Realization of Dominant Reliability in Python

Preetham Reddy.A

October 2018

Abstract

In this document, we will see how Dominant Reliability is implemented in python and Dominant Reliability Polynomial of some graphs, all graphs are considered as finite, undirected, and simple.

1 Dominant Reliability

A Graph $G = (V, E)$ whose vertices V fail randomly and independently with equal probability p ,whereas the edges are perfectly reliable and $q = 1 - p$ then Dominant Reliability[1] is:

$$DRel(G, p) = \sum_{\substack{J \subseteq V \\ N_G[J]=V}} p^{|J|} q^{n-|J|} \tag{1}$$

where n is the number of vertices in Graph G,$N_G[J]$ is the closed neighbourhood of J in the graph G.

2 Realization in Python

Package requirements to run the Dominant Reliability :

1.Python
2.Networkx
3.Sympy

In main function a finite, undirected and simple graph is created by adding vertices and edges to the empty graph G. using the list of vertices $V(G)$ we generate a combination of vertices. we use combination of vertices, graph and list of vertices variable to generate dominant reliability polynomial.

```
def main():
    # start time
    time_start = time.time()


    # creation of empty graph
    G = nx.Graph()
    # adding vertices to graph G
    G.add_nodes_from(['a','b','c','d','e','f','g'])
    # adding edges to graph G
    G.add_edges_from([('a','b'),('c','d'),('d','e'),('b','f'),('c','g')])


    #  list for vertices V(G) of graph G
    G_vertices=G.nodes()
    # print the list of vertices
    print("Graph G vertices V(G):\n",G_vertices)
    #  list for edges E(G) of graph G
    G_edges=G.edges()
```

```
    # print the list of edges
    print("Graph G edges E(G):\n",G_edges)
    # pass the vertices list V(G) to create combinations of vertices
    combi_vertices=combi(G_vertices)
     # passing the combination of vertices ,Graph G, vertices list V(G) to calculate the
                                                    dominant reliability polynomial
    ply1=polynomial(combi_vertices,G,G_vertices)
    # print the Dominant Reliability Polynomial
    print("Dominant Reliability Polynomial of graph G:\n",sympy.simplify(ply1))
    # printing computational time
    print("computational time:%s seconds"%(time.time() - time_start))
    return 0
```

Function to create a combination of vertices using the list of vertices V(G)

```
def combi(G_vertices):
    '''
    function to create a combination of vertices

    argument: list of vertices V(G)

    return: combination of vertices
    '''
    # empty list of combination of vertices
    combi_vertices = []
    # for loop to generate the combination of vertices iterating
    #the list of vertices V(G)
    for i in range(1, len(G_vertices) + 1):
        p = list(itertools.combinations(G_vertices, i))
        combi_vertices.extend(p)

    return combi_vertices
```

Function to generate dominant reliability polynomial using combination of vertices, graph G and vertices list of graph V(G)

```
def polynomial(combi_vertices, G, G_vertices):
    '''
    Function to generate dominant reliability polynomial

    argument: combination of vertices ,graph G and vertices list of graph V(G)

    return: polynomial equation
    '''
    # polynomial intialization
    ply1 = 0
    l = []
    # printing the number of combinations of vertices
    print('number of combinations of vertices:', len(combi_vertices))
    # loop for iterating each combination of vertices
    for k in range(0, len(combi_vertices)):

        each_combi = list(combi_vertices[k])

        l2 = []
        #  for loop to check each combination of vertex is a closed neighbhourhood of J
                                                        in the graph G
        for i in range(0, len(each_combi)):
            # get neighbhourhood of J for each combination of vertex in the graph G
            c = nx.all_neighbors(G, combi_vertices[k][i])

            l1 = list(c)

            # create a list of neighbhourhood of each combination
            #of vertex in the graph G
            c33 = [combi_vertices[k][i]]

            # closed neighbhourhood of each combination of vertex in the graph G
            l2.extend(l1)
            l2.extend(c33)
```

```
        # if closed neighbhourhood of J in the graph G is equal to vertices of
        #graph V(G) then generate polynomial equation
        if set(l2) == set(G_vertices):

            l.extend([combi_vertices[k]])

            len_J = len(combi_vertices[k])

            # polynomial equation if the closed neighbhourhood of the J is equal to
                                                        vertices of graph V(G)
            ply = (p ** len_J) * ((1 - p) ** (len(G_vertices) - len_J))
            # summation of all   polynomial equation
            ply1 = ply1 + ply


    return ply1
```

3 Results

Example 1: For a tree graph mentioned in Remark:4.3[1] the Dominant Reliability Polynomial is shown in the below figure:1

```
(graph) preetham@preetham-Inspiron-N4050:~$ python3 /home/preetham/graph/domrel.py
Graph G vertices v(G):
 ['f', 'c', 'e', 'd', 'g', 'a', 'b']
Graph G edges E(G):
 [('f', 'b'), ('c', 'd'), ('c', 'g'), ('e', 'd'), ('a', 'b')]
number of combinations of vertices: 127
Dominant Reliability Polynomial of graph G:
 p**3*(-p**4 + 5*p**3 - 7*p**2 + 4)
computational time:0.26146841049194336 seconds
(graph) preetham@preetham-Inspiron-N4050:~$
```

Figure 1: Dominant Reliability Polynomial for a tree graph

Example 2: For a cycle graph with 20 vertices Dominant Reliability Polynomial is shown in the below figure:2

```
(graph) preetham@preetham-Inspiron-N4050:~$ python3 /home/preetham/graph/domrel.py
Graph G vertices v(G):
 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Graph G edges E(G):
 [(0, 1), (0, 19), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10), (10, 11), (11, 12), (12, 13), (13, 14), (14, 15), (
15, 16), (16, 17), (17, 18), (18, 19)]
number of combinations of vertices: 1048575
Dominant Reliability Polynomial of graph G:
 p**7*(3*p**13 - 40*p**12 + 230*p**11 - 680*p**10 + 790*p**9 + 1392*p**8 - 7190*p**7 + 13280*p**6 - 13305*p**5 + 6640*p**4 - 244*p**3 - 1300*p*
*2 + 405*p + 20)
computational time:121.7087652683258 seconds
(graph) preetham@preetham-Inspiron-N4050:~$
```

Figure 2: Dominant Reliability Polynomial for a cycle graph with 20 vertices

4 Computational complexity

As the number of vertices increases in the graph the computational time increase exponentially from example 1 and 2. Calculation of Dominant Reliability Polynomial is NP-hard.

References

[1] Klaus Dohmen , Peter Tittmann *Dominant Reliability*

[2] Python Documentation `https://docs.python.org/3/`

[3] Networkx Documentation `https://networkx.github.io/documentation/stable/index.html`

[4] Sympy Documentation `https://docs.sympy.org/latest/index.html`