```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!
!! Carrier Broker Optimisation: Link Flow Model
!!
!!- The following script implements the LFM defined in Section 6.1
!!  of Mari Holmen's and Sindre Møgster Braaten's masters thesis
!!
!!- Authors: Mari Holmen and Sindre Møgster Braaten
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

model CarrierBrokerLFM

options explterm
options noimplicit
uses "mmxprs", "mmsystem";

parameters
    ! Data file to read from
    Data = 'data/s2-mosel-link.txt';
    ! Minimum proportion of total backup requirement reserved on an arc (aka. beta)
    MinBackupProportion = 0.25;
    ! Time limit for runtime,  maximum number of seconds for optimisation
    TimeLimit = -1;
end-parameters

writeln("Model Parameters:");
writeln("Data:", Data);
writeln("MinBackupProportion(beta):", MinBackupProportion);
writeln("TimeLimit:", TimeLimit);

declarations
    timetracker:    real; ! used to log timestamps for time consumption output
end-declarations

writeln("Building model...");
timetracker := timestamp;

!setparam("XPRS_presolve", 0);  ! uncomment to turn of presolve
if(TimeLimit>0.0) then
    setparam("XPRS_maxtime", TimeLimit);
end-if

setparam("XPRS_verbose", true); ! Turn on message printing
setparam("XPRS_MIPLOG", 2); ! 2: print information for each solution found
                            !(ALT: 0: no log, 1: summary in end, 3: log each node, -N: log every Nth node)
!!!!!!!!!!!!!!!!!!!!!!!!
! SETS
!!!!!!!!!!!!!!!!!!!!!!!!

declarations
! Set sizes
    n_Customers:            integer; ! number of customers
    n_Services:             integer; ! number of services
    n_Providers:            integer; ! number of providers
    n_Nodes:                integer; ! number of nodes in total

! Sets
    Customers:              set of integer;
    Providers:              set of integer;
    ! Used as shorthand for 'cc in Customers, ss in S_ServicesForCustomer(cc)' when cc is not needed
    Services:               set of integer;
    ! Set of nodes in the network.
    ! - First we have the customer nodes, then the internal nodes, the the provider nodes.
    Nodes:                  set of integer;
    ! Set of internal nodes in the network + all customer nodes
    ! - usage for internal nodes for each customer cc: 'nn in I_Nodes | nn<>cc'
    I_Nodes:                set of integer;
end-declarations

initializations from Data
    n_Customers;
    n_Services;
    n_Providers;
    n_Nodes;
end-initializations

Customers:= 1..n_Customers;
Services:= 1..n_Services;
Providers:= 1..n_Providers;
```

```
Nodes:= 1..n_Nodes;
I_Nodes:= 1..(n_Nodes-n_Providers);

finalize(Customers);
finalize(Services);
finalize(Providers);
finalize(Nodes);
finalize(I_Nodes);

! INDEXED SETS

declarations
    ! set of services of for each customer cc
    S_ServicesForCustomer:      set of set of integer;
end-declarations

initialisations from Data
    S_ServicesForCustomer;
end-initialisations

!!!!!!!!!!!!!!!!!!!!!!
! PARAMETERS
!!!!!!!!!!!!!!!!!!!!!!

declarations
! Parameters
    ! Price per used capacity between nodes
    K_CapPrice:                 dynamic array(Nodes,Nodes) of real;
    ! R_Revenue from serving each customer
    R_Revenue:                  dynamic array(Customers) of real;
    ! Price of placing a service at a provider
    H_PlacePrice:               dynamic array(Services,Providers) of real;
    ! Latency requirement for each service from customer to provider
    G_LatencyReq:               array(Services) of real;
    ! Bandwidth requirement for each service from customer to provider
    B_BandwidthReqUp:               array(Services) of real;
    ! Bandwidth requirement for each service from provider to customer
    B_BandwidthReqDown:         array(Services) of real;
    ! Minimum avarage availability for each service
    Y_AvailabilityReq:          array(Services) of real;
    ! Lateny between each pair of nodes
    T_LinkLatency:              dynamic array(Nodes,Nodes) of real;
    ! Bandwidth capacity between each pair of nodes
    F_BandwidthCap:             dynamic array(Nodes,Nodes) of real;
    ! Expected availability for each owned link between each pair of nodes
    D_AvailabilityExp:          dynamic array(Nodes,Nodes) of real;
    ! Node for each provider
    E_ProviderNode:             set of integer;

! Network data interpretation configuration
    Symmetric:              boolean;

end-declarations

initialisations from Data
    K_CapPrice;
    R_Revenue;
    H_PlacePrice;
    G_LatencyReq;
    B_BandwidthReqUp;
    B_BandwidthReqDown;
    Y_AvailabilityReq;
    T_LinkLatency;
    F_BandwidthCap;
    D_AvailabilityExp;

    Symmetric;
end-initialisations

! Provider nodes are the n_Providers last nodes in network
E_ProviderNode:=(n_Nodes-n_Providers+1)..n_Nodes;
finalize(E_ProviderNode);

! If Symmetric is set to true in provided dataset
! - duplicate all arcs in dataset in its opposite direction if opposite not already specified
if(Symmetric) then
    forall(nn in Nodes, mm in Nodes) do

        if(exists(K_CapPrice(nn,mm)) and not exists(K_CapPrice(mm,nn))) then
            create(K_CapPrice(mm,nn));
            K_CapPrice(mm,nn):=K_CapPrice(nn,mm);
```

```
        end-if

        if(exists(T_LinkLatency(nn,mm)) and not exists(T_LinkLatency(mm,nn))) then
            create(T_LinkLatency(mm,nn));
            T_LinkLatency(mm,nn):=T_LinkLatency(nn,mm);
        end-if

        if(exists(F_BandwidthCap(nn,mm)) and not exists(F_BandwidthCap(mm,nn))) then
            create(F_BandwidthCap(mm,nn));
            F_BandwidthCap(mm,nn):=F_BandwidthCap(nn,mm);
        end-if

        if(exists(D_AvailabilityExp(nn,mm)) and not exists(D_AvailabilityExp(mm,nn))) then
            create(D_AvailabilityExp(mm,nn));
            D_AvailabilityExp(mm,nn) :=D_AvailabilityExp(nn,mm);
        end-if
    end-do
end-if

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! VARIABLES
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

declarations
    !Variables
    ! - x: binary, placement of service at provider
    x_Placement:        dynamic array(Services, Providers)          of mpvar;
    ! - u: binary, use of arc for service for uplink (and oppsite arc for downlink)
    u_UsePrimary:       dynamic array(Nodes,Nodes,Services)         of mpvar;
    ! - y: binary, serving of a customer
    y_Serve:            dynamic array(Customers)                    of mpvar;
    ! - b : binary, use of arc for service for backup uplink (and opposite arc for downlink)
    b_UseBackup:        dynamic array(Nodes,Nodes,Services)         of mpvar;
    ! - r : binary, is service s needs backup on its path to provider p
    r_RequireBackup:    dynamic array(Services,Providers)           of mpvar;
    ! - l (lambda): continuous, bandwidth reserved for backup on a (owned) link
    l_BackupRes:        dynamic array(Nodes,Nodes)                  of mpvar;
    ! - l: binary, indicates if two services have overlapping primary paths
    l_Overlap:          dynamic array(Services,Services)            of mpvar;
end-declarations

! - for all valid combinations of service and provider
forall (ss in Services, pp in Providers | exists(H_PlacePrice(ss,pp))) do
    create (x_Placement(ss,pp));
    x_Placement(ss,pp) is_binary;
    create(r_RequireBackup(ss,pp));
    r_RequireBackup(ss,pp) is_binary;
end-do

! - for evary arc in network
forall(ii in Nodes, jj in Nodes) do
    create(l_BackupRes(ii,jj));
end-do

! - for every service
forall(cc in Customers, ss in S_ServicesForCustomer(cc)) do
    ! - for every arc in network
    ! -- EXCEPT: arcs in to customer node of service, as paths from customer to provider will
    !           never traverse these links
    forall(ii in Nodes, jj in Nodes | jj<>cc and exists(F_BandwidthCap(ii,jj))) do
        create(u_UsePrimary(ii,jj,ss));
        u_UsePrimary(ii,jj,ss) is_binary;
        create(b_UseBackup(ii,jj,ss));
        b_UseBackup(ii,jj,ss) is_binary;
    end-do
end-do

! - for all customers
forall(cc in Customers) do
    create(y_Serve(cc));
    y_Serve(cc) is_binary;
end-do

! - for every distinct pair of two services
forall (ss in Services, tt in Services | ss < tt) do
    create(l_Overlap(ss,tt));
end-do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! CONSTRAINTS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
declarations
! Objective function
    Total_Profits:                                                      linctr;

! Constraints
    ServeCustomer:              dynamic array(Services)                  of linctr;
    ArcCapacity:                dynamic array(Nodes,Nodes)               of linctr;
    PrimaryStartRequirement:    dynamic array(Services)                  of linctr;
    BackupStartRequirement:     dynamic array(Services)                  of linctr;
    BandwidthFlowPrimary:       dynamic array(Services,I_Nodes)          of linctr;
    BandwidthFlowBackup:        dynamic array(Services,I_Nodes)          of linctr;
    PrimaryEndRequirement:      dynamic array(Services,Providers)        of linctr;
    BackupEndRequirement:       dynamic array(Services,Providers)        of linctr;
    PrimaryLatencyRequirement:  dynamic array(Services,Providers)        of linctr;
    BackupLatencyRequirement:   dynamic array(Services,Providers)        of linctr;
    AllocateBackupPath:         dynamic array(Services,Providers)        of linctr;
    AvailabilityRequirement:    dynamic array(Services)                  of linctr;
    SumBackupLimit:             dynamic array(Nodes,Nodes)               of linctr;
    SingleBackupLimit:          dynamic array(Nodes,Nodes,Services)      of linctr;
    LinkDisjoint:               dynamic array(Nodes,Nodes,Services)      of linctr;
    PrimaryOverlap:             dynamic array(Nodes,Nodes,Services,Services)  of linctr;
    BackupOverlap:              dynamic array(Nodes,Nodes,Services,Services)  of linctr;
end-declarations

! OBJECTIVE FUNCTION
! - total profits from serving customers
Total_Profits := (
        sum (cc in Customers) (
            ! R_Revenue from serving customer (if served)
            R_Revenue(cc)*y_Serve(cc)
            -
            ! costs associated with customer's required services
            sum (ss in S_ServicesForCustomer(cc)) (
                ! placement cost
                sum (pp in Providers) (
                    H_PlacePrice(ss,pp)*x_Placement(ss,pp)
                )
                +
                ! network usage cost
                sum (nn in Nodes, mm in Nodes | exists(K_CapPrice(nn,mm))) (
                    K_CapPrice(nn,mm)
                    *
                    (
                        B_BandwidthReqUp(ss)*u_UsePrimary(nn,mm,ss)
                        +
                        B_BandwidthReqDown(ss)*u_UsePrimary(mm,nn,ss)
                    )
                )
            )
        )
        -
        !Backup use cost
        sum (nn in Nodes, mm in Nodes | exists(K_CapPrice(nn,mm))) (
            K_CapPrice(nn,mm)*l_BackupRes(nn,mm)
        )
    );

! SERVE CUSTOMER CONSTRAINT
! Customers can only be served if all services for customer is provided
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc)) do
        ServeCustomer(ss) := sum (pp in Providers) x_Placement(ss,pp) - y_Serve(cc) = 0;
    end-do
end-do

! ARC TOTAL CAPACITY CONSTRAINT
! Use of an arc must not exceed its capacity (primary + backup cap)
forall (nn in Nodes, mm in Nodes| exists(F_BandwidthCap(nn,mm)))do
    ArcCapacity(nn,mm) := (
        sum (ss in Services) (
            B_BandwidthReqUp(ss)*u_UsePrimary(nn,mm,ss)
            +
            B_BandwidthReqDown(ss)*u_UsePrimary(mm,nn,ss)
        )
        +
        l_BackupRes(nn,mm)
        <=
        F_BandwidthCap(nn,mm)
    );
end-do
```

```
! CUSTOMER NODE ROUTING START CONSTRAINTS:
! primary / backup must select arc from customer node if chosen
forall (cc in Customers, ss in S_ServicesForCustomer(cc)) do
    PrimaryStartRequirement(ss) := (
            sum(mm in Nodes | mm<>cc) (u_UsePrimary(cc,mm,ss) )
             - y_Serve(cc)
        )= 0;
    BackupStartRequirement(ss) := (
            sum(mm in Nodes | mm<>cc) b_UseBackup(cc,mm,ss)
            -
            sum(pp in Providers) r_RequireBackup(ss,pp)
        ) =0;
end-do

! ROUTING FLOW CONSTRAINTS
! - routing in to a node for a service must be equal to the routing out
!   (unless a it is a provider node or the service's customer node)
forall (cc in Customers, ss in S_ServicesForCustomer(cc), nn in I_Nodes | nn<>cc ) do
    BandwidthFlowPrimary(ss,nn) := (
            sum (mm in Nodes | exists(F_BandwidthCap(nn,mm))) u_UsePrimary(nn,mm,ss)
            -
            sum(mm in Nodes | exists(F_BandwidthCap(mm,nn))) u_UsePrimary(mm,nn,ss)
        ) = 0;
    BandwidthFlowBackup(ss,nn) := (
            sum (mm in Nodes | exists(F_BandwidthCap(nn,mm))) b_UseBackup(nn,mm,ss)
            -
            sum(mm in Nodes | exists(F_BandwidthCap(mm,nn))) b_UseBackup(mm,nn,ss)
        ) = 0;
end-do

! PLACEMENT SIDE ROUTING END CONSTRAINTS
! primary / backup must select arc in to placement node if chosen, or act as transit node
! if not selected / not able to be selected
! and primary and backup routing must end at same provider
forall (ss in Services, pp in Providers) do
    PrimaryEndRequirement(ss,pp) := (
            sum (nn in Nodes | nn<>E_ProviderNode(pp)) (
                u_UsePrimary(nn, E_ProviderNode(pp),ss)
            )
            -
            sum(mm in Nodes | mm<>E_ProviderNode(pp)) (
                u_UsePrimary(E_ProviderNode(pp),mm,ss)
            )
            -
            x_Placement(ss,pp)
        ) = 0;
    BackupEndRequirement(ss,pp) := (
            sum (nn in Nodes | nn<>E_ProviderNode(pp)) (
                b_UseBackup(nn, E_ProviderNode(pp),ss)
            )
            -
            sum(mm in Nodes | mm<>E_ProviderNode(pp)) (
                b_UseBackup(E_ProviderNode(pp),mm,ss)
            )
            -
            r_RequireBackup(ss,pp)
    ) = 0;

    if(exists(H_PlacePrice(ss,pp))) then
        ! can only have backup paths to same provider as primary
        AllocateBackupPath(ss,pp):= r_RequireBackup(ss,pp) - x_Placement(ss,pp) <= 0;
    end-if
end-do

! LATENCY REQUIREMENT CONSTRAINTS
! - user -> placement: for each service, latency for any used path must meet latency requirements
forall (ss in Services, pp in Providers) do
    PrimaryLatencyRequirement(ss,pp) :=
            sum(nn in Nodes, mm in Nodes) (
                T_LinkLatency(nn,mm)*(u_UsePrimary(nn,mm,ss) + u_UsePrimary(mm,nn,ss))
            )
            <= G_LatencyReq(ss);

    BackupLatencyRequirement(ss,pp) :=
            sum(nn in Nodes, mm in Nodes) (
                T_LinkLatency(nn,mm)*(b_UseBackup(nn,mm,ss) + b_UseBackup(mm,nn,ss))
            )
            <= G_LatencyReq(ss);
end-do
```

```
! AVAILABILITY CONSTRAINTS
! Primary path must have sufficient availability or a link disjoint backup path must be provided
! - linearised by using logarithms
forall (ss in Services) do
    AvailabilityRequirement(ss) := (
        sum( nn in Nodes, mm in Nodes | exists(D_AvailabilityExp(nn,mm))) (
            ln(D_AvailabilityExp(nn,mm)) * u_UsePrimary(nn,mm,ss)
        )
        +
        sum(pp in Providers) r_RequireBackup(ss,pp)
        >=
        ln(Y_AvailabilityReq(ss)));
end-do

! SUM BACKUP REQUIREMENT
! must reserve a certain proportion of the sum of backup requirements on an arc
forall (nn in Nodes, mm in Nodes) do
    SumBackupLimit(nn,mm) := (
        MinBackupProportion*
        sum(ss in Services) (
            B_BandwidthReqUp(ss)*b_UseBackup(nn,mm,ss)
            +
            B_BandwidthReqDown(ss)*b_UseBackup(mm,nn,ss)
        )
        -
        l_BackupRes(nn,mm)
        <= 0);
end-do

forall (ii in Nodes, jj in Nodes, ss in Services ) do
    ! MAXIMUM BACKUP CONTRAINT
    ! Must reserve backup capacity at least as high as the maximal single backup requirement
    SingleBackupLimit(ii,jj,ss) := (
        B_BandwidthReqUp(ss)
        *b_UseBackup(ii,jj,ss)  ! 1 if ii,jj is used in way UP
        +
        B_BandwidthReqDown(ss)
        *b_UseBackup(jj,ii,ss)  ! 1 if ii,jj is used in way DOWN
        -
        l_BackupRes(ii,jj)
        <= 0
    );

    ! LINK DISJOINT CONSTRAINTS
    ! The primary and backup path (if given) for a service must be link disjoint
    LinkDisjoint(ii,jj,ss) := b_UseBackup(ii,jj,ss) + u_UsePrimary(ii,jj,ss) <=1;
end-do

! SERVICE PATH OVERLAP CONSTRAINTS
! To services have overlapping main paths if for any arc both paths are represented
! for every service combination
forall(ss in Services, tt in Services | ss < tt) do
    ! for every LINK ( (i,j) in A | i < j)
    forall(ii in Nodes, jj in Nodes | ii < jj and exists(F_BandwidthCap(ii,jj))) do

        ! PRIMARY PATH OVERLAP CONSTRAINTS
        ! Two services have overlapping primary paths if for any LINK both services
        ! has selected one of the link's two arcs
        PrimaryOverlap(ii, jj, ss, tt):=
            u_UsePrimary(ii,jj,ss)+u_UsePrimary(jj,ii,ss) ! 1 if ss uses link
            +
            u_UsePrimary(ii,jj,tt)+u_UsePrimary(jj,ii,tt) ! 1 if tt uses link
            -
            l_Overlap(ss, tt)
            <= 1;

        ! BACKUP PATH OVERLAP CONSTRAINT
        ! backup paths may not overlap at any LINK if their primary paths overlap anywhere
        BackupOverlap(ii, jj, ss,tt):=
            b_UseBackup(ii,jj,ss)+b_UseBackup(ii,jj,ss) ! 1 if ss uses link
            +
            b_UseBackup(ii,jj,tt)+b_UseBackup(ii,jj,tt) ! 1 if tt uses link
            +
            l_Overlap(ss, tt)
            <= 2;
    end-do
end-do

writeln("Model building completed in ", timestamp - timetracker, " seconds");

writeln("Solving model...");
```

```
timetracker := timestamp;
maximize(XPRS_PRI, Total_Profits);

if (getprobstat=XPRS_OPT) then
    writeln("Model solved in ", timestamp - timetracker," seconds");
else
    writeln("Model was not solved after ", timestamp - timetracker," seconds");
end-if

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!   Solution output:
! - this following part contains logic for outputting the solution as human
!   readable text and is not part of the model itself.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

writeln("\nTotal Profits: ", getobjval);
writeln("\nBackup Costs: ",
    sum(nn in Nodes, mm in Nodes) getsol(l_BackupRes(nn,mm))*K_CapPrice(nn,mm)
);

! for all customers being served
forall(cc in Customers | getsol(y_Serve(cc)) > 0.1) do
    ! output customer information and generated profits (excluding backup costs)
    writeln("\n\nCustomer ", cc, " (node ",cc,") is being served\n - R_Revenue: ",
        R_Revenue(cc)*getsol(y_Serve(cc)),"\n - profits: ",(
            R_Revenue(cc)*getsol(y_Serve(cc))
            -
            ! costs associated with customer's required services
            sum (ss in S_ServicesForCustomer(cc))
                (
                    ! placement cost
                    sum (pp in Providers)
                        H_PlacePrice(ss,pp)*getsol(x_Placement(ss,pp))
                    +
                    ! network usage cost
                    sum (nn in Nodes, mm in Nodes) (
                            K_CapPrice(nn,mm)
                            *(
                                getsol(u_UsePrimary(nn,mm,ss)*B_BandwidthReqUp(ss))
                                +getsol(u_UsePrimary(mm,nn,ss)*B_BandwidthReqDown(ss)))
                            )
                )
        )
        );

    ! for all services of the served customer
    forall(ss in S_ServicesForCustomer(cc)) do
        ! for the provider selected for the service (x only > 0.1 for one)
        forall(pp in Providers | getsol(x_Placement(ss,pp)) > 0.1) do
            ! output information about service and placement
            writeln(
                "\n - Service ",ss,":\n  - Costs: ",
                (   ! Calculate costs for this specific service
                    H_PlacePrice(ss,pp)*getsol(x_Placement(ss,pp))
                    +
                    sum (nn in Nodes, mm in Nodes) (
                        K_CapPrice(nn,mm)
                        *(
                            getsol(u_UsePrimary(nn,mm,ss))*B_BandwidthReqUp(ss)
                            +getsol(u_UsePrimary(mm,nn,ss))*B_BandwidthReqDown(ss)
                        )
                    )
                ),
                "\n  - placement: provider #", pp, " (node ",(n_Nodes-n_Providers+pp),
                ") - Cost: ",H_PlacePrice(ss,pp),
                "    -Availability without backup: ",
                exp(
                    sum( nn in Nodes, mm in Nodes | (exists(D_AvailabilityExp(nn,mm))) ) (
                        getsol(u_UsePrimary(nn,mm,ss))*ln(D_AvailabilityExp(nn,mm))
                    )
                ),
                "    -Availability requirement: ", Y_AvailabilityReq(ss)
            );
        end-do

        ! output primary path network routing information for service
        ! - up-link
        writeln("   - ARCS:\n      - primary usage up:");
        forall(nn in Nodes, mm in Nodes) do
```

```
            if (getsol(u_UsePrimary(nn,mm,ss)) > 0.1) then
                writeln("        - (", nn, ",", mm, ") : ",
                        B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss)),
                        "    (",
                        K_CapPrice(nn,mm)*B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss)),")" );
            end-if
        end-do
        ! - down-link
        writeln("      - primary usage down:");
        forall(nn in Nodes, mm in Nodes) do
            if (getsol(u_UsePrimary(nn,mm,ss)) > 0.1) then
                writeln("        - (", mm, ",", nn, ") : ",
                        B_BandwidthReqDown(ss)*getsol(u_UsePrimary(nn,mm,ss)),
                        "    (",
                        K_CapPrice(nn,mm)*B_BandwidthReqDown(ss)*getsol(u_UsePrimary(nn,mm,ss)),")" );
            end-if
        end-do

        ! if this service requires a backup path (given its primary path routing)
        if (
            sum( nn in Nodes, mm in Nodes | (exists(D_AvailabilityExp(nn,mm))) ) (
                getsol(u_UsePrimary(nn,mm,ss))*ln(D_AvailabilityExp(nn,mm))
            )
            < ln(getsol(Y_AvailabilityReq(ss)))
        ) then
            ! output backup path network routing information
            ! - up-link
            writeln("      - backup usage up:");
            forall (nn in Nodes, mm in Nodes) do
                if (getsol(b_UseBackup(nn,mm,ss))=1) then
                    writeln("        - (",nn,",",mm,"): ", getsol(l_BackupRes(nn,mm)));
                end-if
            end-do
            ! - down-link
            writeln("      - backup usage down:");
            forall (nn in Nodes, mm in Nodes) do
                if (getsol(b_UseBackup(nn,mm,ss))=1) then
                    writeln("        - (",mm,",",nn,"): ", getsol(l_BackupRes(mm,nn)));
                end-if
            end-do
        end-if
    end-do
end-do

!! Output information about total bandwidth usage on arcs
! arcs with high bandwidth usage
writeln("\n\nArcs with high utilisation of capacity (>=90%):");
forall(nn in Nodes, mm in Nodes | exists(F_BandwidthCap(nn,mm))) do
    if (sum(ss in Services) B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) >=
F_BandwidthCap(nn,mm)*0.9 then
        writeln(
            " - (",nn,",",mm,") ",
            (
                100*sum(ss in Services) (
                    B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))
                ) / F_BandwidthCap(nn,mm)
            ),
            " %"
        );
    end-if
end-do
! arcs with medium bandwidth usage
writeln("\n\nArcs with medium utilisation of capacity (< 10%, < 90%):");
forall(nn in Nodes, mm in Nodes | exists(F_BandwidthCap(nn,mm))) do
    if ((sum(ss in Services)B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) >
F_BandwidthCap(nn,mm)*0.1 and
        (sum(ss in Services)B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))) <
F_BandwidthCap(nn,mm)*0.9) then
        writeln(
            " - (",nn,",",mm,") ",
            (
                100*sum(ss in Services)(
                    B_BandwidthReqUp(ss)*getsol(u_UsePrimary(nn,mm,ss))
                ) / F_BandwidthCap(nn,mm)
            ),
            " %"
        );
    end-if
end-do

end-model
```