Energy-efficiency vs. resilience Code for optimization procedures and simulations

P. Chołda and P. Jaglarz

To reproduce the steps of the algorithm presented in the paper, save attached files to a common directory and provide required tools.

CONTENTS

The authors are with the AGH University of Science and Technology, Department of Telecommunications, Krakow, Poland. E-mail: {piotr.cholda}@agh.edu.pl.

SOFTWARE REQUIREMENTS

- CPLEX (including OPL Interpreter): http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
- MATLAB (including MATLAB Compiler): http://www.mathworks.com/products/matlab/

MAIN.M

This MATLAB script represents the main loop of the optimization algorithm

```
%% GENERAL INFORMATION
% Optimization module using the Dijkstra's shortest path algorithm
% to compute the (near) optimal routing scheme.
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

%% INPUT

% Clear the environment

        close all;
    clear all;
        clc;

 for protection = [1]
% Read the complete environment snapshot from a .mat file


        clearvars -except protection;
    load( 'input' );

%% MODIFIED YAGED'S ALGORITHM
% (Pioro --> Algorithm 5.12 with the proposed modification 5.6.2)

    k = 0;                                          % Iteration counter
        F = Inf;                                        % Global cost
        F_new = Inf;                                    % Global cost -> new value
        Pd = zeros( length( d ), length( A ) );     % Shortest paths for the corresponding traffic demands
    backup_paths = zeros( length( d ), length( A ) );
    backup_loc_paths = zeros( length( A ), length( A ) );

    while ( 1 )

                % For each traffic demand, find the shortest path and increase the load
                % of the corresponding links
        y = zeros( length( A ), 1 );                        % Link loads
        y3 = cell( length( A ), 1 );
        y2 = zeros( length( A ), 1 );
        for i = 1 : length( A )
                y3{ i } = 0;
        end

        for i = 1 : length( d )
            if protection == 0 || protection == 2 || protection == 4
                [ cost, path ] = dijkstra( A, links, y, d( i, 1 ), d( i, 2 ), d( i, 3 ) );
            elseif protection == 1 || protection == 3
                [ cost, path, backup ] = suurballe( A, links, y, d( i, 1 ), d( i, 2 ), d( i, 3 ) );
            end

            if cost == Inf
                                warning( 'There is no path between nodes %d and %d.\n', d( i, 1 ), d( i, 2 ) );
            else
                Pd( i, : ) = zeros( 1, length( A ) );                   % Remove the previous shortest path entry
                for m = 1 : length( path )
                    y( path( m ) ) = y( path( m ) ) + d( i, 3 );        % Increase the link load
                    Pd( i, m ) = path( m );                                         % Update the shortest path for the demand (link predecessor sequence)
                end

                if protection == 1
```

```
                    backup_paths( i, : ) = zeros( 1, length( A ) );
                    for m = 1 : length( backup )
                        y( backup( m ) ) = y( backup( m ) ) + d( i, 3 );    % optional - backup path cost
                        y2( backup( m ) ) = y2( backup( m ) ) + d( i, 3 );
                        backup_paths( i, m ) = backup( m );
                    end
                end

                if protection == 3
                    backup_paths( i, : ) = zeros( 1, length( A ) );
                    for m = 1 : length( backup )
                        y3{ backup( m ) } = [ y3{ backup( m ) } d( i, 3 ) ];    % optional - backup path cost
                        backup_paths( i, m ) = backup( m );
                    end
                end

            end
        end

        if protection == 2
            y2 = zeros( length( A ), 1 );

            for i = 1 : length( A )
                A( i, 6 ) = 0;
                [ cost, backup_loc ] = dijkstra( A, links, y, A( i, 1 ), A( i, 2 ), y( i ) );
                A( i, 6 ) = 1;

                backup_loc_paths( i, : ) = zeros( 1, length( A ) );
                for m = 1 : length( backup_loc )
                    y2( backup_loc( m ) ) = y2( backup_loc( m ) ) + y( i );    % optional - backup path cost
                    backup_loc_paths( i, m ) = backup_loc( m );
                end
            end

            y = y + y2;
        end

        if protection == 3
            % dedicated global protection
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            y3 = cell( length( A ), 1 );

            for i = 1 : length( A )
                [ r, ~ ] = find( Pd == i );
                XXX = zeros( length( d ), length( A ) );
                XXX( r, : ) = backup_paths( r, : );

                for j = 1 : length( A )
                    [ r, ~ ] = find( XXX == j );

                    y3{ j } = [ y3{ j } sum( d( r, 3 ) ) ];
                end
            end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            y2 = zeros( length( A ), 1 );

            for i = 1 : length( y2 )
                y2( i ) = max( y3{ i } );
            end

            y = y + y2;
        end

        if protection == 4
            y2 = zeros( length( A ), 1 );
```

```
127
128             for i = 1 : length( A )
129                 A( i, 6 ) = 0;
130                 [ cost, backup_loc ] = dijkstra( A, links, y, A( i, 1 ), A( i, 2 ), y( i ) );
131                 A( i, 6 ) = 1;
132
133                 backup_loc_paths( i, : ) = zeros( 1, length( A ) );
134                 for m = 1 : length( backup_loc )
135                     y3{ backup_loc( m ) } = [ y3{ backup_loc( m ) } y( i ) ];    % optional - backup path cost
136                     backup_loc_paths( i, m ) = backup_loc( m );
137                 end
138             end
139
140             for i = 1 : length( y2 )
141                 y2( i ) = max( y3{ i } );
142             end
143
144             y = y + y2;
145         end
146
147                 % Compute the global cost of the network
148
149                 F = F_new;
150                 F_new = global_cost( Cx, Cy, y );
151         disp( F_new );
152         y_sum = sum( y > 0 );
153         disp( y_sum );
154
155         % Determine whether another iteration is needed
156
157                 if F_new == F || ( protection == 2 && k == 1 ) || ( protection == 3 && k == 2 ) || ( protection == 4 && k == 0 )
158
159             if protection == 0
160                 uniq = unique( Pd );
161                 sleep = 1 : length( A );
162                 sleep = setdiff( sleep, uniq );
163             end
164
165             % STOP
166                         break;
167                 end
168
169                 % For each link, compute the derivatives (link unit costs)
170
171                 for i = 1 : length( A )
172                         A( i, 3 ) = get_avg_derivative_of_cost( Cx( i, : ), Cy( i, : ), y( i ), k );
173                 end
174
175                 k = k + 1;
176     end
177
178         fprintf( 'k = %d\n\n', k );
179
180
181 %% SAVE THE SELECTED VARIABLES INTO A .mat FILE
182
183     if protection == 0
184         save( 'output', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'y', 'Pd', 'sleep' );
185     elseif protection == 1
186         save( 'output_prot_glob', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'y', 'Pd', 'backup_paths' );
187     elseif protection == 2
188         save( 'output_prot_loc', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'y', 'Pd', 'backup_loc_paths' );
189     elseif protection == 3
190         save( 'output_ws_prot_glob', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'y', 'y2', 'Pd',
```

```
            ↪'backup_paths' );
    elseif protection == 4
        save( 'output_ws_prot_loc', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'y', 'y2', 'Pd', '↵
            ↪backup_loc_paths' );
    end

end
```

GENINPUTDATA.M

This MATLAB script represents function to retrieve the data from the .XML file

```
%% GENERAL INFORMATION
% Input data generator for the optimization module using the Dijkstra's
% shortest path algorithm to compute the (near) optimal routing scheme.
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

%% INPUT DATA FOR THE OPTIMIZATION MODULE

% Clear the environment

        close all;
        clear all;
        clc;

% Network topology (undirected graph)

    % XML file containing the topology data (compatible with SNDlib format)

    filename = 'polska.xml';

    tree = xmlread( filename );

    % Read names of all the network nodes from the XML file
    % Array of structures, each containing the following fields:
    %   xml_network_nodes(n).name - descriptive node name
    %   xml_network_nodes(n).x - x coordinate
    %   xml_network_nodes(n).y - y coordinate

    xml_network_nodes = xml_get_network_nodes( tree );
    nodes = cell( [ xml_network_nodes.name ] );

    % Read names of all the network links from the XML file
    % Array of structures, each containing the following fields:
    %   xml_network_links(n).name - descriptive link name
    %   xml_network_links(n).src - source node descriptive name
    %   xml_network_links(n).dst - destination node descriptive name
    %   xml_network_links(n).throughput - link throughput
    %   xml_network_links(n).weight - link weight (for example: cost, distance)

    xml_network_links = xml_get_network_links( tree );

    % Read names of all the network demands from the XML file
    % Array of structures, each containing the following fields:
    %   xml_network_demands(n).name - descriptive demand name
    %   xml_network_demands(n).src - source node descriptive name
    %   xml_network_demands(n).dst - destination node descriptive name
    %   xml_network_demands(n).value - demand traffic value

    xml_network_demands = xml_get_network_demands( tree );

    % Translate the data into the appropriate structures
    % Links (arcs) - array of structures, each containing the following fields:
    %   A(n,1) - src - source node index
    %   A(n,2) - dst - destination node index
    %   A(n,3) - weight - link weight (for example: cost, distance)
    %   A(n,4) - throughput - link throughput
    %   A(n,5) - distance - link distance
    %   A(n,6) - status - the current status of the link:
    %            0: down
    %            1: up
    %            2: sleeping (? - may be useful when substituting a node..)

    A = zeros( length( xml_network_links ), 6 );

    for i = 1 : length( xml_network_links )
```

```
67          % XML file from SNDlib contains only links in one direction!
68
69          for j = 1 : length( xml_network_nodes )
70              if strcmp( xml_network_nodes( j ).name, xml_network_links( i ).src ) == 1
71                  A( i, 1 ) = j;
72                  dx = str2double( xml_network_nodes( j ).x );
73                  dy = str2double( xml_network_nodes( j ).y );
74
75                  break;
76              end
77          end
78
79          for j = 1 : length( xml_network_nodes )
80              if strcmp( xml_network_nodes( j ).name, xml_network_links( i ).dst ) == 1
81                  A( i, 2 ) = j;
82                  dx = str2double( xml_network_nodes( j ).x ) - dx;
83                  dy = str2double( xml_network_nodes( j ).y ) - dy;
84
85                  break;
86              end
87          end
88
89          % A( i, 3 ) = str2double( xml_network_links( i ).weight );
90          A( i, 3 ) = round( sqrt( dx^2 + dy^2 ) * 111 );
91          % A( i, 4 ) = str2double( xml_network_links( i ).throughput );
92          A( i, 4 ) = Inf;
93          A( i, 5 ) = round( sqrt( dx^2 + dy^2 ) * 111 );
94          A( i, 6 ) = 1;
95
96      end
97
98      % Traffic demands - array of structures, each containing the following fields:
99      %   d(n,1) - src - source node index
100     %   d(n,2) - dst - destination node index
101     %   d(n,3) - value - size of the traffic demand
102
103     %% step: 1 - unidirectional (single) demand / 2 - directional demands
104     step = 1;
105     counter = 1;
106     d = zeros( step * length( xml_network_demands ), 3 );
107     for i = 1 : length( xml_network_demands )
108
109         for j = 1 : length( xml_network_nodes )
110             if strcmp( xml_network_nodes( j ).name, xml_network_demands( i ).src ) == 1
111                 d( counter, 1 ) = j;
112
113                 break;
114             end
115         end
116
117         for j = 1 : length( xml_network_nodes )
118             if strcmp( xml_network_nodes( j ).name, xml_network_demands( i ).dst ) == 1
119                 d( counter, 2 )= j;
120
121                 break;
122             end
123         end
124
125         d( counter, 3 ) = str2double( xml_network_demands( i ).value );
126
127         if step == 2
128             % The same for the opposite direction
129             d( counter + 1, 2 ) = d( counter, 1 );
130             d( counter + 1, 1 ) = d( counter, 2 );
131             d( counter + 1, 3 ) = d( counter, 3 );
132         end
133
134         counter = counter + step;
135     end
```

```
136     d = unique( d, 'rows' );
137
138     [ ~, ind ] = sort( d( :, 3 ), 'descend' );
139     d = d( ind, : );
140
141 %     % Optional manual demands
142 %     d = [ 2 7 200; 7 2 200; 5 7 300; 7 5 300; 2 6 250; 6 2 250; 4 8 80; 8 4 80; 1 3 350;
143 %         3 1 350; 6 7 150; 7 6 150; 4 3 90; 3 4 90; 5 1 100; 1 5 100; 12 3 200; 3 12 200;
144 %         1 8 150; 8 1 150; 10 9 250; 9 10 250; 8 6 200; 6 8 200; 6 10 300; 10 6 300 ];
145
146
147     %% Remove repeated demands if SNDlib file contains demands in two directions
148
149     remove_repeated_demands = true;
150     if remove_repeated_demands == 1
151
152         for i = 1 : length( d )
153             [ ~, Locb] = ismember( [ d( i, 2) d( i, 1 ) d( i, 3 ) ], d, 'rows' );
154             if Locb > 0
155                 d( Locb, : ) = 0;
156             end
157         end
158
159         d = setdiff( d, [ 0 0 0 ], 'rows' );
160
161         % optional
162         % d( :, 3 ) = 2 * d( :, 3 );
163     end
164
165
166     % Convert A matrix to links adjacency matrix
167     links = zeros( length( nodes ) );
168     for i = 1 : length( A )
169         links( A( i, 1 ), A( i, 2 ) ) = i;
170         links( A( i, 2 ), A( i, 1 ) ) = i;
171     end
172
173
174     % Link cost function lookup table (one row per link)
175     %   Cx - array of arguments (link load)
176     %   Cy - array of values (link cost for the corresponding load)
177     % Assumption: the first column of Cx always contains zeros.
178
179     step = 10;
180     Max = 150000;
181     Cx = zeros( length( A ), 1 + Max / step );
182
183     for j = 1 : length( A )
184        Cx( j, : ) = 0 : step : Max;
185     end
186
187         Cy = sqrt( Cx );
188
189
190 %% SAVE ALL THE VARIABLES INTO A .mat FILE
191
192     save( 'input', 'A', 'd', 'nodes', 'links', 'Cx', 'Cy', 'filename' );
```

XMLGETNETWORKDEMANDS.M

This MATLAB script represents function to to retrieve the data from the .XML file

```
%% GENERAL INFORMATION
% Function that extracts network demands from an XML data file provided
% by the SNDlib project (http://sndlib.zib.de).
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

%%

function result = xml_get_network_demands( root )

result = struct( ...
    'name', {}, ...
        'src', {}, ...
        'dst', {}, ...
    'value', {} ...
);

if root.hasChildNodes
        % Enter the 'network' level

        child_nodes = root.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'network' ) == 1
                        xml_level_network = child_nodes.item( z );

                        break;
                end
        end

        % Enter the 'demands' level

        child_nodes = xml_level_network.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
        if strcmp( child_nodes.item( z ).getNodeName(), 'demands' ) == 1
                        xml_level_demands = child_nodes.item( z );

                        break;
        end
        end


        % Get names of all the network nodes

        child_nodes = xml_level_demands.getChildNodes();
        struct_counter = 1;

        for i = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( i ).getNodeName(), 'demand' ) == 0
                        continue;
                end

                result( struct_counter ).name = child_nodes.item( i ).getAttribute( 'id' );

        % Demand parameters

                sub_child_nodes = child_nodes.item( i ).getChildNodes();

        for j = 0 : ( sub_child_nodes.getLength() - 1 )
                        if strcmp( sub_child_nodes.item( j ).getNodeName(), '#text' ) == 1
                                continue;
                        end

                        sub_element_name = char( sub_child_nodes.item( j ).getNodeName() );
```

```
67
68                          switch sub_element_name
69                                  case 'source'
70                                          result( struct_counter ).src = sub_child_nodes.item( j
                                                ↳ ).getFirstChild().getNodeValue();
71                                  case 'target'
72                                          result( struct_counter ).dst = sub_child_nodes.item( j
                                                ↳ ).getFirstChild().getNodeValue();
73                  case 'demandValue'
74                                          result( struct_counter ).value = sub_child_nodes.item(
                                                ↳ j ).getFirstChild().getNodeValue();
75                          end
76          end
77
78                  struct_counter = struct_counter + 1;
79          end
80  end
```

XMLGETNETWORKLINKS.M

This MATLAB script represents function to to retrieve the data from the .XML file

```
%% GENERAL INFORMATION
% Function that extracts the selected network link parameters from an XML
% data file provided by the SNDlib project (http://sndlib.zib.de).
%
% Author: Andrzej Kamisinski (andrzej.kamisinski@ktd.krakow.pl)
% Date: 12.2013

%%

function result = xml_get_network_links( root )

result = struct( ...
        'name', {}, ...
        'src', {}, ...
        'dst', {}, ...
        'throughput', {}, ...
        'weight', {} ...
);

if root.hasChildNodes
        % Enter the 'network' level

        child_nodes = root.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'network' ) == 1
                        xml_level_network = child_nodes.item( z );

                        break;
                end
        end

        % Enter the 'networkStructure' level

        child_nodes = xml_level_network.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'networkStructure' ) == 1
                        xml_level_network_structure = child_nodes.item( z );

                        break;
                end
        end

        % Enter the 'links' level

        child_nodes = xml_level_network_structure.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'links' ) == 1
                        xml_level_links = child_nodes.item( z );

                        break;
                end
        end

        % Get the selected parameters of all the network links

        child_nodes = xml_level_links.getChildNodes();
        struct_counter = 1;

        for i = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( i ).getNodeName(), 'link' ) == 0
                        continue;
                end

```

```
67                  result( struct_counter ).name = child_nodes.item( i ).getAttribute( 'id' );
68
69                  % Link parameters
70
71                  sub_child_nodes = child_nodes.item( i ).getChildNodes();
72
73                  for j = 0 : ( sub_child_nodes.getLength() - 1 )
74                          if strcmp( sub_child_nodes.item( j ).getNodeName(), '#text' ) == 1
75                                  continue;
76                          end
77
78                          sub_element_name = char( sub_child_nodes.item( j ).getNodeName() );
79
80                          switch sub_element_name
81                                  case 'source'
82                                          result( struct_counter ).src = sub_child_nodes.item( j↵
                                                ↳ ).getFirstChild().getNodeValue();
83                                  case 'target'
84                                          result( struct_counter ).dst = sub_child_nodes.item( j↵
                                                ↳ ).getFirstChild().getNodeValue();
85                                  case 'additionalModules'
86                                          % CONSIDERATION: Handle multiple capacity modules (?)
87
88                                          sub_sub_child_nodes = sub_child_nodes.item( j ).↵
                                                ↳getChildNodes();                       % "↵
                                                ↳addModule" blocks
89                                          sub_sub_sub_child_nodes = sub_sub_child_nodes.item( 1 ↵
                                                ↳).getChildNodes();        % "capacity", "cost" ↵
                                                ↳elements in the first addModule block
90
91                                          for z = 0 : ( sub_sub_sub_child_nodes.getLength() - 1 ↵
                                                ↳)
92                                                  if strcmp( sub_sub_sub_child_nodes.item( z ).↵
                                                        ↳getNodeName(), 'capacity' ) == 1
93                                                          result( struct_counter ).throughput = ↵
                                                                ↳sub_sub_sub_child_nodes.item( z ↵
                                                                ↳).getFirstChild().getNodeValue()↵
                                                                ↳;
94                                                  elseif strcmp( sub_sub_sub_child_nodes.item( z↵
                                                        ↳ ).getNodeName(), 'cost' ) == 1
95                                                          result( struct_counter ).weight = ↵
                                                                ↳sub_sub_sub_child_nodes.item( z ↵
                                                                ↳).getFirstChild().getNodeValue()↵
                                                                ↳;
96                                                  end
97                                          end
98                                  otherwise
99                                          %disp( 'Unhandled link parameter - ignore.' )
100                         end
101                 end
102
103                 struct_counter = struct_counter + 1;
104         end
105 end
```

XMLGETNETWORKNODES.M

This MATLAB script represents function to to retrieve the data from the .XML file

```
%% GENERAL INFORMATION
% Function that extracts network node names from an XML data file provided
% by the SNDlib project (http://sndlib.zib.de).
%
% Author: Andrzej Kamisinski (andrzej.kamisinski@ktd.krakow.pl)
% Date: 12.2013

%%

function result = xml_get_network_nodes( root )

result = struct( ...
        'name', {}, ...
    'x', {}, ...
    'y', {} ...
);

if root.hasChildNodes
        % Enter the 'network' level

        child_nodes = root.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'network' ) == 1
                        xml_level_network = child_nodes.item( z );

                        break;
                end
        end

        % Enter the 'networkStructure' level

        child_nodes = xml_level_network.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'networkStructure' ) == 1
                        xml_level_network_structure = child_nodes.item( z );

                        break;
                end
        end

        % Enter the 'nodes' level

        child_nodes = xml_level_network_structure.getChildNodes();

        for z = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( z ).getNodeName(), 'nodes' ) == 1
                        xml_level_nodes = child_nodes.item( z );

                        break;
                end
        end

        % Get names of all the network nodes

        child_nodes = xml_level_nodes.getChildNodes();
        struct_counter = 1;

        for i = 0 : ( child_nodes.getLength() - 1 )
                if strcmp( child_nodes.item( i ).getNodeName(), 'node' ) == 0
                        continue;
                end

                result( struct_counter ).name = child_nodes.item( i ).getAttribute( 'id' );

```

```
67          % Node parameters
68
69                  sub_child_nodes = child_nodes.item( i ).getChildNodes();
70          sub_child_nodes = sub_child_nodes.item( 1 ).getChildNodes();
71
72          for j = 0 : ( sub_child_nodes.getLength() - 1 )
73                      if strcmp( sub_child_nodes.item( j ).getNodeName(), '#text' ) == 1
74                              continue;
75                      end
76
77                      sub_element_name = char( sub_child_nodes.item( j ).getNodeName() );
78
79                      switch sub_element_name
80                              case 'x'
81                                      result( struct_counter ).x = sub_child_nodes.item( j )↵
                                            ↳.getFirstChild().getNodeValue();
82                              case 'y'
83                                      result( struct_counter ).y = sub_child_nodes.item( j )↵
                                            ↳.getFirstChild().getNodeValue();
84                      end
85          end
86
87                  struct_counter = struct_counter + 1;
88          end
89  end
```

DIJKSTRA.M

This MATLAB script represents function to find the shortest paths

```
%% GENERAL INFORMATION
% Dijkstra's shortest path algorithm
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

function [ cost, path ] = dijkstra( A, links, load, src, dst, traffic, sleep )

n = length( links );        % Number of nodes in the network
visited( 1 : n ) = 0;    % Visited nodes
dist( 1 : n ) = Inf;     % It stores the shortest distance between the source node and any other node
prev( 1 : n ) = 0;       % Previous node, informs about the best previous node known to reach each network node

dist( src ) = 0;


for j = 1 : n - 1

    candidate = inf( 1, n );
    for i = 1 : n
        if visited( i ) == 0
            candidate( i ) = dist( i );
        end
    end

    [ ~, s ] = min( candidate );
    visited( s ) = 1;
    for d = 1 : n
        if links( s, d ) == 0
            continue;
        end

        if ( dist( s ) + A( links( s, d ), 3 ) ) < dist( d ) && A( links( s, d ), 6 ) == 1
            if load( links( s, d ) ) + traffic <= A( links( s, d ), 4 )
                dist( d ) = dist( s ) + A( links( s, d ), 3 );
                prev( d ) = s;
            else
                % warning( 'Link %d: Link capacity is full ...\n', links( s, d ) );
            end
        end
    end

end

cost = dist( dst );
path = 0;




% powtarzamy ze wznowionymi linkami
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if cost == Inf

visited( 1 : n ) = 0;    % Visited nodes
dist( 1 : n ) = Inf;     % It stores the shortest distance between the source node and any other node
prev( 1 : n ) = 0;       % Previous node, informs about the best previous node known to reach each network node

dist( src ) = 0;


for j = 1 : n - 1
```

```
63
64      candidate = inf( 1, n );
65      for i = 1 : n
66          if visited( i ) == 0
67              candidate( i ) = dist( i );
68          end
69      end
70
71      [ ~, s ] = min( candidate );
72      visited( s ) = 1;
73      for d = 1 : n
74          if links( s, d ) == 0
75              continue;
76          end
77
78          if ( dist( s ) + A( links( s, d ), 3 ) ) < dist( d ) && ( A( links( s, d ), 6 ) == 1 ↵
                ↳|| A( links( s, d ), 6 ) == 2 )
79              if load( links( s, d ) ) + traffic <= A( links( s, d ), 4 )
80                  dist( d ) = dist( s ) + A( links( s, d ), 3 );
81                  prev( d ) = s;
82              else
83                  % warning( 'Link %d: Link capacity is full ...\n', links( s, d ) );
84              end
85          end
86      end
87
88  end
89
90  cost = dist( dst );
91  path = 0;
92
93  end
94  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
95
96
97  if cost ~= Inf
98      % Node path
99      npath = dst;
100     while npath( 1 ) ~= src
101
102         if prev( npath( 1 ) ) > 0 && npath( 1 ) ~= prev( prev( npath( 1 ) ) )
103             npath = [ prev( npath( 1 ) ) npath ];
104         else
105             error( 'Error␣in␣npath␣...\n' );
106         end
107
108     end
109
110     % Convert node path to link path
111     path = zeros( 1, length( npath ) - 1 );
112         for i = 1 : length( npath ) - 1
113         path( i ) = links( npath( i ), npath( i + 1 ) );
114         end
115 end
```

SUURBALLE.M

This MATLAB script represents function to find the shortest pair of disjoint paths

```
%% GENERAL INFORMATION
% Suurballe's shortest cycle algorithm
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

function [ cost, primary, backup ] = suurballe( A, links, load, src, dst, traffic )

% STEP 1
DG = inf( length( links ) );
for i = 1 : length( A )
    DG( A( i, 1 ), A( i, 2 ) ) = A( i, 3 );
    DG( A( i, 2 ), A( i, 1 ) ) = A( i, 3 );
end

[ cost, path1, npath1, dist ] = dijkstra_md( A, DG, links, load, src, dst, traffic );
if cost == Inf
    error( 'No path1 from %d to %d - infinite cost\n', src, dst );
end


% STEP 2
for i = 2 : length( npath1 )
    links( npath1( i - 1 )  , npath1( i ) ) = -1;
end

for i = 1 : length( A )
    DG( A( i, 1 ), A( i, 2 ) ) = A( i, 3 ) - dist( A( i, 2 ) ) + dist( A( i, 1 )  );
    DG( A( i, 2 ), A( i, 1 ) ) = A( i, 3 ) - dist( A( i, 1 ) ) + dist( A( i, 2 )  );
end


% STEP 3
[ cost2, path2 ] = dijkstra_md( A, DG, links, load, src, dst, traffic );
if cost2 == Inf
    error( 'No path2 from %d to %d - infinite cost\n', src, dst );
end


% STEP 4
% check for unique
path = [ setdiff( path1, path2 ) setdiff( path2, path1 ) ];

if length( path ) == length( [ path1 path2 ] )

    primary = path1;
    backup = path2;

else

    links = zeros( length( links ) );
    for i = 1 : length( path )
        links( A( path( i ), 1 ), A( path( i ), 2 ) ) = path( i );
        links( A( path( i ), 2 ), A( path( i ), 1 ) ) = path( i );
    end

    [ cost, primary ] = dijkstra( A, links, load, src, dst, traffic );
    if cost == Inf
        error( 'There is no primary path between nodes %d and %d.\n', src, dst );
    end

    for i = 1 : length( primary )
        links( A( primary( i ), 1 ), A( primary( i ), 2 ) ) = 0;
        links( A( primary( i ), 2 ), A( primary( i ), 1 ) ) = 0;
    end

```

```
    [ cost2, backup ] = dijkstra( A, links, load, src, dst, traffic );
    if cost2 == Inf
        error( 'There is no backup path between nodes %d and %d.\n', src, dst );
    end

end




function [ cost, path, npath, dist ] = dijkstra_md( A, DG, links, load, src, dst, traffic )

n = length( links );        % Number of nodes in the network
visited( 1 : n ) = 0;    % Visited nodes
dist( 1 : n ) = Inf;     % It stores the shortest distance between the source node and any other node
prev( 1 : n ) = 0;       % Previous node, informs about the best previous node known to reach each network node

dist( src ) = 0;


for j = 1 : n - 1

    candidate = inf( 1, n );
    for i = 1 : n
        if visited( i ) == 0
            candidate( i ) = dist( i );
        end
    end

    [ ~, s ] = min( candidate );
    visited( s ) = 1;
    for d = 1 : n
        if links( s, d ) == 0
            continue;
        end

        if ( dist( s ) + DG( s, d ) ) < dist( d ) && links( s, d ) > 0 && A( links( s, d ), 6 ) == 1
            if load( links( s, d ) ) + traffic <= A( links( s, d ), 4 )
                dist( d ) = dist( s ) + DG( s, d );
                prev( d ) = s;
            else
                warning( 'Link %d: Link capacity is full ...\n', links( s, d ) );
            end
        end
    end

end

cost = dist( dst );
path = 0;


if cost ~= Inf
    % Node path
    npath = dst;
    while npath( 1 ) ~= src

        if prev( npath( 1 ) ) > 0 && npath( 1 ) ~= prev( prev( npath( 1 ) ) )
            npath = [ prev( npath( 1 ) ) npath ];
        else
            error( 'Error in npath ...\n' );
        end

    end

    % Convert node path to link path
```

```
133     path = zeros( 1, length( npath ) - 1 );
134         for i = 1 : length( npath ) - 1
135         path( i ) = links( npath( i ), npath( i + 1 ) );
136         end
137 end
```

GETAVGDERIVATIVEOFCOST.M

This MATLAB script represents an auxiliary function to calculate the costs

```
%% GENERAL INFORMATION
% This function returns the average 'derivative' of cost in a given point (on the grounds of a lookup table).
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014


function dc = get_avg_derivative_of_cost( Cx_row, Cy_row, load, k )

N = 10;
index = N - k + 1;

if index <= 1
    error( 'Index of iterations is too small ...' );
end

dc = 0;

if load > Cx_row( index )

        for m = index + 1 : length( Cx_row )
                if Cx_row( m ) == load
                        dc_1 = ( Cy_row( 1, m ) - Cy_row( 1, m - 1 ) ) / ( Cx_row( 1, m ) - Cx_row( 1, m - 1 ) );
                        dc_2 = ( Cy_row( 1, m + 1 ) - Cy_row( 1, m ) ) / ( Cx_row( 1, m + 1 ) - Cx_row( 1, m ) );

                        dc = ( dc_1 + dc_2 ) / 2;

                        break;
                elseif Cx_row( 1, m ) > load
                        dc = ( Cy_row( 1, m ) - Cy_row( 1, m - 1 ) ) / ( Cx_row( 1, m ) - Cx_row( 1, m - 1 ) );

                        break;
                end
        end

else
        dc = Cy_row( index ) / Cx_row( index );
end
```

GLOBALCOST.M

This MATLAB script represents an auxiliary function to calculate the costs

```
%% GENERAL INFORMATION
% This function returns the global cost value for the network.
%
% Input:
%       y - vector of link loads
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

function gc = global_cost( Cx, Cy, y )

gc = 0;

for i = 1 : length( y )
        gc = gc + link_cost( Cx( i, : ), Cy( i, : ), y( i ) );
end
```

LINKCOST.M

This MATLAB script represents an auxiliary function to calculate the costs

```
%% GENERAL INFORMATION
% This function returns the estimated cost of a link on the grounds of the provided parameters.
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

function c = link_cost( Cx_row, Cy_row, load )

if load == 0
        c = 0;
else
        c = Inf;

        for m = 1 : length( Cx_row )
                if Cx_row( m ) == load
                        c = Cy_row( m );

                        break;
                elseif Cx_row( m ) > load
                        c = ( Cy_row( m - 1 ) + Cy_row( m ) ) / 2;

                        break;
        elseif load > Cx_row( end )
            warning( 'Cx range is too small ...' );

            break;
                end
        end
end
```