APPENDIX II

MATLAB Code

MATLAB is a commonly used program for computer modeling. Its code is relatively straightforward. So even though you may not use MATLAB, it has a pseudocode flavor that should be easy to translate into your favorite programming language. If you wish to learn about MATLAB or reference all the manuals on line, go to www.mathworks.com for help. There are many demos, free software, and other useful items as well as all the MATLAB documentation you would ever need. An excellent version is also available for students. Many of the programs we have used in this book are listed in this appendix and come on the included CD. All the plots and graphs in this book were created with MATLAB version 6.5. We have listed the MATLAB code in the appendix in case the CD gets separated from the book.

PROGRAM 1: BINARY GENETIC ALGORITHM

```
%              Binary Genetic Algorithm
%
%  minimizes the objective function designated in ff
%  Before beginning, set all the parameters in parts
   I, II, and III
%  Haupt & Haupt
%  2003
clear
%_____________________________________________________
%                    I. Setup the GA
ff='testfunction';  % objective function
npar=2;             % number of optimization variables
%_____________________________________________________
%                 II. Stopping criteria
maxit=100;                 % max number of iterations
mincost=-9999999;          % minimum cost
```

Practical Genetic Algorithms, Second Edition, by Randy L. Haupt and Sue Ellen Haupt.
ISBN 0-471-45565-2

```
%_______________________________________________________
%                  III. GA parameters
popsize=16;                      % set population size
mutrate=.15;                     % set mutation rate
selection=0.5;                   % fraction of population
                                 % kept
nbits=8;                         % number of bits in each
                                 % parameter
Nt=nbits*npar;                   % total number of bits
                                 % in a chormosome
keep=floor(selection*popsize);   % #population members
                                 % that survive

%_______________________________________________________
%              Create the initial population
iga=0;                             % generation counter
                                   % initialized
pop=round(rand(popsize,Nt));       % random population of
                                   % 1s and 0s
par=gadecode(pop,0,10,nbits);      % convert binary to
                                   % continuous values
cost=feval(ff,par);                % calculates population
                                   % cost using ff
[cost,ind]=sort(cost);             % min cost in element 1
par=par(ind,:);pop=pop(ind,:);     % sorts population with
                                   % lowest cost first
minc(1)=min(cost);                 % minc contains min of
                                   % population
meanc(1)=mean(cost);               % meanc contains mean
                                   % of population

%_______________________________________________________
%              Iterate through generations
while iga<maxit
  iga=iga+1;           % increments generation counter

%_______________________________________________________
%                     Pair and mate
M=ceil((popsize-keep)/2);             % number of matings
prob=flipud([1:keep]'/sum([1:keep]));% weights
                                      % chromosomes based
                                      % upon position in
                                      % list
odds=[0 cumsum(prob(1:keep))'];       % probability
distribution function
```

```
pick1=rand(1,M);                          % mate #1
pick2=rand(1,M);                          % mate #2

% ma and pa contain the indicies of the chromosomes
% that will mate
ic=1;
while ic<=M
 for id=2:keep+1
   if pick1(ic)<=odds(id) & pick1(ic)>odds(id-1)
     ma(ic)=id-1;
   end % if
   if pick2(ic)<=odds(id) & pick2(ic)>odds(id-1)
     pa(ic)=id-1;
   end % if
 end % id
 ic=ic+1;
end % while

%_______________________________________________________
%    Performs mating using single point crossover
ix=1:2:keep; % index of mate #1
xp=ceil(rand(1,M)*(Nt-1)); % crossover point
pop(keep+ix,:)=[pop(ma,1:xp) pop(pa,xp+1:Nt)];
                              % first offspring
pop(keep+ix+1,:)=[pop(pa,1:xp) pop(ma,xp+1:Nt)];
               % second offspring

%_______________________________________________________
%                Mutate the population
nmut=ceil((popsize-1)*Nt*mutrate);     % total number
                                       % of mutations
mrow=ceil(rand(1,nmut)*(popsize-1))+1; % row to mutate
mcol=ceil(rand(1,nmut)*Nt);    % column to mutate
for ii=1:nmut
  pop(mrow(ii),mcol(ii))=abs(pop(mrow(ii),mcol(ii))-1);
                                       % toggles bits
end                                    % ii

%_______________________________________________________
%      The population is re-evaluated for cost
par(2:popsize,:)=gadecode(pop(2:popsize,:),0,10,nbits);
% decode
cost(2:popsize)=feval(ff,par(2:popsize,:));
```

```
%_______________________________________________________
%       Sort the costs and associated parameters
[cost,ind]=sort(cost);
par=par(ind,:); pop=pop(ind,:);

%_______________________________________________________
%     Do statistics for a single nonaveraging run
minc(iga+1)=min(cost);
meanc(iga+1)=mean(cost);

%_______________________________________________________
%                  Stopping criteria
if iga>maxit | cost(1)<mincost
  break
end

[iga cost(1)]

end %iga

%_______________________________________________________
%                 Displays the output
day=clock;
disp(datestr(datenum(day(1),day(2),day(3),day(4),day(5),
day(6)),0))
disp(['optimized function is ' ff])
format short g
disp(['popsize = ' num2str(popsize) ' mutrate = '
num2str(mutrate) ' # par = ' num2str(npar)])
disp(['#generations=' num2str(iga) ' best cost='
num2str(cost(1))])
disp(['best solution'])
disp([num2str(par(1,:))])
disp('binary genetic algorithm')
disp(['each parameter represented by ' num2str(nbits)
' bits'])
figure(24)
iters=0:length(minc)-1;
plot(iters,minc,iters,meanc,'-');
xlabel('generation');ylabel('cost');
text(0,minc(1),'best');text(1,minc(2),'population
average')
```

PROGRAM 2: CONVERTS BINARY CHROMOSOME TO CONTINUOUS VARIABLES

```
%    gadecode.m
%       Decodes binary encripted parameters
%
%              f=gadecode(chrom,lo,hi,bits,gray)
%              chrom = population
%              lo  = minimum parameter value
%              hi  = maximum parameter value
%              bits = number of bits/parameter

% Haupt & Haupt
% 2003

function f=gadecode(chrom,lo,hi,bits)

[M,N]=size(chrom);
npar=N/bits;                    % number of variables
quant=(0.5.^[1:bits]');         % quantization levels
quant=quant/sum(quant);         % quantization levels
normalized
ct=reshape(chrom',bits,npar*M)';% each column contains
                                % one variable
par=((ct*quant)*(hi-lo)+lo);    % DA conversion and
                                % unnormalize varaibles
f=reshape(par,npar,M)';         % reassemble population
```

PROGRAM 3: CONTINUOUS GENETIC ALGORITHM

```
%              Continuous Genetic Algorithm
%
%  minimizes the objective function designated in ff
%  Before beginning, set all the parameters in parts
%  I, II, and III
%  Haupt & Haupt
%  2003

%_______________________________________________________
%                    I Setup the GA
ff='testfunction';  % objective function
npar=2;             % number of optimization variables
varhi=10; varlo=0;  % variable limits
```

```
%_______________________________________________________
%                II Stopping criteria
maxit=100;          % max number of iterations
mincost=-9999999;   % minimum cost

%_______________________________________________________
%                 III GA parameters
popsize=12;     % set population size
mutrate=.2;     % set mutation rate
selection=0.5;  % fraction of population kept
Nt=npar;        % continuous parameter GA Nt=#variables

keep=floor(selection*popsize);    % #population
                                  % members that survive
nmut=ceil((popsize-1)*Nt*mutrate);  % total number of
                                    % mutations
M=ceil((popsize-keep)/2);           % number of matings

%_______________________________________________________
%            Create the initial population
iga=0;                  % generation counter
initialized
par=(varhi-varlo)*rand(popsize,npar)+varlo;   % random
cost=feval(ff,par);     % calculates population cost
                        % using ff
[cost,ind]=sort(cost);  % min cost in element 1
par=par(ind,:);         % sort continuous
minc(1)=min(cost);      % minc contains min of
meanc(1)=mean(cost);    % meanc contains mean of
population

%_______________________________________________________
%             Iterate through generations
while iga<maxit
  iga=iga+1;           % increments generation counter

%_______________________________________________________
%                    Pair and mate
M=ceil((popsize-keep)/2);          % number of matings
prob=flipud([1:keep]'/sum([1:keep])); % weights
                                      % chromosomes
odds=[0 cumsum(prob(1:keep))'];       % probability
                                      % distribution
                                      % function
pick1=rand(1,M);                      % mate #1
pick2=rand(1,M);                      % mate #2
```

```
% ma and pa contain the indicies of the chromosomes
% that will mate
ic=1;
while ic<=M
 for id=2:keep+1
    if pick1(ic)<=odds(id) & pick1(ic)>odds(id-1)
      ma(ic)=id-1;
    end
    if pick2(ic)<=odds(id) & pick2(ic)>odds(id-1)
      pa(ic)=id-1;
    end
 end
 ic=ic+1;
end


%_______________________________________________________
%    Performs mating using single point crossover
ix=1:2:keep;                         % index of mate #1
xp=ceil(rand(1,M)*Nt);               % crossover point
r=rand(1,M);                         % mixing parameter
for ic=1:M
 xy=par(ma(ic),xp(ic))-par(pa(ic),xp(ic));  % ma and pa
                                            % mate
 par(keep+ix(ic),:)=par(ma(ic),:);    % 1st offspring
 par(keep+ix(ic)+1,:)=par(pa(ic),:);    % 2nd offspring
 par(keep+ix(ic),xp(ic))=par(ma(ic),xp(ic))-r(ic).*xy;
% 1st
par(keep+ix(ic)+1,xp(ic))=par(pa(ic),xp(ic))+r(ic).*xy;
% 2nd
  if xp(ic)<npar % crossover when last variable not
selected
    par(keep+ix(ic),:)=[par(keep+ix(ic),1:xp(ic))
    par(keep+ix(ic)+1,xp(ic)+1:npar)];
    par(keep+ix(ic)+1,:)=[par(keep+ix(ic)+1,1:xp(ic))
    par(keep+ix(ic),xp(ic)+1:npar)];
  end % if
end

%_______________________________________________________
%                 Mutate the population
mrow=sort(ceil(rand(1,nmut)*(popsize-1))+1);
mcol=ceil(rand(1,nmut)*Nt);
```

```
for ii=1:nmut
   par(mrow(ii),mcol(ii))=(varhi-varlo)*rand+varlo;
% mutation
end % ii

%_______________________________________________________
%   The new offspring and mutated chromosomes are
% evaluated
cost=feval(ff,par);

%_______________________________________________________
%      Sort the costs and associated parameters
[cost,ind]=sort(cost);
par=par(ind,:);

%_______________________________________________________
%     Do statistics for a single nonaveraging run
  minc(iga+1)=min(cost);
  meanc(iga+1)=mean(cost);

%_______________________________________________________
%                 Stopping criteria
if iga>maxit | cost(1)<mincost
  break
end

[iga cost(1)]

end %iga

%_______________________________________________________
%                Displays the output
day=clock;
disp(datestr(datenum(day(1),day(2),day(3),day(4),day(5),
day(6)),0))
disp(['optimized function is ' ff])
format short g
disp(['popsize = ' num2str(popsize) ' mutrate = '
num2str(mutrate) ' # par = ' num2str(npar)])
disp(['#generations=' num2str(iga) ' best cost='
num2str(cost(1))])
disp(['best solution'])
disp([num2str(par(1,:))])
disp('continuous genetic algorithm')
```

```
figure(24)
iters=0:length(minc)-1;
plot(iters,minc,iters,meanc,'-');
xlabel('generation');ylabel('cost');
text(0,minc(1),'best');text(1,minc(2),'population
average')
```

PROGRAM 4: PARETO GENETIC ALGORITHM

```
%              Pareto Genetic Algorithm
%
%   minimizes the objective function designated in ff
%   All optimization variables are normalized between 0
%   and 1.
%  ff must map variables to actual range
%   Haupt & Haupt
%   2003

%_______________________________________________________
%                     Setup the GA
ff='testfunction';  % objective function
npar=4;             % number of optimization variables

%_______________________________________________________
%                   Stopping criteria
maxit=50;                % max number of iterations
mincost=.001;            % minimum cost

%_______________________________________________________
%                     GA parameters
selection=0.5;           % fraction of population kept
popsize=100;
keep=selection*popsize;
M=ceil((popsize-keep)/2);   % number of matings
odds=1;
for ii=2:keep
   odds=[odds ii*ones(1,ii)];
end
odds=keep+1-odds;
Nodds=length(odds);
mutrate=0.1;             % mutation rate
```

```
%_______________________________________________________
%             Create the initial population
iga=0;                      % generation counter initialized
pop=rand(popsize,npar);  % random population of
                            % continuous values
fout=feval(ff,pop);         % calculates population cost
                            % using ff

%_______________________________________________________
%              Iterate through generations
while iga<maxit
  iga=iga+1;                        % increments
generation counter
cost(1:4,:)
    [g,ind]=sort(fout(:,1));
    pop=pop(ind,:);  % sorts chromosomes
    h=fout(ind,2);
    ct=0; rank=1;
    q=0;
    while ct<popsize
        for ig=1:popsize
            if h(ig)<=min(h(1:ig))
                ct=ct+1;
                q(ct)=ig;
                cost(ct,1)=rank;
            end
            if rank==1
                px=g(q);py=h(q);
                elite=length(q);
            end
            if ct==popsize; break; end
        end
        rank=rank+1;
    end
    pop=pop(q,:);
    figure(1); clf;plot(g,h,'.',px,py); axis([0 1 0 1]);
    axis square pause
    [cost,ind]=sort(cost);
    pop=pop(ind,:);

    % tournament selection
    Ntourn=2;
    picks=ceil(keep*rand(Ntourn,M));
    [c,pt]=min(cost(picks));
    for ib=1:M
```

```
        ma(ib)=picks(pt(ib),ib);
    end
    picks=ceil(keep*rand(Ntourn,M));
    [c,pt]=min(cost(picks));
    for ib=1:M
         pa(ib)=picks(pt(ib),ib);
    end

%_____________________________________________________________________
%                   Performs mating
   ix=1:2:keep;              % index of mate #1
   xp=floor(rand(1,M)*npar); % crossover point
   r=rand(1,M);              % mixing parameter
   xy=pop(ma+popsize*xp)-pop(pa+popsize*xp);
   % mix from ma and pa
   pop(keep+ix+popsize*xp)=pop(ma+popsize*xp)-r.*xy;
   % 1st offspring
   pop(keep+ix+1+popsize*xp)=pop(pa+popsize*xp)+r.*xy;
   % 2nd offspring
   for ic=1:M/2
   if xp(ic)<npar % perform crossover when last
                  % variable not selected
   pop(keep+ix(ic),:)=[pop(ma(ic),1:xp(ic))
   pop(pa(ic),xp(ic)+1:npar)];
   pop(keep+ix(ic)+1,:)=[pop(pa(ic),1:xp(ic))
   pop(ma(ic),xp(ic)+1:npar)];
   end % if
   end % end ic
   pop(1:4,:)

%_____________________________________________________________________
%                 Mutate the population
nmut=ceil((popsize-elite)*npar*mutrate);% total number
                                        % of mutations
mrow=ceil(rand(1,nmut)*(popsize-elite))+elite;
mcol=ceil(rand(1,nmut)*npar);
mutindx=mrow+(mcol-1)*popsize;
pop(mutindx)=rand(1,nmut);

%_____________________________________________________________________
%   The new offspring and mutated chromosomes are
evaluated for cost
row=sort(rem(mutindx,popsize));
iq=1; rowmut(iq)=row(1);
for ic=2:nmut
```

```
    if row(ic)>keep;break;end
    if row(ic)>rowmut(iq)
        iq=iq+1; rowmut(iq)=row(ic);
    end
end
if rowmut(1)==0;rowmut=rowmut(2:length(rowmut));end
fout(rowmut,:)=feval(ff,pop(rowmut,:));
fout(keep+1:popsize,:)=feval(ff,pop(keep+1:popsize,:));
fout(keep+1:popsize,:)=feval(ff,pop(keep+1:popsize,:));

%_______________________________________________________
%                  Stopping criteria
if iga>maxit
  break
end

[iga cost(1) fout(1,:)]

end %iga

%_______________________________________________________
%                 Displays the output
  day=clock;

  disp(datestr(datenum(day(1),day(2),day(3),day(4),day(
  5),day(6)),0))
  disp(['optimized function is ' ff])
  format short g
  disp(['popsize = ' num2str(popsize) ' mutrate = '
  num2str(mutrate) ' # par = ' num2str(npar)])
  disp(['Pareto front'])
  disp([num2str(pop)])
     disp('continuous parameter genetic algorithm')
```

PROGRAM 5: PERMUTATION GENETIC ALGORITHM

```
%             Genetic Algorithm for permuation problems
%
%  minimizes the objective function designated in ff

clear
global iga x y

%  Haupt & Haupt
%  2003
```

```
%_______________________________________________________
%                        Setup the GA
ff='tspfun';                        % objective function
npar=20;                 % # optimization variables
Nt=npar;                 % # columns in population matrix
rand('state',3)
x=rand(1,npar);y=rand(1,npar);  % cities are at
                                % (xcity,ycity)

%_______________________________________________________
%                     Stopping criteria
maxit=10000;                  % max number of iterations

%_______________________________________________________
%                        GA parameters
popsize=20;                % set population size
mutrate=.1;                % set mutation rate
selection=0.5;             % fraction of population kept

keep=floor(selection*popsize);   % #population members
                                 % that survive
   M=ceil((popsize-keep)/2);     % number of matings
   odds=1;
   for ii=2:keep
       odds=[odds ii*ones(1,ii)];
   end
   Nodds=length(odds);

%_______________________________________________________
%              Create the initial population
iga=0;             % generation counter initialized
for iz=1:popsize
   pop(iz,:)=randperm(npar); % random population
end

cost=feval(ff,pop);        % calculates population cost
                           % using ff
[cost,ind]=sort(cost);     % min cost in element 1
pop=pop(ind,:);            % sort population with lowest
                           % cost first
minc(1)=min(cost);         % minc contains min of
                           % population
meanc(1)=mean(cost);  % meanc contains mean of population
```

```
%_______________________________________________________
%              Iterate through generations
while iga<maxit
   iga=iga+1;       % increments generation counter

%_______________________________________________________
%                     Pair and mate
        pick1=ceil(Nodds*rand(1,M)); % mate #1
        pick2=ceil(Nodds*rand(1,M)); % mate #2

% ma and pa contain the indicies of the parents
        ma=odds(pick1);
        pa=odds(pick2);

%_______________________________________________________
%                    Performs mating
for ic=1:M
 mate1=pop(ma(ic),:);
 mate2=pop(pa(ic),:);
 indx=2*(ic-1)+1;    % starts at one and skips every
                     % other one
 xp=ceil(rand*npar);    % random value between 1 and N
 temp=mate1;
 x0=xp;

 while mate1(xp)~=temp(x0)
   mate1(xp)=mate2(xp);
   mate2(xp)=temp(xp);
   xs=find(temp==mate1(xp));
   xp=xs;
 end

     pop(keep+indx,:)=mate1;
     pop(keep+indx+1,:)=mate2;
 end

%_______________________________________________________
%                 Mutate the population
nmut=ceil(popsize*npar*mutrate);

for ic = 1:nmut
 row1=ceil(rand*(popsize-1))+1;
 col1=ceil(rand*npar);
 col2=ceil(rand*npar);
```

```
 temp=pop(row1,col1);
 pop(row1,col1)=pop(row1,col2);
 pop(row1,col2)=temp;
 im(ic)=row1;
end
cost=feval(ff,pop);

%_______________________________________________________
%      Sort the costs and associated parameters
part=pop; costt=cost;
[cost,ind]=sort(cost);
pop=pop(ind,:);

%_______________________________________________________
%                  Do statistics
  minc(iga)=min(cost);
  meanc(iga)=mean(cost);

end %iga

%_______________________________________________________
%                Displays the output
  day=clock;
  disp(datestr(datenum(day(1),day(2),day(3),day(4),day(
  5),day(6)),0))
  disp([‘optimized function is ‘ ff])
  format short g
  disp([‘popsize = ‘ num2str(popsize) ‘ mutrate = ‘
  num2str(mutrate) ‘ # par = ‘ num2str(npar)])
  disp([‘ best cost=’ num2str(cost(1))])

  disp([‘best solution’])
  disp([num2str(pop(1,:))])
     figure(2)
     iters=1:maxit;
     plot(iters,minc,iters,meanc,’--’);
     xlabel(‘generation’);ylabel(‘cost’);

figure(1);plot([x(pop(1,:)) x(pop(1,1))],[y(pop(1,:))
y(pop(1,1))],x,y,’o’);axis square
```

PROGRAM 6: TRAVELING SALESPERSON PROBLEM COST FUNCTION

```
% cost function for traveling salesperson problem

%  Haupt & Haupt
%  2003

function dist=tspfun(pop)

global iga x y

[Npop,Ncity]=size(pop);
tour=[pop pop(:,1)];

%distance between cities
for ic=1:Ncity
    for id=1:Ncity
        dcity(ic,id)=sqrt((x(ic)-x(id))^2+(y(ic)-
        y(id))^2);
    end % id
end %ic

% cost of each chromosome
for ic=1:Npop
    dist(ic,1)=0;
    for id=1:Ncity

        dist(ic,1)=dist(ic)+dcity(tour(ic,id),tour(ic,i
        d+1));
    end % id
end % ic
```

PROGRAM 7: PARTICLE SWARM OPTIMIZATION

```
% Particle Swarm Optimization - PSO

% Haupt & Haupt
% 2003

clear
ff = 'testfunction'; % Objective Function

% Initializing variables
popsize = 10;    % Size of the swarm
```

```
npar = 2;        % Dimension of the problem
maxit = 100;     % Maximum number of iterations
c1 = 1;          % cognitive parameter
c2 = 4-c1;       % social parameter
C=1;             % constriction factor

% Initializing swarm and velocities
par=rand(popsize,npar);       % random population of
                              % continuous values
vel = rand(popsize,npar);     % random velocities

% Evaluate initial population
cost=feval(ff,par); % calculates population cost using
                    % ff
minc(1)=min(cost);       % min cost
meanc(1)=mean(cost);     % mean cost
globalmin=minc(1);       % initialize global minimum

% Initialize local minimum for each particle
localpar = par;          % location of local minima
localcost = cost;        % cost of local minima

% Finding best particle in initial population
[globalcost,indx] = min(cost);
globalpar=par(indx,:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Start iterations
iter = 0;          % counter

while iter < maxit
   iter = iter + 1;

% update velocity = vel
   w=(maxit-iter)/maxit; %inertia weiindxht
   r1 = rand(popsize,npar);     % random numbers
   r2 = rand(popsize,npar);     % random numbers
   vel = C*(w*vel + c1 *r1.*(localpar-par) +
c2*r2.*(ones(popsize,1)*globalpar-par));

% update particle positions
   par = par + vel;     % updates particle position
   overlimit=par<=1;
```

```
    underlimit=par>=0;
    par=par.*overlimit+not(overlimit);
    par=par.*underlimit;

% Evaluate the new swarm
    cost = feval(ff,par);    % evaluates cost of swarm

% Updating the best local position for each particle
    bettercost = cost < localcost;
    localcost = localcost.*not(bettercost) +
cost.*bettercost;
    localpar(find(bettercost),:) =
par(find(bettercost),:);

% Updating index g
    [temp, t] = min(localcost);
if temp<globalcost
    globalpar=par(t,:); indx=t; globalcost=temp;
end
    [iter globalpar globalcost]     % print output each
                                    % iteration
    minc(iter+1)=min(cost);         % min for this
                                    % iteration
    globalmin(iter+1)=globalcost;   % best min so far
    meanc(iter+1)=mean(cost);       % avg. cost for
                                    % this iteration

end% while

figure(24)
iters=0:length(minc)-1;
plot(iters,minc,iters,meanc,'–',iters,globalmin,':');
xlabel('generation');ylabel('cost');
text(0,minc(1),'best');text(1,minc(2),'population
average')
```

PROGRAM 8: ANT COLONY OPTIMIZATION

```
%  ACO: ant colony optimization for solving the
traveling salesperson
%  problem

% Haupt & Haupt
% 2003
```

```
clear
rand('state',11)
Ncity=30;       % number of cities on tour
Nants=Ncity;    % number of ants=number of cities

% city locations
xcity=rand(1,Ncity);ycity=rand(1,Ncity); % cities are
located at (xcity,ycity)

%distance between cities
for ic=1:Ncity
    for id=1:Ncity
dcity(ic,id)=sqrt((xcity(ic)-xcity(id))^2+(ycity(ic)-
ycity(id))^2);
    end % id
end %ic

vis=1./dcity;                 % visibility equals inverse
                              % of distance
phmone=.1*ones(Ncity,Ncity);% initialized pheromones
                              % between cities
maxit=600;                    % max number of iterations

% a1=0 - closest city is selected
% be=0 - algorithm only works w/ pheromones and not
% distance of city
% Q - close to the lenght of the optimal tour
% rr - trail decay
a=2;b=6;rr=0.5;Q=sum(1./(1:8));dbest=9999999;e=5;

% initialize tours
for ic=1:Nants
    tour(ic,:)=randperm(Ncity);
end % ic
tour(:,Ncity+1)=tour(:,1); % tour ends on city it
starts with

for it=1:maxit
% find the city tour for each ant
% st is the current city
% nxt contains the remaining cities to be visited
    for ia=1:Nants
        for iq=2:Ncity-1
            [iq tour(ia,:)];
            st=tour(ia,iq-1); nxt=tour(ia,iq:Ncity);
```

```
prob=((phmone(st,nxt).^a).*(vis(st,nxt).^b)).
sum((phmone(st,nxt).^a).*(vis(st,nxt).^b));
    rcity=rand;
    for iz=1:length(prob)
        if rcity<sum(prob(1:iz))
        newcity=iq-1+iz;                    % next city
to be visited
                    break
                end % if
            end % iz
            temp=tour(ia,newcity); % puts the new city
                                  % selected next in line
            tour(ia,newcity)=tour(ia,iq);
            tour(ia,iq)=temp;
        end % iq
    end % ia
% calculate the length of each tour and pheromone
distribution
phtemp=zeros(Ncity,Ncity);
for ic=1:Nants
    dist(ic,1)=0;
    for id=1:Ncity

dist(ic,1)=dist(ic)+dcity(tour(ic,id),tour(ic,id+1));
        phtemp(tour(ic,id),tour(ic,id+1))=Q/dist(ic,1);
    end % id
end % ic

[dmin,ind]=min(dist);
if dmin<dbest
    dbest=dmin;
end % if

% pheromone for elite path
ph1=zeros(Ncity,Ncity);
    for id=1:Ncity
        ph1(tour(ind,id),tour(ind,id+1))=Q/dbest;
    end % id

% update pheromone trails
phmone=(1-rr)*phmone+phtemp+e*ph1;
 dd(it,:)=[dbest dmin];
[it dmin dbest]
end %it
```

```
[tour,dist]
figure(1)
plot(xcity(tour(ind,:)),ycity(tour(ind,:)),xcity,ycity,'
o')
axis square
figure(2);plot([1:maxit],dd(:,1),[1:maxit],dd(:,2),'-')
```

PROGRAM 9: TEST FUNCTIONS

```
% Test functions for optimization
% These are the test functions that appear in Appendix I.
% Set funnum to the function you want to use.
% funnum=17 is for a MOO function

% Haupt & Haupt
% 2003

function f=testfunction(x)

funnum=16;

if funnum==1     %F1
   f=abs(x)+cos(x);
elseif funnum==2     %F2
   f=abs(x)+sin(x);
elseif funnum==3     %F3
   f=x(:,1).^2+x(:,2).^2;
elseif funnum==4     %F4
   f=100*(x(:,2).^2-x(:,1)).^2+(1-x(:,1)).^2;
elseif funnum==5     %F5
   f(:,1)=sum(abs(x')-10*cos(sqrt(abs(10*x'))))';
elseif funnum==6     %F6
   f=(x.^2+x).*cos(x);
elseif funnum==7     %F7
   f=x(:,1).*sin(4*x(:,1))+1.1*x(:,2).*sin(2*x(:,2));
elseif funnum==8     %F8
   f=x(:,2).*sin(4*x(:,1))+1.1*x(:,1).*sin(2*x(:,2));
elseif funnum==9     %F9

f(:,1)=x(:,1).^4+2*x(:,2).^4+randn(length(x(:,1)),1);
elseif funnum==10    %F10
   f(:,1)=20+sum(x'.^2-10*cos(2*pi*x'))';
elseif funnum==11    %F11
   f(:,1)=1+sum(abs(x').^2/4000)'-prod(cos(x'))';
```

```
elseif funnum==12   %F12
f(:,1)=.5+(sin(sqrt(x(:,1).^2+x(:,2).^2).^2)-
.5)./(1+.1*(x(:,1).^2+x(:,2).^2));
elseif funnum==13   %F13
    aa=x(:,1).^2+x(:,2).^2;
    bb=((x(:,1)+.5).^2+x(:,2).^2).^0.1;

f(:,1)=aa.^0.25.*sin(30*bb).^2+abs(x(:,1))+abs(x(:,2));
elseif funnum==14   %F14
    f(:,1)=besselj(0,x(:,1).^2+x(:,2).^2)+abs(1-
x(:,1))/10+abs(1-x(:,2))/10;
elseif funnum==15   %F15
f(:,1)=-exp(.2*sqrt((x(:,1)-1).^2+(x(:,2)-
1).^2)+(cos(2*x(:,1))+sin(2*x(:,1))));
elseif funnum==16   %F16
f(:,1)=x(:,1).*sin(sqrt(abs(x(:,1)-(x(:,2)+9))))-
(x(:,2)+9).*sin(sqrt(abs(x(:,2)+0.5*x(:,1)+9)));
elseif funnum==17   %MOO function
    x=x+1;
    f(:,1)=(x(:,1)+x(:,2).^2+sqrt(x(:,3))+1./x(:,4))/8.5;
    f(:,2)=(1./x(:,1)+1./x(:,2)+x(:,3)+x(:,4))/6;
end
```