Reliability Improvements for Automation Systems

Vincent Débieux
École Polytechnique Fédérale de Lausanne
vincent.debieux@alumni.epfl.ch

Thanikesavan Sivanthi, Yvonne-Anne Pignolet
ABB Corporate Research Switzerland
firstname.lastname@ch.abb.com

***Abstract*—Automation systems occupy an important place in todays industry, from manufacturing to electrical grid protection and control and much more. They perform complex functions with high reliability requirements. The reliability analysis of such systems is a demanding task for the system engineers. Consequently efficient means to compute the reliability and ways to automatically suggest reliability improvements are needed. To this end, we propose two methods for automatic suggestion of reliability improvements based on duplication and upgrades of automation and communication equipment. The first method identifies critical devices in the system to improve reliability in a greedy manner and the second method recommends reliability improvements using a genetic algorithm. Both methods are evaluated on synthetic and real automation networks topologies, and they identify reliability improvement solutions in a timely manner.**

***Index Terms*—Reliability analysis, Reliability improvement, System Reliability**

I. Introduction

Automation systems comprise of multiple interconnected devices that perform various mission critical functions. These functions have high reliability requirements in order to guarantee the safety and efficiency of such systems. The introduction of standards in automation systems, such as IEC 61850 [1], has facilitated interoperability and flexible allocation of functions. This implies that automation system solutions often include devices from multiple vendors, and they can be highly customized to fit to the customer needs. Lately, there is also an increasing cost pressure on vendors to reduce the capital and maintenance cost of automation systems. Accordingly, the system design engineers are confronted with the challenge of designing a solution that fulfills the reliability requirements of functions, while keeping the cost low. To this end, the design engineers have to analyze the reliability and cost of different options. In case a solution does not fulfill the reliability requirements, then the required reliability is typically achieved by means of introducing redundancy at various levels of the system. This manual exploration of the solution space is tedious and laborious due to the high degree of topological complexity and network redundancy. An automatic exploration of the solution space for reliability improvements can not only reduce the time required for system design but also can help to make better decisions.

While previous work has focused on the network design, the placement of equipment or the addition of communication links, this paper studies how existing automation networks can be improved by *replacing devices with more reliable devices or by duplicating them*. We present two methods for automatic suggestions of reliability improvements. The first method identifies the most critical device in the system and tries to improve the reliability in a greedy manner. The second method generates sets of non-dominated solutions with respect to improvement cost and reliability gain using the Non-dominated Sorting Genetic Algorithm II [8]. Both methods improve the reliability by applying different levels of redundancy to the devices of a given automation system. The performance of both methods are evaluated against different randomly generated industrial automation networks, and a real world example. Note that the same methods can be applied for availability improvement suggestions as well by replacing the reliability calculation with an availability calculation.

II. Background and Related Work

Reliability computation is typically divided into three tasks: first the input (system description, including network topology) is transformed into a graph, then mincuts or minpaths on this graph are enumerated. They are then used to compute the probability that the system will be up at a given time, as opposed to be in a failed state (down). A *cut* is a set of nodes that disconnects a graph when removed. It is minimal (*mincut*) if no subset is also a cut. A path is a set of devices and links that chain from a source to a target. It is minimal (*minpath*) if no subset is also a path. In other words, the probability that the vertices of the graph are in a good state is calculated for all mincuts. Typically, the vertices of the constructed graph are the computing and I/O devices and also the networking components, while the edges represent the dependencies of the components. Thus a cut is a set of vertices that make the system fail if they are all down. There are two main issues when computing the reliability of the system based on this approach. Firstly, minpath/mincut enumeration is a NP-hard problem (Ball [4]). Secondly, the elements in minpaths or mincuts are not necessarily disjoint. As a result, the reliability is typically computed using the classic inclusion and exclusion formula that entails substantial computational complexity (Rausand and Hyland [22]).

Several techniques have been presented over the years in order to tackle the complexity of reliability computation. The reliability computation methods can be classified into two categories: reliability approximation methods and exact reliability methods. Reliability approximation methods (Gobien [10], Sebastio et al. [23]) focus on finding the most relevant minpaths

or mincuts instead of enumerating all of them. The upper and lower bounds are then computed based on these mincuts and minpaths. These methods provide the exact reliability value when run to completion. However, these methods use a large amount of memory and only focus on two-terminal reliability, i.e. the reliability between a pair of devices. Exact reliability computation methods focus on computing the reliability based on non-disjoint minpaths or mincuts. The methods used Sum of Disjoint Product (SDP) (Hariri and Raghavendra [11], Yeh [27] [28]) which are impractical already for medium-sized networks, because of the exponential number of minpaths/mincuts. To this end, factoring/decomposition methods were devised (Misra [17], Wood [26]). They are more efficient than SDP-based methods but still not good enough for medium sized networks. Currently, many exact reliability methods use Binary Decision Diagrams (BDD) (Chang et al. [6], Lin et al. [15]). BDDs (Akers [2], Bryant [5]) are considered to be the state-of-the-art data structure to represent Boolean expressions. A BDD is a rooted, directed, acyclic graph in which every non-leaf node has two children. The reliability can be recursively computed on both children of each BDD node, starting from the root node. Lin et al. [15] propose an efficient *recursive merge* mincut algorithm for computing the reliability and availability of networks. This method was later adapted to the evaluation of reliability analysis of functions by Sivanthi and Görlitz [24]. The advantage of this method is that it enumerates the mincuts using the recursive merge algorithm on $s-t$ graphs (graphs with special source/target nodes s and t) constructed using communication paths between the I/O and computing devices. This prevents the recursive merge algorithm from loosing computation time on irrelevant parts of the network. After the mincut computation, a BDD is used to recursively compute the reliability of the function.

Several methods for reliability improvement in networks based on heuristics algorithms are presented in the literature. Kim and Gen [13] propose a heuristic approach based on genetic algorithms to find near-optimal solutions in designing spanning tree networks. They rearrange the connections of user nodes to a backbone network to optimize the reliability and the cost of the network. Kumar et al. [14] also propose a reliability optimization method based on genetic algorithm to solve a network expansion problem by rearranging the possible connections between the nodes. Cheng [7] uses a genetic algorithm approach to minimize the cost of links in a backbone network, knowing beforehand the set of node and the price of the links between each pair of nodes and with the constraint that the network must be tolerant against single point of failure. The approaches presented in [7], [13], [14] tackle the reliability optimization problem using a genetic algorithm approach but only consider the links connecting the nodes together. In this paper, we tackle the problem by applying different levels of redundancy at the device level, allowing for duplication or upgrade of devices.

This paper aims at providing automatic suggestion of reliability improvements based on the aforementioned modifications for automation system functions. To the best of our knowledge, this is the first work that considers this problem with device duplication and upgrades for arbitrary automation networks. In power systems research, the placement of (electrical) switches for fault isolation in radial distribution systems [16] and planning for distributed generators and cross-connections are studied [30]. Other related work in this area focuses on adding links only and is targeted at road [29] or communication networks [3].

III. System Model

An automation network contains multiple input and output devices as well as computing devices, e.g., controllers. These devices are interconnected with each other via networking components, i.e. network devices and links. As a consequence, a subset of all these devices and links need to be operational at all times for the correct operation of the system function. We assume that the devices can be in a working or failed state. For simplicity, we also assume independent failures and a constant failure rate for the devices and links of industrial automation systems, however, our methods can easily be extended to incorporate other failure models.

In order to compute the reliability of an industrial automation system function and to suggest improvements with an associated cost, our model describes the devices and links in the system together with their MTTF and cost.

Functions model the information necessary to provide services the system has been designed for. They describe which devices act as inputs/outputs (*I/O nodes*) and which devices execute the function (*dependent nodes*). This information is expressed in the form of a dependency expression with AND and OR operators. In the example function in Fig. 1, $d1$ is the dependent node (e.g., a controller) and $d5$ and $d6$ are I/O nodes (e.g., sensors or actuators). Furthermore, the OR operator ("|") between $d5$ and $d6$ implies that they are redundant: at least one of them needs to be in a working state in order for the function to execute successfully. An AND operator ("&") means that both devices need to be in a working state, i.e. they are not redundant. The reliability of each function is considered individually i.e. one at a time.

For the reliability computation it is not necessary to distinguish between automation and communication equipment. As a consequence the following entities are part of the model *devices*, *links* and *function*. The devices and links represent the topology of the automation network. Devices include automation and control equipment (e.g., controllers, sensors and actuators) as well as communication equipment (e.g., switches and gateways). The size of a network is measured with respect to the number of devices, i.e. a network of size x contains x devices. The model is summarized in Fig. 1.

In industrial automation networks, it is common to specify static routing paths for information flow when deploying the network. (Redundant) paths connecting the dependent node

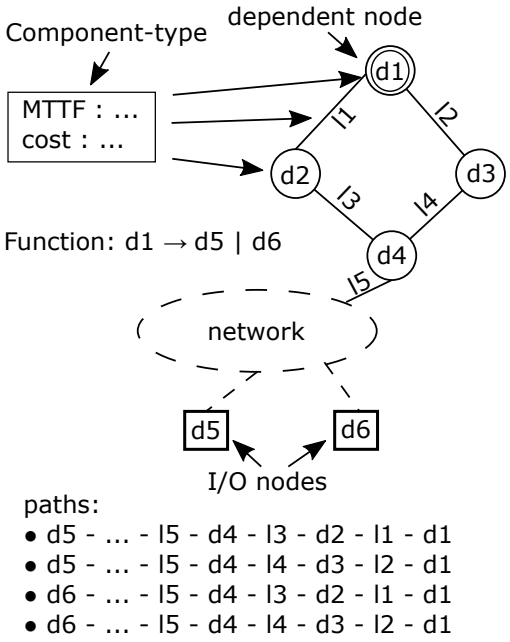

Figure 1. Overview of the system model

and the I/O nodes are thus precomputed. This allows us to focus the computation on the reliability analysis and not on path discovery, which is another complex problem on its own. All networks considered in this paper satisfy the following assumptions:

- The topology is connected.
- Component failures are independent from each other and their failure rate is constant.
- A component is either in the working state (up) or in the failed state (down).
- There can be one or more paths between the dependent node and I/O nodes.
- All paths from any I/O node to dependent nodes are precomputed off-line.
- There are no software failures.

IV. Reliability Computation

An industrial automation system function is reliable if all components required for successful operation of the function are reliable. The components are the I/O devices, the device which executes the function (dependent node), the network devices and links along the path to I/O devices. To this end, for each function only the relevant part of network topology is considered for the network model. The generation of mincuts for computing the function reliability can be complex since the function can have AND and OR dependencies on multiple groups of input/output devices, which can be redundant, and can be connected to the dependent node via redundant paths. Therefore, a two step approach is used for generating the mincuts and is described in Algorithm 1.

The first step is to transform the dependency expression into a graph $\mathcal{DG}$ in which the AND elements are represented by a series structure and OR elements are represented by a parallel structure. The recursive merge algorithm for enumerating mincuts [15] is executed on this graph to generate mincuts MCS_1. Each mincut thus generated consists of a set of devices, which when failing together will cause the function to fail.

Note that the function is not operational when not only the devices in the cut set jointly fail but also when all communication paths to the devices fail. Therefore, for each mincut generated in the first step, the paths between devices in the mincut and the dependent node are considered. This second step generates a graph $\mathcal{SG}$ for each mincut in MCS_1, where $\mathcal{SG}$ represents the elements of the mincut and the communication paths in between. The recursive merge algorithm is then executed on all $\mathcal{SG}$ graphs corresponding to the mincuts in MCS_1, and the resulting mincuts are stored in MCS_2. MCS_2 now contains all mincuts for computing the function reliability.

The mincuts in MCS_2 are not necessarily mutually disjoint, i.e. an element can appear in more than one mincut, this makes the reliability calculation more complex. To reduce the complexity, a Binary Decision Diagram (BDD) is then constructed. An inherent property of BDD is that all paths from its root node to its leaf nodes are mutually exclusive. The BDD for the reliability computation is constructed from MCS_2, where the elements of each mincut in MCS_2 are represented using BDD AND operators. As each mincut in MCS_2 can individually make the function fail, this is represented using BDD OR operations.

Let $R_X(t)$ denote the probability that a component X is in operation state at time t, and MTTF be the Mean Time To Failure of the component. Under the assumption that the failures are independent and the failure rates are constant this corresponds to (1). The reliability of the function is then computed by $1 - \overline{P}_{root}$, i.e. from the root node of the BDD, where $\overline{P}_v$ in (2) is the unreliability calculated recursively on the left children ($\overline{P}_{n_l}$) and right children ($\overline{P}_{n_r}$) of BDD node v.

$$R_X(t) = e^{\frac{-t}{X_{MTTF}}} \tag{1}$$

$$\overline{P}_v = (1 - R_{X_v}(t)) \cdot \overline{P}_{v_l} + R_{X_v}(t) \cdot \overline{P}_{v_r} \tag{2}$$

The computation time for generating all mincuts can be reduced by splitting the graphs at the articulation points and running the recursive merge algorithm on the biconnected components. This provides a speed up of 50% to 85% when applied to randomly generated industrial automation networks [9].

V. Reliability Improvements

This section discusses two approaches that improve function reliability by exploring different modifications to the network

Algorithm 1 Reliability Computation

Input: $\mathcal{G}$ ▷ Network graph
Input: $\mathcal{D}$ ▷ Dependency function
$MCS_1 \leftarrow \{\}$ ▷ mincuts
Step 1:
$\mathcal{DG}$ = CreateDependencyGraph($\mathcal{D}$)
MCS_1 = RecursiveMerge($\mathcal{DG}$)
Step 2:
for all $mcs \in MCS_1$ **do**
    $\mathcal{P}$ = DeducePaths($\mathcal{G}$, mcs)
    $\mathcal{SG}$ = CreateGraph($\mathcal{P}$,mcs)
    MCS_2 = MCS_2 $\cup$ RecursiveMerge($\mathcal{SG}$)
end for
Step 3:
$\mathcal{BDD}$ = CreateBDD(MCS_2)
ComputeReliability($\mathcal{BDD}$)

in line with the following assumptions.[1]

Assumption 1. *Devices can be upgraded*

The simplest modification is to replace a device by another device that fulfills the same function and has a higher MTTF. This implies when a device is upgraded, it is replaced by a component in the same category with a better MTTF.

Assumption 2. *Two devices cannot be merged*

In a series structure, the total reliability is the product of each component's reliability. Thus it might be interesting to consider merging two devices together. However some devices must stay physically close to some other device, i.e two devices A and B connected over WiFi or Bluetooth. Merging A with another device C would increase the distance between A and B, lowering their ability to communicate.

Assumption 3. *Devices can be duplicated*

Another possible modification is to create redundancy by duplicating a device and all its connections. This will add paths between the sources and targets and thus improve the reliability of the function. More precisely, all paths going via the original device are replicated through the duplicated device.

Link duplication and upgrade are also valid modifications but considering them implies a larger search space. In order to keep computation times low, they are not taken into account.

Assumption 4. *For device upgrades the difference between their costs is spent.*

We assume that upgrading device x to device y yields a cost of $cost(y) - cost(x)$ and duplicating device x yields a cost of $cost(x) + \sum_i cost(link_{x,i})$, for all nodes i which are neighbors of x. The impact of these modifications on the topology as well as their implications on the cost model are shown in Fig. 2.

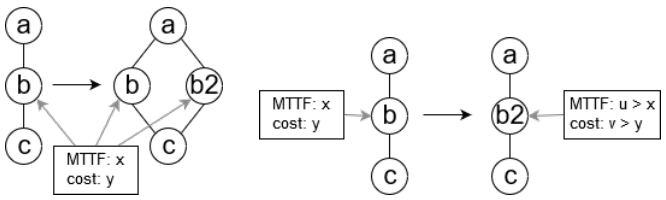

Figure 2. Topology changes due to duplications (left) and upgrades (right)

A. Greedy approach

The first approach towards reliability improvement suggestion is based on selecting components greedily for a duplication or upgrade based on the importance of the components for the function reliability. We first list the importance measures we considered and then describe the greedy algorithm in more detail.

1) Importance measure: Rausand and Høyland [22] describe methods to compute the importance of devices in a system. The methods are summarized briefly here.

(A) *Birnbaum measure*: The Birnbaum importance measure $I^B(i,t)$ is computed as the difference between the system reliability given that the component i is functioning at time t and the reliability of the system with component i in failed state. The measure is dependent on the structure of the network and the reliabilities of the other components.

$$I^B(i,t) = R(1_i,t) - R(0_i,t) \tag{3}$$

(B) *Improvement potential (IP)*: The IP measure is obtained similarly to the Birnbaum measure with the difference that it takes the reliability of the component into account. It is computed as the difference between the system reliability considering that the component i is replaced by an ideal one and the actual system reliability.

$$I^{IP}(i,t) = R(1_i,t) - R(t) \tag{4}$$

(C) *Risk achievement worth (RAW)*: The RAW measure is the ratio of system unreliability with the component removed from the system and the unreliability of the actual system. The higher the ratio, the more important is the component for system reliability.

$$I^{RAW}(i,t) = \frac{1 - R(0_i,t)}{1 - R(t)} \tag{5}$$

(D) *Risk reduction worth (RRW)*: The RRW measure is the ratio of system unreliability and the unreliability of the system considering that component i is replaced by an ideal one. It represents how much the unreliability can be decreased if the component is upgraded.

$$I^{RRW}(i,t) = \frac{1 - R(t)}{1 - R(1_i,t)} \tag{6}$$

[1]The addition of a communication link between two devices may shorten the path between an I/O node and a dependent node for a relatively low cost and thus increase reliability significantly. As this aspect of reliability improvements has been studied in related work (see Section II), we do not consider link additions in this paper.

(E) *Fussell-Vesely's measure (FV)*: FV's measure calculates the probability that at least one minimal cut set D_j^i that contains component i has failed at time t given that the system has failed at time t as shown in (7), where $P(D_j^i, t)$ is the probability that the minimal cut set D_j^i containing component i has not failed at time t.

$$I^{FV}(i,t) = \frac{1 - \prod_j P(D_j^i, t)}{1 - R(t)} \tag{7}$$

2) Algorithm: The greedy approach for reliability improvement exploits the importance of components. It greedily improves the reliability by duplicating or upgrading devices in the network one at a time. The algorithm's execution can be decomposed in three steps. First, the most critical device is identified using one of the importance measures described in Section V-A. Second, two new instances of the network are created, in one instance the device and its connections are duplicated and in the other instance the device is upgraded. Finally, the reliability of each instance is computed and the network instance with highest reliability is considered for the next iteration. These steps are repeated for a given number of iterations. The number of paths using the device is doubled, with each device duplication. In order to avoid an explosion of the number of paths leading to complex graphs for the mincut algorithm, the number of duplications is limited. Once this limit is reached, the greedy algorithm stops duplicating devices. It then simply creates only one network instance in which the device with the highest importance measure is upgraded. It should be noted that duplicating a device modifies the topology and thus the mincuts need to be enumerated again. On the other hand, upgrading a device only requires to recompute the reliability on the BDD because the topology stays unchanged. The algorithm stops when the maximum number of iterations has been reached or no further improvements are possible because the number of duplications has been reached and no devices can be upgraded anymore. The pseudocode of this method is shown in Algorithm 2.

Note that this algorithm can be adapted to take the cost of the improvements into account by incorporating the cost in the importance function and/or by selecting the instance to keep using a weighted sum of cost and reliability instead of deciding only based on reliability.

Algorithm 2 Greedy Algorithm

```
input network, function, max_iterations, max_duplications,
importance measure
it = 0, duplications = 0
while it < max_iterations ∧ improvements possible do
    Instances = {}
    /*Descending order of importance*/
    C = Sort(Importance(network, function))
    /*Get the most critical device*/
    dev = C[0]
    if duplications < max_duplications then
        Instances.add(network.Upgrade(dev))
        Instances.add(network.Duplicate(dev))
    else
        Instances.add(network.Upgrade(dev))
    end if
    /*Select the instance with highest reliability*/
    network = max(Reliability(Instances))
    if max(Reliability(instances) is a duplication then
        duplications++
    end if
    it++
end while
```

B. Evolutionary Approach

The principle behind genetic algorithms is to mimic the process of natural evolution to solve optimization problems. A genetic algorithm works on a population of solutions called *chromosomes*. Pairs of chromosomes are selected from the initial population to generate new chromosomes by crossover. The newly generated chromosomes undergo gene mutation before being added to the new population. Only the best half of the population is kept for the next round of generations. The probabilities of both mutation and crossover, the population size, and the maximum number of generations can be configured. In order to find different trade-offs between the cost due to modifications and the reliability gain, the situation is posed as a multi-objective optimization problem that is solved using Non-dominated Sorting Genetic Algorithm (NSGA2) [8]. The NSGA2 algorithm sorts the individual solutions of the population based on their ranks. The rank of a solution is defined by the number of solutions that it dominates [21]. In our problem setting, the solutions with higher ranks have lower cost and higher relative reliability gain than lower rank solutions. The members that have equal rank are grouped into the same front. The algorithm finds multiple fronts where the members of the fronts with higher ranks dominate the members of the fronts with lower ranks. The population for the next run is selected using binary tournament selection based on the rank and crowding distance, which is a measure of how close the member is to its neighbors with respect to their chromosomes. The first half of the rank sorted population is used to generate new off springs by means of crossover and mutation. The parents and the offspring constitute the population for the next run. The algorithm continues until the maximum number of iteration is reached and at the end the solutions from the first front are stored in the result set. Thus this method considers not only the reliability of improvements but also their costs. The pseudocode of this method is shown in Algorithm 3.

The advantage of using NSGA2 is that it requires no knowledge of the problem at hand and is thus often used as a black box, i.e. an encoding scheme is used and the meta-heuristics does not know what the encoding represents. Binary is the most commonly used encoding and it is valid

Table I
DESCRIPTION OF BITSTRINGS

Binary	Modification
000	Device stays unchanged
001	Device is upgraded once
010	Device is upgraded twice
011	Device is upgraded thrice
100	Device is duplicated once
101	Device is duplicated once and then one is upgraded once
110	Device is duplicated once and then one is upgraded twice
111	Device is duplicated once and then both are upgraded once

for our problem at hand. Following the assumptions made earlier, we devise an encoding scheme illustrated in Fig. 3. A chromosome comprises of sequence of all devices considered for improvements. Each of these devices is represented by a sequence of 3 bits whose meaning is shown in Table I.

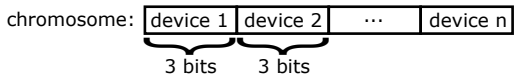

Figure 3. Encoding scheme

The encoding allows up to two duplicates for every device or to upgrade the device up to three times. To allow for more upgrades, one can add more encoded bits to the bit string. Allowing for more duplications however will greatly increase the computation time as the minimum cuts need to be enumerated anew. For the same reasons as in the greedy approach, only a limited number of duplications is allowed in one chromosome. If a suggested solution involves the upgrade of a device that is not upgradable, i.e. it is at the top of the upgrade chain, or duplicates more devices than allowed, then the chromosome is declared invalid and a new one is generated. Not invalidating these chromosomes would allow distinct chromosomes to have the same value in the non-dominated front.

Algorithm 3 Genetic Algorithm

input network, function, max_iterations, max_duplications, population_size, crossover_probability, mutation_probability
iterations = 0, duplications = 0
$\mathcal{R} = \{\}$ ▷ Set of results
$\mathcal{P}$ = CreateInitialPopulation(network, population_size)
SortPopulationByRank($\mathcal{P}$)
while iterations $<$ max_iterations **do**
  $\mathcal{C}$ = SelectFirstHalf($\mathcal{P}$)
  $\mathcal{O}$ = GenerateOffspring($\mathcal{C}$, crossover_probability, mutation_probability)
  $\mathcal{P} = \mathcal{C} \cup \mathcal{O}$
  SortPopulationByRank($\mathcal{P}$)
end while
$\mathcal{R}$ = GetFirstRankSolution($\mathcal{P}$)

VI. EVALUATION

In this section we describe our results obtained from the evaluation of the greedy and the evolutionary approach (denoted by GA in this section) on synthetic automation networks. We compare the computation time, the relative reliability gain and the relative improvement cost of both approaches. All experiments are conducted on a 64-bit Linux machine with 2.2 GHz Intel CPU and 64 GB RAM. We considered measurements on 100 different randomly generated instances for experiments with synthetic automation networks. We used the jMetalCpp[2] NSGA2 library for the evolutionary approach. Our implementation of the *recursive merge* algorithm uses the Boost Graph Library[3]. The reliability computation uses the Colorado University Decision Diagram (CUDD) library presented by Somenzi [25]. *Box-and-Whisker* plots are used to represent the data resulting from the following experiments. The box spans from the 1st quartile to the 3rd quartile. The median (2nd quartile) is represented by an horizontal line inside the box. The length of the whiskers is set to 1.5 times the interquartile range (width of the box).

A. Synthetic Automation Networks

Typical automation networks with high dependability requirements often contain ring network parts (HSR [12]) and parallel network parts (PRP [12]). Furthermore they are sparse and often follow a hierarchical layout. Thus they can be represented by generalized block graphs with cycles and complete bi-partite graphs as motifs [19]. Here PRP networks correspond to a $K_{2,n}$ complete bi-partite graph, with $n-1$ leaf nodes and one aggregating node. Analogously a cycle corresponds to a HSR ring. In our experiments, we thus generate random synthetic automation networks using blockgraphs.

The synthetic networks are generalized block graphs. Four motifs are considered in place of the cliques in standard block graphs: series, ring networks (HSR [12]), star networks and parallel networks (PRP [12], complete bipartite graphs, $K_{2,k+1}$). The generation process starts with one of these motifs and recursively appends more of the above motifs to randomly chosen nodes until the maximal network size (i.e. the number of devices) is reached. To each of the devices and links the lowest cost and MTTF of their component category are assigned. Once the network is constructed, the next task is to create a dependency function and to define all *path*s.

The I/O devices are chosen randomly among all the devices according to the given proportion of network size, and the dependency expression is built by recursively splitting this set of devices in two groups and adding an AND or OR operator in the middle. In order to keep the networks analyzable, the OR operators involve at most four devices. In other words, OR operators are only allowed in the lower levels of the dependency expression. Finally, up to k shortest paths between each I/O device and the dependent node are extracted. In case there are

[2]http://jmetalcpp.sourceforge.net/
[3]http://www.boost.org/doc/libs/1_63_0/libs/graph/doc/index.html

Table II
CATEGORIES OF COMPONENTS

Category	Cost [kUSD] min	Cost [kUSD] max	MTTF [years] min	MTTF [years] max
High-end computing device	5	10	45	55
Low-end computing device	0.1	5	40	50
Networking devices	0.5	3.5	18	22
Special devices	20	50	250	310
Links	0.1	1	50	60

more than k paths available, then the first k shortest paths are chosen.

Reliability improvement suggestion requires a cost function to evaluate the cost of a solution. The cost function is the sum of the cost of all components in the network. In our evaluation the components are modeled by categories. The number of components n and the range of values for their cost and MTTF can be set in each category. The idea behind this model is that a more reliable component with similar functionality costs more. Following this reasoning, n costs and n MTTF values are drawn from a uniform distribution ranging from the minimum to the maximum values corresponding to the category. Finally, n components are created with the n MTTF values. The costs are assigned such that the most reliable component is also the most expensive, the second most reliable is the second most expensive, and so on until all costs are assigned. Such a model allows for a higher level abstraction of the actual cost model. Furthermore, it implicitly creates an order for component upgrades that can be used for reliability improvement, i.e. a component can be upgraded to the next better one in the cost/MTTF scale.

Table II provides the values used in the cost function. The *Special devices* category represents highly reliable and expensive devices such as circuit breakers or current transformers. The cost and MTTF values are provided by industry references. The MTTF value of components in one category vary from +/- 10%. The number of components per category is set to four to keep it simple. Furthermore, randomly generated networks considered for reliability improvements always start with every device and link having the lowest value in the component category. This is to see the potential gain starting from a totally less reliable network to an optimized one.

The greedy approach and the GA approach are evaluated using random generated synthetic automation network instances, that have 30% nodes as I/O devices and use only AND operators in the dependency function. For the generated networks 2 communication paths between each I/O node and the dependent node are precomputed where possible. The number of iterations in both algorithms is set equal to two times the network size. In order to evaluate the algorithms for reliability improvements in a reasonable computation time, the total number of duplications in the network is set to 6 in both algorithms. The networks considered for reliability improvements start with every device and link having the lowest values in the component entity type.

1) Importance Measures: First, the different importance measures and their impact on the reliability improvement suggestions of the greedy approach are evaluated. To this end, the greedy algorithm is run with different importance measures on the same network instances. The given importance measure is used to rank and select the most critical device in each iteration of the greedy algorithm. The relative cost for improvements, relative reliability gain and the computation time of the greedy algorithm for the different measures are shown in Fig. 4, Fig. 5, and Fig. 6 respectively.

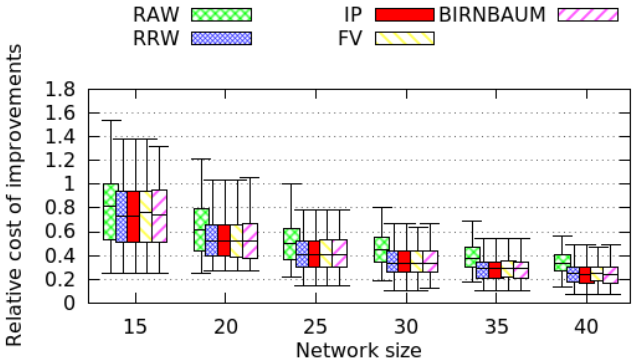

Figure 4. Relative cost for improvements of the different importance measures

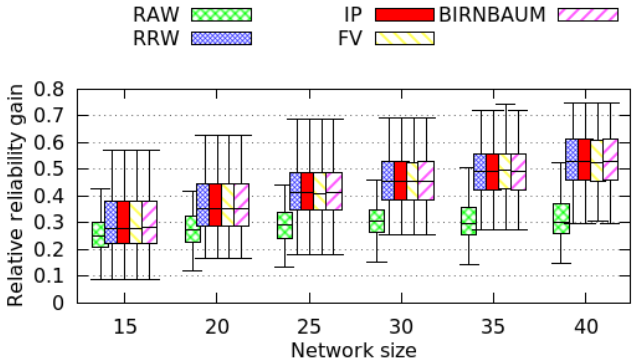

Figure 5. Relative reliability gain of the different importance measures

For all network sizes, all except RAW output higher relative reliability gain for a lower cost. This is because in RAW all components in series will have the same importance irrespective of their individual reliability values, and for components in parallel RAW gives higher importance to the component with highest reliability. This results in lower relative reliability gains for higher cost. All other measures output similar relative cost for improvements and relative reliability gain, but the greedy approach computes faster when using RRW or IP as importance measure. The Birnbaum and FV measures are slower that RRW or IP and the slowest is RAW. The reason RRW and IP are

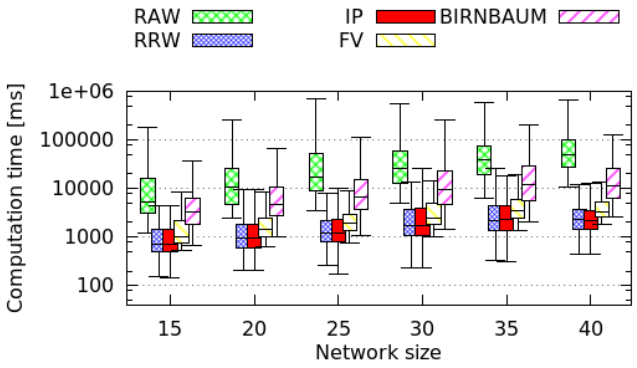

Figure 6. Computation time of the different importance measures

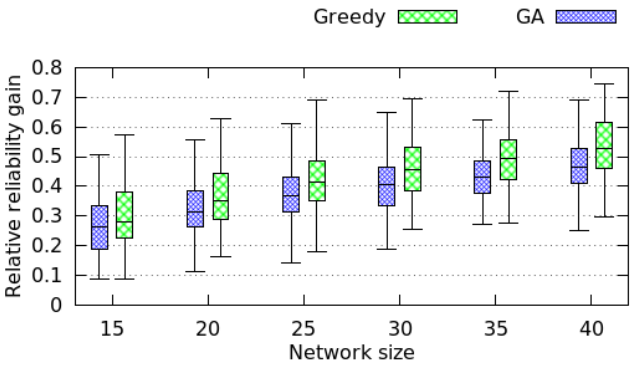

Figure 8. Relative reliability gain of Greedy and GA

faster is because both measures are based on upgrades, which implies that the network topology is not changed and the reliability can be simply calculated by using the new value for the upgraded device. On the other hand, Birnbaum, FV and RAW measures require recalculation of reliability without the component. This implies that the IP and RRW are the measures of choice when looking for the most critical components in the network from the reliability improvement point of view.

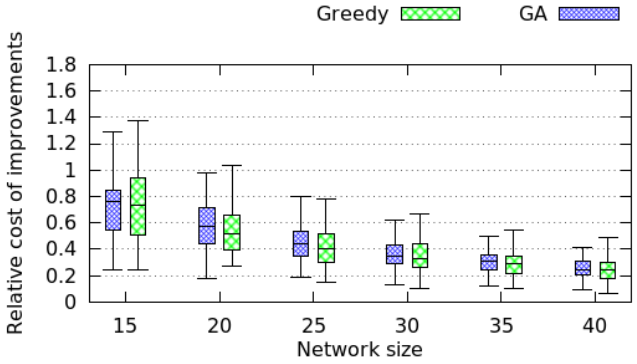

Figure 7. Relative improvement cost of Greedy and GA

2) Reliability Improvements: Let us now compare the performance of the GA and Greedy approach. Greedy focuses on the reliability gain, while NSGA2 takes the cost into account as well. NSGA2 has quite a few parameters that affect the outcome. Since it is not the focus of this paper to find the best parameters, we have used the typical values for mutation and crossover probabilities which are 1% and 90% respectively. The relative cost for improvements, relative reliability gain and the computation time of the greedy algorithm using RRW and GA are shown in Fig. 7, Fig. 8, and Fig. 9 respectively. It can be seen from Fig. 8 that Greedy approach results in a higher reliability gain than GA for all network sizes. The median relative reliability gain is 6 to 12% lower than that of Greedy. Moreover, GA produced results that have slightly higher cost than the Greedy approach. The median relative improvement

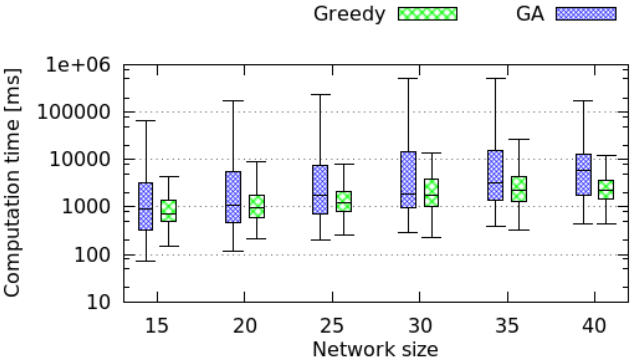

Figure 9. Computation time of Greedy and GA

cost of GA is 3 to 11% higher when compared to Greedy. The computation time of GA is also higher than that of Greedy and also has a larger variance in comparison to the Greedy approach.

The individual solutions obtained by both approaches for an example network instance of size 40 is shown in Fig. 10. The figure shows the final set of solutions in the non-dominated front of GA as well as the solutions produced in the different iterations of the Greedy algorithm. The Greedy approach results in a solution that has higher reliability than GA. Moreover, for an equivalent cost, the solutions produced by GA have higher reliability gains than that of the solutions produced by Greedy algorithm as can be seen from Fig. 10. When given enough time, GA would eventually find the solution with higher reliability or even a better one than Greedy. Hence, if the objective is to find a solution that improves the reliability quickly, then it is better to use the Greedy approach. On the other hand, the non-dominated front of the GA is best suited for analyzing the Pareto trade-offs between cost and reliability.

B. Industrial Case Study

As a real-world case study we analyzed the NanoTera smart grid project [18], a smart grid automation system at the EPFL campus. The system performs real-time monitoring of a

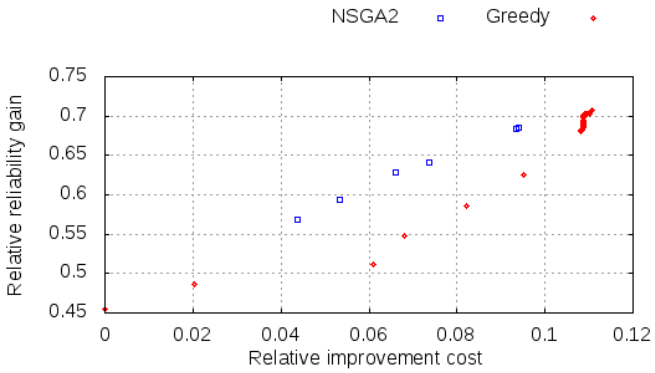

Figure 10. Comparison of solutions generated by GA and Greedy approach

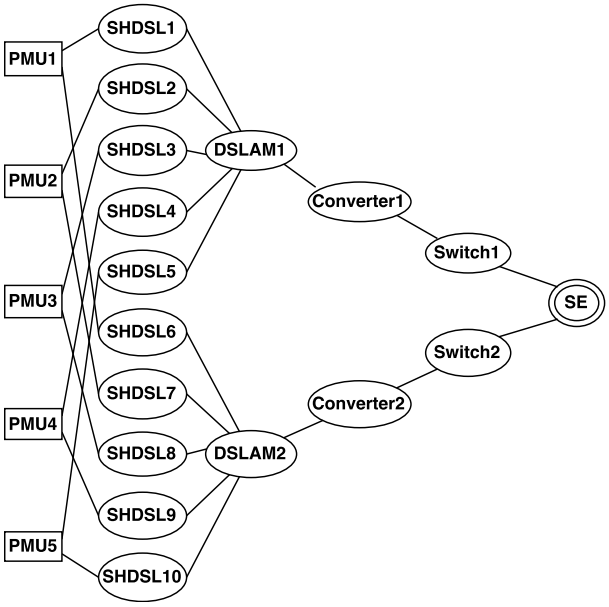

Figure 11. NanoTera network

medium voltage grid using Phasor Measurement Units (PMUs). The PMUs provide synchronized phasor measurements of different feeders in the smart grid to the State Estimator (SE), which estimates the state of the smart grid. The communication network for this project is built on previously existing copper telephone lines and thus the PMUs are directly connected to Symmetrical High-speed Digital Subscriber Line modems (SHDSL). At the other end of the telephone lines, a Digital Subscriber Line Access Multiplexer (DSLAM) merges the traffic into a single line and a converter takes care of translating the packets from SHDSL to Ethernet. Finally a switch forwards packets from the Converter to the SE. The PMUs communicate with the SE using the iPRP protocol [20], which is an adaptation of the PRP protocol for IP networks. In short, for every packet the PMUs send they also send a duplicate on a disjoint path to the destination. Thus, the entire communication network described above is duplicated,

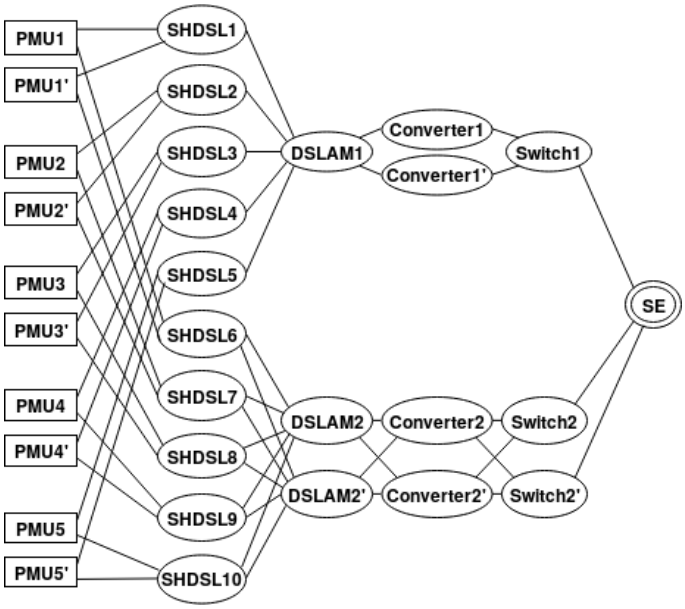

Figure 12. Improved NanoTera network

as shown in Fig. 11. See [18] for more details. The function that is analyzed considers all the PMUs as I/O nodes and the SE as the dependent node i.e. the dependency expression is SE $\rightarrow$ PMU1 & PMU2 & PMU3 & PMU4 & PMU5.

The MTTF and the cost estimates of all components are presented in Table III. Due to the symmetry in the network, the importance of the devices depends on their types. The converters have the highest importance because of their low MTTF. Since the PMUs are not duplicated, they are crucial as well. On the other hand the SE is of lower importance because its MTTF is very high.

Table IV shows the values corresponding to the maximum reliability achieved using greedy algorithm and GA. The maximum number of duplications and the number of iterations are set to 10 and 20 respectively. The greedy algorithm uses RRW measure for selection of critical components. The results show that both the cost and the reliability improvement achieved for NanoTera using the greedy algorithm are higher than that of the GA. Fig. 12 shows an improved reliability solution that duplicates the converters, the PMUs, one of the DSLAMs and a Switch.

Table III
CLASSES OF COMPONENTS

	price [USD]		MTTF [years]	
Class	min	max	min	max
PMU	1200	1600	50	55
SHDSL	160	210	25	30
DSLAM	4000	5000	45	50
Converter	100	150	8	11
Switch	300	400	35	40
State Estimator	3000	4000	80	90
Links	30	100	50	55

Table IV
RELIABILITY IMPROVEMENTS THROUGH GREEDY APPROACH

Method	Reliability [%]		Cost [kUSD]		Computation
	Initial	Final	Initial	Final	time [s]
Greedy	75.67	96.93	541	1141	52
GA	75.67	91.04	541	976	102

VII. CONCLUSION

This paper proposes two methods for reliability improvement recommendations for automation system functions by applying modifications to the automation system network. The first method improves reliability using a greedy algorithm and the second method uses genetic algorithm for making reliability improvement recommendations. The greedy method converges quickly to a solution with a higher reliability. It can be used for quickly finding a solution with higher reliability gain. The approach based on genetic algorithm is best suited for analyzing the Pareto trade-offs between cost and reliability. As future work, it would be interesting to parallelize the evaluation of solutions of both approaches and to use approximation algorithms as proposed in [23] and adapt them for the computation of function reliability.

ACKNOWLEDGMENT

The authors would like to thank Jean-Yves Le Boudec, École Polytechnique Fédérale de Lausanne for his inputs and support.

REFERENCES

[1] IEC 61850 Communication Networks and Systems in Substations, 2003.
[2] S. B. Akers. Binary decision diagrams. *IEEE Trans. on Computers*, C-27(6):509–516, June 1978.
[3] M. M. Atiqullah and S. S. Rao. Reliability optimization of communication networks using simulated annealing. *Microelectronics reliability*, 33(9):1303–1319, 1993.
[4] M. O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. on Reliability*, 35(3), 1986.
[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug. 1986.
[6] Y.-R. Chang, H.-Y. Lin, I.-Y. Chen, and S. yen Kuo. A cut-based algorithm for reliability analysis of terminal-pair network using obdd. In *Proc. 27th Computer Software and Applications Conf. COMPAC*, 2003.
[7] S.-T. Cheng. Topological optimization of a reliable communication network. *IEEE Transactions on Reliability*, 47(3):225–233, Sept. 1998.
[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, Apr. 2002.
[9] V. Débieux, Y.-A. Pignolet, and T. Sivanthi. Faster exact reliability computation. In *Proc. Int. Conf. Dependable Systems and Networks*, 2017.
[10] W. Dotson and J. Gobien. A new analysis technique for probabilistic graphs. *IEEE Trans. on Circuits and Systems*, 26(10):855–865, Oct. 1979.
[11] S. Hariri and C. S. Raghavendra. Syrel: A symbolic reliability algorithm based on path and cutset methods. *IEEE Trans. on Computers*, C-36(10):1224–1232, Oct. 1987.
[12] IEC. Industrial Communication Networks High Availability Automation Networks Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR), 2010.
[13] J. R. Kim and M. Gen. Genetic algorithm for solving bicriteria network topology design problem. In *Proc. Congress Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, page 2279 Vol. 3, 1999.
[14] A. Kumar, R. M. Pathak, and Y. P. Gupta. Genetic-algorithm-based reliability optimization for computer network expansion. *IEEE Transactions on Reliability*, 44(1):63–72, Mar. 1995.
[15] H.-Y. Lin, S.-Y. Kuo, and F.-M. Yeh. Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD. In *Proc. Eighth IEEE Symp. Computers and Communications. ISCC*, 2003.
[16] Y. Mao and K. N. Miu. Switch placement to improve system reliability for radial distribution systems with distributed generation. *IEEE Transactions on Power Systems*, 18(4):1346–1352, 2003.
[17] K. B. Misra. An algorithm for the reliability evaluation of redundant networks. *IEEE Trans. on Reliability*, 19(4), 1970.
[18] M. Pignati, M. Popovic, S. Barreto, R. Cherkaoui, G. D. Flores, J. Y. L. Boudec, M. Mohiuddin, M. Paolone, P. Romano, S. Sarri, T. Tesfay, D. C. Tomozei, and L. Zanni. Real-time state estimation of the epfl-campus medium-voltage grid by using pmus. In *Proc. IEEE Power Energy Society Innovative Smart Grid Technologies Conf. (ISGT)*, pages 1–5, Feb. 2015.
[19] Y. A. Pignolet, S. Schmid, and G. Tredan. Adversarial Topology Discovery in Network Virtualization Environments: a Threat for ISPs? *Distributed Computing*, 28(2):91–109, 2015.
[20] M. Popovic, M. Mohiuddin, D. C. Tomozei, and J. Y. L. Boudec. iprp —the parallel redundancy protocol for IP networks: Protocol design and operation. *IEEE Transactions on Industrial Informatics*, 12(5):1842–1854, Oct. 2016.
[21] R. V. Rao. Teaching-learning-based optimization algorithm. In *Teaching Learning Based Optimization Algorithm*, pages 9–39. Springer, 2016.
[22] M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. JOHN WILEY & SONS INC, 2003.
[23] S. Sebastio, K. S. Trivedi, D. Wang, and X. Yin. Fast computation of bounds for two-terminal network reliability. *European Journal of Operational Research*, 238(3):810–823, 2014.
[24] T. Sivanthi and O. Görlitz. A systematic approach for calculating reliability of substation automation system functions. In *Proc. IEEE Int. Energy Conf. and Exhibition (ENERGYCON)*, pages 957–962, Sept. 2012.
[25] F. Somenzi. Cudd: Cu decision diagram package release 3.0.0. http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf, Dec. 2015. [Online; accessed 21-February-2017].
[26] R. K. Wood. Factoring algorithms for computing k-terminal network reliability. *IEEE Trans. on Reliability*, 35(3), 1986.
[27] W.-C. Yeh. A new sum-of-disjoint-products technique to determine network reliabilities with known minimal paths. In *Proc. Third Int. Conf. Information Technology and Applications (ICITA)*, 2005.
[28] W. C. Yeh. An improved sum-of-disjoint-products technique for symbolic multi-state flow network reliability. *IEEE Trans. on Reliability*, 64(4):1185–1193, Dec. 2015.
[29] Y. Yin, W. H. K. Lam, and M. A. Miller. A simulation-based reliability assessment approach for congested transit network. *Journal of Advanced Transportation*, 38(1):27–44, 2004.
[30] I. Ziari, G. Ledwich, A. Ghosh, and G. Platt. Integrated distribution systems planning to improve reliability under load growth. *IEEE Transactions on power delivery*, 27(2):757–765, 2012.