APPENDIX J **Code listing for weapon system optimization algorithms.**

```
//        +++++  Main Weapon System Optimization  +++++

  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/header.h"
//
//                Global debug
//
  ofstream G_debug_out("Z_Global_debug" );
  int G_debug_cntl = FALSE;

  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzSet.h"
  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzHedge.h"
  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzVariable.h"
  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzRule.h"
  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzIssue.h"
  #include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzDecision.h"

  #include "MES_random.h"
  #include "MES_ES_opt.h"
  #include "MES_Weapon.h"
  #include "MES_Mission.h"
  #include "MES_Targeting.h"
  #include "MES_WS_Opt.h"
//                ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    int main( int argc, char *argv[])

  {
  cout << "+++++++++ Begin  Assessment +++++++++\n"<<flush;
  ifstream weap_in( argv[2] );
  cout << "Input data file : " << argv[2] << "\n" << flush ;
  WeaponOpt NuSys;
  NuSys.Load_Data( weap_in ) ;
//
//                Set a new output file based on info in argv[1]
//
  strcpy( NuSys.scenario[0].Results_file, argv[1] ) ;
  NuSys.WS_Opt();

  for( int k=0; k<NuSys.Ivec[NuSys.total_optimize]; k++ )
    cout << "\n" << NuSys.soln.es_mean[0][k] ;
  cout << "\n";
  cout << "+++++++++ End of Assessment +++++++++\n";
 } ;      // Block: 14
```

// +++++ Weapon System Allocation (Targeting) +++++

// ++

//

// **Weapon System Allocation**

//

// This class defines the allocation activities associated with weapon
// system optimization or for simple weapon to target allocation
// activities. This constitutes one of the three major calculational
// activities which comprise weapon system optimization but also a
// stand-alone sub function.

//

// The algorithm must be able to treat multiple scenarios in support of
// the system optimization activities. For simple allocations this capa
// bility is not required.

//

// **References**:

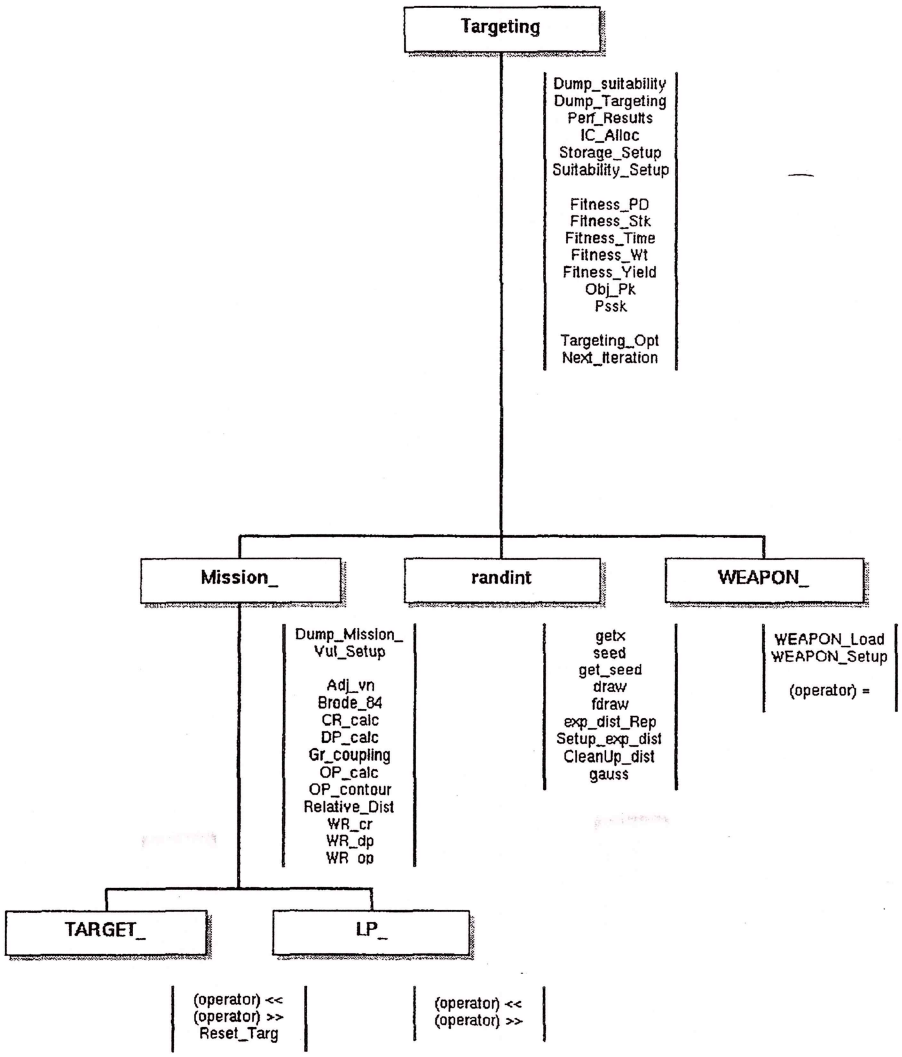
//

// Defense Intelligence Agency, "Physical Vulnerability Handbook for
// NuclearWeapons",OGA-2800-23-92
// Cox,E., *The Fuzzy Systems Handbook,* AP Professional
// (a div of Harcourt Brace & Co.), 1994, ISBN 0-12-194270-8
// Klir, G.J., Folger, T.A., Fuzzy Sets, Uncertainty & Informtion,
// Prentice Hall, 1988, ISBN 0-13-345984-5
// R.R. Yager; "Multiple objective decision-making using fuzzy sets",
// Iona Colledge, Int J Man-Machine Studies, 1977 9,375-382.
// JSTPS TR 91-1, "PDCALC A Computer Program For The
// Probability Of Damage Calculations", 1991.
// H. Brode, S. Speicher, "Air Blast From Nuclear Bursts-analytic
// Approximations", Pacific-Sierra Research Corp.,
// PSR Report 1419-1.
// S. Speicher, H. Brode, "Extremely High Overpressure Analytic
// Expression For Burst Height, Range And Time - Over An Ideal
// Surface", Pacific-Sierra Research Corp., PSR Note 611
// Capt. G.Martelle, "3DPD HANDBOOK, Formulas and Algorithms for
// Computing The Probability of Damage of New Generation
// Nuclear Weapons",AF Operational Test & Evaluation Center.

//

// **Code Structure :**

//



//

//

//

// **ProgramConstruction:**

//

// The driver for the allocation algorithms is simply a main program
// in which an object is declared, data is loaded, and the optimization
// is performed. When this algorithm is used in conjunction with the
// weapon system optimization, control is treated in the **WeaponOpt class**. IN stand-alone mode an example of a driver for the
// optimization follows:

//

```
//    ifstream allo_in( argv[1] );
//    ofstream d_out("Validate1.dat");
//    Targeting tst_allo;
//    allo_in >> tst_allo;
//    tst_allo.IC_Alloc();
```

```
//          tst_allo.Dump_suitability( d_out );
//          tst_allo.Targeting_Opt( );
//
//      The first 2 statements define the input and output files.  The third
//      line defines the Targeting class object, we then load the data via
//      the overloaded operator >>.  The function call tst_allo.IC_Alloc()
//      is the call which presets and defines information for use in the
//      allocation effort..  Finally, pre-processed information is dumped to
//      file and the optimization algorithm is fired.
//
//      Input Format: (typical dataset)
//
//          scenario1
//          Weapons 2 0
//          Yield 333 Inventory 75 Options 0 2  Response 1.0 eoi
//          Yield 111 Inventory 225 Options 4 0 CEPs 450 400 750 350
//              Response 1.0 eoi
//          Dump
//          Convergence 50
//          Acceptance 1.0
//
//          Targets Targets 250
//          608 41L8 0.7317663603 61.93145  44.1054146  46.25955643
//          465 37M6 0.5177064626 58.98407 101.7777439 179.97161811
//          500 37M6 0.4310704167 51.18029 157.6116195 229.35555875
//          ...
//          eoi
//
//      Input Keywords include:
//      Weapons, Yield, Inventory, Options , CEPs, Response,
//      Convergence, ObjectivePD, Mutation, Seed, Dump, Debug,
//      & eoi for termination.
//
//      Targeting Member Functions:
//
//      Setup, input and output of information:
//      IC_Alloc, Storage_Setup, Suitability_Setup
//      Dump_suitability, Dump_Targeting,
//
//      Fitness and Performance evaluations are performed in:
//      Fitness, Obj_Time, Obj_Yield, Obj_Pk, Perf_Results, Pssk
//
//      Optimization evaluations :
//      Next_Iteration, Targeting_Opt
//
//      ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

class Targeting

```
 {
  public:
//
//              Define addiitonal model parameters:
//
  int exists;
  char* obj_name;
  char* Results_file ;

  long RN_seed;
  int DEBUG, DUMP, FUZZY_TACTIC ;
  int APPEND_rslts;
  float acceptance, Mission_acceptance, Mission_fitness;

  int max_targets ;
  int base_line_WH, new_WH, total_WH ;
  int total_inventory, no_option;
  int Restart_iter, iterations ;
  float pd_objective, F1_form_factor, beta[4];
  float Scale_Inv ;

  WEAPON_ *sys;
  Mission_ startX;
  randint random;
  int **opt_index;
  int *allo_vec;
  float **suitability;
  FzDecision tactic;

  Targeting();                          // Constructor
//
//              overload the input and output operators
//
  friend istream& operator>>( istream& in,  Targeting&  );
  friend ostream& operator<<( ostream& out, Targeting&  );
//
//              Define addiitonal member functions:
//
  float Dist_Gamma( float, float, float ) ;
  float Dist_Sigmoid( float, float, float, float ) ;
  void Dump_suitability( );
  void Dump_Targeting( ostream& out );
```

```
    void IC_Alloc( );
    float Fitness(int, int, float*, float, int );// Goal Fitness Function
    float Next_Iteration( );
    float Obj_HOB( int, int, int );
    float Obj_Pk( int, float, float );
    float Obj_Time( int, int );          // Time Urgency Fitness Function
    float Obj_Yield( float );            // Minimum Yield Fitness Function
    void Perf_Results( );
    float Pssk( int, float, float );     // Determine the probability of "single shot" kill
    void Storage_Setup( );
    void Suitability_Setup( );
    float Targeting_Opt(  );             // Opt. The Targeting Of Weapons To Targets

};    // Block: 15
```

Targeting::Targeting( )

```
{
  obj_name = new char[10];
  Results_file = new char[ 25 ];
//
//              Define default conditions
//
  exists=FALSE;
  strcpy( obj_name,"None" );
  strcpy( Results_file,"Zresults" );
  RN_seed=1111111111;
  DEBUG=FALSE;
  DUMP=FALSE;
  FUZZY_TACTIC= FALSE ;
  APPEND_rslts=TRUE;
  acceptance=1.25;

  Restart_iter=0;
  iterations=100;

  max_targets=1;
  base_line_WH=1;
  new_WH=0 ;
  total_WH= base_line_WH+new_WH ;
  no_option=0;

  pd_objective=0.75;
  F1_form_factor=0.025;
  Scale_Inv = 1.0 ;
//
```

```
// Com 7        Beta Is The Decision Model Weighting Coeficients.
//
  for( int i=0; i<4; i++ )
    beta[i]=1.0;
  } ;      // Block: 16
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

istream& operator>>(istream& in, Targeting& adum)

```
  {
   char tmp_in[15];
   char tmp_file[15];
   adum.exists=TRUE;
   int tmp;
   in >> adum.obj_name;
   in >> tmp_in;
   while( strcmp( tmp_in,"eoi" ) !=0 )
   {
//
//              Output Of The Keywords Read For Targeting
//
   cout << "----- Targeting Keyword : " << tmp_in << "\n" << flush;
     if( strcmp( tmp_in,"Targets" ) == 0 )
       {
       in >> adum.startX;
       adum.max_targets = adum.startX.max_targets ;
       }
     else if( strcmp( tmp_in,"Weapons" ) == 0 )
       {
//
//              First load baselines and other Targeting parameters
//              Next load the CEP values and associated stockpile assignments
//
//              Note:  for input optimization, sensitivity studies we are allowing
//              for a scaleing of the inventories.  Thsi can be done is ScaledInv
//              is loaded prior to the weapons.
//
       in >> adum.base_line_WH >> adum.new_WH;
       adum.total_WH = adum.base_line_WH + adum.new_WH;
       adum.sys= new WEAPON_[ adum.total_WH+1 ];
       for( int k=1; k<= adum.total_WH; k++ )
         {
         adum.sys[ k ].WEAPON_Load( in ) ;
         adum.sys[ k ].inventory = int( float( adum.sys[ k ].inventory ) *
           adum.Scale_Inv ) ;
         }
```

```
        }
      else if( strcmp( tmp_in,"ScaledInv" ) == 0 )
        {
        in >> adum.Scale_Inv ;
        }
      else if( strcmp( tmp_in,"Convergence" ) == 0 )
        in >> adum.iterations;
      else if( strcmp( tmp_in,"FuzzyTactics" ) == 0 )
        {
        in >> tmp_in;
        adum.tactic.Load_Model( tmp_in );
        adum.FUZZY_TACTIC= TRUE ;
        }
      else if( strcmp( tmp_in,"Acceptance" ) == 0 )
        in >> adum.acceptance;
      else if( strcmp( tmp_in,"Debug" ) == 0 )
        adum.DEBUG = TRUE ;
      else if( strcmp( tmp_in,"Dump" ) == 0 )
        adum.DUMP = TRUE ;
      else if( strcmp( tmp_in,"ObjectivePD" ) == 0 )
//
//              Objective Mission Performance
//
        in >> adum.pd_objective;
      else if( strcmp( tmp_in,"PD_fitness" ) == 0 )
//
//              Factor Defining The Degree Of Acceptance Of The Objective Pd
//
        in >> adum.F1_form_factor;
      else
 {
        cerr << "Error: (Targeting Input) unaceptable input option : " << tmp_in
        << "\n" << flush;
 } ;    // Block: 17
  in >> tmp_in;
 } ;    // Block: 18
//
//              Need To Initialize storage
//
  cout << "----- Targeting :: Input Complete:  Storage_Setup  next\n" << flush;
  adum.Storage_Setup();
  cout << "----- Targeting :: Storage_Setup Complete\n" << flush;
  return in;
 } ;    // Block: 19
//              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ostream& operator<<(ostream& out, Targeting& A)

```
 {
//
//              provide coding to represent the ouput desired for this object
//
  out << "\n\n" << A.obj_name << "\n";

  out << "Objective Pd  " <<  A.pd_objective << "\n";
  out << "Targets " << A.max_targets << "\n";
  out << "Weapons " << A.total_WH << "\n";
  for(  int k=1; k<= A.total_WH; k++ )
    out << A.sys[k] <<" "<< "\n";
  out << "Convergence  " <<  A.iterations << "\n";
  return out;
 } ;     // Block: 20
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Dist_Gamma(float alp, float bet, float x)

```
 {
//
//              This Routine returns values associated with a gamma distribution
//
  float rslt ;
  float F5, Fnorm, mode_val ;
  float gam_alf= 1.0;
//
//              Evaluate for the probabiity given a timing objective.
//
  for( int alf=alp-1; alf>0; alf-- )
    gam_alf *= alf;
  F5= pow( bet, -alp) * pow( x, (alp-1)) * exp( -x/bet ) / gam_alf ;
//
//              Need To Normalize The Timing Fitness To Mitigate Gamma
//              Distribution Effects.
//
  mode_val= bet*(alp-1) ;
  Fnorm= pow( bet, -alp) * pow( mode_val, (alp-1)) *
    exp( -mode_val/bet ) / gam_alf ;
  rslt = F5/Fnorm;
  return (rslt);
 } ;     // Block: 21
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Dist_Sigmoid(float alp, float bet, float gam,

```
    float x )

  {
//
//              This Routine returns values associated with a sigmoid distribution.
//              We will assume that a decreasing sigmoid ( IC type response )
//              will utilize a standard form.  An increasing sigmoid begins with a
//              value of zero progresses through the inflection point to a value
//              of one. ( questionable assumption, a PI function might be better ).
//
  float rslt ;
//
//              Is It An Increasing Or Decreasing Sigmoid ?
//
//              beta < 0  =>  Decreasing !!
//              beta > 0  =>  Increasing !!
//
  rslt = 1.0 / ( 1.0 + exp( -(x-alp)/bet ) ) ;
//
//              Provide for a cutoff
//
  if( x >= gam )
    rslt=0.005 ;
  return (rslt);
  } ;     // Block: 22
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    void Targeting::Dump_suitability( )

  {
//
//              This Routine Dumps Information Associated With The Allocation
//              Of Weapons To Targets..
//

  G_debug_out <<"\n Suitability Pre-Calculations \n"<< " ----- total warheads : "
    << total_WH << "\n";
  for( int k=0; k<max_targets; k++ )
    {
    G_debug_out << startX.targs[k].vntk << "  " << startX.targs[k].catcode << " : ";
    for( int sys_index=0; sys_index<=total_WH; sys_index++ )
      G_debug_out <<" ( "<<opt_index[k][sys_index]<< " "
      <<suitability[k][sys_index]<< " ) ";
    G_debug_out << "\n";
    }
  G_debug_out <<"\nGoal Acceptance ( Unconstrained stockpiles ) :  "
```

```
  <<Mission_acceptance<<" ( Acceptance level = "<<acceptance<< " )\n";
  G_debug_out <<"Goal Fitness ( Unconstrained stockpiles ) : "
  <<Mission_fitness<<"\n";
  return ;
 } ;     // Block: 23
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Targeting::Dump_Targeting(ostream& out)

```
 {
//
//              This Routine Dumps Information Associated With The Targeting
//              Object Setup.
//
  out << "----  ::  Targeting Dump (bgn)  ::  ---- " << obj_name << "\n";
  if( DUMP)
  {
    out << "----  ::  Targeting Dump (---)  ::  ---- "
      << "Storage Parameters : (max_targets, xxx_WH(3), total_inventory) \n" ;
      out << "----  ::  Targeting Dump (---)  ::  ---- " << max_targets << " "
      << base_line_WH << "\t" << new_WH << "\t" << total_WH << "\t"
      << total_inventory << "\n";

    out << "----  ::  Targeting Dump (---)  ::  ---- Control Parameters :  "
      << "(DEBUG, APPEND_rslts, Restart_iter, iterations, RN_seed) \n"
      << "----  ::  Targeting Dump (---)  ::  ---- " << DEBUG << "\t"
      << APPEND_rslts << "\t" << Restart_iter << "\t" << iterations << "\t"
      << RN_seed << "\n";

    out << "----  ::  Targeting Dump (---)  ::  ---- Convergence Parameters : "
      << "(acceptance, Mission_acceptance, Mission_fitness,pd_objective)\n" ;
    out << "----  ::  Targeting Dump (---)  ::  ---- "
      << acceptance<< "\t" << Mission_acceptance << "\t" << Mission_fitness
      << "\t" << pd_objective << "\n" ;
    out << "----  ::  Targeting Dump (---)  ::  ---- Convergence Parameters : "
      << "(F1_form_factor, beta[3])  \n" ;
    out << "----  ::  Targeting Dump (---)  ::  ---- " << F1_form_factor << "\t"
      << beta[0] << "\t" << beta[1] << "\t" << beta[2] << "\t" << "\n" ;

    out << "----  ::  Targeting Dump (---)  ::  ---- Weapon Parameters : \n";
      for( int k=0; k<= total_WH; k++ )
        out << sys[k] ;
 }

 out << "----  ::  Targeting Dump (---)  ::  ---- Mission Parameters : \n";
 startX.Dump_Mission_( out );
```

```
  out << "----  ::  Targeting Dump (end)  ::  ---- \n";
  return ;
 } ;     // Block: 24
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Targeting::IC_Alloc( )

```
 {
//
//              This Routine Provides Setups For The Targeting Problem.
//
  int k_option;
//
//              The First Step Of The Process Is To Use The Fuzzy "Tactical" Model
//              To Establish Target Priorities And Strategic Time Line Criteria.
//
//              Model tracking information is stored in file "Z_tactics"
//              The objective is to loop through each target in the scenario and
//              estimate the "weight" parameter and the "time urgency" parameter
//              associated with each target.  Run_model() must accept the
//              parameters and transfer the results for restorage in the appropriate
//              target information.
//
  if( FUZZY_TACTIC )
    {
    float in[10];
    float out[10];
    for( int fz=0; fz<max_targets; fz++ )
      {
      in[0]= startX.targs[fz].lethality;
      in[1]= startX.targs[fz].mission;
      in[2]= startX.targs[fz].function;

      tactic.Assess_Issues( in,out );

      startX.targs[fz].Wt = out[0];
      startX.targs[fz].Tm_obj = out[1];
      startX.targs[fz].HOB_obj = out[2];
      }
    }
//
//              Setup An Experimental Distribution To Identify What The "Weight"
//              Cutoff Will Be.  (Target Rich Environments)
//
//
```

```
//              Identify Total Warhead Inventories
//
  int total_stockpile=0 ;
  for( int k2=1; k2<=total_WH; k2++ )
    total_stockpile += sys[k2].inventory ;
  total_inventory= total_stockpile;
//
//              Note:  space allocation for tmp_wt should be the max of either
//              max_targets or total_inventory not the sum.  The sum is easier to
//              execute and will not degrade performance.
//
  float *tmp_wt;
  tmp_wt= new float[ max_targets+total_inventory ];
  for( int fz=0; fz<max_targets; fz++ )
    tmp_wt[fz]= startX.targs[fz].Wt ;
  random.Sort( tmp_wt, max_targets );
  float ratio, wt_cutoff ;
  ratio = float( total_inventory )/float( max_targets ) ;
  if( ratio >=1.0 )
    ratio= 1.0 ;
  wt_cutoff= tmp_wt[ total_inventory ] ;
//
//              Create An Experimental Distribution For Defining Initial Targetings
//
  int *tmp_hist;
  tmp_hist= new int[ total_inventory ];
  int indx=0;
  for( int m=1; m<=total_WH; m++ )
    for( int n=0; n<sys[ m ].inventory; n++ )
      {
      tmp_hist[indx]=m;
      indx ++;
      }
  random.Setup_exp_dist( tmp_hist, total_inventory );
//
//              Set The Initial Targetings
//
    for( int k=0; k<max_targets; k++ )
      {
      if( startX.targs[k].Wt  >  wt_cutoff )
        {
        k_option= int( random.exp_dist( ) ) ;
        allo_vec[k] = k_option;
        }
      else
        allo_vec[k] = no_option;
```

```
      }
//
//              Next Calculat The Suitabilities Of Potential Soutions
//
  Suitability_Setup();
//
//              Cleanup storage
//
  delete tmp_hist;
  return ;
 } ;     // Block: 25
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Fitness(int targ_indx, int sys_indx, float *cep,

float yld, int fz_opts)

```
 {
//
//              This routine provides the fitness based on goal or mission targets
//              damage expectancy.  Too little or too much is not a good solution.
//
  float Pk, F0,F1,F2,F3,F4, Fsum ;
  int indx=0;
  float max_pk=0.0;
  float min_pk=99999.0;
  int indx_cep=0;
  float max_cep=0.0;
  for( int k0=0; k0< fz_opts; k0++ )
    if( cep[k0] >= max_cep )
      {
      max_cep= cep[k0];
      indx_cep=k0;
      }
//
//              Assess System response time Suitability
//
  F1= beta[1]*Obj_Time( targ_indx, sys_indx );
//
//              Assess Yield Suitability
//
  F2= beta[2]*Obj_Yield( yld );
//
//              Assess Probability Of Kill Suitability
//
  for( int k=0; k< fz_opts; k++ )
```

```
    {
    F0 = beta[0]*Obj_Pk( targ_indx, cep[k], yld );
    F3 = beta[3]*Obj_HOB( targ_indx, sys_indx, k );
    F4 = sys[sys_indx].relia[ k ] ;
//
//              Note:  Discovered that some fitnesses are true trades, Pk vs yield
//              while others are critical.  Trades allow summation while the critical
//              conditions should be a product type total fitness.
//
  Fsum= F1*F4*(F0+F2+F3) ;

    if( Fsum <min_pk )
      min_pk= Fsum;
    if( Fsum > max_pk )
      {
      max_pk=Fsum;
      indx= k;
      }
    if( ( max_pk-min_pk) <= 0.0001 )
      indx= indx_cep;
    };
  opt_index[ targ_indx ][sys_indx] = indx;
  return( max_pk );
 } ;     // Block: 26
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Next_Iteration( )

```
 {
//
//              This Routine Performs Series Of Iterations In An Effort To Relax
//              The Problem To A Near Optimal Solution.
//
  int *draw_from;
  draw_from= new int[ max_targets ];
  int indx= 0;
  int m ;
  for( int df=0; df<max_targets; df++ )
    {
    m= allo_vec[df];
    if( suitability[df][m] <= acceptance )
      {
      draw_from[ indx ] = df;
      indx++;
      }
    }
```

```
//
//              Perform Statistical Modifications To The Allocation Vector
//
  int loc1, loc2, loc3, tmp, k_option ;
  float sOr, sNr, sO, sN, pr2 ;
  for( int k=0; k< indx; k++ )
    {
    loc1=draw_from[ k ];
    pr2 = random.fdraw();
    loc2 = int( pr2*indx  );
    loc3= draw_from[ loc2 ];
    k_option= 0;
//
//              Check To See If Exchange or Replacement Is A Gain
//
//              NOTE:  Using average suitabilities insures that an exchange is
//              better overall.
//
    sO= ( suitability[ loc1 ][ allo_vec[ loc1 ] ] +
      suitability[ loc3 ][ allo_vec[ loc3 ] ] )/2.0;
    if( random.e_siz > 1 )
      {
      k_option= int( random.exp_dist( ) ) ;
      sN= ( suitability[ loc1 ][ k_option ] +
        suitability[ loc3 ][ allo_vec[ loc3 ] ] )/2.0;
      }
    else
      {
      k_option= 0;
      sN= 0.0;
      }
    sNr= ( suitability[ loc1 ][ allo_vec[ loc3 ] ] +
      suitability[ loc3 ][ allo_vec[ loc1 ] ] )/2.0;
//
//              Is There An Improvement ??  Then Which Is The Best Exchange
//              Or Replacement
//
    if( sO < sNr || sO < sN )
      {
      if( sNr >= sN )
        {
        tmp= allo_vec[ loc1 ];
        allo_vec[ loc1 ]= allo_vec[ loc3 ];
        allo_vec[ loc3 ]= tmp ;
        }
      else
```

```
        {
        tmp= allo_vec[loc1];
        allo_vec[loc1] = k_option;
        random.exp_dist_Rep( tmp );
        k_option=0;
        }
      }
    if( k_option != 0 )
      random.exp_dist_Rep( k_option );
    }
  float mission_fit =0;
  indx= 0;
  for( df=0; df<max_targets; df++ )
    {
    m= allo_vec[df];
    if( suitability[df][m] >= acceptance )
      indx++;
    }
  mission_fit = float( indx )/ float( max_targets ) ;
//
//              Cleanup Storage
//
  delete draw_from;
  return mission_fit ;
 } ;     // Block: 27
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Obj_HOB(int targ_indx, int sys_indx, int opt_indx)

```
 {
//
//              Simple Evaluation Of System Performance For A Weapon Concept
//              Against a Target (targ_indx ):  HOB constraints are evaluated.
//
  float FzHt, TgHt, F1 ;
  float F2_form_factor ;
  FzHt=sys[sys_indx].hobs[ opt_indx ] ;
  TgHt= startX.targs[ targ_indx ].HOB_obj ;
  F2_form_factor= 0.5*FzHt ;
  if( FzHt > 99998.0 && FzHt <100000.0 )
    F1 = 1.0;
  else
    F1 = exp(-(FzHt -TgHt)*(FzHt -TgHt)/F2_form_factor ) ;
  return( F1 );
 } ;     // Block: 28
```

```
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

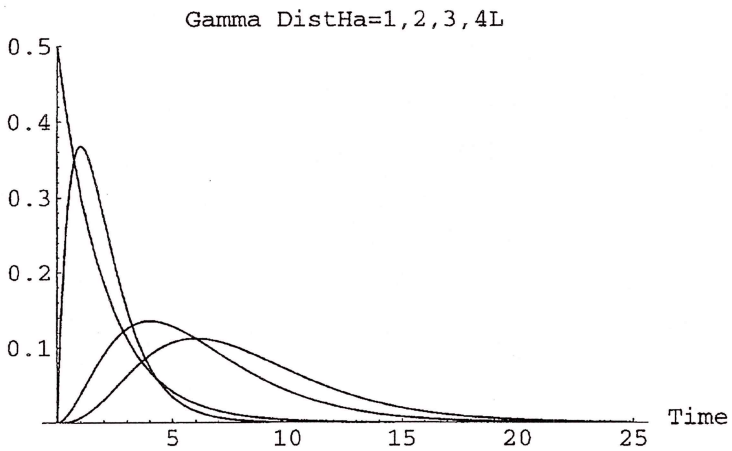
float Targeting::Obj_Pk(int targ_indx, float cep, float yld)

```
 {
//
//              Simple Evaluation Of System Performance For A Weapon Concept
//              Against a Target (targ_indx ):  Pssk constraints evaluated.
//
  float Pk, F1 ;
  Pk = Pssk( targ_indx, cep, yld );
  F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
  return( F1 );
 } ;     // Block: 29
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

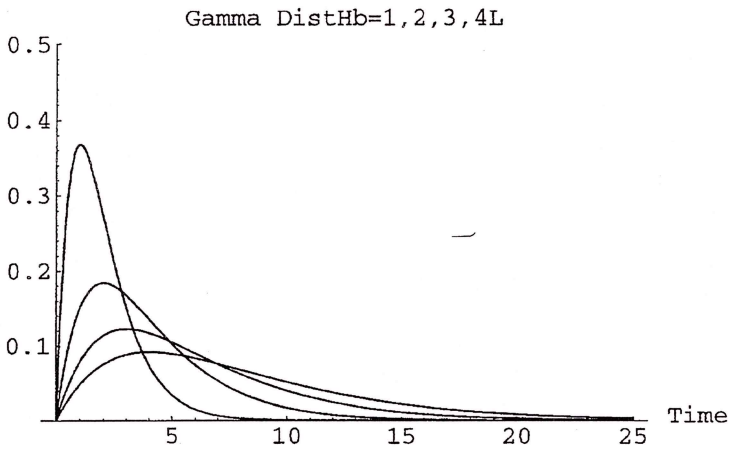
float Targeting::Obj_Time(int targ_indx, int sys_indx)

```
 {
//
//              This routine captures fitness for a time urgency metric; ie.
//              reconstitution targets, time urgent targets, and those of uniform
//              importance as functions of time.
//
//              We are assuming that the system response can be captured
//              by a representative "gamma" distribution.  This distribution can
//              and has been used in queing problems and will be suitability for
//              this application.   Gamma distributions require two parameters,
//              α and β They should be reasonably easy to define since:
//
//              Mean = α β
//              Variance = α β^2
//              Mode = β (α–1)  if α >= 1 zero (0) otherwise
//
//              Knowing any two of the three characteristics can define the
//              distribution form.  The function assesses the probability of
//              launching a system based on the targeting requirements
//              (time-urgency).
//              NOTE:  correlation assumes α is an integer.
```

//



//



//
// The routine has been expanded, 5/11/98, to consider sigmoid
// functions as well as the gamma function. This routine will call
// either the Dist_Gamma or the Dist_Sigmoid function.
//

```
  float F5 ;
  float alp,bet,gam,  response_tm ;
  alp = sys[ sys_indx ].alpha ;
  bet = sys[ sys_indx ].beta ;
  gam = sys[ sys_indx ].gamma ;
  response_tm= startX.targs[ targ_indx ].Tm_obj ;
//
//              Evaluate for the probabiity given a timing objective.
//
  if( gam < 0.0 )
    F5 = Dist_Gamma( alp, bet, response_tm ) ;
  else
```

```
    F5 = Dist_Sigmoid( alp, bet, gam, response_tm ) ;
   return( F5 );
  } ;     // Block: 30
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Obj_Yield(float yld)

```
  {
//
//              This routine provides a rule to minimize yield applied to a target.
//              Note:  This Routine Is Valid If The Yield Options
//                   Are Not Less Than 100
//
   float min_yield=25.0 ;
   float F2;
   F2 = pow( (min_yield/yld),0.3333) ;
   if ( F2 > 1.0 )
     F2 = 1.0;

   return( F2 );
  } ;     // Block: 31
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

void Targeting::Perf_Results( )

```
  {
   ofstream Monitor_out(Results_file,ios::app);

   int sys_index, indx1, avn, ak, wr_out ;
   char att;
   float Pk, yld, tmp_fl ;
   float mission_PD=0.0 ;
   float cep, hob ;
   Monitor_out.precision(6);
//
//              Create A Database Of Mission Performance And Targeting
//
   Monitor_out << "\nPerformance results: \n" << "   :: vntk  :: " ;
   if( DEBUG )
     Monitor_out << " catcode  :: r95 :: T depth :: Leth :: Mis  :: Func  :: ";
   Monitor_out << " Wt  :: Timing  :: HOB Pref  :: wr :: index :: yield :: cep :: ";
   Monitor_out << " hob :: Pk ::\n";
   for( int k=0; k<max_targets; k++ )
     {
     sys_index = allo_vec[k];
     avn= startX.targs[ k ].vn;
```

```
      att= startX.targs[ k ].tc;
      ak= startX.targs[ k ].k;
      if( sys_index > 0 )
        {
        indx1 = opt_index[k][sys_index] ;
        cep= sys[ sys_index ].ceps[ indx1 ];
        hob= sys[ sys_index ].hobs[ indx1 ];
        yld= sys[ sys_index ].yield;
        Pk =Pssk( k, cep, yld );
        wr_out= startX.wr0;
        }
      else
        {
        Pk=0.0;
        cep=1e10;
        wr_out=0.0;
        }
      mission_PD += Pk ;
      if( att == 'G' )
        Monitor_out << startX.targs[ k ].gvn1 << att << startX.targs[ k ].dsig1
          << startX.targs[ k ].tg1 << "\t" ;
      else
        Monitor_out << avn << att << ak << " \t " ;

          if( DEBUG )
            {
            Monitor_out << startX.targs[ k ].catcode << "  " << startX.targs[ k ].r95
            << "\t" << startX.targs[ k ].targ_depth << "\t";
            Monitor_out << startX.targs[ k ].lethality << "  "
            << startX.targs[ k ].mission << "  " << startX.targs[ k ].function << "\t";
            }
          Monitor_out << startX.targs[ k ].Wt << "\t" << startX.targs[ k ].Tm_obj
          << "\t" << startX.targs[ k ].HOB_obj << "\t" ;
          Monitor_out  << wr_out << "\t" << sys_index << "    " << yld << "    "
          << cep << "    " << hob << "\t" << Pk << "\n";
      }
//
//              Create A Series Of Allocaation Databases
//
  for( int ka=1; ka<=total_WH; ka++ )
    {
    Monitor_out << "\nWeapon Targetings : ( inv no. " << ka << " )\n";
    for( int kb=0; kb<max_targets; kb++ )
      {
      sys_index = allo_vec[kb];
      indx1 = opt_index[kb][sys_index] ;
```

```
        if( sys_index == ka )
          {
          cep= sys[ sys_index ].ceps[indx1];
          hob= sys[ sys_index ].hobs[indx1];
          yld= sys[ sys_index ].yield;
          avn= startX.targs[ kb ].vn;
          att= startX.targs[ kb ].tc;
          ak= startX.targs[ kb ].k;
          Pk =Pssk( kb, cep, yld );
          wr_out= startX.wr0;

          if( att == 'G' )
            {
            Monitor_out << startX.targs[ kb ].gvn1 << att << startX.targs[ kb ].dsig1
              << startX.targs[ kb ].tg1 << "\t";
            Monitor_out << startX.targs[ kb ].gvn2 << att
              << startX.targs[ kb ].dsig2  << startX.targs[ kb ].tg2 << "\t" ;
            }
          else
            Monitor_out << avn << att << ak << "\t" ;

          if( DEBUG )
            {
            Monitor_out << startX.targs[ kb ].catcode << "  " << startX.targs[ kb ].r95
            << "\t" << startX.targs[ kb ].targ_depth << "\t";
            Monitor_out << startX.targs[ kb ].lethality << "  "
            << startX.targs[ kb ].mission << "  " << startX.targs[ kb ].function << "\t";
            }
          Monitor_out << startX.targs[ kb ].Wt << "\t" << startX.targs[ kb ].Tm_obj
          << "\t" << startX.targs[ kb ].HOB_obj << "\t" ;
          Monitor_out  << wr_out << " \t " << sys_index << "    " << yld << "    "
          << cep << "    " << hob << " \t " << Pk << "\n";
          }
        }
      }
//
//              Print Out Allocated Inventories
//
  float mission_fit =0;
  int m ;
  int indx= 0;
  for( int df=0; df<max_targets; df++ )
    {
    m= allo_vec[df];
    mission_fit += suitability[df][m] ;
    if( suitability[df][m] >= acceptance )
```


15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
        att= startX.targs[ k ].tc;
        ak= startX.targs[ k ].k;
        if( sys_index > 0 )
          {
          indx1 = opt_index[k][sys_index] ;
          cep= sys[ sys_index ].ceps[ indx1 ];
          hob= sys[ sys_index ].hobs[ indx1 ];
          yld= sys[ sys_index ].yield;
          Pk =Pssk( k, cep, yld );
          wr_out= startX.wr0;
          }
        else
          {
          Pk=0.0;
          cep=1e10;
          wr_out=0.0;
          }
        mission_PD += Pk ;
        if( att == 'G' )
          Monitor_out << startX.targs[ k ].gvn1 << att << startX.targs[ k ].dsig1
            << startX.targs[ k ].tg1 << "\t" ;
        else
          Monitor_out << avn << att << ak << " \t " ;

            if( DEBUG )
              {
              Monitor_out << startX.targs[ k ].catcode << "  " << startX.targs[ k ].r95
              << "\t" << startX.targs[ k ].targ_depth << "\t";
              Monitor_out << startX.targs[ k ].lethality << "  "
              << startX.targs[ k ].mission << "  " << startX.targs[ k ].function << "\t";
              }
            Monitor_out << startX.targs[ k ].Wt << "\t" << startX.targs[ k ].Tm_obj
            << "\t" << startX.targs[ k ].HOB_obj << "\t" ;
            Monitor_out  << wr_out << "\t" << sys_index << "    " << yld << "    "
            << cep << "    " << hob << "\t" << Pk << "\n";
      }
//
//              Create A Series Of Allocaation Databases
//
    for( int ka=1; ka<=total_WH; ka++ )
      {
      Monitor_out << "\nWeapon Targetings : ( inv no. " << ka << " )\n";
      for( int kb=0; kb<max_targets; kb++ )
        {
        sys_index = allo_vec[kb];
        indx1 = opt_index[kb][sys_index] ;
```

```
            if( sys_index == ka )
              {
              cep= sys[ sys_index ].ceps[indx1];
              hob= sys[ sys_index ].hobs[indx1];
              yld= sys[ sys_index ].yield;
              avn= startX.targs[ kb ].vn;
              att= startX.targs[ kb ].tc;
              ak= startX.targs[ kb ].k;
              Pk =Pssk( kb, cep, yld );
              wr_out= startX.wr0;

              if( att == 'G' )
                {
                Monitor_out << startX.targs[ kb ].gvn1 << att << startX.targs[ kb ].dsig1
                  << startX.targs[ kb ].tg1 << "\t";
                Monitor_out << startX.targs[ kb ].gvn2 << att
                  << startX.targs[ kb ].dsig2  << startX.targs[ kb ].tg2 << "\t" ;
                }
              else
                Monitor_out << avn << att << ak << "\t" ;

              if( DEBUG )
                {
                Monitor_out << startX.targs[ kb ].catcode << "  " << startX.targs[ kb ].r95
                << "\t" << startX.targs[ kb ].targ_depth << "\t";
                Monitor_out << startX.targs[ kb ].lethality << "  "
                << startX.targs[ kb ].mission << "  " << startX.targs[ kb ].function << "\t";
                }
              Monitor_out << startX.targs[ kb ].Wt << "\t" << startX.targs[ kb ].Tm_obj
              << "\t" << startX.targs[ kb ].HOB_obj << "\t" ;
              Monitor_out  << wr_out << " \t " << sys_index << "    " << yld << "    "
              << cep << "    " << hob << " \t " << Pk << "\n";
              }
            }
          }
//
//              Print Out Allocated Inventories
//
  float mission_fit =0;
  int m ;
  int indx= 0;
  for( int df=0; df<max_targets; df++ )
    {
    m= allo_vec[df];
    mission_fit += suitability[df][m] ;
    if( suitability[df][m] >= acceptance )
```


Scottish Campaign for Nuclear Disarmament
15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org

```
      else
        {
        total_fitness=0.0;
        opt_index[ k ][sys_index] = 0;
        }
      suitability[k][sys_index]=total_fitness ;
      }
//
//              Examine Allocation If Unconstrained
//
  int tmp_indx;
  indx= 0;
  Mission_fitness=0.0;
  for( k=0; k< max_targets; k++ )
    {
    tmp_indx=0;
    for( int jj=1; jj<= total_WH; jj++ )
      {
      if( suitability[k][ jj ] > suitability[k][ tmp_indx ] )
        tmp_indx=jj ;
      }
    if( suitability[k][tmp_indx] >= acceptance )
      indx++;
    Mission_fitness +=suitability[k][tmp_indx] / float( max_targets ) ;
    }
//
//              Optimum Fitness Acheivable With Unlimited Inventories
//
  Mission_acceptance = float( indx )/ float( max_targets ) ;

//
//              Cleanup Storage
//
  return ;
 } ;     // Block: 35
//          ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

float Targeting::Targeting_Opt( )

```
 {
  float mission_fitness;
  float mission_fit ;
  float fit;
  float debug_fitness;

  IC_Alloc( );
```

```
    if( DEBUG )  Dump_suitability( );

    float *fit_TmSr;
    fit_TmSr = new float[ iterations ];
    float run_ave=Mission_fitness ;
    int term_crit=FALSE ;

    for( int iter=Restart_iter; iter<iterations; iter++ )
      {
//
//              Determin the total Targeting of weapons for  a chromosome
//
    for( int k=0; k<= total_WH; k++ )
      {
      sys[ k ].allocated =0;
      for( int kop=0; kop < sys[ k ].total_fz_opt ; kop++ )
        sys[ k ].opt_inv[ kop ] =0 ;
      }
//
// Com 9        Note: allo_vec Provides The Mapping Between Targets And WH's
//
    int jj, indx0 ;
    for( k=0; k< max_targets; k++ )
      {
      jj = allo_vec[ k ] ;
      sys[ jj ].allocated +=1;
      indx0 = opt_index[k][jj] ;
      if( jj != 0 ) sys[ jj ].opt_inv[ indx0 ] += 1 ;
      }
//
// Com 10       Create The Next Iteration
//
      mission_fitness= Next_Iteration(  );
      mission_fit =0;
      for( int df=0; df<max_targets; df++ )
        mission_fit += suitability[df][ allo_vec[df] ] ;
      mission_fit /= float( max_targets ) ;
//
//              Evaluate convergence criteria (deviation from running average)
//
      float conv_fac=10000.0 ;
      fit_TmSr[ iter ] = mission_fit ;
      if( abs( run_ave-mission_fit )*conv_fac <= 1.0 )
        term_crit= TRUE ;

      if( run_ave > 0.0000001 )
```

```
      run_ave =(run_ave + mission_fit)/2.0 ;
    else
      run_ave = mission_fit ;

//
//              Setup For A Debug Calculation And Output
//
    if( DEBUG )
      {
      G_debug_out << "Iter : "<<iter<< " Mission Acceptance = "
      << mission_fitness << " Mission Fitness = "<< mission_fit
      <<" \nTargeting Vector \n" ;
      int indx1,indx2 ;
      for( int kb=0; kb<max_targets; kb++ )
        {
        indx2= allo_vec[kb] ;
        indx1= opt_index[kb][ indx2 ] ;
        G_debug_out << " (";
        G_debug_out << " "<< indx2 << " ";
        G_debug_out <<indx1<< " "<< suitability[kb][indx2] << " ) ";
        }
      G_debug_out << "\n";
      G_debug_out << "\nWeapon System Inventories \n";
      for( int kd=0; kd<= total_WH; kd++ )
      G_debug_out << kd << " \t " << sys[ kd ] << "\n";
      }
//
//              Provide tracking information for normal and post analysis
//
//              Tracking output controlled by WS_Opt object
      if( G_debug_cntl )
        {
        for(  int kd=1; kd<= total_WH; kd++ )
          {
          cout << "\nWarHead " << kd << "  " << mission_fit << "\t " ;
          for(  int kd2=0; kd2< sys[ kd ].total_fz_opt; kd2++ )
            cout <<" "<< sys[ kd ].ceps[kd2] <<" :: "<< sys[ kd ].opt_inv[kd2]<< "   ";
          cout << "\n" << flush ;
          }
        G_debug_cntl=FALSE ;
        }
    if( term_crit || iter>=iterations-1)
      {
      G_debug_out << "\n+++++ +++++ Target Opt Iter : "<<iter ;
      G_debug_out << " (Population fitness : " << mission_fit <<" )\t";
      for( int kd=0; kd<= total_WH; kd++ )
```

```
            {
            G_debug_out << "\n" << kd << "\t " ;
            for( int kd2=0; kd2< sys[ kd ].total_fz_opt; kd2++ )
              G_debug_out <<" "<< sys[ kd ].ceps[kd2] <<" :: "<< sys[ kd ].opt_inv[kd2]
              << "    ";
            G_debug_out << "\n";
            }
          break;
          }
      }
//
//                Output The Results Of The Targeting
//
  if( APPEND_rslts == TRUE )
    Perf_Results( );
//
//                Cleanup Storage
//
  random.CleanUp_dist( );
  return( mission_fit ) ;
 } ;      // Block: 36
```


Scottish Campaign for Nuclear Disarmament
15 Barrland Street • Glasgow • G41 1QH •
Tel: 0141 423 1222 Fax: 0141 433 2821 e-mail: scnd@banthebomb.org