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/

MAINSIMULATION.M

This MATLAB script represents the main loop of the simulation

```
%% GENERAL INFORMATION
% Discrete-time simulator module.
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

%% INPUT

% Clear the environment

%       close all;
%       clear all;
%       clc;

function [ total_d, number_d, wake_time, wake_number, sim_time, timeline, ↵
    ↳connections_availability, m_link_load, m_link_load_backup, m_Global_cost, m_Backup_cost,↵
    ↳ m_Path_cost ] = main_simulation( N, distribution, recovery_metod, percent )

% Generate a data set for the discrete-time simulator module.

    % [ id, timeline, timeline_add, filename ] = gen_input_data( N );
    [ ~, timeline, timeline_add, ~ ] = gen_input_data( N, distribution );


% Recovery meteod (0-off, 1-prot_ded_glob, 2-prot_ded_loc, 3-prot_sh_glob, 4-prot_sh_loc, 5-↵
    ↳global_restoration, 6-local_restoration)


%% SIMULATION

    % Read the complete environment snapshot from a .mat file
    % time = clock;
    sleep = 0;
    switch recovery_metod
        case 0
            load( '../modul_optymalizacyjny/output' );
        case 1
            load( '../modul_optymalizacyjny/output_prot_glob' );
        case 2
            load( '../modul_optymalizacyjny/output_prot_loc' );
        case 3
            load( '../modul_optymalizacyjny/output_ws_prot_glob' );
        case 4
            load( '../modul_optymalizacyjny/output_ws_prot_loc' );
        case 5
            load( '../modul_optymalizacyjny/output' );
            A( sleep, 6 ) = 2;
            A( :, 4 ) = ( 1 + ( percent / 100 ) ) .* y;
            A( sleep, 4 ) = Inf;
        case 6
            load( '../modul_optymalizacyjny/output' );
            A( sleep, 6 ) = 2;
            A( :, 4 ) = ( 1 + ( percent / 100 ) ) .* y;
            A( sleep, 4 ) = Inf;
    end

    if percent == Inf
        A( :, 4 ) = Inf;
    end

        % Downtime array for connections (traffic demands) & connections availability matrix

        downtime = zeros( 2, length( d ) );
    wakeup = zeros( 2, length( A ) );
        connections_availability =  true( length( d ), length( timeline ) ) ;
    link_load = zeros( length( A ), 1 );
```

```
64      % link_load_pr = zeros( length( A ), 1 );
65      link_load_ba = zeros( length( A ), 1 );
66      Global_cost = 0;
67      Backup_cost = 0;
68          Path_cost = zeros( length( d ), 1 );
69
70
71      % Main loop
72
73          for i = 1 : length( timeline ) - 1
74
75          % Matrix declaration
76          Pd_now = zeros( length( d ), length( A ) );
77          link_load_primary = zeros( length( A ), 1 );
78          link_load_backup = zeros( length( A ), 1 );
79          backup_link_load = zeros( length( A ), 1 );
80          if recovery_metod == 5 || recovery_metod == 6
81              pos = ( A == 20 );
82              A( sleep, 6 ) = 2;
83              A( pos ) = 20;
84              wakes = [];
85          end
86
87
88                  % Iterate through the 'events' array to perform required actions
89                  % for links that either went up or down (or any other valid status) at ↵
                        ↳timeline( i )
90
91          for k = 1 : size( timeline_add, 2 )
92              if timeline_add( i, k ) == 0
93                  break;
94              end
95
96              j = timeline_add( i, k );
97
98              if A( j, 6 ) == 0
99                  A( j, 6 ) = 1;
100             elseif A( j, 6 ) == 1
101                 A( j, 6 ) = 0;
102             elseif A( j, 6 ) == 2
103                 A( j, 6 ) = 20;
104             elseif A( j, 6 ) == 20
105                 A( j, 6 ) = 2;
106             end
107
108         end
109
110
111
112         for j = 1 : length( d )
113                         path_ok = true;
114
115                         % Check status of the working path
116
117                         for k = 1 : length( A )
118                                 if Pd( j, k ) == 0
119                                         % Padding detected - interrupt processing
120
121                                         break;
122                                 end
123
124                                 if A( Pd( j, k ), 6 ) == 0
125                                         % Link failure detected, try to find a suitable backup↵
                                                ↳ path..
126                                         path_ok = false;
127
128                                         break;
129                                 end
130                         end
```

```
131
132                         % Backup paths - check only if needed
133
134             if path_ok == false
135
136                 switch recovery_metod
137                     case 1
138                         path = [];
139                         for z = 1 : length( A )
140                             if backup_paths( j, z ) == 0
141                                 % Padding detected - interrupt processing (we have found a backup path)
142                                 path_ok = true;
143
144                                 % Increase the link load
145                                 for s = 1 : length( path )
146                                     link_load_backup( path( s ) ) = link_load_backup( path( s ) ) + d( j, 3 );
147                                     Pd_now( j, s ) = path( s );
148                                 end
149
150                                 break;
151                             end
152
153                             if A( backup_paths( j, z ), 6 ) == 0
154                                 % Link failure detected, there is no working backup path..
155
156                                 break;
157                             end
158                             path = [ path backup_paths( j, z ) ];
159                         end
160
161                     case 2
162                         paths = [];
163                         for k = 1 : length( A )
164                             if Pd( j, k ) == 0
165                                 path_ok = true;
166
167                                 % Increase the link load
168                                 for s = 1 : length( paths )
169                                     link_load_backup( paths( s ) ) = link_load_backup( paths( s ) ) + d( j, 3 );
170                                     Pd_now( j, s ) = paths( s );
171                                 end
172
173                                 break;
174                             end
175
176                             if A( Pd( j, k ), 6 ) == 0
177                                 path_fail = false;
178                                 path = [];
179                                 for z = 1 : length( A )
180                                     if backup_loc_paths( Pd( j, k ), z ) == 0
181                                         % Padding detected - interrupt processing
182
183                                         break;
184                                     end
185
186                                     if A( backup_loc_paths( Pd( j, k ), z ), 6 ) == 0
187                                         % Link failure detected, there is no working backup path..
188                                         path_fail = true;
189                                         break;
190                                     end
191                                     path = [ path backup_loc_paths( Pd( j, k ), z ) ];
192                                 end
193
194                                 if path_fail
195                                     break;
```

```
196                                 else
197                                     paths = [ paths path ];
198                                 end
199
200                             else
201                                 paths = [ paths Pd( j, k ) ];
202                             end
203                         end
204
205                     case 3
206                         path = [];
207                         for z = 1 : length( A )
208                             if backup_paths( j, z ) == 0
209                                 % Padding detected - interrupt processing (we have found a ↵
                                    ↳backup path)
210                                 path_ok = true;
211
212                                 % Increase the link load
213                                 for s = 1 : length( path )
214                                     link_load_backup( path( s ) ) = link_load_backup( path( s ↵
                                        ↳) ) + d( j, 3 );
215                                     Pd_now( j, s ) = path( s );
216                                 end
217
218                                 break;
219                             end
220
221                             if A( backup_paths( j, z ), 6 ) == 0 || backup_link_load( ↵
                                    ↳backup_paths( j, z ) ) + d( j, 3 ) > y2( backup_paths( j, z ↵
                                    ↳) )
222                                 % Link failure detected, there is no working backup path..
223
224                                 break;
225                             end
226                             path = [ path backup_paths( j, z ) ];
227                             backup_link_load( backup_paths( j, z ) ) = backup_link_load( ↵
                                    ↳backup_paths( j, z ) ) + d( j, 3 );
228                         end
229
230                     case 4
231                         paths = [];
232                         for k = 1 : length( A )
233                             if Pd( j, k ) == 0
234                                 path_ok = true;
235
236                                 % Increase the link load
237                                 for s = 1 : length( paths )
238                                     link_load_backup( paths( s ) ) = link_load_backup( paths( ↵
                                        ↳s ) ) + d( j, 3 );
239                                     Pd_now( j, s ) = paths( s );
240                                 end
241
242                                 break;
243                             end
244
245                             if A( Pd( j, k ), 6 ) == 0
246                                 path_fail = false;
247                                 path = [];
248                                 for z = 1 : length( A )
249                                     if backup_loc_paths( Pd( j, k ), z ) == 0
250                                         % Padding detected - interrupt processing
251
252                                         break;
253                                     end
254
255                                     if A( backup_loc_paths( Pd( j, k ), z ), 6 ) == 0 || ↵
                                            ↳backup_link_load( backup_loc_paths( Pd( j, k ), z ) ↵
                                            ↳) + d( j, 3 ) > y2( backup_loc_paths( Pd( j, k ), z ↵
                                            ↳) )
```

```
256                                     % Link failure detected, there is no working backup ↵
                                            ↳path..
257                                     path_fail = true;
258                                     break;
259                                 end
260                                 path = [ path backup_loc_paths( Pd( j, k ), z ) ];
261                                 backup_link_load( backup_loc_paths( Pd( j, k ), z ) ) = ↵
                                        ↳backup_link_load( backup_loc_paths( Pd( j, k ), z ) ↵
                                        ↳) + d( j, 3 );
262                             end
263
264                             if path_fail
265                                 break;
266                             else
267                                 paths = [ paths path ];
268                             end
269
270                         else
271                             paths = [ paths Pd( j, k ) ];
272                         end
273                     end
274
275                 case 5
276                     [ cost, path ] = dijkstra( A, links, link_load_now, d( j, 1 ), d( j, 2↵
                            ↳ ), d( j, 3 ), sleep );
277
278                     if cost ~= Inf
279                         % Increase the link load
280                         for s = 1 : length( path )
281                             link_load_backup( path( s ) ) = link_load_backup( path( s ) ) ↵
                                    ↳+ d( j, 3 );
282                             Pd_now( j, s ) = path( s );
283                         end
284
285                         wake = path( ismember( path, sleep ) );
286                         A( wake, 6 ) = 1;
287                         wakes = [ wakes wake ];
288
289                         path_ok = true;
290                     end
291
292                 case 6
293                     paths = [];
294                     wake2 = [];
295                     for k = 1 : length( A )
296                         if Pd( j, k ) == 0
297                             path_ok = true;
298
299                             % Increase the link load
300                             for s = 1 : length( paths )
301                                 link_load_backup( paths( s ) ) = link_load_backup( paths( ↵
                                        ↳s ) ) + d( j, 3 );
302                                 Pd_now( j, s ) = paths( s );
303                             end
304
305                             wakes = [ wakes wake2 ];
306
307                             break;
308                         end
309
310                         if A( Pd( j, k ), 6 ) == 0
311                             [ cost, path ] = dijkstra( A, links, link_load_now, A( Pd( j, ↵
                                    ↳k ), 1 ), A( Pd( j, k ), 2 ), d( j, 3 ), sleep );
312
313                             if cost == Inf
314                                 A( setdiff( wake2, wakes ), 6 ) = 2;
315                                 break;
316                             else
317                                 paths = [ paths path ];
```

```
318
319                                     wake = path( ismember( path, sleep ) );
320                                     A( wake, 6 ) = 1;
321                                     wake2 = [ wake2 wake ];
322                                 end
323
324                             else
325                                 paths = [ paths Pd( j, k ) ];
326                             end
327                         end
328
329                 end
330
331                 % We have not found any suitable backup path --> increase downtime
332                 if path_ok == false
333
334                                         connections_availability( j, i ) = 0;
335
336                     downtime( 1, j ) = downtime( 1, j ) + ( timeline( i + 1 ) - timeline( i ) ↵
                        ↳);
337                     downtime( 2, j ) = downtime( 2, j ) + 1;
338
339                 end
340
341             else
342                 % Primary path is OK
343                 for k = 1 : length( A )
344                     if Pd( j, k ) ~= 0
345                         link_load_primary( Pd( j, k ) ) = link_load_primary( Pd( j, k ) ) + d(↵
                            ↳ j, 3 );
346                         Pd_now( j, k ) = Pd( j, k );
347                     else
348                         break;
349                     end
350                 end
351
352             end
353             link_load_now = link_load_primary + link_load_backup;
354         end
355
356
357         delta = timeline( i + 1 ) - timeline( i );
358
359         if recovery_metod == 5 || recovery_metod == 6
360             wakeup( 1, unique( wakes ) ) = wakeup( 1, unique( wakes ) ) + delta;
361             wakeup( 2, unique( wakes ) ) = wakeup( 2, unique( wakes ) ) + 1;
362         end
363
364
365         Global_cost = Global_cost + delta * global_cost( Cx, Cy, link_load_now );
366         Path_cost = Path_cost + delta * path_cost( Cx, Cy, Pd_now, link_load_now, d( :, 3 ) );
367         backup_cost_now = sqrt( link_load_now ) .* link_load_backup ./ link_load_now;
368         backup_cost_now( isnan( backup_cost_now ) ) = 0;
369         Backup_cost = Backup_cost + delta * sum( backup_cost_now );
370         link_load = link_load + delta * link_load_now;
371         % link_load_pr = link_load_pr + link_load_primary;
372         link_load_ba = link_load_ba + delta * link_load_backup;
373         end
374
375
376 %       summary;
377     sim_time = timeline( end );
378     total_d = 10 * 365 * 24 * downtime( 1, : )' ./ sim_time;
379     number_d = 10 * 365 * 24 * downtime( 2, : )' ./ sim_time;
380     wake_time = 10 * 365 * 24 * wakeup( 1, : )' ./ sim_time;
381     wake_number = 10 * 365 * 24 * wakeup( 2, : )' ./ sim_time;
382     m_link_load = link_load ./ sim_time;
383     m_link_load_backup = link_load_ba ./ sim_time;
384     m_Global_cost = Global_cost / sim_time;
```

```
385     m_Backup_cost = Backup_cost / sim_time;
386     m_Path_cost = Path_cost ./ sim_time;
387
388
389 %% SAVE VARIABLES INTO A .mat FILE
390
391     % save( test.m, 'downtime', 'd', 'Pd', 'sleep', 'connections_availability', 'link_load', '↵
        ↳timeline', 'Global_cost', 'Path_cost' );
```

GENINPUTDATASIMULATION.M

This MATLAB script represents function to retrieve the data for simulation

```
%% GENERAL INFORMATION
% Generate a data set for the discrete-time simulator module.
%
% Author: Piotr Jaglarz (pjaglarz@student.agh.edu.pl)
% Date: 04.2014

%% INPUT DATA FOR THE SIMULATOR MODULE

function [ id, timeline, timeline_add, filename ] = gen_input_data( N, distribution )


% Load the parameters generated by the optimization module
        load( '../modul_optymalizacyjny/input' );

% Number of failure-repair pairs (down/up events) for each link

        % N = 1600;

% Events (arcs) - array of failure and repair model:
%       Events(n).failure_model - distribution of TTF (Time-to-Failure) times for the link:
%               0: exponential distribution (values in range: [param_1, param_2]; param_1 < ↵
    ↳param_2)
%               1: normal distribution (param_1 = mean, param_2 = std dev)
%       Events(n).failure_model_param_1 - the first parameter for the failure_model
%       Events(n).failure_model_param_2 - the second parameter for the failure_model
%       Events(n).repair_model - distribution of TTR (Time-to-Repair) times for the link:
%               0: uniform distribution (values in range: [param_1, param_2]; param_1 < ↵
    ↳param_2)
%               1: normal distribution (param_1 = mean, param_2 = std dev)
%       Events(n).repair_model_param_1 - the first parameter for the repair_model
%       Events(n).repair_model_param_2 - the second parameter for the repair_model

    Events = struct( ...
                        'failure_model', {}, ...
                        'failure_model_param_1', {}, ...
                        'failure_model_param_2', {}, ...
                        'repair_model', {}, ...
                        'repair_model_param_1', {}, ...
                        'repair_model_param_2', {} ...
        );

% Line Failures
%  Line failures can, for instance, be caused by a failure in the fiberoptic cable or the
%  failure of an OA or WDM line system. An assumption often made is that fiber
%  failures within a single fiberoptic cable are completely dependent, because most
%  failures are caused by dig ups, affecting all fibers within the cable. For physical
%  cables the MTBF can be specified using the cable cut (CC) metric. This is the
%  average cable length that results in a single cable cut per year (e.g., CC ? 450 km
%  means that per 450km cable, there will be on average one cable cut each year). This
%  expresses the fact that the probability to have a cable cut is larger for a longer link.
%
% The MTBF of the cable is then calculated as
%   MTBF(hours) = (CC * 365 * 24) / Length of the cable

        for i = 1 : length( A )
        if strcmp( distribution, 'exp' )
            Events( i ).failure_model = 0;
            Events( i ).failure_model_param_1 = ( 0.5 * 450 * 365 * 24 ) / A( i, 5 );
            Events( i ).repair_model = 0;
            Events( i ).repair_model_param_1 = 12;
        elseif strcmp( distribution, 'pareto' )
            Events( i ).failure_model = 1;
            Events( i ).failure_model_param_1 = 1 / ( 3.18443e-07 * ( 1 + 9 * A( i, 5 ) / max(↵
                ↳ A( :, 5 ) ) ) );
            Events( i ).repair_model = 1;
            Events( i ).repair_model_param_1 = 60 * ( 1 + 9 * A( i, 5 ) / max( A( :, 5 ) ) );
```

```
64              Events( i ).repair_model_param_2 = 2.3;
65          end
66          end
67
68  % For each link, generate a vector of time intervals (expressed as the
69  % number of time units) between consecutive status changes (down/up events).
70  % Please note that TTF and TTR times usually have different distributions.
71
72          events = zeros( length( Events ), 2*N );
73          failures = zeros( length( Events ), N );
74          repairs = zeros( length( Events ), N );
75
76      rng('shuffle');      % Generate Random Numbers That Are Different
77          for i = 1 : length( Events )
78
79                  % Failure model
80
81                  if Events( i ).failure_model == 0
82                          % Exponential distribution
83                          failures( i, : ) = exprnd( Events( i ).failure_model_param_1, 1, N );
84
85                  elseif Events( i ).failure_model == 1
86                          % % Pareto distribution
87                          % %failures( i, : ) = gprnd( 1 / Events( i ).failure_model_param_2, ↵
                                ↳Events( i ).failure_model_param_1 / Events( i ).↵
                                ↳failure_model_param_2, Events( i ).failure_model_param_1, 1, N )↵
                                ↳;
88              % failures( i, : ) = prnd( Events( i ).failure_model_param_1, Events( i ).↵
                    ↳failure_model_param_2, 1, N ) / 60;
89
90              % Exponential distribution
91                          failures( i, : ) = exprnd( Events( i ).failure_model_param_1, 1, N ) /↵
                                ↳ 60;
92
93                  end
94
95                  % Repair model
96
97                  if Events( i ).repair_model == 0
98                          % Exponential distribution
99                          repairs( i, : ) = exprnd( Events( i ).repair_model_param_1, 1, N );
100
101                 elseif Events( i ).repair_model == 1
102                         % Pareto distribution
103                         %repairs( i, : ) = gprnd( 1 / Events( i ).repair_model_param_2, Events↵
                                ↳( i ).repair_model_param_1 / Events( i ).repair_model_param_2, ↵
                                ↳Events( i ).repair_model_param_1, 1, N ) / 60;
104             repairs( i, : ) = prnd( Events( i ).repair_model_param_1, Events( i ).↵
                    ↳repair_model_param_2, 1, N ) / 60;
105
106                 end
107
108         end
109
110         % Join the time intervals to form failure-repair pairs
111
112         z = 1;
113         for j = 1 : N
114                 % Copy a column from the 'failures' matrix
115                 events( :, z ) = failures( :, j );
116                 z = z + 1;
117
118                 % Copy the corresponding column from the 'repairs' matrix
119                 events( :, z ) = repairs( :, j );
120                 z = z + 1;
121         end
122
123 % Generate the timeline for the simulation, and, at the same time, convert
124 % each time interval between consecutive events into the number of time
```

```
% units counted from the reference time (start of the simulation)

        timelines = zeros( size( events, 1 ) * size( events, 2 ), 2 );
    timelines( 1, : ) = 0;
    z = 2;

        for i = 1 : size( events, 1 )
                time_counter = 0;

                for j = 1 : size( events, 2 )
                        time_counter = time_counter + events( i, j );
                        events( i, j ) = time_counter;
                        timelines( z, 1 ) = time_counter;
            timelines( z, 2 ) = i;
                        z = z + 1;
                end
        end

        % Strip duplicate entries, sort in ascending order
    timelines = sortrows( timelines );
    timeline = zeros( 1, length( unique( timelines) ) - length( A ) );
    timeline_add = zeros( length( unique( timelines) ) - length( A ), 2 );
    timeline( 1 ) = 0;
    timeline_add( 1 ) = 0;

    y = 1;
    for i = 2 : length( timelines )
        if timelines( i - 1, 1 ) ~= timelines( i, 1 )
            y = y + 1;
            timeline( y ) = timelines( i, 1 );
            z = 1;
            timeline_add( y, z ) = timelines( i, 2 );
        else
            z = z + 1;
            timeline_add( y, z ) = timelines( i, 2 );
        end
    end

    % Generate failures data id
    id = round( 10000 * rand );

%% SAVE THE SELECTED VARIABLES INTO A .mat FILE

%     save( 'events', 'id', 'N', 'events', 'timeline', 'timeline_add', 'filename', 'failures',↵
    ↳ 'repairs' );
```