

```
#####
# Optimization Model File
# HSA JCSG - DFAS
#
# Michael Bowes
# Revised Jul 28 2005 - Now reflects capacity cost
#####

##### Sites, Business Line Functions #####
set Sites;                      # List of DFAS Sites
set Functions;                  # List of functional work area
set Acct within Functions; # List of primary accounting functions
set Line within Functions; # Primary business line functions

#####
# Milval, Capacity, Requirements, etc
#####
param MV{Sites};          # Military values for Sites

param capac{Sites} >=0;      # Site capacities (Usable Sq Ft)
param safes{Sites};        # Add to capac to get full capacity
param capacity{s in Sites} := capac[s] + safes[s];
param total_capacity := sum {s in Sites} capacity[s];
                                # Total capacity (Usable Sq Ft)
param requirement{f in Functions};
                                # Staff requirements, by function
param useRate   := 160;       # Space use - 160 sf per capita

param centers{s in Sites, f in Line};
                                # Identify biggest current business line sites

param Mil{Sites};           # Sites on military installation
param ATFP{Sites};         # Off-base Sites that meet security needs (it's not really ATFP)
param Secure{s in Sites} := min(Mil[s],ATFP[s]);

param Cost{Sites};          # Cost per square foot for facilites support & lease

param AvgCost := sum{s in Sites}(capacity[s]*Cost[s])/total_capacity;

param minAssignLine;        # Ensure balance in two major line centers
param minCenterSpt;        # Ensures support in the major centers
param minAssignSpt;         # Assigns other support units in workable size

#####
# Parameters used to control the optimization
#
param rho_resource;        # Penalty on retained square footage
param rho_build;            # Penalty on new construction
param rho_expand;           # Penalty on takeover of available space
param rho_func;             # Penalty on number of functional locations open
                            # used to encourage concentration of business lines
param rho_move;             # Encourage locating where function is now (avoids
                            # excessive relocation and reduce risk of losing expertise)
param rho_secure;           # Discourage locating at sites with inadequate security

param Keep{Sites};          # Can be used to force open
param Close{Sites};         # Can be used to force closure

param Count;                 # used to generate alternative closure scenarios

#####
# Variables
#
var Open{Sites} binary >= 0;    # Track which sites are kept open

var OpenFunc{Sites, Functions} binary >= 0;
                                # Track whether a business line function is present at a site

var Assign{Sites,Functions} integer >=0; # Staff of each type assigned to a site
var Major{Sites, Line} binary >= 0;    # Track major centers for line functions
var MajorSite{Sites} binary >= 0;       # Track site with major centers

#####
# Special space needs
#####
param Vault;
param VaultDoD;
```

```

param SCIF;
param SCIFDoD;
param TrainArmy;
param TrainDoD;
param TrainFld;
param VTC;

var a_VTC{Sites} binary >=0;      # VTC needed if a line function present
var a_Vault{Sites} binary >=0;    # Vault 18701 needed if DoD Acct present
var a_VaultDoD{Sites} binary >=0; # Vault 2744 needed if other accounting present
var a_SCIF{Sites} binary >=0;     # Secure room 3150 needed if DoD Acct present
var a_SCIFDoD{Sites} binary >=0; # Secure room 1575 needed if other Acct prese
var a_TrainArmy{Sites} binary >=0;# Training room 17600 if Army accnt
var a_TrainDoD{Sites} binary >=0; # Training room 10000 if DoD accnt
var a_TrainFld{Sites} binary >=0; # Training room 8500 if field accnt
# Note that warehouse space is assumed to be available if needed

#####
# Parameters and variables related to expansion
#####
param maxExp{Sites} default 0;   # Extra space available at the site
param maxBuild{Sites} default 0;  # Potential size of new building

var expRes{Sites} >= 0;          # Expansion into extra space already available
var buildRes{Sites} >= 0;        # Newly constructed space

# OBJECTIVE FUNCTION
#####
maximize Military_Value :

sum {s in Sites} (MV[s]* Open[s])

- rho_resource*sum{s1 in Sites}
  Cost[s1]*(capacity[s1]*Open[s1] + expRes[s1])/(AvgCost*total_capacity)

- rho_expand*sum{s2 in Sites} expRes[s2]/total_capacity

- rho_build*sum{s3 in Sites} buildRes[s3]/total_capacity

- rho_func*sum{s4 in Sites, f4 in Line} (OpenFunc[s4,f4])/card(Sites))
  # This is applied to the primary business line functions. It favors specialization.

+ rho_move*sum{s5 in Sites, f5 in Line} (centers[s5,f5]*Assign[s5,f5]/requirement[f5])
  # Favor keeping line staff in sites that already have significant staff doing that work

- rho_secure*sum{s6 in Sites} (1-Secure[s6])/card(Sites)
  # Discourages assigning staff to Sites that are not secure
;

# CONSTRAINTS
#####

#1. Can be used to force Sites Open or Closed
subject to Force_open {a in Sites: Keep[a]==1}: Open[a] == 1;
subject to Force_close {a in Sites: Close[a]==1}: Open[a] == 0;

#2. Meet requirements - staff must be assigned space somewhere
subject to Meet_Requirements {f in Functions}:
  sum {s in Sites} Assign[s,f] = requirement[f];

#3. Resource use cannot exceed available capacity (including expansion)
subject to Resources_Available {s in Sites}:
  sum {f in Functions} (Assign[s,f]*useRate)
  + a_VTC[s]*VTC + a_Vault[s]*Vault + a_VaultDoD[s]*VaultDoD
  + a_SCIF[s]*SCIF + a_SCIFDoD[s]*SCIFDoD
  + a_TrainArmy[s]*TrainArmy + a_TrainDoD[s]*TrainDoD + a_TrainFld[s]*TrainFld
  <= (capacity[s]*Open[s] + expRes[s] + buildRes[s]);

#4a Expansion constraints - expansion to other open office space can't exceed availability
subject to Expansion_Space_Limits {s in Sites}:
  expRes[s] <= maxExp[s]*Open[s];

#4b Expansion constraints - new building can't exceed the given limit
subject to Expansion_Build_Limits {s in Sites}:
  buildRes[s] <= maxBuild[s]*Open[s];

#5 Do not assign functions to "closed" Sites.
subject to Functions_to_Open_Sites {f in Functions, s in Sites}:
  OpenFunc[s,f] <= Open[s];

```

```

#6. Do not assign functional staff to location without that function present
subject to Assign_to_Open {f in Functions, s in Sites}:
  Assign[s,f] <= OpenFunc[s,f]*requirement[f];

#7. Make sure unused Sites (no functions assigned) are forced closed !!!
subject to Close_Unused_Sites {s in Sites}:
  Open[s] <= sum {f in Functions} Assign[s,f];

#7a. Make sure unused functions (no staff of that type assigned) are forced to zero !!!
subject to Close_Unused_Functions {s in Sites, f in Functions}:
  OpenFunc[s,f] <= Assign[s,f]*requirement[f];

#8. Special space needs
#####
#8a  VTC space
subject to VTC_Space {s in Sites}:
  a_VTC[s] >= sum{f in Line} ( OpenFunc[s,f]/(card(Sites)*card(Line)) );

#8b  Vault space, if no DoD accounting
subject to Vault_Space1 {s in Sites}:
  a_Vault[s] >= sum{f in (Acct diff {'AcctDoD'})} ( OpenFunc[s,f]/(card(Sites)*card(Line)) )
    - OpenFunc[s,'AcctDoD'];

#8c  Vault space, if DoD accounting
subject to Vault_Space2 {s in Sites}:
  a_VaultDoD[s] >= OpenFunc[s,'AcctDoD']/ (card(Sites)*card(Line));

#8d  Vault space total
subject to Vault_Space3 {s in Sites}:
  a_Vault[s] + a_VaultDoD[s] <= 1;

#8e  SCIF space, if no DoD accounting
subject to SCIF_Space1 {s in Sites}:
  a_SCIF[s] >= sum{f in (Acct diff {'AcctDoD'})} ( OpenFunc[s,f]/(card(Sites)*card(Line)) )
    - OpenFunc[s,'AcctDoD'];

#8f  SCIF space, if DoD accounting
subject to SCIF_Space2 {s in Sites}:
  a_SCIFDoD[s] >= OpenFunc[s,'AcctDoD']/ (card(Sites)*card(Line));

#8g  SCIF space total
subject to SCIF_Space3 {s in Sites}:
  a_SCIF[s] + a_SCIFDoD[s] <= 1;

#8h  Training space, if Army accounting
subject to Train_Space1 {s in Sites}:
  a_TrainArmy[s] >= OpenFunc[s,'AcctArmy']/ (card(Sites)*card(Line));

#8i  Training space, if DoD accounting and no Army
subject to Train_Space2 {s in Sites}:
  a_TrainDoD[s] >= OpenFunc[s,'AcctDoD']/ (card(Sites)*card(Line))
    - OpenFunc[s,'AcctArmy'];

#8j  Training space, if Field accounting only
subject to Train_Space3 {s in Sites}:
  a_TrainFld[s] >= OpenFunc[s,'AcctFld']/ (card(Sites)*card(Line))
    - OpenFunc[s,'AcctDoD'] - OpenFunc[s,'AcctArmy'];

#8k  Training space total
subject to Train_Space4 {s in Sites}:
  a_TrainArmy[s] + a_TrainDoD[s] + a_TrainFld[s] <= 1;

#9. At least two locations for each major business lines
#####
subject to At_Least_2_Major_by_Line {f in Line}:
  sum {s in Sites} Major[s,f] >= 2;

subject to Not_Major_if_not_Open {s in Sites, f in Line}:
  OpenFunc[s,f] >= Major[s,f];

subject to MajorSite_if_MajorFunction {s in Sites}:
  MajorSite[s] >= sum{f in Line} Major[s,f]/card(Line);

subject to Balance_in_Majors {s in Sites, f in Line}:
  Assign[s,f] >= Major[s,f]*minAssignLine;
# Ensure enough balance in the 2 major centers so the second could take over for the first
# Original runs required at least 600 of the line staff in each major center

```

```

#10c HQ - maintain HQ with 149 people (the actual location of the HQ
# may change. It is to be considered outside the optimization. This simply keeps the group together)
#####
##### subject to Arlington_Assign:
Assign['Arlington_VA__', 'HQAr1'] == requirement['HQAr1'];

subject to Arlington_Open:
Open['Arlington_VA__'] == 1;

subject to Arlington_OpenFunc:
OpenFunc ['Arlington_VA__', 'HQAr1'] == 1;

subject to Arlington_Func2 {f in Functions: f <> 'HQAr1'}:
OpenFunc['Arlington_VA__', f] == 0;

subject to Arlington_Func3 {s in Sites: s <> 'Arlington_VA__'}:
OpenFunc[s, 'HQAr1'] == 0;

#11. Linkages
#####
#11a Assign WHS to location with DOD Accounting
subject to WHS {s in Sites}:
OpenFunc[s, 'WHS'] <= OpenFunc[s, 'AcctDoD'];

#11b Assign DoDEA. DTRA, DMPOS to location with Army Accounting
subject to DoDEA {s in Sites}:
OpenFunc[s, 'DoDEA'] <= OpenFunc[s, 'AcctArmy'];

subject to DTRA {s in Sites}:
OpenFunc[s, 'DTRA'] <= OpenFunc[s, 'AcctArmy'];

subject to DMPOS {s in Sites}:
OpenFunc[s, 'DMPOS'] <= OpenFunc[s, 'AcctArmy'];

#11c Army NAF accounting with Army accounting
subject to NAF {s in Sites}:
OpenFunc[s, 'NAFAcct'] <= OpenFunc[s, 'AcctArmy'];

#11d Some accounting if any Commercial Pay
subject to Acct_if_Commercial_Pay {s in Sites}:
sum {f in Acct} OpenFunc[s, f] >= OpenFunc[s, 'CmPay'];

#11e Some Acct people, if any CivPay
subject to Acct_if_MilCiv_Pay {s in Sites}:
sum {f in Acct} OpenFunc[s, f] >= OpenFunc[s, 'MCPay'];

#12. Distribute admin
#####
#Admin only if line function (except for HQ)
subject to Support_only_if_Line {s in Sites}:
OpenFunc[s, 'CorpSpt'] <= sum {f in Line} OpenFunc[s, f];

# Some Admin if Major Line
subject to Assign_Support_if_Major {s in Sites}:
Assign[s, 'CorpSpt'] >= sum {f in Line} Major[s, f] / card(Line);

# Min Support team assignments - 300 to major centers, at least 75 to others (if any assigned)
subject to Min_Assign_Support {s in Sites}:
Assign[s, 'CorpSpt'] >= OpenFunc[s, 'CorpSpt']*minAssignSpt;

subject to Min_Assign_Major_Support {s in Sites}:
Assign[s, 'CorpSpt'] >= MajorSite[s]*minCenterSpt;

# Max assignments
subject to Less_than_Line {s in Sites}:
Assign[s, 'CorpSpt'] <= sum {f1 in Line} Assign[s, f1];

#13. Alternatives Generation
#####
subject to Number_of_Sites_Open :
sum {s in Sites} Open[s] <= card(Sites) - Count;

```

```

#####
# Optimization Run File
# HSA JCSG - DFAS
#
#
# Michael Bowes
# Revised 28 Jul 2005
# Allows forced closures or opens
#####

reset;

model dfas.mod;
data dfas.dat;
option cplex_options 'timelimit=240 solutionlim=500000 mipgap=.0000001 integrality=0';

param KpCls symbolic default "";
param KpCls1 symbolic default "";
param KpCls2 symbolic default "";
param KpCls3 symbolic default "";
param KpCls4 symbolic default "";
param KpCls5 symbolic default "";

#### ENTER RUN PARAMETERS
#####
let rho_resource := 5 ; # penalty on total capacity retained
let rho_build := 20 ; # penalty on new construction
let rho_expand := 15 ; # penalty on moving into available space nearby (renovation)
let rho_func := .01 ; # favor concentration
let rho_move := .01 ; # avoid moves, favor existing concentration
let rho_secure := 0 ; # penalty on sites with security issues

#Set Closure or Openings
#let Close[{"Denver_CO____"}] := 1; let KpCls1 := "_noDenv"; # Close Denver
#let Close[{"Cleveland_OH___"}] := 1; let KpCls2 := "_noClev"; # Close Cleveland
#let Keep[{"Limestone_ME___"}] := 1; let KpCls3 := "_kpLime"; # Keep Limestone
#let Keep[{"Rome_NY_____"}] := 1; let KpCls4 := "_kpRome"; # Keep Rome

let minAssignLine := 500;
let minCenterSpt := 300;
let minAssignSpt := 75;

# Upper and lower limits on number of closure alternatives to generate
param C_up; let C_up := 22; # Retain 4 = 26-22 sites sites (3 plus HQ)
param C_lo; let C_lo := 20; # Retain 6 = 26-20 sites (6 plus HQ)
#####

param i_c; # used to track passes through loop

#####
#Build Output file name
param RUN symbolic;
let RUN := "(r_" & rho_resource & ",b_" & rho_build & ",e_" & rho_expand &
           ",f_" & rho_func & ",m_" & rho_move & ",s_" & rho_secure & ")";

param OutFile symbolic;
let KpCls := KpCls1 & KpCls2 & KpCls3 & KpCls4;
let OutFile := "dfas" & KpCls & RUN & ".out";

#To store for results
set METRICS := {"_AvgMV", "%CapRtnd", "_Build", "_Renovate"};
set OUTPUTS := Sites union METRICS;

param Results{OUTPUTS, 1..10} ; # The 10 may have to be changed if loop is bigger
for {s in OUTPUTS, i in (1..10)} let Results[s,i] := -0.1; # initialize
#####

#### Here's the main solution loop - Count is incremented on each pass
#####
let Count := C_up;
let i_c := 1; # Tracks the number of passes through loop

repeat {
solve;

printf "===== \n" > (OutFile);
printf " NEW RUN \n" > (OutFile);

```

```
printf "=====\\n" > (OutFile);

#####
# BEGIN IF statement - output for feasible solutions only
if (solve_result_num<104 or solve_result_num=422) then {
#####
printf "\\n" > (OutFile);

printf "NUMBER OF SITES RETAINED = %2.0f of %2.0f \\n",
      sum {s in Sites} Open[s], card(Sites)
    > (OutFile);
printf "\\n" > (OutFile);

printf "Penalties used in the run \\n" > (OutFile);
printf ===== \\n" > (OutFile);
printf "          Penalty      ObjValue \\n" > (OutFile);
printf ===== \\n" > (OutFile);
printf "Penalty on Capacity   = %13.3f      %9.2f \\n", rho_resource,
      rho_resource* sum {s in Sites} (capacity[s]*Open[s] + expRes[s] + buildRes[s])/(total_capacity)
    > (OutFile);
printf "Penalty on Expansion  = %13.3f      %9.2f \\n", rho_expand,
      rho_expand * sum {s in Sites} expRes[s]/(total_capacity)
    > (OutFile);
printf "Penalty on Building   = %13.3f      %9.2f \\n", rho_build,
      rho_build * sum {s in Sites} buildRes[s]/(total_capacity)
    > (OutFile);
printf "Reward Specialization = %13.3f      %9.2f \\n", rho_func,
      rho_func * sum {s in Sites, f in Line } OpenFunc[s,f]/(card(Sites))
    > (OutFile);
printf "Penalize Moving Staff = %13.3f      %9.2f \\n", rho_move,
      rho_move * sum {s in Sites, f in Line}
        (centers[s,f]*Assign[s,f]/requirement[f])
    > (OutFile);
printf "Penalty on non-secure = %13.3f      %9.2f \\n", rho_secure,
      rho_secure * sum {s in Sites} (Secure[s]/card(Sites))
    > (OutFile);
printf ===== \\n" > (OutFile);

printf "Military Value: Total MV = %6.3f  Average MV = %6.3f \\n",
      sum {s in Sites} MV[s]*Open[s],
      (sum {s in Sites} MV[s]*Open[s])/(sum {s in Sites} Open[s])
    > (OutFile);

printf "Overall Objective Function Value: = %9.3f \\n", Military_Value > (OutFile);
printf "\\n" > (OutFile);
printf "\\n" > (OutFile);

printf "Overall Capacity: Initial = %7.0f  Final = %7.0f  Retained = %5.1f(%%) \\n",
      total_capacity, sum {s in Sites} (capacity[s]*Open[s] + expRes[s] + buildRes[s]),
      sum {s in Sites} 100*(capacity[s]*Open[s] + expRes[s] + buildRes[s])/total_capacity
    > (OutFile);

printf "Expansion to available space = %7.0f  New construction = %7.0f \\n",
      sum {s in Sites} expRes[s], sum {s in Sites} buildRes[s]
    > (OutFile);
printf "\\n" > (OutFile);

printf "\\n" > (OutFile);

printf "Site Results (1 identifies retained sites) \\n" > (OutFile);

display Open, MV > (OutFile);;
display Open, MV;

printf "\\n" > (OutFile);
printf "           Initial      Take      Build      Final |  Staff  Special      Total \\n" > (OutFile);
printf "           Capacity     Space     Space     Capacity |  Use    Use      Used \\n" > (OutFile);
printf "           ===== \\n" > (OutFile);

for {s in Sites}
  { printf "%14s  %8.0f %8.0f %8.0f %9.0f %8.0f %8.0f %8.0f \\n",
    s, capacity[s], expRes[s], buildRes[s], capacity[s]*Open[s] + expRes[s] + buildRes[s],
    sum{f in Functions} Assign[s,f]*useRate,
    (Vault*a_Vault[s] + VaultDoD*a_VaultDoD[s] + SCIF*a_SCIF[s] + SCIFDoD*a_SCIFDoD[s] +
    TrainArmy*a_TrainArmy[s] + TrainDoD*a_TrainDoD[s] + TrainFld*a_TrainFld[s] +
    VTC*a_VTC[s] ),
    (Vault*a_Vault[s] + VaultDoD*a_VaultDoD[s] + SCIF*a_SCIF[s] + SCIFDoD*a_SCIFDoD[s] +
    TrainArmy*a_TrainArmy[s] + TrainDoD*a_TrainDoD[s] + TrainFld*a_TrainFld[s] +
    VTC*a_VTC[s] )}
```

```

TrainArmy*a_TrainArmy[s] + TrainDoD*a_TrainDoD[s] + TrainFld*a_TrainFld[s] +
VTC*a_VTC[s] + sum{f in Functions} Assign[s,f]*useRate)

    > (OutFile);
};

printf "\n" > (OutFile);
printf "          Acct    Acct    Acct    Comm      MC      NAF      Def     Sppt | Total \n" > (OutFile);
printf "          Field   DoD     Army     Pay       Pay     Acct   Agncy   Staff | Staff \n" > (OutFile);
printf "===== \n" > (OutFile);

for {s in Sites}
{ printf "%14s %6.0f %6.0f %6.0f %6.0f %6.0f %6.0f | %6.0f \n",
  s, Assign[s,'AcctFld'], Assign[s,'AcctDoD'], Assign[s,'AcctArmy'], Assign[s,'CmPay'],
  Assign[s,'MCPay'],
  Assign[s,'NAFAcct'], Assign[s,'DMPOS'] + Assign[s,'DoDEA'] + Assign[s,'DTRA'] + Assign[s,'WHS'],
  Assign[s,'CorpSpt'] + Assign[s,'HQAr1'],
  sum {f in Functions} Assign[s,f]
  > (OutFile);
};

for {s in Sites} let Results[s,i_c] := Open[s];

let Results["_AvgMV",i_c] :=      (sum {s in Sites} MV[s]*Open[s])/(sum {s in Sites} Open[s]) ;
let Results["_CapRtnd",i_c] :=  sum {s in Sites} 100*(capacity[s]*Open[s] + expRes[s] + buildRes[s])/total_capacity ;
let Results["_Build",i_c] :=    sum {s in Sites} buildRes[s];
let Results["_Renovate",i_c] :=  sum {s in Sites} expRes[s] ;

#####
}

else
printf "Attempted to retain %2.0f of %2.0f. The run was infeasible\n",
card(Sites) - Count, card(Sites) > (OutFile);

# This ends the if statement on output for feasible solutions
#####

printf "===== \n\n\n" > (OutFile);

let Count := Count - 1;
let i_c := i_c + 1;

} while Count >= C_lo;      # Ends the repeat Count

printf "\n" > (OutFile);
display Results > (OutFile);
printf "Results that show as -0.1 can be discarded \n" > (OutFile);
printf "\n" > (OutFile);

close (OutFile);
;

```

DCN: 11874