Dynamic Programming for Minimal Cost Topology with Reliability Constraint

Basima Elshqeirat, Sieteng Soh, Suresh Rai, and Mihai Lazarescu

***Abstract*—This paper addresses an NP-hard problem, called NTD-CR, to design a minimal-cost communication network topology that satisfies a pre-defined reliability constraint. Since reliability is always a major issue in the network design, the problem is practical for critical applications requiring minimized cost. The paper formulates a *dynamic programming* (DP) scheme to solve NTD-CR problem. DP approach, called DPCR-ST, generates the topology using a selected set of spanning trees of the network, STX_{min}. We propose three greedy heuristics to generate and order only k spanning trees of the network. Each heuristic allows DPCR-ST to enumerate STX_{min} using only k spanning trees, which improves the time complexity while producing near optimal topology. Simulations based on fully connected networks that contain up to 2.3×10^9 spanning trees show the merits of ordering methods and the effectiveness of our algorithm vis-à-vis four existing state-of-the-art techniques; DPCR-ST produces 81.5% optimal results, while using only 0.77%of the spanning trees contained in network.**

***Index Terms*—Dynamic programing, network optimization, network reliability, network topology design.**

I. Introduction

A well-designed communication network is inseparable from the effective running of user applications. For critical applications (*e.g.*, emergency system, rescue and military operations) it is important that the communication network topology is as reliable as possible since in practice network components (*e.g.*, links) are failure-prone. A more reliable topology will make the communication network operate effectively and without interruption, even in the presence of the component failures [1].Further, some applications may need to run on a topology with a guaranteed minimum reliability, R_{min}, to properly operate. However, constructing a reliable topology incurs higher installation cost. Given a set of various centers (nodes), their possible connecting links, link failure rate and installation cost, NTD-CR selects the most suitable set of links such that the resulting model meets its required reliability R_{min} while minimizing its installation cost. This paperconsiders network reliability [2], also called all-terminal reliability, as the measure of reliability.

The NTD-CR problem has been shown NP-hard [3], and thus one must use heuristic and/or approximation solutions to design large sized topologies. There are many proposed solutions for NTD-CR problem. The existing algorithms that generate approximation solution are mainly based on meta-heuristic techniques, *e.g.*, Genetic Algorithm [2], [3], Swarm Particle [4] and Ant Colony [5]. While the metaheuristic algorithms can significantly reduce time complexity, they still require numerous iterations to converge and thus use a considerable computational effort while producing only up to 63.1% optimal solutions. Thus, approach that can produce better results is still needed, especially for use in large scale networks.

The main contribution of this paper is two folds. First, it uses a *dynamic programming* (DP) formulation to generate topology based on the proposed algorithm, DPCR-ST. Second, this paper proposes three heuristics to enumerate only $k \le n$ spanning trees, which are used by DPCR-ST to significantly reduce its time complexity; n is the total number of spanning trees in the network.

The layout of this paper is as follows. Section II discusses the network model and notations. Section III formulates the NTD-CR problem and provides assumptions. Section IV describes our proposed solutions while Section V presents the simulation results. Finally, Section VI concludes the paper and discusses the future work.

II. Network Model and Notations

A communication network can be modeled by a probabilistic bidirectional simple graph G=(V, E), in which each vertex/node $v_i \in V$ represents a network component (*e.g.*, router or computer site) and each edge $e_j \in E$ represents the connecting media (*e.g.*, cable or communication link) between the network components. It is assumed that all node locations and connecting links are given. Each e_j has a cost $c_j>0$ that represents the cost to install e_j, and reliability $0 \le r_j \le 1$ that represents the probability that e_j is functioning (UP); all nodes are always UP and use no setup costs. Edge failures are assumed statistically independent and without repair. Fig. 1 (a) shows an example of the graph model of a network with four fixedly positioned nodes and five links; Table I provides cost (c_j) and reliability (r_j) values for an edge e_j.

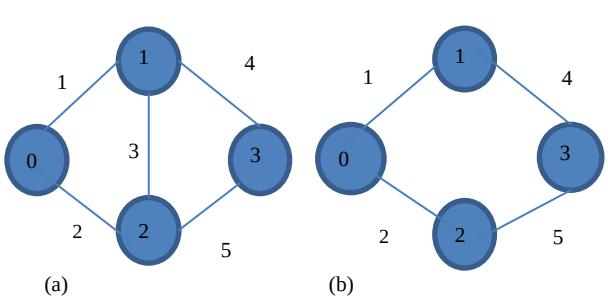

Fig. 1. An example network and optimal solution.

A spanning tree i, ST_i, is a subgraph of G, which is a tree and contains all vertices in G. A spanning tree in a network

Manuscript received February 20, 2013; revised August 1, 2013.

B. Elshqeirat, S. Soh, and M. Lazarescu are with Department of Computing, Curtin University, Perth, Western Australia; (e-mail: Basima.elshoqeirat@postgrad.curtin.edu.au, S.Soh@curtin.edu.au, M.Lazarescu@curtin.edu.au; tel.: +61 8 9266 2984; fax: +61 8 9266 2819).

S. Rai is with Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, USA. (e-mail: srai@lsu.edu).

with |V| nodes has (|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 (a).

TABLE I: LINK WEIGHT AND SPANNING TREE SET FOR NETWORK IN FIG. 1 (A)

	ST_G			Link Weight		
i	ST_i	Rel(ST_i)	Cost(ST_i)	e_j	c_j	r_j
1	(1,3,5)	0.567	13	1	5	0.9
2	(1,3,4)	0.567	11	2	3	0.6
3	(2,3,4)	0.378	9	3	2	0.7
4	(1,2,5)	0.486	14	4	4	0.9
5	(2,3,5)	0.378	11	5	6	0.9
6	(2,1,4)	0.486	12			
7	(1,4,5)	0.729	15			
8	(2,5,4)	0.486	13			

Let Cost(ST_i) denote the cost of installing all links in spanning tree ST_i, computed by taking the sum of c_j of each e_j in ST_i. The cost of a network topology G, Cost(G), is obtained using the sum of all c_j for each $e_j$in G. Let Rel(ST_i) denote the reliability of spanning tree ST_i;it is 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 another word, it is the probability that a set of operational links provides communication path between every pair of nodes. Calculating Rel(G), in general, is an NP-hard problem [5]; Section III.B provides details about computing Rel(G). Notice that G can be constructed using nodes in V and all links in ST_G, and thus this paper

$$\text{usesCost}(ST_G)=\text{Cost}(G)=\text{Cost}(E)$$

and

$$\text{Rel}(ST_G)=\text{Rel}(G)=\text{Rel}(E).$$

Similarly, we consider that

$$\text{Cost}(ST_i)=\text{Cost}(L_i)$$

and

$$\text{Rel}(ST_i)=\text{Rel}(L_i).$$

III. NETWORK DESIGN PROBLEM AND SOLUTION

Let X_i be a decision variable {0, 1} that indicates if spanning tree ST_i in Gis selected (X_i=1) or not selected (X_i=0). The following equations describe the NTD-CR problem.

$$\text{Minimize Cost}\left(\bigcup_{i=1}^{|ST_G|} ST_iX_i\right) \quad (1)$$

$$\text{Subject to Rel}\left(\bigcup_{i=1}^{|ST_G|} ST_iX_i\right) \geq R_{min} \quad (2)$$

Equation (1) calculates the minimum cost of the network using only the selected spanning trees ST_i from (2). One may generate all 2^n possible combinations of spanning trees that meet the constraint in(2). Then, for each combination that has reliability at least R_{min}, use(1) to calculate its cost and select the topology with the minimum cost as G_{min} with Rel(G_{min})$\geq R_{min}$. This solution is prohibitive for use in large networks since a general network contains $n=O(|V|^{|V|})$ spanning trees [6]. In Section IV.A, we propose aDP approach to solve (1) and (2).

To illustrate the NTD-CR problem, consider the network in Fig. 1 (a). For R_{min}=0.87, Fig. 1 (b) shows the optimal network topology, G_{min}, whose links form a set of spanning trees{(2, 5, 4), (1, 4, 5), (2, 1, 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.

IV. PROPOSED DYNAMIC PROGRAMMING-BASED SOLUTION

A. Dynamic Programming Formulation for NTD-CR

Let STX_i, for i=1, 2, …, n, be a set of spanning trees selected from n-i+1 spanning trees in {ST_i, ST_{i+1}, …, ST_n} and G_i=(V, $E_i \subseteq$E) be its induced graph whose links comprise of all links in STX_i. We use STX_i and G_i interchangbly since one can be generated from the other. Note that $0 \leq |STX_i| \leq n-i+1$, and there are 2^n different STX_i, and we aim to select STX_1 with a reliability of at least R_{min}, *i.e.*, Rel(G_1)$\geq R_{min}$ and minimum Cost(G_1).

Let DP[1.. n, 0.. $\check{R}_{min}$] be a 2-dimension DPtable, where $\check{R}_{min}=round(\delta \times R_{min})$, for a positive integer multiplier δ and a function *round*(•) that returns the closest integer value of (•). 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≤R[i, $\check{r}$]≤1.0, ST[i, $\check{r}$]$\subseteq ST_G$, a set of links L[i, $\check{r}$]$\subseteq$E, and an integer index 0≤J[i,$\check{r}$]≤δ. In essence, the columns of DP table partition the reliability constraint R_{min}into δconsecutive reliability constraints, *i.e.*, R_{min}/δ, (2×R_{min})/δ,…,δ×R_{min}/δ=R_{min}. In other words, each column index $\check{r}$=0, 1, …, $\check{R}_{min}$, corresponds to a reliability constraint r=0, 1/δ, …, ($\check{R}_{min}$/δ)≈R_{min}, *i.e.*, r=$\check{r}$/δ and $\check{r}$=*round*(δ×r), and each DP[i,$\check{r}$] is used to store four pieces of information of each selected topology G_i that has Rel(G_i)≥r. Specifically, for each Rel(G_i)≥r, we set C[i,$\check{r}$]=Cost(G_i), R[i, $\check{r}$]=Rel(G_i), ST[i, $\check{r}$]=STX_i, and L[i, $\check{r}$]=E_i. For Rel(G_i)<r, we set C[i, $\check{r}$]=∞, R[i, $\check{r}$]=0, ST[i, $\check{r}$]={}, and L[i, $\check{r}$]=( ). Note that C[i,$\check{r}$]=0 is not possible since each link is assumed to have a non-zero cost. Since C[1, $\check{R}_{min}$] is the cost of G_1=(V, $E_1 \subseteq$E) with Rel(G_1)≥R_{min}, NTD-CR aims to generate DP[1, $\check{R}_{min}$] that contains the minimum C[1, $\check{R}_{min}$], 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 J[i,$\check{r}$]=$\check{r}2$. Thus, index J[i,$\check{r}$]=0, 1, 2, …,100 marks the ending column of a range of columns that have the same reliability. For example, we store J[i,$\check{r}$]=38 at columns $\check{r}$=0 to $\check{r}$=38 if R[i, 0]=R[i, 1]= … =R[i, 38]. Note that we set 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:

i=n:

$$\text{C}[i, \check{r}]=\text{Cost}(ST_i); \text{ if Rel}(ST_i) \geq r \quad (3)$$

$$\text{C}[i, \check{r}]=\infty; \text{ if Rel}(ST_i) < r \quad (4)$$

i<n and Rel(ST_i)≥ r:

$$\text{C}[i, \check{r}]=\text{Min}(\text{C}[i+1, \check{r}], \text{Cost}(ST_i)) \quad (5)$$

$i<n$ and Rel(L[i+1, j]∪L$_i$)≥ r:

$$C[i, \check{r}]=\text{Min}(C[i+1, \check{r}], \text{Cost}(L[i+1, j] \cup L_i)) \quad (6)$$

Without loss of generality, we consider the spanning tree selection start from the last spanning tree ST$_n$. In (3), when the last spanning tree has reliability of at least r, it should be selected, giving C[n, $\check{r}$]=Cost(ST$_n$). In contrast, when Rel(ST$_n$)<r,ST$_n$is not selected because it does not meet the constraint r; thus (4) sets C[n,$\check{r}$]=∞ to denote that no spanning tree is selected.

Equation (5) and (6) are used for each remaining ST$_i$, for i=n-1, n-2, …, 1. Equation (5) considers two options, selecting or not selecting ST$_i$, when Rel(ST$_i$)≥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 since 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, (5) 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 (6) considers the case when selecting ST$_i$ together with some previously selected trees STX$_j$ satisfies the required reliability r, *i.e.*, Rel(L[i+1, j]∪L$_i$)≥r, for each possible j=J[i,$\check{r}$]=0, 1, …, 100. Like (5), (6) also considers the minimum cost between either selecting or not selecting ST$_i$; the former produces Cost(L[i+1, j]∪L$_i$) and the latter 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, j]∪L$_i$) if ST$_i$ is selected. Further, (6) 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 it will select ST$_i$.

The DP formulation in (3) to (6) is similar to the DP solution for the well-known NP-complete 0/1 knapsack problem [7]. In the 0/1 knapsack problem, there are n items where each *item* has capacity and value and its goal is to select a set of *items* that have the *maximumtotal value* while having *total capacityno larger* than a given capacity constraint. In contrast, NTD-CR aims to select a set of *spanning trees* whose induced topology has *minimumtotal cost* while having *network reliability no less* than a given reliability constraint R_{min}. However, unlike for knapsack 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 (6) must consider all possible values of j, *i.e.*,J[i, $\check{r}$]. Further, while the total capacity of two items in Knapsack 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$_h$∪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.

B. DPCR-ST Algorithm

Fig. 2 shows our proposed DP algorithm, called DPCR-ST, that directly applies (3) to(6).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×Ř$_{min}$.As shown in Fig. 2, DPCR-ST keeps only two consecutive rows, called *row*1 and *row*2, and therefore it requires only a table of size 2×Ř$_{min}$. Specifically, DPCR-ST computes C[1, j] and R[1, j] in *row*1 using the information in C[2, $\check{r}$] and R[2, $\check{r}$] in *row*2, for all relevant columns $\check{r}$ and j. After copying the contents of *row*1 to *row*2, it repeats the step until all spanning trees are considered.

Line 1 implements (4) while Line 2 to 8 are based on (3). The remainder of the code is used to implement (5) and (6). Specifically, (5) is solved in Line 9 to 21, (6) in Line 22 to 38, and Line 39 to 45 copies the contents of *row*1 to *row*2.

C. DPCR-ST Analysis

The time complexity of DPCR-ST can be computed as follows.The Cost(X) function requires all unique links in the set of spanning trees X.

DPCR-ST Algorithm:

```
1.  Initialize C[2,ř]=∞, R[2, ř]=0, ST[2,ř]={},L[2,ř]=( ),J[2, ř]=Ř_min,
    forRel(ST_n)< r// Equation. (4)
2.  for (ř← 0 to round(δ×Rel(ST_n)) do  // Equation. (3)
3.      C[2,ř] ← Cost(ST_n)
4.      R[2,ř] ← Rel(ST_n)
5.      ST[2,ř] ←ST_n
6.      L[2,ř] ← L_n
7.      J[2, ř]←round(δ×R[2,ř])
8.  end for ř
9.  for (i←n-1 downto 1) do  // Eqs (5)-(6)
10.     for (ř←0 to round(δ×Rel(ST_i)) do // Equation. (5)
11.         C[1,ř] ←Min(C[2, ř], Cost (ST_i))
12.         if C[2, ř]< Cost (ST_i)
13.             ST[1,ř] ← ST[2,ř]
14.             L[1,ř] ← L[2,ř]
15.         else
16.             ST[1,ř] ←ST_i
17.             L[1,ř] ← L_i
18.         end if
19.         R[1,ř] ←Rel(L[1, ř])
20.J[1, ř]← round(δ×R[1,ř])
21.     end for ř
22.     for (y ← 0 toŘ_min) do // Equation.(6)
23.if (J[2, y] ≠J[2, y+1])
24.             j=J[2, y]
25.             if Rel(L[2,  j] ∪ L_i)≥ř
26.                 C[1,ř] ← Min(C[2, ř], Cost (L[2, j] ∪ {L_i}))
27.                 if C[2, ř]<Cost (L[2, j] ∪ L_i)
28.                     ST[1,ř] ← ST[2,ř]
29.                     L[1,ř] ← L[2,ř]
30.                 else
31.                     ST[1,ř] ← ST[2, j] ∪ST_i
32.                     L[1,ř] ← L[2, j] ∪ L_i
33.end if
34.                 R[1,ř] ←Rel(L[1, ř])
35.J[1, ř]← round(δ×R[1,ř])
36.             end if
37.         end if
38.end  fory
39.     for ( y  ← 0toŘ_min) do // copyrow1 to row 2
40.C[2, y ] ←C[1, y ]
41.         R[2, y ] ← R[1, y ]
42.         ST[2, y ] ← ST[1, y ]
43.         L[2, y ] ← L[1, y ]
44.         J[2, y ] ←J[1, y ]
45.     end for y
46. end for i
```

Fig. 2. DPCR-ST Pseudocode

For each $\check{r}$, Cost(X) returns the sum of C[i+1, $\check{r}$] and the cost of links in ST$_i$ that are not inL[i+1, $\check{r}$]. Using the bit

implementation [8], one requires only one bit OR and one bit XOR operation to obtain the links in ST_i that are not inL[i+1,$\check{r}$], and thus for any X, Cost(X) 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(X) function can be implemented using any exact reliability calculation [8], heuristic technique [9] or approximation (bounding) method [2]. In this paper, we use Monte Carlo simulation [9] with time complexity $O(b \times |V|^4)$ [1]to estimate Rel(X) of each candidate network; b is the number of replication. Notice that Rel(X) is used only for each different j in each row i. Hence, in total, the time complexity of using Rel(X) is $O(\psi \times b \times |V|^4)$, where ψ is the total number of different j 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. Improving the Efficiency 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. Using smaller k will reduce the time complexity of the algorithm.

For a given graph G(V, E), we first compute link weight 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 [10] 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 the following orders for the spanning trees in Table I; CR1:(ST_2, ST_7, ST_1, ST_3, ST_6, ST_8, ST_4, ST_5), CR2:(ST_7, ST_2, ST_1, ST_8, ST_6, ST_4, ST_5, ST_3) and CR3:(ST_3, ST_2, ST_5, ST_6, ST_1, ST_8, ST_4, ST_7).Note that (3) to (6) consider spanning trees starting from ST_n, and thus DPCR-ST sets ST_n as the least weighted spanning tree, ST_{n-1} as, the second least weighted, *etc*. Since Yen's algorithm requires a time complexity of $O(k \times |V| \times (|E| + |V| \times \log|V|))$, DPCR-ST requires an extra $O(n \times |V| \times (|E| + |V| \times \log|V|))$ time complexity for the improvement, *i.e.*, $O(\psi \times b \times |V|^4 + n \times |E| \times \check{R}_{min} + n \times |V| \times (|E| + |V| \times \log|V|))$.Note that our DPCR-ST generates only the first k least weight spanning trees. Thus, this improvement does not require all spanning trees a priori, which improves DPCR-ST's time complexity, *i.e.*, $O(\psi \times b \times |V|^4 + n \times |E| \times \check{R}_{min} + k \times |V| \times (|E| + |V| \times \log|V|))$.

V. Simulation and Discussion

We have implemented our DPCR-ST in C language to generate the topology of the 76 fully connected networks in [5] with the number of nodes, links and spanning trees range from 6 to 11, 15 to 55, and 1269 to 2.3×10^9, respectively. We obtained 76 cost matrices from the authors in [5], and use them for all link costs of all networks; the authors [5] randomly generated the integer costs with values between 1 and 100. Like in [5], we set R_{min} to either 0.9 or 0.95 and equal link reliability with value of either 0.9 or 0.95. All simulations using DPCR-ST were run on Intel Core i5 with 2.53 GHz with 4 GB of RAM, running Linux (Ubuntu Core 11.10).

For each of the 76 fully connected network topologies in [5],we first generated its spanning trees in four different orders: random, CR1, CR2, and CR3, described in Section IV.D. We have used Prim's algorithm [10] 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 $C_{best}$is the *minimum* among the costs of topologies generated using random, CR1, CR2, CR3, and column *Rel* stores its reliability.

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

DPCR-ST 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 in Table II and III. The tables show the total number of topologies generated with cost $Cbes_t$ and their cost optimality with respect to C_{min} – the cost of G_{min}, *i.e.*, $C_{best} > C_{min}$, $C_{best} = C_{min}$, $C_{best} < C_{min}$. Note that $C_{min}$is the minimum cost of each topology with reliability at least R_{min} as reported in [5]. As stated in [2], the reliability of each topology with cost C_{min} was estimated using a Monte Carlo method that produces result within 1% of R_{min}.

As shown in Table II, 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 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 and/or CR3.

TABLE II: 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%	4863.1 %	28 36.8 %
CR3	25 32.8%	24 31.5%	6 7.8%	55 72.3%	21 27.6%

As shown in Table II, our DPCR-ST can produce topology with $C_{best} < C_{min}$ because we *round* off each reliability to its closest integer and use a Monte Carlo method [9] that computes reliability within 0.5% of R_{min}. The table also shows that CR1, CR2, and CR3 produce 69.7%, 53.8%, 64.3% of topologies with cost less or equal than C_{min}, respectively. Thus, in term of optimality, CR1 (CR2) is the best (worst) performer.

Table III 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 DPCR-ST to use the three ordering criteria, CR1, CR2 and CR3, and select the best among their results to generate topologies with lower cost. As shown in the table, such approach produces only 18.4% topologies with less optimal cost.

TABLE III: DISTRIBUTION OF CBEST GENERATED USING ONE OR MORE 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%)

B. DPCR-ST versus Existing Approaches

Table IV compares the effectiveness of our DPCR-ST against four state-of-the-art approaches, *i.e.*, NGA [2], LS/NG [3], ACO-SA [5], and BDD [11], using the 76 fully connected networks. Since we were unable to obtain the source codes for the four approaches in [2], [3], [5], and [11], we have used their reported results in our comparisons.

As shown in Table IV, while NGA, LS/NGA and ACO-SA could generate optimal solutions for 16, 24 and 48 out of 76 instances, respectively, and BDD obtain optimal solutions for 14 out of 45 instances, DPCR-ST could produce 62 out of 76 optimal results (81.5%), significantly improving the effectiveness of the existing algorithms.

TABLE IV: COMPARISON BETWEEN DPCR-ST, NGA, LS/NGA, ACO_SA AND BDD

	DPCR-ST	NGA	LS_GA	ACO_SA	BDD
$C_{best}=C_{min}$	32 (42.1%)	16 (21%)	24 (31.5%)	48 (63.1%)	14 (33%)
$C_{best}<C_{min}$	30 (39.4%)	N/A	N/A	N/A	NA

Further, 30 of 62 C_{best} generated using DPCR-ST 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 [2], [3], [5], and [11].

V. CONCLUSION

We have defined a network topology design problem, NTD-CR, to generate topology that has the minimum cost subject to a reliability constraint R_{min}. We have proposed a heuristic based on *dynamic programming*, to solve NTD-CR. Our method DPCR-ST incrementally generates only a selected *k* spanning trees from the network, and is scalable on networks having large number of spanning trees. We have proposed to sort the spanning trees using three different orders to optimize our method's effectiveness and efficiency. The experimental study shows that the DPCR-ST approach is able to generate 81.5% optimal solutions. We plan to design an alternative DP approach that heuristically deletes links from the original topology to find an optimal design.

REFERENCES

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

[2]. B. Dengiz, F. Altiparmak, and A. Smith, "Efficient optimization of all-terminal reliable networks," *IEEE Trans. on Reliability*, vol. 41, no. 1, pp. 18-26, 1997.

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

[4]. C. Papagianni and K. Papadopoulos, "Communication Network Design Using Particle Swarm Optimization," in *Proc. International Multi Conference on Computer Science and Information Technology*, Poland, 2008, pp. 915-920.

[5]. F. Altiparmak, B. Dengiz, O. Belgin, "A hybrid Ant Colony Optimization approach for the Design of Reliable Networks," *IIE Transactions*, vol. 42, pp. 273-287, 2010.

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

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

[8]. 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.

[9]. S. Yeh, J. Lin, W. Yeh, "New Monte Carlo method for estimating network reliability," in *Proc. the 16th International Conference on Computers and Industrial Engineering*, Japan, 1994,pp. 723-726.

[10]. Snippets Dzone. [Online]. Available: http://www.dzone.com/snippets/prims-algorithm-finding.

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

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 area of computer network, 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 Honoursin 1996, and his Ph.D in computer Science from the 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 an 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.