

# *Complex System Reliability*

## *A Graph Theory Approach*

**Todd Silvestri**

We demonstrate a method of generating an exact analytical expression for the reliability of a complex system using a directed acyclic graph to represent the system's reliability block diagram. Additionally, we show how statistical information stored in a reliability block diagram can be used to transform an analytical expression into a time-dependent function for system reliability.

### ■ Introduction

Among its many interpretations, the term reliability most commonly refers to the ability of a device or system to perform a task successfully when required. More formally, it is described as the probability of functioning properly at a given time and under specified operating conditions [1]. Mathematically, the reliability function is defined by

$$R(t) = P(T > t) \text{ for } t \geq 0,$$

where  $T$  is a nonnegative random variable representing the device or system lifetime.

For a system composed of at least two components, the system reliability is determined by the reliability of the individual components and the relationships among them. These relationships can be depicted using a reliability block diagram (RBD).

Simple systems are usually represented by RBDs with components in either a series or parallel configuration. In a series system, all components must function satisfactorily in order for the system to operate. For a parallel system to operate, at least one component must function correctly. Systems can also contain components arranged in both series and parallel configurations. If an RBD cannot be reduced to a series, parallel, or series-parallel configuration, then it is considered a complex system.

This article deals with the generation of an exact analytical expression for the reliability of a complex system. The demonstrated method relies on finding all paths between the source and target vertices in a directed acyclic graph (i.e., RBD), as well as the inclusion-exclusion principle for probability.

### A Note on Timings

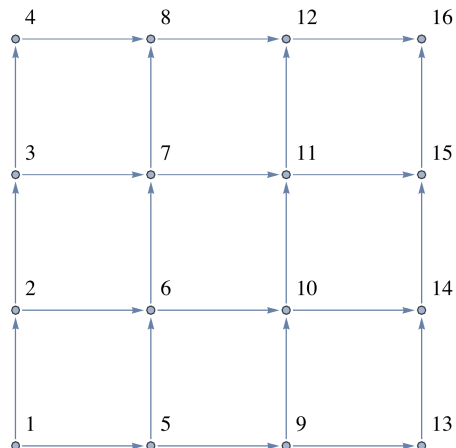
The timings reported in this article were measured on a custom workstation PC using the built-in function `Timing`. The system consists of an Intel® Core™ i7 CPU 950 @ 4 GHz and 24 GB of DDR3 memory. It runs Microsoft® Windows™ 7 Professional (64-bit) and scores 1.32 on the *MathematicaMark9* benchmark.

## ■ Finding All $(s, t)$ -Paths in a Directed Acyclic Graph

We begin by considering a directed graph  $G = (V, E)$  that consists of a finite set  $V$  of vertices together with a finite set  $E$  of ordered pairs of vertices called directed edges. The built-in function `Graph` can be used to construct a graph from explicit lists of vertices and edges.

```
vertices = Range[16];
edges = {1 → 2, 1 → 5, 2 → 3, 2 → 6, 3 → 4, 3 → 7, 4 → 8,
        5 → 6, 5 → 9, 6 → 7, 6 → 10, 7 → 8, 7 → 11, 8 → 12,
        9 → 10, 9 → 13, 10 → 11, 10 → 14, 11 → 12, 11 → 15,
        12 → 16, 13 → 14, 14 → 15, 15 → 16};

G["GridGraph"] = Graph[vertices, edges,
  GraphLayout → {"GridEmbedding", "Dimension" → {4, 4}},
  ImagePadding → 10, ImageSize → 200, VertexLabels → "Name"]
```



This two-dimensional grid graph, labeled  $G_{4,4}$ , can be constructed much more efficiently by using the built-in function `GridGraph`. Throughout this section, we utilize it to illustrate our functions.

Now, for a vertex  $v \in V$ , we define the set  $N_G^+(v)$  of out-neighbors as

$$N_G^+(v) = \{w \in V \mid vw \in E\},$$

where  $vw$  is taken to mean a directed edge from  $v$  to  $w$ . This is implemented in the function `VertexOutNeighbors`.

```
VertexOutNeighbors[g_?DirectedGraphQ, v_] :=  
  Cases[EdgeList[g], DirectedEdge[v, w_] -> w]
```

```
VertexOutNeighbors[g_?DirectedGraphQ] :=  
  Map[VertexOutNeighbors[g, #] &, VertexList[g]]
```

`VertexOutNeighbors` behaves similarly to the built-in function `VertexOutDegree`. That is, given a graph  $G$  and a vertex  $v$ , the function returns a list of out-neighbors for the specified vertex.

```
VertexOutNeighbors[G["GridGraph"], 1]  
  
{2, 5}
```

If, however, only the graph  $G$  is specified, the function will give a list of vertex out-neighbors for all vertices in the graph.

```
VertexOutNeighbors[G["GridGraph"]]  
  
{{2, 5}, {3, 6}, {4, 7}, {8}, {6, 9}, {7, 10}, {8, 11}, {12},  
 {10, 13}, {11, 14}, {12, 15}, {16}, {14}, {15}, {16}, {}}
```

The order in which the out-neighbors are displayed is determined by the order of vertices returned by `VertexList`.

```
VertexList[G["GridGraph"]]  
  
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

We can implement similar functions to obtain the set  $N_G^-(v)$  of in-neighbors by simply changing `DirectedEdge[v, w_] -> w` to `DirectedEdge[u_, v] -> u`.

The next step toward our goal is to consider a method of traversing a graph. One common approach of systematically visiting all vertices of a graph is known as depth-first search (DFS). In its most basic form, a DFS algorithm involves visiting a vertex, marking it as “visited,” and then recursively visiting all of its neighbors [2]. The function `DepthFirstSearch` implements this algorithm for directed graphs.

```

DepthFirstSearch[g_?DirectedGraphQ, s_] :=
  Block[{$RecursionLimit = 1024, visited = {}},
    dfs[v_] :=
      If[! MemberQ[visited, v],
        AppendTo[visited, v];
        Scan[dfs[#] &, VertexOutNeighbors[g, v]]
      ];

    dfs[s];

    visited
  ]

```

Given a graph  $G$  and a starting vertex  $s$ , `DepthFirstSearch` returns a list of vertices in the order in which they are visited.

```

DepthFirstSearch[G["GridGraph"], 1]

{1, 2, 3, 4, 8, 12, 16, 7, 11, 15, 6, 10, 14, 5, 9, 13}

```

We compare this with the result of the built-in function `DepthFirstScan`.

```

Reap[DepthFirstScan[G["GridGraph"], 1,
  {"PrevisitVertex" -> Sow}]] [[2, 1]]

{1, 2, 3, 4, 8, 12, 16, 7, 11, 15, 6, 10, 14, 5, 9, 13}

```

Next, let us define the function `DirectedAcyclicGraphQ`.

```

DirectedAcyclicGraphQ[g_] :=
  DirectedGraphQ[g] && AcyclicGraphQ[g]

```

If the graph  $G$  is both directed and acyclic, `DirectedAcyclicGraphQ` yields `True`. Otherwise, it yields `False`.

```

DirectedAcyclicGraphQ[G["GridGraph"]]

True

```

Finally, we consider the problem of finding all paths in a directed acyclic graph  $G$  between two arbitrary vertices  $s, t \in V$ . Typically, we refer to  $s$  as the source and  $t$  as the target. A path in  $G$  is defined as a sequence of vertices  $\{v_1, v_2, \dots, v_k\}$  such that  $v_i v_{i+1} \in E$  for  $i = 1, 2, \dots, k - 1$ . Since we have constrained ourselves to a directed acyclic graph, all paths are simple. That is to say, all vertices in a path are distinct.

By modifying the depth-first search algorithm, we arrive at a solution.

```

FindPaths[g_?DirectedAcyclicGraphQ, s_, t_] :=
  Block[$RecursionLimit = 1024, paths = {}],
  fp[path_List, v_] :=
    If[v != t,
      Scan[fp[Append[path, #], #] &,
        VertexOutNeighbors[g, v]],
      (* else *)
      AppendTo[paths, path];
  ];

fp[{s}, s];

paths
]
```

Like the original DFS algorithm, we visit a vertex and then recursively visit all of its neighbors. However, instead of checking if a vertex has been marked “visited,” we compare the current vertex to the target. If they do not match, we continue to traverse the graph. Otherwise, the target has been reached and we store the path for later output.

For a given directed acyclic graph  $G$ , a source vertex  $s$ , and a target vertex  $t$ , `FindPaths` returns a list of all paths connecting  $s$  to  $t$ .

```

FindPaths[G["GridGraph"], 1, 16]

{{1, 2, 3, 4, 8, 12, 16}, {1, 2, 3, 7, 8, 12, 16},
 {1, 2, 3, 7, 11, 12, 16}, {1, 2, 3, 7, 11, 15, 16},
 {1, 2, 6, 7, 8, 12, 16}, {1, 2, 6, 7, 11, 12, 16},
 {1, 2, 6, 7, 11, 15, 16}, {1, 2, 6, 10, 11, 12, 16},
 {1, 2, 6, 10, 11, 15, 16}, {1, 2, 6, 10, 14, 15, 16},
 {1, 5, 6, 7, 8, 12, 16}, {1, 5, 6, 7, 11, 12, 16},
 {1, 5, 6, 7, 11, 15, 16}, {1, 5, 6, 10, 11, 12, 16},
 {1, 5, 6, 10, 11, 15, 16}, {1, 5, 6, 10, 14, 15, 16},
 {1, 5, 9, 10, 11, 12, 16}, {1, 5, 9, 10, 11, 15, 16},
 {1, 5, 9, 10, 14, 15, 16}, {1, 5, 9, 13, 14, 15, 16}}
```

In this particular instance, the function takes approximately 0.85 milliseconds to return the result.

`FindPaths` works for any pair of vertices.

```

FindPaths[G["GridGraph"], 3, 8]

{{3, 4, 8}, {3, 7, 8}}
```

If no path is found, the function returns an empty list.

```
FindPaths[G["GridGraph"], 8, 3]
{ }
```

## ■ Minimal Paths, Inclusion-Exclusion, and System Reliability

Up to this point, we have been working with graphs in an abstract, mathematical sense. We now make the transition from directed acyclic graph to reliability block diagram by associating vertices with components in a system and edges with relationships among them.

Consider a single component in an RBD. Let us imagine a “flow” moving from a source, through the component, to a target. The component is deemed to be functioning if the flow can pass through it unimpeded. However, if the component has failed, the flow is prevented from reaching the target.

The “flow” concept can be extended to an entire system. A system is considered to be functioning if there exists a set of functioning components that permits the flow to move from source to target. We define a path in an RBD as a set of functioning components that guarantees a functioning system. Since we have chosen to use a directed acyclic graph to represent a system’s RBD, all paths are minimal. That is to say, all components in a path are distinct.

Once the minimal paths of a system’s RBD have been obtained, the principle of inclusion-exclusion for probability can be employed to generate an exact analytical expression for reliability. Let  $\{A_1, A_2, \dots, A_n\}$  be the set of all minimal paths of a system. At least one minimal path must function in order for the system to function. We can write the reliability of the system  $R_S$  as the probability of the union of all minimal paths:

$$R_S = P\left(\bigcup_{i=1}^n A_i\right) = \sum_{k=1}^n (-1)^{k-1} \sum_{\substack{\emptyset \neq I \subseteq [n] \\ |I|=k}} P\left(\bigcap_{i \in I} A_i\right).$$

This is implemented in the function `SystemReliability`.

```
P[λ_List] :=
Apply[Times, Map[R# &, Apply[Union, λ]]]
```

```

SystemReliability[g_?DirectedAcyclicGraphQ, s_, t_] :=
Module[{minimalPaths, n, Δ, k},
  minimalPaths = FindPaths[g, s, t];

  n = Length[minimalPaths];
  Δ = Range[n];

  Sum[
    (-1)k-1 Total[Map[P[minimalPaths[[#]] &, Subsets[Δ, {k}]]],
    {k, n}
  ]
]

```

Given a system's RBD (represented by a directed acyclic graph  $G$ ), a source vertex  $s$ , and a target vertex  $t$ , `SystemReliability` returns an exact analytical expression for the reliability.

## □ Series Systems

Consider the RBD of a simple system with four components in a series configuration.

```

edges = {a → b, b → c, c → d};

RBD["Series"] = Graph[edges, ImagePadding → 10,
  VertexLabels → "Name", VertexShapeFunction → "Square",
  VertexSize → 0.15]

```



The reliability of the system is given in terms of the reliability of its four components.

```

SystemReliability[RBD["Series"], a, d] // TraditionalForm

```

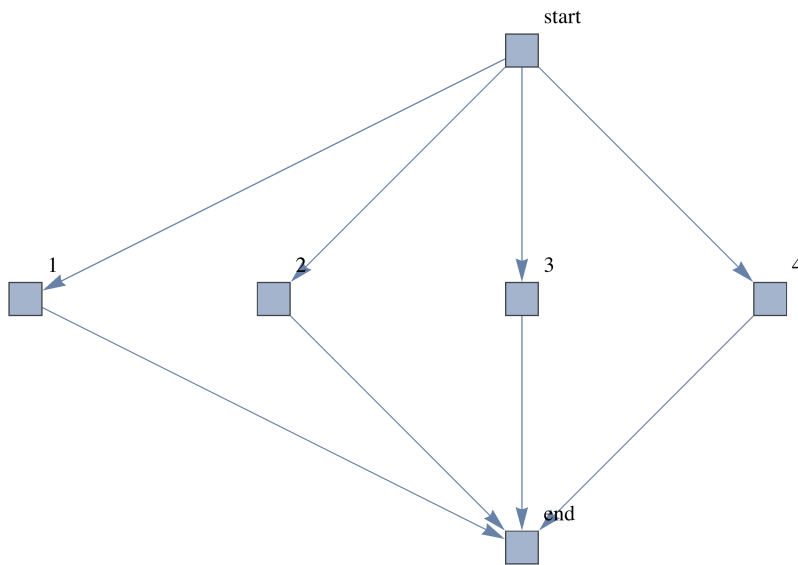
$$R_a R_b R_c R_d$$

## □ Parallel Systems

Consider the RBD of a simple system with four components in a parallel configuration.

```
edges = {start → 1, start → 2, start → 3, start → 4,
         1 → end, 2 → end, 3 → end, 4 → end};

RBD["Parallel"] = Graph[edges, ImagePadding → 10,
                       VertexLabels → "Name", VertexShapeFunction → "Square",
                       VertexSize → 0.15]
```



The “start” and “end” components are not part of the actual system. They are added to ensure the RBD meets the criteria for a directed acyclic graph.

```
DirectedAcyclicGraphQ[RBD["Parallel"]]
```

```
True
```

Furthermore, these nonphysical components are taken to have perfect reliability, that is,  $R = 1$ . Since they have no effect on the system’s reliability, they can be safely removed from the resulting analytical expression. To do so, we simply define a list of replacement rules and apply it to the result of `SystemReliability`.

```
rr = {R_start → 1, R_end → 1};
```



The reliability of the system is given in terms of the reliability of its four components.

```
(SystemReliability[RBD["Parallel"], start, end] /. rr) //
TraditionalForm
```

$$-R_2 R_1 + R_2 R_3 R_1 - R_3 R_1 + R_2 R_4 R_1 - R_2 R_3 R_4 R_1 + R_3 R_4 R_1 - R_4 R_1 + R_1 + R_2 - R_2 R_3 + R_3 - R_2 R_4 + R_2 R_3 R_4 - R_3 R_4 + R_4$$

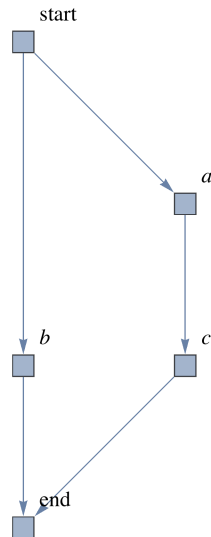
## □ Series-Parallel Systems

Next, we examine the RBDs of two simple systems with components in a series-parallel configuration.

### ■ System 1

```
edges = {start ↔ a, start ↔ b, a ↔ c, b ↔ end, c ↔ end};
```

```
RBD["SeriesParallel1"] =
Graph[edges, EdgeStyle → Arrowheads[{{0.083, 1}}],
ImagePadding → 10, ImageSize → 100, VertexLabels → "Name",
VertexShapeFunction → "Square", VertexSize → 0.15]
```



Component  $c$  is in series with component  $a$ , and both components are in parallel with component  $b$ .

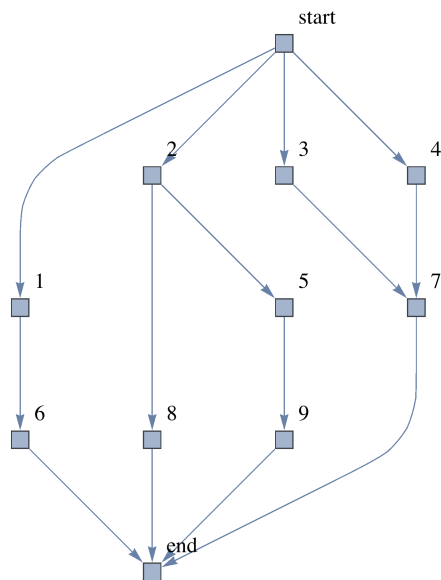
```
(SystemReliability[RBD["SeriesParallel1"], start, end] /.
  rr) // TraditionalForm
```

$$-R_a R_b R_c + R_a R_c + R_b$$

## ■ System 2

```
edges = {start → 1, start → 2, start → 3, start → 4,
  1 → 6, 2 → 5, 2 → 8, 3 → 7, 4 → 7, 5 → 9, 6 → end,
  7 → end, 8 → end, 9 → end};
```

```
RBD["SeriesParallel2"] =
  Graph[edges, EdgeStyle → Arrowheads[{{0.038, 1}}],
  ImagePadding → 10, ImageSize → 200, VertexLabels → "Name",
  VertexShapeFunction → "Square", VertexSize → 0.15]
```



As in previous examples, we use `SystemReliability` to obtain an exact analytical expression for the reliability.

```
(SystemReliability[RBD["SeriesParallel2"], start, end] /.
  rr) // Simplify // TraditionalForm
```

$$R_4 R_7 - R_2 R_4 R_8 R_7 - R_2 R_4 R_5 R_9 R_7 + R_2 R_4 R_5 R_8 R_9 R_7 - \\ R_3 (R_4 - 1) (R_2 (R_8 (R_5 R_9 - 1) - R_5 R_9) + 1) R_7 + R_2 R_8 + R_2 R_5 R_9 - \\ R_2 R_5 R_8 R_9 + R_1 R_6 (R_3 (R_4 - 1) R_7 - R_4 R_7 + 1) (R_2 (R_8 (R_5 R_9 - 1) - R_5 R_9) + 1)$$

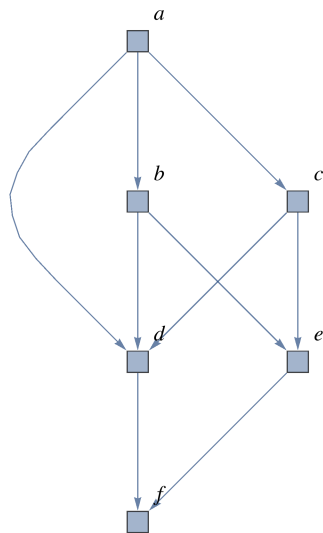
## □ Complex Systems

Finally, we examine the RBDs of two complex systems.

### ■ System 1

```
edges = {a → b, a → c, a → d, b → d, b → e, c → d, c → e,
  d → f, e → f};
```

```
RBD["Complex1"] =
  Graph[edges, EdgeStyle → Arrowheads[{{0.044, 1}}],
  ImagePadding → 10, ImageSize → 150, VertexLabels → "Name",
  VertexShapeFunction → "Square", VertexSize → 0.15]
```



The reliability of the system is given in terms of the reliability of its six components.

```
SystemReliability[RBD["Complex1"], a, f] // TraditionalForm
```

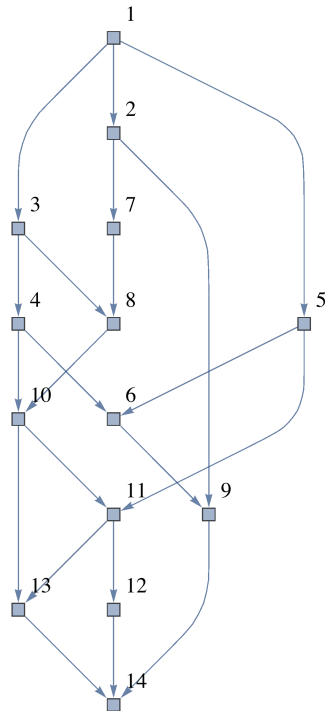
$$R_a R_b R_c R_d R_e R_f - R_a R_b R_c R_e R_f - R_a R_b R_d R_e R_f + R_a R_b R_e R_f - R_a R_c R_d R_e R_f + R_a R_c R_e R_f + R_a R_d R_f$$

The result is returned after approximately 0.59 milliseconds.

## ■ System 2

```
edges = {1 ↔ 2, 1 ↔ 3, 1 ↔ 5, 2 ↔ 7, 2 ↔ 9, 3 ↔ 4, 3 ↔ 8,
  4 ↔ 6, 4 ↔ 10, 5 ↔ 6, 5 ↔ 11, 6 ↔ 9, 7 ↔ 8, 8 ↔ 10,
  9 ↔ 14, 10 ↔ 11, 10 ↔ 13, 11 ↔ 12, 11 ↔ 13, 12 ↔ 14,
  13 ↔ 14};
```

```
RBD["Complex2"] =
Graph[edges, EdgeStyle → Arrowheads[{0.044, 1}],
ImagePadding → 10, ImageSize → 150, VertexLabels → "Name",
VertexShapeFunction → "Square", VertexSize → 0.15]
```



The reliability of the system is given in terms of the reliability of its fourteen components.

```
SystemReliability[RBD["Complex2"], 1, 14] // Simplify //
TraditionalForm
```

$$\begin{aligned}
 &R_1 (R_5 (R_6 R_9 (R_{11} (R_{12} (R_{13} - 1) - R_{13}) + 1) + R_{11} (R_{13} - R_{12} (R_{13} - 1))) + \\
 &\quad R_3 (R_8 R_{10} ((1 - R_5 R_6 R_9) R_{13} + R_{11} ((R_5 - 1) R_{12} (R_{13} - 1) + R_5 (R_6 R_9 - 1) R_{13})) + \\
 &\quad R_4 (- (R_8 - 1) R_{10} (R_{13} + R_{11} ((R_5 - 1) R_{12} (R_{13} - 1) - R_5 R_{13})) - \\
 &\quad R_6 R_9 (R_{10} (R_{13} - R_{11} R_{12} (R_{13} - 1)) + \\
 &\quad R_5 (R_{10} (R_{11} (R_{12} (R_{13} - 1) + (R_8 - 1) R_{13}) - R_8 R_{13}) + 1) - 1)) + \\
 &R_2 (R_9 (-R_5 R_6 + R_5 R_{11} R_{12} R_6 + R_5 R_{11} R_{13} R_6 - R_5 R_{11} R_{12} R_{13} R_6 - R_5 R_{11} R_{12} + \\
 &\quad R_5 R_7 R_8 R_{10} R_{11} R_{12} - R_7 R_8 R_{10} R_{11} R_{12} - R_7 R_8 R_{10} R_{13} - R_5 R_{11} R_{13} + \\
 &\quad R_5 R_7 R_8 R_{10} R_{11} R_{13} + R_5 R_{11} R_{12} R_{13} - R_5 R_7 R_8 R_{10} R_{11} R_{12} R_{13} + \\
 &\quad R_7 R_8 R_{10} R_{11} R_{12} R_{13} + R_3 (R_8 R_{10} ((R_5 R_6 + R_7 - 1) R_{13} + R_{11} \\
 &\quad ((R_5 - 1) (R_7 - 1) R_{12} (R_{13} - 1) - R_5 (R_6 + R_7 - 1) R_{13})) + \\
 &\quad R_4 ((R_8 - 1) R_{10} (R_{13} + R_{11} ((R_5 - 1) R_{12} (R_{13} - 1) - R_5 R_{13})) + \\
 &\quad R_6 (R_{10} (R_{13} - R_{11} R_{12} (R_{13} - 1)) + R_5 (R_{10} (R_{11} (R_{12} (R_{13} - \\
 &\quad 1) + (R_8 - 1) R_{13}) - R_8 R_{13}) + 1) - 1)) + 1) - \\
 &(R_3 - 1) R_7 R_8 R_{10} (R_{13} + R_{11} ((R_5 - 1) R_{12} (R_{13} - 1) - R_5 R_{13}))) R_{14}
 \end{aligned}$$



The result is returned after approximately 0.33 seconds.

## ■ Time-Dependent Reliability of a Complex System





We now turn our attention to the derivation of a time-dependent expression for the reliability of a complex system based on information contained within its reliability block diagram.

Let us imagine that we have a generic system composed of six subsystems and we know the reliability relationships among them. In addition, the underlying statistical distributions and parameters used to model the subsystems' reliabilities are known.

We begin by creating the system's RBD.

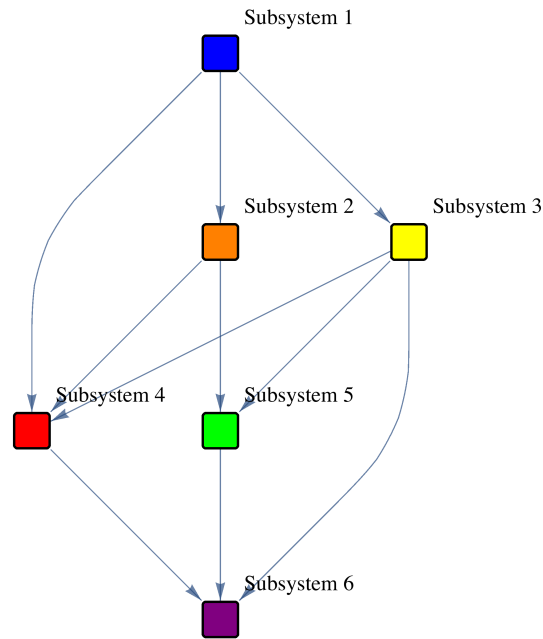
```
vertices = {
  Property[subsys[1],
    {"Distribution" → WeibullDistribution[2.21, 3.44],
     VertexLabels → "Subsystem 1", VertexShape →  }],
  Property[subsys[2],
    {"Distribution" → ExponentialDistribution[1.06],
     VertexLabels → "Subsystem 2", VertexShape →  }],
```

```

Property[subsys[3],
  {"Distribution" → ExponentialDistribution[1.38],
   VertexLabels → "Subsystem 3", VertexShape → }],
Property[subsys[4],
  {"Distribution" → WeibullDistribution[2.68, 2.25],
   VertexLabels → "Subsystem 4", VertexShape → }],
Property[subsys[5],
  {"Distribution" → ExponentialDistribution[0.94],
   VertexLabels → "Subsystem 5", VertexShape → }],
Property[subsys[6],
  {"Distribution" → WeibullDistribution[3.97, 4.36],
   VertexLabels → "Subsystem 6", VertexShape → }],
};
edges = {subsys[1] ↔ subsys[2], subsys[1] ↔ subsys[3],
  subsys[1] ↔ subsys[4], subsys[2] ↔ subsys[4],
  subsys[2] ↔ subsys[5], subsys[3] ↔ subsys[4],
  subsys[3] ↔ subsys[5], subsys[3] ↔ subsys[6],
  subsys[4] ↔ subsys[6], subsys[5] ↔ subsys[6]};

RBD["Generic"] = Graph[vertices, edges,
  EdgeStyle → Arrowheads[{{0.044, 0.999}}],
  ImagePadding → {{12, 66}, {12, 24}}, ImageSize → 250,
  VertexSize → Medium]

```



In defining the RBD, we have made use of the `Property` function to store information associated with each subsystem. For instance, the custom property "Distribution" is used to store a parametric statistical distribution. Labels, images, and other properties can also be specified.

Next, we use `SystemReliability` to generate an exact analytical expression for the reliability.

```
(Rs = SystemReliability[RBD["Generic"], subsys[1],  
subsys[6]]) // Simplify // TraditionalForm
```

$$R_{\text{subsys}(1)} \left( R_{\text{subsys}(2)} R_{\text{subsys}(5)} + R_{\text{subsys}(4)} (1 - R_{\text{subsys}(2)} R_{\text{subsys}(5)}) \right) + R_{\text{subsys}(3)} (R_{\text{subsys}(4)} - 1) (R_{\text{subsys}(2)} R_{\text{subsys}(5)} - 1) R_{\text{subsys}(6)}$$

Now, the reliability function of the  $i^{\text{th}}$  subsystem is given by

$$R_i(t) = 1 - F_i(t),$$

where  $F_i(t)$  is the corresponding cumulative distribution function (CDF). For each subsystem, we use `PropertyValue` to extract the symbolic distribution stored in the RBD, and then use the built-in function `CDF` to construct its reliability function.

```
reliabilityFunctions =  
Map[  
  (1 - CDF[PropertyValue[{RBD["Generic"], #},  
    "Distribution"], t]) &, VertexList[RBD["Generic"]]];
```

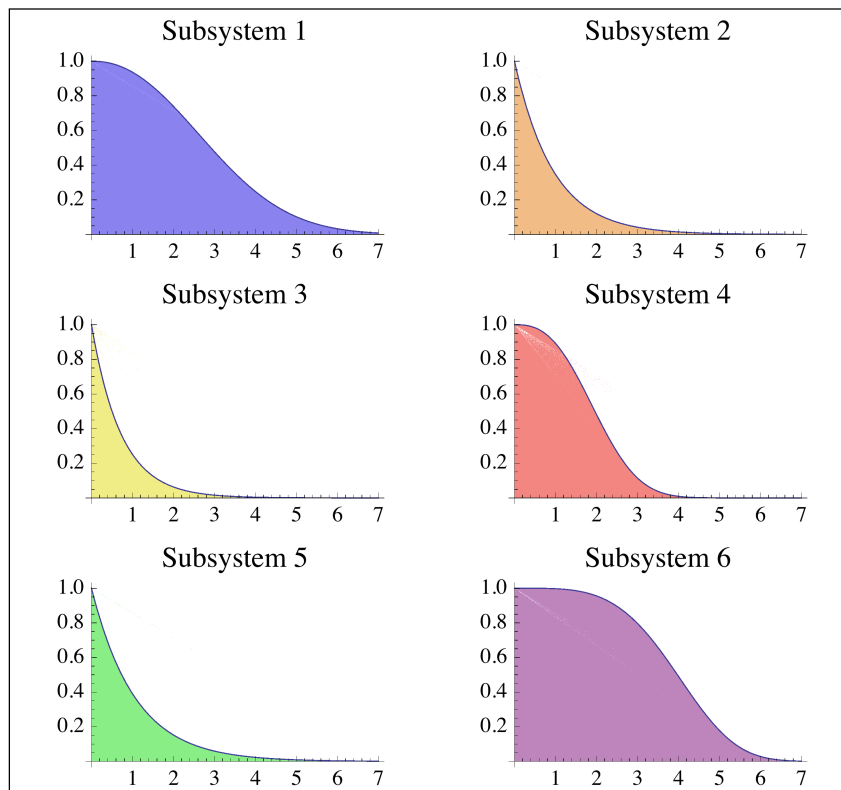
We extract additional information, for example, subsystem labels, from the RBD and combine it with the reliability functions to create plots for comparison.

```

colors =
  Flatten[
    Map[
      DominantColors[PropertyValue[{RBD["Generic"], #},
        VertexShape], 1] &, VertexList[RBD["Generic"]]]];
labels =
  Map[PropertyValue[{RBD["Generic"], #}, VertexLabels] &,
    VertexList[RBD["Generic"]]];

GraphicsGrid[
  Partition[
    Table[Plot[reliabilityFunctions[[i]], {t, 0, 7},
      Filling -> Axis,
      FillingStyle -> Directive[Opacity[0.5], colors[[i]],
      PlotLabel -> labels[[i]], PlotRange -> All],
      {i, Length[VertexList[RBD["Generic"]]]}], 2],
  Frame -> True]

```





In order to transform our static analytical expression into a time-dependent function, we first define a list of replacement rules.

```
symbols = Map[R# &, VertexList[RBD["Generic"]]];
rules = MapThread[Rule, {symbols, reliabilityFunctions}]
```

$$\left\{ \begin{array}{l} R_{\text{subsys}[1]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-0.0651937 t^{2.21}} \quad t > 0 \\ 0 \quad \text{True} \end{array} \right), \\ R_{\text{subsys}[2]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-1.06 t} \quad t \geq 0 \\ 0 \quad \text{True} \end{array} \right), \\ R_{\text{subsys}[3]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-1.38 t} \quad t \geq 0 \\ 0 \quad \text{True} \end{array} \right), \\ R_{\text{subsys}[4]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-0.113802 t^{2.68}} \quad t > 0 \\ 0 \quad \text{True} \end{array} \right), \\ R_{\text{subsys}[5]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-0.94 t} \quad t \geq 0 \\ 0 \quad \text{True} \end{array} \right), \\ R_{\text{subsys}[6]} \rightarrow 1 - \left( \begin{array}{l} 1 - e^{-0.00289227 t^{3.97}} \quad t > 0 \\ 0 \quad \text{True} \end{array} \right) \end{array} \right\}$$

Next, we apply the list of rules to the expression for system reliability.

```
RS[t_] := RS /. rules
```

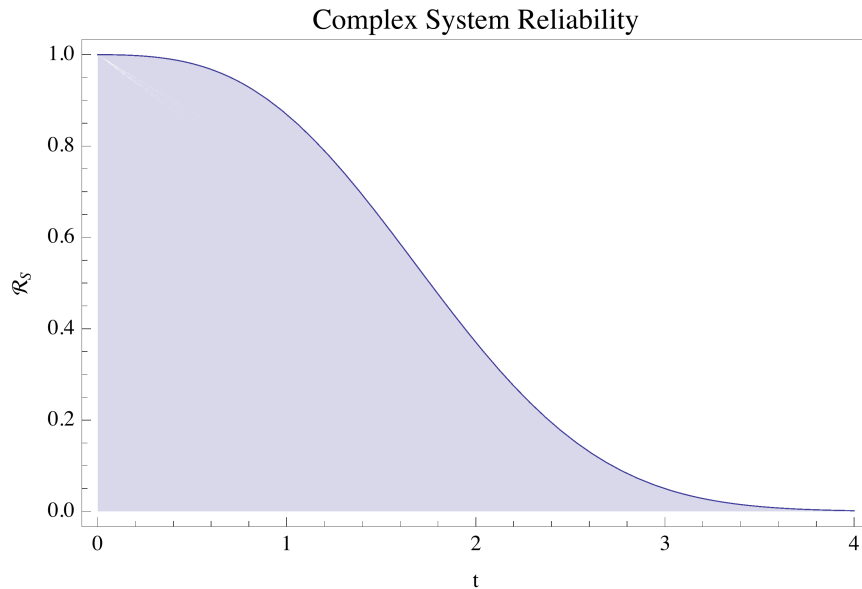
The result is a time-dependent reliability function for the complex system described by the RBD.

```
RS[t] // FullSimplify
```

$$\left[ \begin{array}{l} 1 \\ e^{-3.38 t - 0.0651937 t^{2.21} - 0.113802 t^{2.68} - 0.00289227 t^{3.97}} \\ \left( e^{3.38 t} + (-1 + e^{1.38 t} + e^{2 \cdot t}) (-1 + e^{0.113802 t^{2.68}}) \right) \end{array} \right] \quad \begin{array}{l} t \leq 0 \\ \text{True} \end{array}$$

Finally, we generate a plot of the system's reliability over time.

```
Plot[ $\mathcal{R}_S[t]$ , {t, 0, 4}, Filling -> Axis, Frame -> True,
  FrameLabel -> {" $\mathcal{R}_S$ ", None}, {"t", None}},
  FrameTicks -> {{True, False}, {True, False}},
  PlotLabel -> "Complex System Reliability", PlotRange -> All]
```



## ■ Discussion and Conclusion

We have demonstrated a method of generating an exact analytical expression for the reliability of a complex system using a directed acyclic graph to represent the system's reliability block diagram. In addition, we have shown how to convert an analytical expression for system reliability into a time-dependent function based on statistical information stored in an RBD. While our focus has been on the analysis of complex systems, we have also shown that the combination of path finding and the inclusion-exclusion principle is equally applicable to simple systems in series, parallel, or series-parallel configurations.

Knowing the static analytical expression or time-dependent solution of a system allows us to perform a more advanced reliability analysis. For instance, we can easily calculate the Birnbaum importance

$$I_i^B = \frac{\partial R_S}{\partial R_i}$$

of the  $i^{\text{th}}$  component using the result of `SystemReliability`. Similarly, we can derive the hazard function, or failure rate, from the system's time-dependent reliability function.

There are several ways in which the functionality demonstrated in this article can be improved and expanded:

- Increase the efficiency of `SystemReliability` by implementing improvements to the classical inclusion-exclusion principle [3].
- Add functions related to common tasks in reliability analysis, for example, reliability importance, failure rate, and so on.
- Add support for  $k$ -out-of- $n$  structures, that is, redundancy.
- Add the ability to export and import complete RBDs.
- Add a mechanism, for example, a graphical user interface (GUI), to facilitate the construction and modification of RBDs.

Finally, the code can be combined into a user-friendly package with full documentation.

## ■ References

- [1] W. Kuo and M. Zuo, *Optimal Reliability Modeling: Principles and Applications*, Hoboken, NJ: John Wiley & Sons, 2003.
  - [2] S. Skiena, *The Algorithm Design Manual*, 2nd ed., London, UK: Springer-Verlag, 2008.
  - [3] K. Dohmen, "Improved Inclusion-Exclusion Identities and Inequalities Based on a Particular Class of Abstract Tubes," *Electronic Journal of Probability*, **4**, 1999 pp. 1–12. doi:10.1214/EJP.v4-42.
- T. Silvestri, "Complex System Reliability," *The Mathematica Journal*, 2014. dx.doi.org/doi:10.3888/tmj.16-7.

## About the Author

Todd Silvestri received his undergraduate degrees in physics and mathematics from the University of Chicago in 2001. As a graduate student, he worked briefly at the Thomas Jefferson National Accelerator Facility (TJNAF) where he helped to construct and test a neutron detector used in experiments to measure the neutron electric form factor at high momentum transfer. From 2006 to 2011, he worked as a physicist at the US Army Armament Research, Development and Engineering Center (ARDEC). During his time there, he cofounded and served as principal investigator of a small laboratory focused on improving the reliability of military systems. He is currently working on several personal projects.

### **Todd Silvestri**

*New Jersey, United States*  
*todd.silvestri@optimum.net*