```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!
!! Carrier Broker Optimisation: Path Flow Model
!!
!!- The following script implements the PFM defined in Section 6.2
!!  of Mari Holmen's and Sindre Møgster Braaten's masters thesis
!!
!!- Authors: Mari Holmen and Sindre Møgster Braaten
!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

model CarrierBrokerPFM

uses "mmxprs"; !gain access to the Xpress-Optimizer solver
options explterm
options noimplicit

uses "mmxprs", "mmsystem";

parameters
    ! Data file to read from
    Data = 'data/ml_multi.txt';
    ! Minimum proportion of total backup requirement reserved on an arc
    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; ! assigns current "timestamp" to timetracker

!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
    n_Paths:                integer; ! number of paths

! Sets
    Customers:              set of integer;
    Providers:              set of integer;
    ! Used for shorthand for 'cc in Customers, ss in S_ServiceForCustomer(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;
    Paths:                  set of integer;
end-declarations

initializations from Data
    n_Customers;
    n_Services;
    n_Providers;
    n_Nodes;
    n_Paths;

end-initializations
```

```
Customers:= 1..n_Customers;
Services:= 1..n_Services;
Providers:= 1..n_Providers;
Nodes:= 1..n_Nodes;
Paths := 1..n_Paths;

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

! INDEXED SETS

declarations
    ! set of services of for each customer
    S_ServicesForCustomer:      set of set of integer;
    ! paths for each pair of service and provider
    K_PathsServiceProvider:     dynamic array(Services,Providers)   of set of integer;
    ! set of paths using each link
    L_PathsUsingArc:            dynamic array(Nodes,Nodes)          of set of integer;
end-declarations

initialisations from Data
    S_ServicesForCustomer;
    K_PathsServiceProvider;
    L_PathsUsingArc;
end-initialisations

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

declarations
    ! R_Revenue from serving each customer cc
    R_Revenue:                 dynamic array(Customers)            of real;
    ! Bandwidth capacity between each pair of nodes ii,jj
    F_BandwidthCap:             dynamic array(Nodes,Nodes)          of real;
    ! bandwidth usage on arc ii,jj for path kk
    U_PathBandwidthUsage:       dynamic array(Nodes,Nodes,Paths)    of real;
    ! cost of using path kk
    C_PathCost:                dynamic array(Paths)                of real;
    ! cost per bandwidth used for backup paths on arc ii,jj
    C_BackupCost:              dynamic array(Nodes,Nodes)          of real;
    ! availability of path kk alone
    D_PathAvailability:         dynamic array(Paths)                of real;
    ! 2d array of availability for paths kk and bb ( P(A)P(B|A) )
    D_CombinationAvailability:  dynamic array(Paths,Paths)          of real;
    ! array of availability req for services
    Y_AvailabilityReq:          dynamic array(Services)             of real;

    BigMBackup:                dynamic array(Nodes,Nodes,Services) of real;

    Symmetric:                                                    boolean;
end-declarations


initialisations from Data
    R_Revenue;
    F_BandwidthCap;
    U_PathBandwidthUsage;
    C_PathCost;
    C_BackupCost;
    D_PathAvailability;
    D_CombinationAvailability;
    Y_AvailabilityReq;
    Symmetric;
end-initialisations

if(Symmetric) then
    forall(ii in Nodes, jj in Nodes) do
        if(exists(F_BandwidthCap(ii,jj)) and not exists(F_BandwidthCap(jj,ii))) then
            create(F_BandwidthCap(jj,ii));
            F_BandwidthCap(jj,ii):=F_BandwidthCap(ii,jj);
        end-if
    end-do
end-if

! for every arc in network
forall(ii in Nodes, jj in Nodes | exists(F_BandwidthCap(ii,jj))) do
```

```
    ! for every service
    forall(ss in Services) do
        ! set BigMBackup(ii,jj,ss) to highest bandwidth usage of any k for s in i,j
        create(BigMBackup(ii,jj,ss));
        BigMBackup(ii,jj,ss) := 0.0;
        forall(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) do
            forall(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj))) do
                if(BigMBackup(ii,jj,ss) < U_PathBandwidthUsage(ii,jj,kk)) then
                    BigMBackup(ii,jj,ss):= U_PathBandwidthUsage(ii,jj,kk);
                end-if
            end-do
        end-do
    end-do
end-do

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

declarations
    ! - x: binary, placement of service at provider
    x_Placement:        dynamic array(Services, Providers)      of mpvar;
    ! - y: binary, serving of a customer
    y_Serve:            dynamic array(Customers)                of mpvar;
    ! - u: binary, indicates which paths are used
    u_UsePrimaryPath:   dynamic array(Paths)                    of mpvar;
    ! - v: binary, indicates which backup paths are used
    v_UseBackupPath:    dynamic array(Paths)                    of mpvar;
    ! - o: binary, indicates if a combination of main and backup path is chosen
    o_UseCombination:   dynamic array(Paths,Paths)              of mpvar;
    ! - lambda: continous, amount of capacity reserved on a link for backup
    l_Lambda:           dynamic array(Nodes,Nodes)              of mpvar;
    ! f: binary, indicates if a service has a need to reserve backup capacity on a arc
    f_needsBackupOnArc: dynamic array(Nodes,Nodes,Services)     of mpvar;
    ! - q: continous, amount of backup capacity needed on an arc for a service
    q_backupPerService: dynamic array(Nodes, Nodes, Services)   of mpvar;
    ! - l: binary, indicates if two services have overlapping primary paths
    l_Overlap:          dynamic array(Services,Services)        of mpvar;
end-declarations

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

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

! - for all paths
forall(pp in Paths) do
    create(u_UsePrimaryPath(pp));
    u_UsePrimaryPath(pp) is_binary;
    create(v_UseBackupPath(pp));
    v_UseBackupPath(pp) is_binary;
end-do

! - for every possible combination of two paths as primary and backup
forall (pp in Paths, bb in Paths | exists(D_CombinationAvailability(pp,bb))) do
    create(o_UseCombination(pp,bb));
    o_UseCombination(pp,bb) is_binary;
end-do

! - for every arc
forall (ii in Nodes, jj in Nodes, ss in Services | exists(F_BandwidthCap(ii,jj))) do
    create(l_Lambda(ii,jj));
    create(q_backupPerService(ii,jj,ss));
    create(f_needsBackupOnArc(ii,jj,ss));
    f_needsBackupOnArc(ii,jj,ss) 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));
    l_Overlap(ss,tt) is_binary;
end-do
```

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

declarations
! Objective function
    Total_Profits:                                                             linctr;

! Constraints
    ServeCustomer:              dynamic array(Services)                         of linctr;
    AllocatePrimaryPath:        dynamic array(Services,Providers)               of linctr;
    AllocateBackupPath:         dynamic array(Services,Providers)               of linctr;
    ArcCapacity:                dynamic array(Nodes,Nodes)                      of linctr;
    AvailabilityRequirement:    dynamic array(Services)                         of linctr;
    SumBackupLimit:             dynamic array(Nodes,Nodes)                      of linctr;
    SingleBackupLimit:          dynamic array(Nodes,Nodes,Services)             of linctr;
    NeedsBackupCapacity:        dynamic array(Nodes,Nodes,Services)             of linctr;
    NeededBackupCapacity:       dynamic array(Nodes,Nodes,Services)             of linctr;
    PathComboRequirement:       dynamic array(Paths,Paths)                      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)
        )
        -
        ! for all paths
        sum(kk in Paths) (
            ! add path cost if used as primary
            C_PathCost(kk)*u_UsePrimaryPath(kk)
        )
        -
        ! for eac arc
        sum(ii in Nodes, jj in Nodes | exists(F_BandwidthCap(ii,jj)))(
            ! add cost of bandwidth reserved for backup paths
            C_BackupCost(ii, jj)*l_Lambda(ii, jj)
    );


! 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 | exists(K_PathsServiceProvider(ss,pp))) (
                x_Placement(ss,pp)
            ) = y_Serve(cc);
    end-do
end-do

! for every service-provider pair
forall(ss in Services, pp in Providers | exists(K_PathsServiceProvider(ss,pp))) do
    ! ALLOCATE PRIMARY PATH CONSTRAINT
    ! If a service is placed at a provider, a primary path connecting to that provider location
    ! must be chosen
    AllocatePrimaryPath(ss,pp) :=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            u_UsePrimaryPath(kk)
        ) = x_Placement(ss,pp);

    ! ALLOCATE BACKUP PATH CONSTRAINT
    ! can only select a backup path to a provider if also selected primary path to the same provider
    AllocateBackupPath(ss,pp) :=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            v_UseBackupPath(kk)
        )
        <=
        sum(kk in K_PathsServiceProvider(ss,pp)) (
            u_UsePrimaryPath(kk)
        );
end-do

! ARC CAPACITY CONSTRAINT
! The total used bandwidth for main paths and reserved for backup paths must not exceed the
! arcs capacity
forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    ArcCapacity(ii,jj) :=
        ! for each path
        sum(kk in L_PathsUsingArc(ii,jj)) (
            ! add bw req for path if used as primary
            U_PathBandwidthUsage(ii,jj,kk)*u_UsePrimaryPath(kk)
        )
        ! add bandwidth reserved for backup paths on arc
        +
        l_Lambda(ii,jj)
        <= F_BandwidthCap(ii,jj);
end-do

! AVAILABILITY REQUIREMENT CONSTRAINT
! The selected main path, with possible backup path, must provide an availability equal to
! or higher than requirement
! - single main path: P(A)
! - with backup path: P(A) + P(B) - P(A)P(B|A)
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc)) do
        AvailabilityRequirement(ss) :=
            ! Availability from main (and backup path): P(A) + P(B)
            sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) (
                sum(kk in K_PathsServiceProvider(ss,pp)) (
                    D_PathAvailability(kk)
                    *
                    (
                        u_UsePrimaryPath(kk)
                        +
                        v_UseBackupPath(kk)
                    )
                )
            )
            -
            ! subtract P(A)P(B|A) if backup path is chosen
            sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp))) (
                sum(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp))
                    D_CombinationAvailability(kk,bb)*o_UseCombination(kk,bb)
            )
            >= Y_AvailabilityReq(ss)* y_Serve(cc);
    end-do
end-do

! BACKUP BANDWIDTH RESERVATION CONSTRAINTS
! - for every arc and service
forall (ii in Nodes, jj in Nodes, ss in Services | exists(F_BandwidthCap(ii,jj))) do

    ! NEEDS BACKUP CAPACITY ON ARC CONSTRAINT
    ! Sets the f variable to 1 if there is a need for backup capacity reservation
    !for the service on the arc
    NeedsBackupCapacity(ii,jj,ss) :=
        q_backupPerService(ii,jj,ss) - BigMBackup(ii,jj,ss)*f_needsBackupOnArc(ii,jj,ss) <= 0;

    ! AMOUNT BACKUP CAPACITY NEEDED ON ARC CONSTRAINT
    ! a service will require a backup reservation at an arc equal to its backup arc requirement
    ! minus the capacity used by the primary path at the same arc (as this capacity will be released
    ! if the primary path goes down and the backup is needed)
    NeededBackupCapacity(ii,jj,ss) :=
        ! bandwidth for backup minus bandwidth for primary on arc
        sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp)))(
            sum(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj)))(
                U_PathBandwidthUsage(ii,jj,kk)
                *
                (
                    v_UseBackupPath(kk)
                    -
                    u_UsePrimaryPath(kk)
                )
            )
        )
        -
        q_backupPerService(ii,jj,ss)
        <= 0;
end-do

! SUM SERVICE BACKUP BANDWIDTH CONSTRAINTS
! bandwidth reserved for backup paths on an arc is at least a fraction of the total bandwidth of all
! backup paths using that arc
forall( ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    SumBackupLimit(ii,jj) :=
        MinBackupProportion*
        sum(cc in Customers) (
            sum(ss in S_ServicesForCustomer(cc)) (
                q_backupPerService(ii,jj,ss)
            )
        )
        <= l_Lambda(ii,jj);
end-do

! SINGLE SERVICE BACKUP BANDWIDTH CONSTRAINT
! bandwidth reserved for backup paths must be at least as high as the bandwidth required by the
! the most demaning single service
forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    forall(ss in Services) do
        SingleBackupLimit(ii,jj,ss) :=
            q_backupPerService(ii,jj,ss) <= l_Lambda(ii,jj);
    end-do
end-do

! PATH COMBINATION CONSTRAINT
! if path kk is used as main path and path bb as backup path, the corresponding combo variable
! o_UseCombination(kk,bb) must also be 1
! for any valid combination of two paths; two paths from same service-provider pair
forall(cc in Customers) do
    forall(ss in S_ServicesForCustomer(cc), pp in Providers| exists(K_PathsServiceProvider(ss,pp))) do
        forall(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp)) do
            PathComboRequirement(kk,bb) :=
                u_UsePrimaryPath(kk) + v_UseBackupPath(bb) - o_UseCombination(kk,bb) <= 1;
        end-do
    end-do
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 (every (ii,jj) where ii < jj)
    forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do

        ! only for evey link (ii < jj) (failured happen at link level -> failing both arcs)
        if(ii < jj) then
            ! SERVICE PATH OVERLAP CONSTRAINTS
            ! Two services have overlapping primary paths if for any LINK both paths are represented
            PrimaryOverlap(ii, jj, ss, tt):=
                sum(pp in Providers | exists(K_PathsServiceProvider(ss,pp)))(
                    ! L_PathsUsingArc(ii,jj)=L_PathsUsingArc(jj,ii) -> need only paths using LINK ii,jj
                    sum(kk in (K_PathsServiceProvider(ss,pp)*L_PathsUsingArc(ii,jj))) (
                        u_UsePrimaryPath(kk)
                    )
                )
                +
                sum(pp in Providers | exists(K_PathsServiceProvider(tt,pp)))(
                    sum(kk in (K_PathsServiceProvider(tt,pp)*L_PathsUsingArc(ii,jj))) (
                        u_UsePrimaryPath(kk)
                    )
                )
                -l_Overlap(ss, tt)
                <= 1;
        end-if

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

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

writeln("\nSolving model...");
timetracker := timestamp;
maximize(XPRS_PRI, Total_Profits);

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

writeln("\nTotal Profits: ", getobjval);
writeln(
    "\nTotal Backup Costs: ",
    sum(ii in Nodes)(sum(jj in Nodes)(C_BackupCost(ii, jj)*getsol(l_Lambda(ii, jj))))
);

! for all served customers
forall(cc in Customers | getsol(y_Serve(cc)) > 0.001) do
    ! output served customer and generated profits for customer
    writeln(
        "\nCustomer ", cc, " (node ",cc,") is being served\n - R_Revenue: ",
        R_Revenue(cc)*getsol(y_Serve(cc))
    );
    ! for all services of customer
    forall(ss in S_ServicesForCustomer(cc)) do
        ! for the provider placement selected for service
        forall(pp in Providers | getsol(x_Placement(ss,pp)) > 0.001) do
            writeln(
                "  - Service ",ss," is placed at provider ",pp,
                "     - Availability req.: ", Y_AvailabilityReq(ss),
                "     - Exp. availability: ",
                (   ! calculate expected availability for mapping
                    sum(kk in K_PathsServiceProvider(ss,pp)) (
                        D_PathAvailability(kk)
                        *
                        (
                            getsol(u_UsePrimaryPath(kk))
                            +
                            getsol(v_UseBackupPath(kk))
                        )
                    )
                    -
                    ! subtract P(A)P(B|A) if backup path is chosen
                    sum(kk in K_PathsServiceProvider(ss,pp), bb in K_PathsServiceProvider(ss,pp))
                        D_CombinationAvailability(kk,bb)*getsol(o_UseCombination(kk,bb))
                )
            );
            forall(kk in K_PathsServiceProvider(ss,pp)) do
                if (getsol(u_UsePrimaryPath(kk)) > 0.001) then
                    writeln(
                        "     - Primary path: ", kk, " , cost: ",
                        getsol(u_UsePrimaryPath(kk))*C_PathCost(kk),
                        " (", getsol(u_UsePrimaryPath(kk))*100, " %)"
                    );
                end-if
            end-do
            forall(kk in K_PathsServiceProvider(ss,pp)) do
                if (getsol(v_UseBackupPath(kk)) > 0.001) then
                    writeln(
                        "     - Backup path: ", kk, " (",
                        getsol(v_UseBackupPath(kk))*100, " %)"
                    );
                end-if
            end-do
        end-do
    end-do
end-do

writeln("\nTotal backup usage");
writeln(
    strfmt("arc ",10),
    strfmt("reserved",10),
    strfmt("max req",10),
    strfmt("sum reqs*",10),
    strfmt("cost/bw",10),
    strfmt("paths",10)
);

declarations
    temp: real;
end-declarations

forall(ii in Nodes, jj in Nodes | exists(L_PathsUsingArc(ii,jj))) do
    ! Find the actual single maximal backup requirement on arc
    temp := 0.0;
    forall(ss in Services) do
        if(getsol(q_backupPerService(ii,jj,ss)) > temp) then
            temp := getsol(q_backupPerService(ii,jj,ss));
        end-if
    end-do
    ! Print information about the arc and backup reservations
    if(getsol(l_Lambda(ii,jj)) > 0.001) then
        write(
            strfmt("("+ ii+ ", "+ jj+ ")",10),
            strfmt(getsol(l_Lambda(ii,jj)), 10),
            strfmt(temp, 10),
            strfmt((sum(ss in Services)getsol(q_backupPerService(ii,jj,ss))), 10),
            strfmt(C_BackupCost(ii,jj), 10),
            "     "
        );
        forall(kk in L_PathsUsingArc(ii,jj) | getsol(v_UseBackupPath(kk)) > 0.001) do
            write(kk, ", ");
        end-do
        write("\n");
    end-if
end-do

end-model
```