must consider all possible values of $\check{r}_\Delta$, *i.e.*, J[i, $\check{r}$]. Further, while the total capacity of two items in the 0-1 Knapsack problem equals the sum of each item's capacity, in NTD-CR, Rel(ST_i)+Rel(ST_p)≠Rel(ST_i ∪ ST_p), and Rel(ST_i)>Rel(ST_p) does not always mean Rel(ST_h ∪

ST_i)>Rel($ST_h \cup ST_p$), for any ST_h. Therefore, each C[i, $\check{r}$] is not necessarily minimum, even when it is computed from two optimal sub problems. In Section IV.D.2, we will show that our heuristic DP solution generates an optimal topology when it uses an optimal order of spanning trees, defined later, as its input.

B. DPCR-ST Algorithm

The DPCR-ST algorithm shows our proposed DP algorithm, which directly applies (5)-(8). For a G=(V, E) that contains n spanning trees with reliability constraint R_{min}, DPCR-ST *implicitly* constructs a DP table of size $n \times \check{R}_{min}$. DPCR-ST keeps only two consecutive rows, called *row*1 and *row*2, and therefore it requires only a table of size $2 \times \check{R}_{min}$. Specifically, DPCR-ST computes C[2, $\check{r}_\Delta$] and R[2, $\check{r}_\Delta$] in *row*2 using the information in C[1, $\check{r}$] and R[1, $\check{r}$] in *row*1, for all relevant columns $\check{r}$ and $\check{r}_\Delta$.

DPCR-ST Algorithm

```
Initialize C[1, ř]=∞, R[1, ř]=0, STX[1, ř]={}, L[1, ř]={}, J[1, ř]=Ř_min, for  Rel(ST_1)<r //  (6)
for (ř← 0 to round(δ×Rel(ST_1)) do // (5)
    C[1, ř] ← Cost(ST_1)
    R[1, ř] ← Rel(ST_1)
    STX[1, ř] ← ST_1
    L[1, ř] ← L_1
    J[1, ř] ← round(δ×R[1, ř])
end for ř
 Copy  row1 to row2
 for (i ← 2 to n) do   //  (7)-(8)
     for (ř ← 0 to round(δ×Rel(ST_i )) do // (7)
         C[2, ř] ← Min(C[1, ř], Cost (ST_i))
         if C[2, ř]<Cost(ST_i)
             STX[2, ř] ← STX[1, ř]
             L[2, ř] ← L[1, ř]
         else
             STX[2, ř] ← ST_i
             L[2, ř] ← L_i
         end if
         R[2, ř] ← Rel(L[2, ř])
         J[2, ř] ← round(δ×R[2, ř])
     end for ř
     for (y ← 0 to Ř_min) do // (8)
```

```
24.         if (J[1, y] ≠ J[1, y-1])
25.             ř∆=J[1, y]
26.             if  Rel(L[1,  ř∆] ∪ Li)≥ ř
27.                 C[2, ř] ← Min(C[1, ř], Cost (L[1,  ř∆] ∪ {Li}))
28.                 if C[2, ř]<Cost(L[1,  ř∆] ∪ Li)
29.                     STX[2, ř] ← STX[1, ř]
30.                     L[2, ř] ← L[1, ř]
31.                 else
32.                     STX[2, ř] ← STX[1,  ř∆] ∪ STi
33.                     L[2, ř] ← L[1,  ř∆] ∪ Li
34.                 end if
35.                 R[2, ř] ← Rel(L[2, ř])
36.                 J[2, ř] ← round(δ×R[2, ř])
37.             end if
38.         end if
39.     end  for y
40.     Copy  row2 to row1
41. end for i
```

After copying the contents of *row*1 to *row*2, it repeats the step until all spanning trees are considered. In this paper, we set the integer multiplier δ, described in Section IV.A, to 100, and thus the DP table contains no more than 101 columns. Line 1 implements (6), while Lines 2 through 8 are based on (5). The remainder of the code is used to implement (7) and (8). Specifically, (7) is solved in Lines 11 through 22, (8) in Lines 23 through 39, and Line 40 copies the contents of *row*1 to *row*2

C. Illustrating Example

To illustrate the DPCR-ST algorithm, consider the network in Fig. 1, and R_{min}=0.87. Table I shows the network's link reliability and cost, and spanning tree reliability and cost. Our DPCR-ST algorithm constructs the DP table shown in Table II, and obtains the optimal topology in Fig. 2. For convenience, we show all eight rows, although our implementation creates only two rows. Each row of the table considers a ST_i for possible selection, and its columns are labeled by reliability values from 0 to $\check{R}_{min}$. Due to space limitation, Table II shows only a range of the reliability values $\check{r}$.

Because Rel(ST_1)=0.486, and thus $\check{r}$=49, lines 2 through 8 in the DPCR-ST algorithm set C[1, $\check{r}$]=13, R[1, $\check{r}$]=0.49, STX[1, $\check{r}$]=(ST_1), and L[1, $\check{r}$]={2, 4, 5} for $\check{r}$=0, …, 49 with $\check{r}_{\Delta}$=*J*[1, $\check{r}$]=49; Fig. 3(a) shows the graph representation for STX[1, $\check{r}$]=(ST_1). Further, (6) initializes the first row with C[1, $\check{r}$]=∞, R[1, $\check{r}$]=0, STX[1, $\check{r}$]={}, and L[1, $\check{r}$]={} for $\check{r}$=50, …, $\check{R}_{min}$; and thus *J*[1, $\check{r}$]=$\check{R}_{min}$=87.

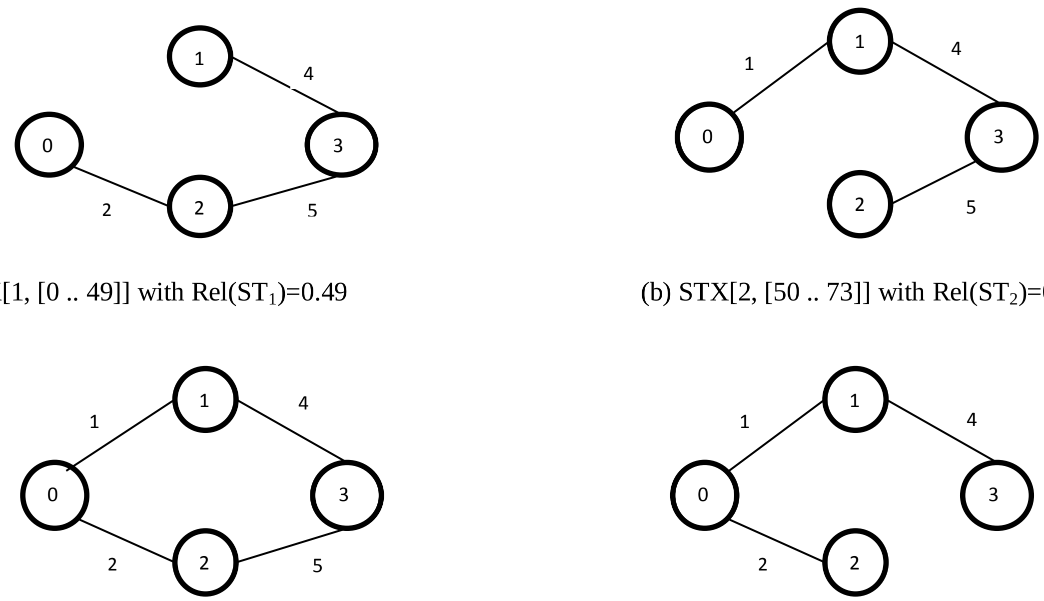
Next, for ST_2 with Rel(ST_2)=0.729, and thus $\check{r}$=73, (7) produces C[2, $\check{r}$]=C[1, $\check{r}$], R[2, $\check{r}$]=R[1, $\check{r}$], STX[2, $\check{r}$]=STX[1, $\check{r}$], and L[2, $\check{r}$]=L[1, $\check{r}$], because C[1, $\check{r}$]=13<Cost(ST_2)=15; thus this step keeps the non-feasible topology shown in Fig. 3(a) for each column $\check{r}$=0, …, 49 in the row. In contrast, for $\check{r}$=50, …, 73, because C[1, $\check{r}$]=∞, (7) sets C[2, $\check{r}$]=15, R[2, $\check{r}$]=0.73, STX[2, $\check{r}$]=(ST_2), L[2, $\check{r}$]={1, 4, 5}, and *J*[2, $\check{r}$]=73, Fig. 3(b) shows the graph representation for STX[2, $\check{r}$]=(ST_2). For $\check{r}$=74, …, $\check{R}_{min}$, selecting ST_2 for each $\check{r}$ in the range is feasible. For this case, we consider $\check{r}_{\Delta}$=*J*[1, $\check{r}$] in *row* 1 that has two possible values, *i.e.*, $\check{r}_{\Delta}$=49, and $\check{r}_{\Delta}$=87 for $\check{r}$=0, …, 49, and $\check{r}$=50, …, $\check{R}_{min}$=87, respectively. For $\check{r}_{\Delta}$=49, Rel((L[*i*-1, $\check{r}_{\Delta}$=49]=L_1) ∪ L_2)=0.88; thus (8) selects both ST_2 and ST_1 at column $\check{r}$=74, ..., $\check{R}_{min}$, and sets C[2, $\check{r}$]=18, R[2, $\check{r}$]=0.88, STX[2, $\check{r}$]=(ST_1, ST_2), L[2, $\check{r}$]={1, 2, 4, 5}, and $\check{r}_{\Delta}$=87. For this case, (8) produces a feasible topology for each column in the range; Fig. 3(c) shows its graph representation. Note that (8) does not select both ST_1 and ST_2 at column $\check{r}$=0, ..., 73, because Cost(L_1 ∪ L_2)=18>C[1, $\check{r}$]=13. For $\check{r}_{\Delta}$=87, we obtain Rel(L[*i*-1, $\check{r}_{\Delta}$]={} ∪ ST_2={1, 4, 5})=0.729, and thus $\check{r}$=73, which is less than each $\check{r}$=74, ..., $\check{R}_{min}$ under consideration. Therefore, (8) produces exactly the same results at columns $\check{r}$=50, ..., 73 as previously obtained using (7).

As another example, for ST_3 with Rel(ST_3)=0.486, and thus $\check{r}$=49, for $\check{r}$=0, ..., 49, (7) selects ST_3, and sets C[3, $\check{r}$]=12, R[3, $\check{r}$]=0.49, STX[3, $\check{r}$]=(ST_3), L[3, $\check{r}$]={1, 2, 4}, and *J*[3, $\check{r}$]=49, because Cost(ST_3)=12<C[2, $\check{r}$]=13; Fig. 3(d) shows the graph representation for STX[3, $\check{r}$]=(ST_3). For (8), there are three possible values for $\check{r}_{\Delta}$=*J*[2, $\check{r}$], *i.e.*, $\check{r}_{\Delta}$=49, $\check{r}_{\Delta}$=73, and $\check{r}_{\Delta}$=87 for

$\check{r}$=0, …, 49, $\check{r}$=50, …, 73, and $\check{r}$=74, …, 87, respectively. For $\check{r}_\Delta$=49, Rel((L[i-1, $\check{r}_\Delta$=49]=L_1) ∪ L_3)=0.88, and thus (8) is used for $\check{r}$=0, …, 87 at row 3. Because Cost(L_1 ∪ L_3)=18>C[2, $\check{r}$]=13 for $\check{r}$=0, …, 49, and Cost(L_1 ∪ L_3)=18>C[2, $\check{r}$]=15 for $\check{r}$=50, …, 73, (8) does not update the DP table for $\check{r}$=0, …, 73.

TABLE II
DP TABLE FOR NETWORK IN FIG. 1

Column / STi	$\check{r}$=0, …, 38	$\check{r}$=39, …, 49	$\check{r}$=50, …, 57	$\check{r}$=58, …, 73	$\check{r}$=74, 75	$\check{r}$=76, …, 85	$\check{r}$=86, 87
ST_1	C[1, $\check{r}$]=13 R[1, $\check{r}$]=0.49 STX[1, $\check{r}$]=(1) L[1,$\check{r}$]={2,4,5} J[1, $\check{r}$]=49	13 0.49 (1) {2,4,5} 49	∞ 0 {} {} 87	∞ 0 {} {} 87	∞ 0 {} {} 87	∞ 0 {} {} 87	∞ 0 {} {} 87
ST_2	13 0.49 (1) {2,4,5} 49	13 0.49 (1) {2,4,5} 49	15 0.73 (2) {1,4,5} 73	15 0.73 (2) {1,4,5} 73	18 0.88 (1,2) {1,2,4,5} 87	18 0.88 (1,2) {1,2,4,5} 87	18 0.88 (1,2) {1,2,4,5} 87
ST_3	12 0.49 (3) {1,2,4} 49	12 0.49 (3) {1,2,4} 49	15 0.73 (2) {1,4,5} 73	15 0.73 (2) {1,4,5} 73	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87
ST_4	11 0.38 (4) {2,3,5} 38	12 0.49 (3) {1,2,4} 49	15 0.73 (2) {1,4,5} 73	15 0.73 (2) {1,4,5} 73	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87
ST_5	11 0.38 (4) {2,3,5} 38	12 0.49 (3) {1,4,5} 49	15 0.73 (2) {1,4,5} 73	15 0.73 (2) {1,4,5} 73	16 0.75 (4,5) {1,2,3,5} 75	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87
ST_6	9 0.38 (6) {2,3,4} 38	12 0.49 (3) {1,2,4} 49	14 0.75 (3,6) {1,2,3,4} 75	14 0.75 (3,6) {1,2,3,4} 75	14 0.75 (3,6) {1,2,3,4} 75	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87
ST_7	9 0.38 (6) {2,3,4} 38	11 0.57 (7) {1,3,4} 57	11 0.57 (7) {1,3,4} 57	14 0.75 (6,7) {1,2,3,4} 75	14 0.75 (6,7) {1,2,3,4} 75	18 0.88 (1,3) {1,2,4,5} 87	18 0.88 (1,3) {1,2,4,5} 87
ST_8	9 0.38 (6) {2,3,4} 38	11 0.57 (7) {1,3,4} 57	11 0.57 (7) {1,3,4} 57	14 0.75 (6,7) {1,2,3,4} 75	14 0.75 (6,7) {1,2,3,4} 75	17 0.85 (7,8) {1,3,4,5} 85	18 0.88 (1,3) {1,2,4,5} 87



(a) STX[1, [0 .. 49]] with Rel(ST_1)=0.49 (b) STX[2, [50 .. 73]] with Rel(ST_2)=0.73

(c) STX[2, [74 .. 87]]=STX[3, [74 .. 87]] with Rel(STX_3)=0.88 (d) STX[3, [0 .. 49]] with Rel(ST_3)=0.49

Fig. 3. Examples of Graph Representations for STX[i=[1 .. n=8], $\check{r}$=[0 .. 87]].

On the other hand, Cost($L_1 \cup L_3$)=C[2, $\check{r}$]=18 for $\check{r}$=74, ..., 87, and therefore the equation includes ST_3 to the selected trees in the previous row, *i.e.*, (ST_1), (ST_2), and (ST_1, ST_2) for columns $\check{r}$=0, ..., 49, $\check{r}$=50, ..., 73, and $\check{r}$=74, ..., 87 respectively. For this case, (8) produces (ST_1, ST_3), (ST_2, ST_3), and (ST_1, ST_2, ST_3) for columns $\check{r}$=0, ..., 49, $\check{r}$=50, ..., 73, and $\check{r}$=74, ..., 87 respectively. Notice that the results, *i.e.*, (ST_1, ST_3), (ST_2, ST_3), (ST_1, ST_2, ST_3), and (ST_1, ST_2) are equivalent feasible solutions with a reliability of 0.88, and cost of 18; the table shows one of the solutions, *i.e.*, C[3, $\check{r}$]=18, R[3, $\check{r}$]=0.88, STX[3, $\check{r}$]=(ST_1, ST_3), L[3, $\check{r}$]={1, 2, 4, 5}, and *J*[3, $\check{r}$]=88 as illustrated in Fig. 3(c).

Repeating steps ST_4 through ST_8, DPCR-ST obtains STX[8, $\check{R}_{min}$]=(ST_1, ST_3) with reliability R[8, $\check{R}_{min}$]=0.88. The optimal topology G_{min}, shown in Fig. 3(c), equivalent to the optimal topology in Fig. 2, is obtained by selecting all links in the spanning trees (ST_1, ST_3).

D. DPCR-ST Analysis

D.1. Time Complexity

The time complexity of DPCR-ST can be computed as follows. The Cost(•) function used in the DPCR-ST algorithm requires all unique links in the set of spanning trees •. For each $\check{r}$, Cost(•) returns the sum of C[i-1, $\check{r}$], and the cost of links in ST_i that are not in L[i-1, $\check{r}$]. Using the bit implementation [21], one requires only one bit OR, and one bit XOR operation to obtain the links in ST_i that are not in L[i-1, $\check{r}$]; and thus, for any •, Cost(•) can be computed in $O(|E|)$. DPCR-ST uses the function at most once for every table entry, and therefore the worst case time complexity for using the function is $O(n\times|E|\times\check{R}_{min})$.

The Rel(•) function used in the DPCR-ST algorithm can be implemented using any exact reliability calculation [25], heuristic technique [8], [11], or approximation (bounding) method [9]. In this paper, we use a MC simulation [26] with time complexity $O(b\times|V|^4)$ to estimate the Rel(•) of each candidate network; b is the number of replication. Notice that Rel(•) is used only for each different $\check{r}_\Delta$ in each row i. Hence, in total, the time complexity of using Rel(•) is $O(\psi\times b\times|V|^4)$, where ψ is the total number of different $\check{r}_\Delta$ in the table. Thus, in the worst case, DPCR-ST requires $O(\psi\times b\times|V|^4+n\times|E|\times\check{R}_{min})$.

*D.*2. Feasible Solutions Using DPCR-ST

In this subsection, we use Lemma 1 and Theorem 1 to show that the DPCR-ST algorithm will always produce feasible solutions for the NTD-CR problem. We first present a definition that is used in the lemma and theorem.

Definition 1. A row i=1, 2, …, n in the DP table is a *non-feasible row* if none of its elements contain a feasible solution, *i.e.*, each R[i, $\check{r}$]<R_{min}, for $\check{r}$=1, 2, …, $\check{R}_{min}$.

Note that R[i, $\check{R}_{min}$]=Rel(STX_i)≠0 for each feasible solution STX_i, while a non-feasible row i has R[i, $\check{R}_{min}$]=0. For example, row i=1 in Table II is a non-feasible row, while the remaining rows are feasible rows.

Lemma 1. A non-feasible row i in DP table must have at least one column $\check{r}$ that contains STX[i, $\check{r}$]=(ST_1, ST_2, …, ST_i), for i=1, 2, …, n, and $\check{r}$=1, 2, …, ($\check{R}_{min}$-1).

Proof. An STX[i, l]=(ST_1, ST_2, …, ST_i) must be generated only from STX[i-1, h]=(ST_1, ST_2, …, ST_{i-1}), for any column $\check{R}_{min}>l\geq h$, because the DPCR-ST algorithm processes the spanning trees in sequence from ST_1, ST_2, …, ST_{i-1}, ST_i.

For row i=1, and Rel(ST_1)<R_{min}, (6) sets R[1, $\check{r}$]=0 and STX[1, $\check{r}$]={} for all r>Rel(ST_1), and thus R[1, $\check{R}_{min}$]=0. Further, (5) sets R[1, $\check{r}$]=Rel(ST_1) and STX[1, $\check{r}$]=(ST_1) in each column $r\leq$Rel(ST_1), and thus row 1 is a non-feasible row, and Lemma 1 is true for i=1.

For i=z, where z=2, 3, …, n, let us assume that there *is* a non-feasible row z which contains STX[z, h]=(ST_1, ST_2, …, ST_{z-1}, ST_z) at column $h<\check{R}_{min}$. We want to show that, if z+1 is a non-feasible row, then it must have at least one column ($\check{R}_{min}$-1)$\geq l\geq h$ that contains STX[z+1, l]=(ST_1, ST_2, …, ST_{z-1}, ST_z, ST_{z+1}). Note that, when Rel(ST_{z+1})$\geq R_{min}$, row z+1 is a feasible row as (7) sets STX[z+1, $\check{R}_{min}$]=(ST_{z+1}) because Cost(ST_{z+1})<C[z, $\check{R}_{min}$]=∞. Further, row z+1 is also a feasible row if Rel($ST_1 \cup ST_2 \cup \ldots \cup ST_{z-1} \cup ST_z \cup ST_{z+1}$)$\geq R_{min}$, even when Rel($ST_{z+1}$)<$R_{min}$. For this case, Cost($ST_1 \cup ST_2 \cup \ldots \cup ST_{z-1} \cup ST_z \cup ST_{z+1}$)<C[$z$, $\check{R}_{min}$]=∞, and thus (8) sets STX[z+1, $\check{R}_{min}$]=(ST_1, ST_2, …, ST_{z-1}, ST_z, ST_{z+1}), and R[z+1, $\check{R}_{min}$]=Rel($ST_1 \cup ST_2 \cup \ldots \cup ST_{z-1} \cup ST_z \cup ST_{z+1}$)$\geq R_{min}$. Thus, row z+1 is a non-feasible row only when Rel($ST_1 \cup ST_2 \cup \ldots \cup ST_{z-1} \cup ST_z \cup ST_{z+1}$)<$R_{min}$. For this case, we consider the largest column $h<\check{R}_{min}$ with STX[z, h]=(ST_1, ST_2, …, ST_{z-1}, ST_z), *i.e.*, R[z, h]=h. Consequently, each column l>h will contain STX[z, l]={}, R[z, l]=0, and C[z, l]=∞ because it is not possible to have any subset of spanning trees from (ST_1, ST_2, …, ST_{z-1}, ST_z) with reliability larger than Rel($ST_1 \cup ST_2 \cup \ldots \cup ST_{z-1} \cup ST_z$). For this case, (8) will set STX[z+1, l]=(ST_1, ST_2, …, ST_{z-1}, ST_z, ST_{z+1}), C[z+1, l]=Cost(ST_1, ST_2, …, ST_{z-1}, ST_z, ST_{z+1}), R[z+1, l]=Rel(ST_1, ST_2, …, ST_{z-1}, ST_z, ST_{z+1}), and L[z+1, l]=$L_1 \cup L_2 \cup \ldots \cup L_{z-1} \cup L_z \cup L_{z+1}$ at columns $l\leq$R[z+1, l]. **Q.E.D.**

Theorem 1. For a network G with Rel(G)$\geq R_{min}$, DPCR-ST always generates a feasible solution for G.

Proof. We want to show that in the worst case DPCR-ST generates G at DP[n, $\check{R}_{min}$] as its feasible solution. Following Lemma 1, each non feasible row z=1, 2, …, n must contain at least one column $\check{r}$=1, 2, …, ($\check{R}_{min}$-1) with STX[z, $\check{r}$]=(ST_1, ST_2, …, ST_{z-1}, ST_z). Thus, in the worst case, there is at least one column r in a non-feasible row n-1 that contains STX[n-1, $\check{r}$]=(ST_1, ST_2, …, ST_{n-1}). For this case, (8) will set STX[n, $\check{r}$]=(ST_1, ST_2, …, ST_{n-1}, ST_n), C[n, $\check{r}$]=Cost(ST_1, ST_2, …, ST_{n-1}, ST_n), R[n, $\check{r}$]=Rel(ST_1, ST_2, …, ST_{n-1}, ST_n), and L[n, $\check{r}$]=$L_1 \cup L_2 \cup \ldots \cup L_{n-1} \cup L_n$ at columns $\check{r}$≤round(R[n, $\check{r}$]). For Rel(G)$\geq R_{min}$, R[n, $\check{r}$]$\geq R_{min}$. Thus, in the worst case, DP[n, $\check{R}_{min}$] will contain a feasible solution. **Q.E.D**.

D.3. Optimality of DPCR-ST

This subsection establishes Theorem 2 and Theorem 3. Theorem 2 states that the DPCR-ST algorithm produces an optimal topology if it uses a given sequence of optimally ordered spanning trees, called STX_{opt}; while Theorem 3 shows that generating STX_{opt} is NP-complete. We first describe two definitions that are used in the theorems.

Definition 2. A sequence of spanning trees $STX_n \subseteq$(ST_1, ST_2, …, ST_n) in graph G is called STX_{min} or *reliability minimal* if (i) Rel(STX_n)$\geq R_{min}$, and (ii) Rel(STX_n-(ST_j))$<R_{min}$ for any $ST_j \in STX_n$.

Definition 3. An STX_{min} is called STX_{opt} or *reliability optimal* if Cost(STX_{min}) is the minimum among all possible STX_{min}.

In Table I, for R_{min}=0.87, there are six STX_{min}:(ST_1, ST_2), (ST_1, ST_3), (ST_1, ST_5), (ST_2, ST_3), (ST_2, ST_5), and (ST_3, ST_5); and six equivalent STX_{opt}:(ST_1, ST_2), (ST_1, ST_3),(ST_1, ST_5), (ST_2, ST_3), (ST_2, ST_5), and (ST_3, ST_5). The six STX_{opt} contain the same set of links, and thus form the same G_{min}. In general, however, a graph G may contain several different G_{min}.

Theorem 2. The DPCR-ST algorithm produces an optimal network topology for a given STX_{opt}.

Proof. Without loss of generality, consider the DPCR-ST algorithm is given $STX_{opt}=(ST_1, ST_2, \ldots, ST_\beta)$, for $\beta=|STX_{opt}|\leq n$. By definition, the feasible solution that DPCR-ST produces from the given STX_{opt} includes all spanning trees in the set. Theorem 1 guarantees that DPCR-ST in the worst case produces $STX[n, \check{R}_{min}]=(ST_1, ST_2, \ldots, ST_n)$ that includes all the spanning trees. Because DPCR-ST adds spanning trees in order, $STX[\beta, \check{R}_{min}]=(ST_1, ST_2, \ldots, ST_\beta)$. Further, because by definition $STX_{opt}=(ST_1, ST_2, \ldots, ST_\beta)$ is the optimal topology, $Cost(STX_{opt})$ is the minimal among all feasible topologies. Therefore, (7) and (8) will keep $STX[i, \check{R}_{min}]=STX[\beta, \check{R}_{min}]$, for $i=\beta+1, \beta+2, \ldots, n$. Therefore, $STX[n, \check{R}_{min}]=STX[\beta, \check{R}_{min}]=(ST_1, ST_2, \ldots, ST_\beta)$ is an optimal solution. **Q.E.D.**

Theorem 3. Generating STX_{min} and STX_{opt} of a general graph G is NP-Complete.

Proof. For STX_{min}, we prove the theorem by reduction from the subset-sum problem [35]. For all links e_j in G, let $\rho(ST_i)$ be the probability that each link $e_j\in ST_i$ is operational while each $e_j\notin ST_i$ fails, *i.e.*, $\rho(ST_i)=\prod_{e_j\in ST_i} r_j \prod_{e_j\notin ST_i}(1-r_j)$. Because all failures are assumed to be statistically independent, $Rel(G)=\rho(ST_1)+\rho(ST_2)+\ldots+\rho(ST_n)$. Given a sequence $(ST_1, ST_2, \ldots, ST_n)$, each $\rho(ST_i)$ can be computed in polynomial time in the order of $|E|$, and the problem to generate STX_{min} is to find a subset of values from a set $\{\rho(ST_1), \rho(ST_2), \ldots, \rho(ST_n)\}$ such that their sum meets the required R_{min}. In other words, we set the target sum to R_{min}, and each value item to $\rho(ST_i)$. Because each STX_{opt} is STX_{min}, one can directly conclude that generating STX_{opt} is also NP complete. **Q.E.D**.

E. Improving the Efficiency of DPCR-ST

Our DPCR-ST algorithm requires all *n* spanning trees of the network, which is not feasible for a network that contains a large number of spanning trees. Further, Theorem 2 states that

DPCR-ST generates an optimal solution only if it is given an optimal sequence of spanning trees, STX_{opt}. Unfortunately, as shown in Theorem 3, generating STX_{opt} is NP-complete. Therefore, to improve the effectiveness and time complexity of DPCR-ST, we propose three different heuristic techniques, each of which sequentially generates only $0 \le k \le n$ spanning trees for its input. Note that each heuristic aims to generate k spanning trees that are expected to contain the trees in STX_{opt}. For a given graph G=(V, E), we first compute link weights w_i for each $e_i \in E$ using one of three different criteria: (i) CR1:$w_i=c_i/r_i$, (ii) CR2:$w_i=c_i$, and (iii) CR3:$w_i=-(\log r_i)$. Then, for each criterion, we use a modified Prim's algorithm [36] to sequentially generate all spanning trees of G, sorted in their increasing weights. Note that the weight of a spanning tree is calculated as the sum of the weight of each link in the spanning tree. As an example, we obtain three orders for the spanning trees in Table I: CR1:(ST_7, ST_2, ST_8, ST_6, ST_3, ST_1, ST_5, ST_4), CR2:(ST_6, ST_7, ST_4, ST_3, ST_8, ST_1, ST_5, ST_2), and CR3:(ST_2, ST_7, ST_8, ST_1, ST_3, ST_5, ST_4, ST_6). Note that (5)-(8) consider spanning trees starting from ST_1, and thus DPCR-ST sets ST_1 as the least weighted spanning tree, ST_2 as the second least weighted, *etc*. Because Prim's algorithm requires a time complexity of $O(|E|\times\log|V|)$, the DPCR-ST algorithm requires an extra $O(n\times(|E|\times\log|V|))$ time complexity for the improvement, *i.e.*, $O(\psi\times b\times|V|^4+n\times|E|\times\check{R}_{min}+n\times(|E|\times\log|V|))$.

Our DPCR-ST algorithm generates only the first k least weight spanning trees, and sets the value of k dynamically as follows. For each of the three heuristics, consider that DPCR-ST has generated and used a sequence of spanning trees ST_1, ST_2, …, ST_i to obtain a network topology G_i with cost C_i, and reliability $R_i \ge R_{min}$, for $i \le n$. In other words, the DPCR-ST algorithm obtains a feasible, but not necessarily optimal, solution G_i. Then, our algorithm generates the next least weight ST_{i+1}; and if it obtains a feasible solution G_{i+1} with reduced cost as compared to G_i, *i.e.*, $Cost(G_i)>Cost(G_{i+1})$, it keeps generating the subsequent spanning trees. The algorithm stops when it generates 10 consecutive spanning trees, none of which can further reduce topology cost.

Specifically, consider the DPCR-ST algorithm has generated a feasible solution G_k from sequence (ST_1, ST_2, …, ST_k). DPCR-ST will terminate and return G_k as its best solution if G_{k+1}, G_{k+2}, …, G_{k+10}, generated respectively from (ST_1, ST_2, …, ST_{k-1}, ST_k), …, (ST_1, ST_2, …, ST_k, ST_{k+1}, …, ST_{k+10}) have Cost(G_k)=Cost(G_{k+1})=Cost(G_{k+2})= … =Cost(G_{k+10}). Notice that using more than k spanning trees does not necessarily make DPCR-ST generate better results due to the heuristic nature of the spanning tree order, except when (ST_1, ST_2, …, ST_k, ST_{k+1}, …, ST_n) is a STX_{opt}. However, this improvement does not require all spanning trees a priori. Thus the DPCR-ST algorithm's time complexity becomes $O(\psi\times b\times|V|^4+k\times|E|\times\check{R}_{min}+k\times(|E|\times\log|V|))$, reducing its running time for smaller values of k. As an example, as discussed in Section V.C for a grid network with 200 nodes and 298 links that contain 1.899^{102} spanning trees, the improvement enables the DPCR-ST algorithm to generate results using only 1214 spanning trees.

V. SIMULATION, AND DISCUSSION

We have implemented our DPCR-ST algorithm in the C language to generate the topology of the 76 fully connected networks in [13] with the number of nodes, links, and spanning trees ranging from 6 to 11, 15 to 55, and 1269 to $2.3\text{x}10^9$, respectively. We obtained 76 cost matrices from the authors in [13], and use them for all link costs of all networks; the authors [13] randomly generated the integer costs with values between 1 and 100. Like in [13], we set R_{min} to either 0.9 or .95, and use equal link reliabilities with values of either 0.9 or 0.95. All simulations using DPCR-ST were run on an Intel Core i5 (2 cores) with 2.53 GHz and 4 GB of RAM, running Linux (Ubuntu Core 11.10).

Table III shows the results using the DPCR-ST algorithm with the three different spanning tree orderings, described in Section IV.E. Each $G_{N,C_{total}}^{p,R_{min}}$ in the first column denotes an N nodes fully connected network G with equal link reliability p, reliability constraint R_{min}, and total link cost C_{total} which is calculated for each network after assigning each link cost using its cost matrix given in [13]. Note that a fully connected network with N nodes contains $N(N\text{-}1)/2$ links. The second column C_{min} is the minimum cost of each topology with reliability at least R_{min} as reported in [13]. As stated in [8], the reliability of each topology with cost C_{min} was estimated using a Monte Carlo method that produces result within 1% of R_{min}. In Table III, columns C_{best} and *Rel* are the results for the 45 networks as reported in [14] using the Binary Decision Diagram (BDD) method; each N/A denotes each of the 31 non-reported values.

TABLE III
THE EXPERIMENTAL RESULTS OF DPCR-ST ON THE 76 NETWORKS IN [5]

		DPCR-ST					BDD	
CN	C_{min}	C_{best}	***Rel***	*k*	***Order***	**CPU sec.**	C_{best}	***Rel***
$G_{6,709}^{0.9,0.9}$	231	231	0.93	62	1,2,3,4	9.08	*N/A*	*N/A*
$G_{6,851}^{0.9,0.9}$	239	239	0.95	70	1,2,3,4	7.74	*N/A*	*N/A*
$G_{6,835}^{0.9,0.9}$	227	227	0.94	15	1,2,3,4	1.15	*N/A*	*N/A*
$G_{6,773}^{0.9,0.9}$	212	212	0.92	99	2	14.1	*N/A*	*N/A*
$G_{6,705}^{0.9,0.9}$	184	184	0.94	99	1,3,4	9.72	*N/A*	*N/A*
$G_{6,709}^{0.9,0.95}$	254	254	0.95	341	1,2,3,4	20.89	*N/A*	*N/A*
$G_{6,851}^{0.9,0.95}$	286	**239**	0.95	89	1,2,3	7.8	*N/A*	*N/A*
$G_{6,15}^{0.9,0.95}$	275	**234**	0.95	95	1,2,3	8.13	*N/A*	*N/A*
$G_{6,773}^{0.9,0.95}$	255	**236**	0.95	104	1,2,3	10.3	*N/A*	*N/A*
$G_{6,705}^{0.9,0.95}$	198	**193**	0.95	76	2	7.03	*N/A*	*N/A*
$G_{6,709}^{0.95,0.95}$	227	**185**	0.95	80	1,3	4.47	*N/A*	*N/A*
$G_{6,851}^{0.95,0.95}$	213	213	0.97	94	1,3,4	7.69	*N/A*	*N/A*
$G_{6,835}^{0.95,0.95}$	190	190	0.97	10	1,2,4	2.6	*N/A*	*N/A*
$G_{6,773}^{0.95,0.95}$	200	200	0.95	99	2	11.6	*N/A*	*N/A*
$G_{6,705}^{0.95,0.95}$	179	**148**	0.95	99	1,2,3	5.94	*N/A*	*N/A*
$G_{7,803}^{0.9,0.9}$	189	189	0.91	61	1,2,3	13.7	*N/A*	*N/A*
$G_{7,1028}^{0.9,0.9}$	184	190	0.95	92	2	14.7	*N/A*	*N/A*
$G_{7,1101}^{0.9,0.9}$	243	**200**	0.93	88	1,3	9.79	*N/A*	*N/A*
$G_{7,2816}^{0.9,0.9}$	129	129	0.91	93	1,2,3,4	7.68	*N/A*	*N/A*
$G_{7,1007}^{0.9,0.9}$	124	124	0.93	66	1,2,3,4	8.11	*N/A*	*N/A*
$G_{7,803}^{0.9,0.95}$	205	205	0.96	61	1,2,3,4	16.13	*N/A*	*N/A*
$G_{7,1028}^{0.9,0.95}$	209	209	0.96	92	1,2,3,4	9.2	*N/A*	*N/A*
$G_{7,1101}^{0.9,0.95}$	268	**264**	0.95	403	1,2,3	18.48	*N/A*	*N/A*
$G_{7,816}^{0.9,0.95}$	143	**139**	0.95	566	1,2,3	28.39	*N/A*	*N/A*
$G_{7,1007}^{0.9,0.95}$	153	153	0.96	94	1,2,3,4	13.42	*N/A*	*N/A*
$G_{7,803}^{0.95,0.95}$	185	**180**	0.95	61	1,3	7.94	*N/A*	*N/A*
$G_{7,1028}^{0.95,0.95}$	182	182	0.95	92	1,2,3	13.32	*N/A*	*N/A*

$G_{7,1101}^{0.95,0.95}$	230	**228**	0.95	489	1,2,3	14.16	*N/A*	*N/A*
$G_{7,816}^{0.95,0.95}$	122	129	0.98	93	1,2,3,4	4.6	*N/A*	*N/A*
$G_{7,1007}^{0.95,0.95}$	124	124	0.99	66	1,2,3,4	17.88	*N/A*	*N/A*
$G_{8,1343}^{0.9,0.9}$	208	**184**	0.90	95	1,2,3,4	11.54	218	0.92
$G_{8,1351}^{0.9,0.9}$	203	203	0.92	48	1,2,3	10.46	213	0.92
$G_{8,1352}^{0.9,0.9}$	211	211	0.93	451	1,3	23.27	211	0.92
$G_{8,1452}^{0.9,0.9}$	291	299	0.90	182	1	19.56	300	0.91
$G_{8,1263}^{0.9,0.9}$	178	178	0.91	99	1,3,4	11.4	181	0.93
$G_{8,1343}^{0.9,0.95}$	247	**223**	0.95	95	1,2,3	12.14	259	0.96
$G_{8,1351}^{0.9,0.95}$	247	**233**	0.95	97	1,3	15.8	253	0.95
$G_{8,1352}^{0.9,0.95}$	245	**232**	0.95	489	1,2,3,4	29.5	245	0.95
$G_{8,1452}^{0.9,0.95}$	336	**332**	0.95	331	1	27.97	351	0.96
$G_{8,1263}^{0.9,0.95}$	202	**181**	0.95	89	1,3	22.66	202	0.95
$G_{8,1343}^{0.95,0.95}$	179	179	0.97	96	1,2,3,4	6.6	179	0.96
$G_{8,1351}^{0.95,0.95}$	194	194	0.97	337	1,3,4	20.07	196	0.96
$G_{8,1352}^{0.95,0.95}$	197	**192**	0.95	499	1,3	25.87	197	0.96
$G_{8,1452}^{0.95,0.95}$	276	282	0.96	341	1	13.36	280	0.96
$G_{8,1263}^{0.95,0.95}$	173	**150**	0.95	99	2	12.29	184	0.97
$G_{9,1859}^{0.9,0.9}$	239	242	0.90	79	1,3	9.10	244	0.93
$G_{9,1897}^{0.9,0.9}$	191	212	0.91	92	2	7.74	194	0.91
$G_{9,1828}^{0.9,0.9}$	257	**252**	0.90	92	3	11.15	273	0.90
$G_{9,1749}^{0.9,0.9}$	171	171	0.90	94	1,2,3,4	14.1	183	0.91
$G_{9,1678}^{0.9,0.9}$	198	**195**	0.91	98	1,3	9.72	198	0.91
$G_{9,1859}^{0.9,0.95}$	286	**279**	0.96	93	1,2,3	10.89	286	0.96
$G_{9,1897}^{0.9,0.95}$	220	236	0.96	99	1,2,3,4	7.8	237	0.95
$G_{9,1828}^{0.9,0.95}$	306	**296**	0.95	242	1,2,3	18.13	306	0.95
$G_{9,1749}^{0.9,0.95}$	219	**200**	0.95	89	1,2,3	20.3	219	0.95
$G_{9,1678}^{0.9,0.95}$	237	**212**	0.95	583	1	37.03	239	0.95
$G_{9,1859}^{0.95,0.95}$	209	214	0.97	79	2	4.47	209	0.97
$G_{9,1897}^{0.95,0.95}$	171	199	0.95	332	2	27.69	171	0.95
$G_{9,1828}^{0.95,0.95}$	233	233	0.97	227	1	20.6	249	0.96
$G_{9,1749}^{0.95,0.95}$	151	151	0.95	95	1,3	11.6	177	0.97
$G_{9,1678}^{0.95,0.95}$	185	**183**	0.98	290	1	15.94	206	0.95
$G_{10,1803}^{0.9,0.9}$	131	131	0.90	104	2	13.7	131	0.91
$G_{10,2155}^{0.9,0.9}$	154	154	0.92	102	1,2,3,4	14.7	154	0.91
$G_{10,1828}^{0.9,0.9}$	267	**250**	0.90	111	1,3	9.79	*N/A*	*N/A*
$G_{10,2546}^{0.9,0.9}$	263	263	0.94	410	1,3	27.68	263	0.94
$G_{10,2517}^{0.9,0.9}$	293	309	0.91	127	1,3,4	18.11	309	0.91
$G_{10,1803}^{0.9,0.95}$	153	153	0.95	102	2	16.13	164	0.95
$G_{10,2155}^{0.9,0.95}$	197	**195**	0.95	364	1,2,3	24.2	205	0.95
$G_{10,1828}^{0.9,0.95}$	311	321	0.95	111	1,2,3,4	18.48	*N/A*	*N/A*
$G_{10,2546}^{0.9,0.95}$	291	291	0.95	421	1,2,3,4	28.39	309	0.96
$G_{10,2517}^{0.9,0.95}$	358	360	0.95	150	1,4	13.42	366	0.96

$G^{0.95,0.95}_{10,1803}$	121	125	0.96	89	1,3,4	7.94	127	0.95
$G^{0.95,0.95}_{10,2155}$	136	<u>136</u>	0.98	222	2	23.32	144	0.90
$G^{0.95,0.95}_{10,1828}$	236	**<u>233</u>**	0.97	103	1,3	14.16	*N/A*	*N/A*
$G^{0.95,0.95}_{10,2546}$	245	**<u>231</u>**	0.97	386	1,2,3,4	24.6	256	0.96
$G^{0.95,0.95}_{10,2517}$	268	296	0.97	114	1	17.88	277	0.96
$G^{0.9,0.9}_{11,2609}$	246	<u>246</u>	0.91	215	2	26.34	*N/A*	*N/A*

For each of the 76 fully connected network topologies in [13], we first generated its spanning trees in four different orders: random, CR1, CR2, and CR3, as described in Section IV.E. We have used Prim's algorithm [36] to generate the randomly ordered spanning trees, and modified the algorithm to generate the spanning trees for the three sorted criteria. Then, we used DPCR-ST on each set of spanning trees to generate its feasible topology with minimum cost. Each of the 76 C_{best} in column 3 is the *minimum* among the costs of topologies generated using random, CR1, CR2, CR3, and column *Rel* stores its reliability. Column CPU shows the CPU time of running our approach for each network.

Our DPCR-ST algorithm produces topologies with C_{best}<C_{min}, and reliability within 0.5% of R_{min}; we used the Monte Carlo method [26] to calculate the reliability. Note that we use the *round* function that will make each topology with cost C_{best} and reliability of 0.745 an acceptable solution for R_{min}=0.75 because *round* (0.745)=75. Column *Order* shows which order, 1=CR1, 2=CR2, 3=CR3, 4=random, can be used to produce each C_{best}; *e.g.*, *Order*={1,3,4} means the corresponding C_{best} can be generated using either random, CR1, CR3, but not CR2. Column *k* shows the maximum number of spanning trees among the four ordering criteria used by the DPCR-ST algorithm to generate their topologies.

As shown in Table III, the DPCR-ST algorithm using CR1, CR2, and CR3 produced 81.5% (62 of 76) optimal results (underlined), 30 topologies of which (bold) have lower cost than the C_{min} reported in [8]. Further, the DPCR-ST algorithm produced the topologies using only 10/1296=0.77% to 583/4782969=0.01% of the spanning trees contained in the networks, using

CPU times ranging between 1.15 and 37.03 seconds, and thus our approach is very efficient. Consistent with its time complexity, described in Section IV.E, as shown in Table III, the DPCR-ST algorithm requires a larger CPU time when it uses a larger number of spanning trees, *k*, to generate its results. Note that we do not compare the performance of the DPCR-ST algorithm, in term of CPU time, against that reported for the other algorithms in [8], [9], [13], and [14] because they were run on different systems, and we were unable to obtain their source codes.

A. The Effect of Spanning Tree Orderings on the Performance of the DPCR-ST Algorithm

Table III shows that the DPCR-ST algorithm with random ordered spanning trees generates C_{best} only in 28 of 76 networks (36.8%), which is the worst as compared to CR1 (82.8%), CR2 (63.1%), and CR3 (72.3%). Further, for each case in which the random order generates C_{best}, at least one of the other three orders was also able to produce the result. This result shows the merit of pre-ordering spanning trees for our DP approach.

To compare the performances of CR1, CR2, and CR3, we summarize their results from Table III in Tables IV and V. The tables show the total number of topologies generated with cost C_{best} and their cost optimality with respect to C_{min}, *i.e.*, $C_{best}>C_{min}$, $C_{best}=C_{min}$, $C_{best}<C_{min}$.

TABLE IV
COMPARISONS AMONG CR1, CR2, AND CR3

Cost / Order	Total number of topologies with cost C_{best}				Total number of topologies with cost C_{best} using the other two sorting criteria
	$C_{best}<C_{min}$	$C_{best}=C_{min}$	$C_{best}>C_{min}$	Total	
CR1	27 (35.5%)	26 (34.2%)	10 (13.1%)	63 (82.8%)	13 (17.2%)
CR2	17 (22.3%)	24 (31.5%)	7 (9.2%)	48 (63.1%)	28 (36.8 %)
CR3	25 (32.8%)	24 (31.5%)	6 (7.8%)	55 (72.3%)	21 (27.6%)

TABLE V
THE DISTRIBUTION OF C_{best} GENERATED USING ONE OR MORE SORTING CRITERIA

Cost / Order	$C_{best}<C_{min}$	$C_{best}=C_{min}$	$C_{best}>C_{min}$
CR1	3	1	4
CR2	2	6	4
CR3	1	0	0

CR1,CR2	0	1	0
CR1,CR3	9	7	3
CR2,CR3	0	0	0
CR1,CR2,CR3	15	17	3
Total	30 (39.4%)	32 (42.1%)	14 (18.4%)

As shown in Table IV, CR1 is the best performer, producing C_{best} 82.8% of the time, followed by CR3 with 72.3%, and CR2 with 63.1%; see the column Total. For each order, the last column in the table shows the total number of topologies with cost C_{best} that can only be generated using its two alternative sorting criteria; *e.g.*, row 1 of the table shows that CR1 produces 13 topologies with cost worse than that produced using CR2 or CR3.

Table V shows the total number of C_{best} uniquely produced using one or more of the three different ordering criteria. The table shows that there are in total 8, 12, and 1 topology with cost C_{best} uniquely generated by CR1, CR2, and CR3, respectively, and the three criteria produce the same topologies 35/76=46% of the time. Further, there are 1, and 19 topologies that can only be generated by either CR1 or CR2, and CR1 or CR3, respectively. The results show that it is important for the DPCR-ST algorithm to use the three ordering criteria, CR1, CR2, and CR3; and select the best among their results to generate topologies with lower costs. As shown in the table, such approach produces only 18.4% topologies with less optimal costs.

B. The DPCR-ST Algorithm versus Existing Approaches

To evaluate the effectiveness of our DPCR-ST algorithm, we compared its results on the 76 fully connected networks with those generated by the state-of-the-art approaches NGA [8], LS-NGA [9], ACO-SA [13], and BDD [14]. Because we were unable to obtain the source codes for the four approaches in [8]-[9], [13], [14], we have used their reported results in our comparisons. Table III shows the C_{best} and *Rel* for the 45 networks generated using BDD as reported in [14]; each N/A denotes each of the 31 non-reported values. However, we cannot present the results from [8]-[9], [13] in the same table because they are presented in a different format. For the same reason, we can't compare the CPU time due to the difference in the CPU processors and simulation environment. As an alternative, we have summarized the performance of all the evaluated algorithms in Table VI.

TABLE VI
COMPARISON BETWEEN DPCR-ST, NGA, LS-NGA, ACO-SA AND BDD

	DPCR-ST	NGA	LS-NGA	ACO-SA	BDD
$C_{best}=C_{min}$	32 (42.1%)	16 (21%)	24 (31.5%)	48 (63.1%)	14 (33.33%)
$C_{best}<C_{min}$	30 (39.4%)	N/A	N/A	N/A	N/A

As shown in Table VI, NGA, LS-NGA, and ACO-SA generate optimal solutions for 16, 24, and 48 out of 76 instances, respectively, while BDD obtains optimal solutions for 14 out of 45 instances. On the other hand, DPCR-ST produces 62 out of 76 optimal results (81.5%), and thus it has significantly higher effectiveness over the existing algorithms. Further, 30 of 62 C_{best} generated using the DPCR-ST algorithm are better than C_{min}. These results show the superiority of our efficient DP approach as compared to the existing state-of-the-arts solutions [8]-[9], [13], [14].

C. The DPCR-ST Algorithm with a Large Number of Spanning Trees

To further evaluate the performance of the DPCR-ST algorithm, we have used the method to solve the NTD-CR for five grid networks in [37] with the number of nodes, links, and spanning trees range from 36 to 200, 57 to 298, and 3.557^{18} to 1.899^{102}, respectively; see Table VII. Note that we have used the formula in [24] to count the number of spanning trees in $Grid_{2xn}$ grid networks. However, the paper does not provide other counting equations for the other grid sizes, *i.e.*, $Grid_{6x6}$ and $Grid_{3x12}$, each containing 36 nodes.

For Table VII, we assume that, when two grid networks have the same number of nodes, the wider grid, *e.g.*, $Grid_{3x12}$, contains more spanning trees than the other, *e.g.*, $Grid_{2x18}$. Thus, the table shows $Grid_{6x6}$, and $Grid_{3x12}$ as $G^{36,60}_{\geq 3.557^{18}}$, and $G^{36,57}_{\geq 3.557^{18}}$, respectively because $Grid_{2x18}$ contains 3.557^{18} spanning trees. We set c_j=1 and r_j=0.9 for all of the five grid networks; and, for each network, we consider five different R_{min} values, *i.e.*, 50%, 60%, 70%, 80%, and 99% of the reliability of the original network, Rel_{max}, *e.g.*, 0.46, 0.55, 0.64, 0.73, and 0.9 for $G^{36,57}_{\geq 3.557^{18}}$, respectively; we obtained Rel_{max} for G_{3x12}, G_{2x20}, and $Grid_{2x100}$ from [38]. To see how fast, in terms of *k*, the DPCR-ST algorithm produces the results for each of the five grid networks, we set R_{min} to Rel_{max}; we consider this scenario the worst case because the resulting topology includes all links in the original network.

TABLE VII
PERFORMANCE DPCR-ST FOR THE LARGE NETWORK RESULTS

CN	R_{min}	*k*	Cost	Rel
$Grid_{3x12}$ $G^{36,57}_{\geq 3.557^{18}}$ k_{max}=45 Rel_{max}=0.9173	0.46	45	44	0.4913
	0.55	45	45	0.5703
	0.64	45	46	0.6590
	0.73	45	48	0.7343
	0.90	45	56	**0.9061(-1.22%)**
$Grid_{6x6}$ $G^{36,60}_{\geq 3.557^{18}}$ k_{max}=194 Rel_{max}=0.9130	0.45	194	46	0.4670
	0.54	194	48	0.6016
	0.63	194	49	0.6863
	0.72	194	50	0.7273
	0.89	194	59	**0.9082(-0.53%)**
$Grid_{3x16}$ $G^{48,77}_{\geq 2.5^{24}}$ k_{max}=187 Rel_{max}=0.7218	0.36	187	52	0.2712
	0.43	187	58	0.4394
	0.50	187	59	0.5021
	0.58	187	62	0.5877
	0.71	187	76	**0.7180(-0.53%)**
$Grid_{2x20}$ $G^{40,58}_{2^{20}}$ k_{max}=248 Rel_{max}=0.7452	0.37	248	28	0.3910
	0.44	248	34	0.4574
	0.52	248	37	0.5362
	0.59	248	39	0.6091
	0.73	248	57	**0.7405(-0.63%)**
$Grid_{2x100}$ $G^{200,298}_{1.899^{102}}$ k_{max}=1214 Rel_{max}=0.251	0.12	1214	184	0.1250
	0.14	1214	195	0.1472
	0.17	1214	224	0.1746
	0.19	1214	268	0.1951
	0.23	1214	297	**0.2383(-5.05%)**

As shown in Table VII, the DPCR-ST algorithm produces each topology using $1214/1.899^{102}$=4.7^{-26}% to $248/2^{20}$=2.3^{-4}% of the spanning trees contained in the networks. The DPCR-ST algorithm is also very fast in producing the results for the networks with the other R_{min} values; see column *k*. Note that the DPCR-ST algorithm requires k_{max} spanning trees to produce

Rel_{max}, and thus k_{max} is the upper bound value of k. As shown in the table, the DPCR-ST algorithm uses only k=k_{max} spanning trees to produce each result. Table VII also shows the reliability value of each generated topology. Because the DPCR-ST algorithm does not calculate the exact reliability values of its generated topologies, we have used MC simulation [26] to compute the estimated reliability (Rel) for each generated topology; we used a sample size of 10^6 in the simulation. However, we are unable to gauge the optimality of the generated topologies for such large networks, except for Cost(G)-1. In this case, for each network, we deleted one link from the network, and used MC simulation to generate its Rel; we repeated the step |E|-1 times, and selected the minimum cost with Rel≥R_{min} as the optimal solution. As shown in Table VII (the numbers in bold in column Rel), the DPCR-ST algorithm is able to generate a topology with a reliability only up to 5.05% off from optimal.

VI. CONCLUSION

We have formally defined a *network topology design* problem, NTD-CR, to generate a topology that has the minimum cost subject to reliability constraint R_{min}. We have proposed a heuristic DP method, DPCR-ST, to solve NTD-CR. The DPCR-ST algorithm incrementally generates only a selected k spanning trees from the network, and thus is scalable on networks with large numbers of spanning trees. We have proposed to sort the spanning trees using three different orders to optimize our method's effectiveness and efficiency. Our simulations on various networks that contain up to 200 nodes, 298 links, and 1.899^{102} spanning trees show the practicality of our techniques. Our approach's results are encouraging, showing that DPCR-ST is suitable for use in larger real world networks. The experimental study shows that the DPCR-ST approach is able to generate 81.5% optimal solutions.

Our DP algorithm incrementally inserts spanning trees, and thus links, to form an optimal topology. We plan to design an alternative DP approach that heuristically deletes links from the original topology to find an optimal design.

REFERENCES

[1]. A. Konak, and E. Smith, “Designing resilient networks using a hybrid genetic algorithm approach,” *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, 2005.

[2]. B.K Lad, M.S. Kulkarni, K.B. Misra, "Optimal reliability design of a system," *Handbook Of Performability Engineering*, Springer, London, pp. 499-519, 2008.

[3]. D. Reichelt, F. Rothlauf, “Reliable communication network design with evolutionary algorithms,” *International Journal of Computational Intelligence and Applications*, vol. 5, no. 2, pp. 1469–0268, 2005.

[4]. H. M. F. Abo ElFotoh, and L. S. Al-Sumait, “A neural approach to topological optimization of communication networks, with reliability constraints,” *IEEE Trans. Reliability*, vol. 50, pp. 397–408, 2001.

[5]. R. Jan, H. Fung, and C. Sheng, “Topology optimization of a communication network subject to a reliability constraint,” *IEEE Trans. Reliability*, vol. 42, no. 1, pp. 63–70, 1994.

[6]. T. Koide, S. Shinmori, and H. Ishii, “Topological optimization with a network reliability constrain,” *Discrete Applied Mathematics*, vol. 115, pp. 135–149, 2001.

[7]. M. Atiqullah, and S. Rao, “Reliability optimization of communication networks using simulated annealing,” *Microelectronics and Reliability*, vol. 33, pp. 1303–1319, 1993.

[8]. B. Dengiz, F. Altiparmak, and A.E. Smith, “Efficient optimization of all-terminal reliable networks,” *IEEE Trans. Reliability*, vol. 41, no. 1, pp. 18–26, 1997.

[9]. B. Dengiz, F. Altiparmak, and A.E. Smith, “Local search genetic algorithm for optimal design of reliable networks,” *IEEE Trans. Evolutionary Computation*, vol. 1, no. 3, pp. 179–188, 1997.

[10]. D. Deeter, and E. Smith, “Economic design of reliable networks,” *IIE Transactions*, vol. 30, pp. 1161–1174, 1998.

[11]. F. Altiparmak, and B. Dengiz, “Cross entropy approach to design of reliable networks,” *European Journal of Operation Research*, vol. 199, pp. 542–552, 2009.

[12]. J. Marquez, and C. Rocco, “All-terminal network reliability optimization via probabilistic solution discovery,” *Reliability Engineering and System Safety*, vol. 93, pp. 1689–1697, 2008.

[13]. F. Altiparmak, B. Dengiz, and O. Belgin, "Design of reliable communication networks: A hybrid ant colony optimization approach for the design of reliable networks," *IIE Transactions*, vol. 42, pp. 273–287, 2010.

[14]. G. Hardy, C. Lucet, and N. Limnios, "A BDD-based heuristic algorithm for design of reliable networks with minimal cost," *International Conference on Mobile Ad Hoc and Sensor Networks*, pp. 13–15, 2006.

[15]. C. Papagianni, K. Papadopoulos, and C. Pappas, "Communication network design using particle swarm optimization," *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 915–920, 2008.

[16]. S. Pierre, M.A. Hyppolite, J.M. Bourjolly, and O. Dioume, "Topological design of computer communication networks using simulated annealing," *Engineering Applications of Artificial Intelligence*, vol.8, pp. 61–69, 1995.

[17]. R. Chelouah, and P. Siarray, "Tabu search applied to global optimization," *European Journal of Operational Research*, vol. 123, pp. 256–270, 2000.

[18]. B. Elshqeirat, S. Soh, S. Rai, M. Lazarescu, "Dynamic programming for minimal cost topology with reliability constraint," *Proceedings of the 5th International Conference on Communication Software and Networks, 2013*.

[19]. A. Kumar, R.M. Pathak, and Y.P. Gupta, "A genetic algorithm for distributed system topology design," *Computers and Industrial Engineering*, vol. 28, pp. 659–670, 1995.

[20]. L. Gen, "A Self-controlled genetic algorithm for reliable communication network design," *Evolutionary Computation IEEE Congress*, pp. 640–647, 2006.

[21]. B. Dengiz, C. Alabas, and O. Dengiz, "A tabu search algorithm for neural networks training," *Journal of Operation Research*, vol. 60, no. 2, pp. 282–291, 2007.

[22]. K. Watcharasitthiwat, S. Pothiya, and P. Wardkein, "Multiple tabu search algorithm for solvingthe topology network design," *Local Search Techniques: Focus on Tabu Search*, I-Tech, pp. 259–264, 2008.

[23]. J. Won, and F. Karray, "Cumulative update of all-terminal reliability for faster feasibility decision," *IEEE Trans. Reliability*, vol. 59, no. 3, pp. 551–562, 2010.

[24]. M. Desjarlais, and R. Molina, "Counting spanning trees in grid graphs," *Congressus Numerantium*, vol. 145, pp. 177–185, 2000.

[25]. S. Rai, and S. Soh, "CAREL: Computer aided reliability evaluator for distributed computer networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 199–213, 1991.

[26]. S. Yeh, J.S. Lin, and W.C. Yeh, "New monte carlo method for estimating network reliability," *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, pp. 723–726, 1994.

[27]. J. M. Won, and F. Karray, "A greedy algorithm for faster feasibility evaluation of all –terminal-reliable networks," *IEEE Trans. System, Man, and Cybernetics-Part B: Cybernetics*, vol. 41, pp. 1600–1611, 2011.

[28]. M. Leslie, C. Hector and R. Gerardo, "A splitting algorithm for network reliability estimation," *IIE Transactions*, vol. 45, pp. 177–189, 2013.

[29]. M. Mutawa, H. Alazemi, and A. Rayes, "A novel steadystate genetic algorithm to the reliability optimization design problem of computer networks," *International Journal of Network Management*, vol. 19, pp. 39–55, 2009.

[30]. F. Shao, X. Shen, and P. Ho, "Reliability optimization of distributed access networks with constrained total cost," *IEEE Trans. Reliability*, vol. 54, no. 3, pp. 421–430, 2005.

[31]. L. Deeter, and E. Smith, "Heuristic optimization of network design considering allterminal reliability," *Proceedings of the Reliability and Maintainability Symposium*, pp. 194-199, 1997.

[32]. B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "A practical algorithm for reliable communication network design," *International Journal of Performability Engineering*, vol. 9, no. 4, pp. 397–408, 2013.

[33]. B. Elshqeirat, S. Soh, S. Rai, and M. Lazarescu, "A Dynamic Programming Algorithm for Reliable Network Design," *IEEE Trans. Reliability*, vol. 63, no. 2, pp. 443–454, 2014.

[34]. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science*, vol. 45, pp. 414–424, 1999.

[35]. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to algorithms," *Cambridge: The MIT Press*, 1990.

[36]. *http://www.dzone.com/snippets/prims-algorithm-finding.*

[37]. S. Kuo, S. Lu, and F. Yeh, "Determining terminal pair reliability based on edge expansion diagrams using OBDD," *IEEE Trans. Reliability*, vol. 48, no. 3, pp. 234–246, 1999.

[38]. G. Hardy, C. Lucet, and N. Limnios, "K-terminal network reliability measures with binary decision diagrams," *IEEE Trans on Reliability*, vol. 56, no. 3, pp. 506–515, 2007.

Basima Elshqeirat received the BS (2004), and MS (2008) degrees in Computer Science from the University of Jordan, Jordan. She is currently working toward the Ph.D degree in Computer Science at Curtin University. Her research interests include network reliability, and network design.

Sieteng Soh received a B.S. degree in Electrical Engineering from the University of Wisconsin, Madison, and M.S. and Ph.D in Electrical Engineering from the Louisiana State University, Baton Rouge. He was a faculty member (1993–2000), and the director of the Research Institute (1998–2000) at Tarumanagara University-Indonesia. He is currently a Senior Lecturer with the Department of Computing at Curtin University. Dr. Soh has published papers in refereed international journals, and conference proceedings in the areas of computer networks, network reliability, and parallel and distributed processing. He is a member of the IEEE.

Suresh Rai received the PhD degree in Electronics and Communication Engineering from Kurukshetra University, Kurukshetra, Haryana, India, in 1980. He is currently a professor with the Division of Electrical and Computer Engineering, School of Electrical Engineering and Computer Science at Louisiana State University, Baton Rouge. His research interests include network traffic, wavelet-based compression, watermarking in audio and video, and network reliability and security. He is a senior member of the IEEE, and a member of the IEEE Computer Society.

Mihai Lazarescu received his B.S. (Computer Science) degree with First Class Honours in 1996, and his Ph.D in Computer Science from Curtin University, in 2000. He has been a senior member of the IMPCA research institute for 10 years, and is currently the Head of Department of Computing and Associate Professor at Curtin University. He has published over 60 papers in refereed international journals, and conference proceedings in the areas of artificial intelligence, machine vision, data mining, and network reliability.