

```

/*****
/*
/*          COM-Ned
/*          (evr7_all6.c)
/*          Zhiang (John) Lin
/*          12/2006
*****/

/*****
/* search1: satisficing (based on minimal resource requirements)
/* search2: economic (based on resource match)
/* search3: behavioral (based on past interactions and whether the potential
partner has received contribution from the focal node before)
*/
/* search4: integrated (based on both economic and behavioral)
/* event:  randomly started events with normal distribution shape
/* resources: a firm transfers own resources to the event to reduce its demand
before searching for external relations*/
/* ERO set at 5 */
/* network: whole network with all nodes checked
/* exchange: one unit per dimension but can contribute more if received from
that partner more earlier*/
*****/
#include<stdio.h>
#include<math.h>

/*****experimental parameters*****/
#define RUN 1          /*number of experimental runs*/
#define NM 30         /*max number of nodes*/
#define TUNIT 30      /*number of micro time units within event year*/
#define YEAR 80       /*number of calendar years*/
#define EYEAR 20      /*number of macro event years used for output records:
notice the difference from last version*/
#define RI 10         /*maximum input types; can change based on experiment*/
#define RO 10         /*maximum output types; can change based on experiment*/
#define ERO 5         /*maximum event dimension, i.e., resource types*/

int SEARCH; /*type of searches: 1 for satisficing, 2 for economic, 3 for
behavioral, 4 for integrated*/
int MAXIN; /*maximum possible amount for an input dimension*/
int MAXOUT; /*maximum possible amount for an output dimension*/
double THRESHOLD; /*no need to use threshold in this version*/
double GROWTH; /*growth rate of the industry resources*/

/*****node characteristics and temporary storages*****/
double resource_in[NM+1][RI+1],resource_out[NM+1][RO+1]; /*nodes in & output*/

double temp_resin[NM+1][RI+1],temp_resout[NM+1][RO+1]; /*temporary storage*/
double rem_resout[NM+1]; /*available output resources*/
double inires_in[NM+1],inires_out[NM+1];

/*****define event*****/
double event[NM+1][ERO+1]; /*note the new event arrays*/

/*****network matrices and temporary storages*****/

double resource_match[NM+1][NM+1]; /*resource match between i&j*/

```

```

double temp_resmatch[NM+1][NM+1]; /*temporary storage*/

double contact[NM+1][NM+1]; /*contacts between i&j*/
double temp_cont[NM+1][NM+1]; /*temporary storage*/

double int_cont[NM+1][NM+1]; /*integrated based on resource and contact*/
double temp_intcont[NM+1][NM+1]; /*temporary storage*/

/*****performance measures*****/
double ex_resource[NM+1],netcontact[NM+1][NM+1];

/*****flags and other variables*****/
int trigger,location,multi;
int time,temp_t,num_tunit,year,syear[NM+1],eyear[NM+1];
int tot_partner[NM+1],curr_partner[NM+1],last_partner[NM+1],diff_partner[NM+1];
double event_dmd[NM+1];
double differs[NM+1],ownres[NM+1],last_exres[NM+1],new_exres[NM+1],
shortage[NM+1];
int run,node;

/*****network measures*****/
double shole[NM+1],centr[NM+1],g_cenmin,g_cenmax,g_cenmm,g_stderr;

FILE *op, *fopen();

/*****main function*****/
/*          main function          */
/*****main function*****/
main()
{

/*during each event year there are TUNIT time units. Within each time unit, each
node takes turns for resource searching*/

/*experiments for satisficing search*/
/*low growth rate*/

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=1;
MAXIN=1;
MAXOUT=1;
THRESHOLD=.20;
GROWTH=.05;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=1;
MAXIN=5;
MAXOUT=5;
THRESHOLD=.20;
GROWTH=.05;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

```

```

}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=1;
  MAXIN=10;
  MAXOUT=10;
  THRESHOLD=.20;
  GROWTH=.05;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

/*high growth rate*/
for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=1;
  MAXIN=1;
  MAXOUT=1;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=1;
  MAXIN=5;
  MAXOUT=5;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=1;
  MAXIN=10;
  MAXOUT=10;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);

  evolution_run();
}

/*experiments for economic search*/
/*low growth rate*/

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=1;
  MAXOUT=1;
  THRESHOLD=.20;
  GROWTH=.05;
  printf("\nsearch=%3d run=%3d",SEARCH,run);

  evolution_run();
}

```

```

}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=5;
  MAXOUT=5;
  THRESHOLD=.20;
  GROWTH=.05;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=10;
  MAXOUT=10;
  THRESHOLD=.20;
  GROWTH=.05;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

/*high growth rate*/
for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=1;
  MAXOUT=1;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=5;
  MAXOUT=5;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=2;
  MAXIN=10;
  MAXOUT=10;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

/*experiments for behavioral search*/
/*low growth rate*/

```

```
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=1;  
  MAXOUT=1;  
  THRESHOLD=.20;  
  GROWTH=.05;  
  printf("\nsearch=%3d run=%3d",SEARCH,run);  
  evolution_run();  
}
```

```
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=5;  
  MAXOUT=5;  
  THRESHOLD=.20;  
  GROWTH=.05;  
  printf("\nsearch=%3d run=%3d",SEARCH,run);  
  evolution_run();  
}
```

```
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=10;  
  MAXOUT=10;  
  THRESHOLD=.20;  
  GROWTH=.05;  
  printf("\nsearch=%3d run=%3d",SEARCH,run);  
  evolution_run();  
}
```

```
/*high growth rate*/  
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=1;  
  MAXOUT=1;  
  THRESHOLD=.20;  
  GROWTH=.25;  
  printf("\nsearch=%3d run=%3d",SEARCH,run);  
  evolution_run();  
}
```

```
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=5;  
  MAXOUT=5;  
  THRESHOLD=.20;  
  GROWTH=.25;  
  printf("\nsearch=%3d run=%3d",SEARCH,run);  
  evolution_run();  
}
```

```
for (run=1;run<=RUN;run++){ /*experimental runs*/  
  SEARCH=3;  
  MAXIN=10;  
  MAXOUT=10;  
  THRESHOLD=.20;  
  GROWTH=.25;
```

```

printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

/*experiments for integrated search*/
/*low growth rate*/

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=4;
MAXIN=1;
MAXOUT=1;
THRESHOLD=.20;
GROWTH=.05;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=4;
MAXIN=5;
MAXOUT=5;
THRESHOLD=.20;
GROWTH=.05;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=4;
MAXIN=10;
MAXOUT=10;
THRESHOLD=.20;
GROWTH=.05;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

/*high growth rate*/
for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=4;
MAXIN=1;
MAXOUT=1;
THRESHOLD=.20;
GROWTH=.25;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

for (run=1;run<=RUN;run++){ /*experimental runs*/
SEARCH=4;
MAXIN=5;
MAXOUT=5;
THRESHOLD=.20;
GROWTH=.25;
printf("\nsearch=%3d run=%3d",SEARCH,run);
evolution_run();
}

```

```

}

for (run=1;run<=RUN;run++){ /*experimental runs*/
  SEARCH=4;
  MAXIN=10;
  MAXOUT=10;
  THRESHOLD=.20;
  GROWTH=.25;
  printf("\nsearch=%3d run=%3d",SEARCH,run);
  evolution_run();
}

}

/*****main function*****/
evolution_run()
{
  int i;

/*****initialize all storages*****/
  matrices_initial();

/*****create resources for each node along input and output dimensions*****/
  resource_initial();

/*****create internode contact matrix for each pair of nodes*****/
  inter_contact(); /*for integrated search*/

for (i=1;i<=NM;i++)
  syear[i]=random()%(YEAR-EYEAR)+1; /*nodes event starting year*/
/*at any year about one third facing events*/

/*****macro: event year process starts *****/
for (year=1;year<=YEAR;year++){ /*calendar year*/

  printf("\nyear=%3d",year);

  for (node=1;node<=NM;node++){
    tot_partner[node]=0; /*initialize first*/
    diff_partner[node]=0; /*number of different partners for the year*/
  }

/*****generate resource demanding events *****/
for (node=1;node<=NM;node++){
  location=node;
  event_generator();
}

/*****micro: time search process starts *****/
for (time=1;time<=TUNIT;time++){ /*make TUNIT=NM?*/

temp_t=time;

/*****at each time unit each node can find a partner if triggered*****/

```

```

/*****at the next time unit, if trigger still positive*****/
/*****each node can take turns to find the next partner again.*****/

/*****adjust resource match matrix for each pair of nodes after each round*/

resource_matrix();

/*for (i=1;i<=NM;i++){
  ran_node[i]=random()%NM;
}*/

last_partner[node]=node; /*last periods partner initialize as self*/

for (node=1;node<=NM;node++){

  location=node; /*taking turns or random if needed*/

  resource_check(); /*check resources to see if event triggers search*/

  if (trigger==1){ /*event resource demand not met; if met skip this node*/

if (SEARCH==1)
  satisficing_search();

if (SEARCH==2)
  economic_search();

if (SEARCH==3)
  behavioral_search();

if (SEARCH==4)
  integrated_search();

  resource_exchange();

  tot_partner[node]+=1; /*number of partners increases by one*/
}

  /*individual_output();*/ /*end of one node*/
}

} /*end of micro loop: for all nodes*/

/*****reallocating resources and updating matrices based on industry growth***/
industry_growth(); /*update relevant resource matrices*/

/*****event year output *****/

net_output(); /*calculating network measures*/

```

```

for (node=1;node<=NM;node++){
  location=node;
  macro_output();
}
} /*end of macro year*/
}

```

```

/*****
/*          Functions          */
/*****

```

```

/*****
/*          initialize all storages          */
/*****

```

```

matrices_initial()
{

```

```

int i,j,ri,ro,ero;

```

```

for(i=1;i<=NM;i++){
  rem_resout[i]=0;
  inires_in[i]=0;
  inires_out[i]=0;
  ex_resource[i]=0;

```

```

  tot_partner[i]=0;
  diff_partner[i]=0;
  last_partner[i]=0;
  curr_partner[i]=0;

```

```

  ownres[i]=0;
  differs[i]=0;
  last_exres[i]=0;
  new_exres[i]=0;
  shortage[i]=0;

```

```

  shole[i]=0;
  centr[i]=0;

```

```

  syear[i]=0;
  eyear[i]=0;

```

```

  for(ri=1;ri<=RI;ri++){
    resource_in[i][ri]=0;

```

```

    temp_resin[i][ri]=0;
  }

```

```

  for(ro=1;ro<=RO;ro++){
    resource_out[i][ro]=0;
    temp_resout[i][ro]=0;
  }

```

```

  for (ero=1;ero<=ERO;ero++){

```

```

    event[i][ero]=0;
}
}
for(i=1;i<=NM;i++){
for(j=1;j<=NM;j++){
    resource_match[i][j]=0;
    temp_resmatch[i][j]=0;
    contact[i][j]=0;
    temp_cont[i][j]=0;
    netcontact[i][j]=0;
    int_cont[i][j]=0;
    temp_intcont[i][j]=0;
}
}
}

/*****
/*          create initial state of all nodes          */
*****/
resource_initial()
{
int i,j,ri,ro;
int tempi,tempj;

for(i=1;i<=NM;i++){
    for(ri=1;ri<=RI;ri++){ /*input resource dimensions and amount*/
        resource_in[i][ri]=random()%(MAXIN+1); /*0 to MAXIN*/
        temp_resin[i][ri]=resource_in[i][ri];
    }

    for(ro=1;ro<=RO;ro++){ /*output resource dimensions and amount*/
        resource_out[i][ro]=random()%(MAXOUT+1); /*0 to maxout*/
        temp_resout[i][ro]=resource_out[i][ro];
    }
}

for (i=1;i<=NM;i++){ /*availability of output resources*/
    for(ro=1;ro<=RO;ro++) /*output resource dimensions and amount*/
        rem_resout[i]+=resource_out[i][ro];
}

for (i=1;i<=NM;i++){ /*record initial resources*/
    for (ri=1;ri<=RI;ri++)
        inires_in[i]+=resource_in[i][ri];
}

for(ro=1;ro<=RO;ro++)
    inires_out[i]+=resource_out[i][ro];
}
}

```

```

/*****
/*          record resource match for each pair of nodes          */
/*****
resource_matrix()
{
int i,j,ri,ro;

for(i=1;i<=NM;i++){
for(j=1;j<=NM;j++){
if (i!=j){
for(ri=1;ri<=RI;ri++){
for(ro=1;ro<=RO;ro++){
if ((ri==ro)&&(resource_in[i][ri]>0)){
resource_match[i][j]+=resource_out[j][ro]/(1.0*RO);/*based on available
output*/
temp_resmatch[i][j]+=resource_out[j][ro]/(1.0*RO);
}
}
}
}
}

for(i=1;i<=NM;i++){
for(j=1;j<=NM;j++){
int_cont[i][j]=resource_match[i][j]+contact[i][j]; /*integrated*/
temp_intcont[i][j]=int_cont[i][j];
}
}

}

/*****
/*          record internode contacts          */
/*****
inter_contact()
{
int i,j;

for (i=1;i<=NM;i++){
for (j=1;j<=NM;j++){

contact[i][j]=0; /*to update later*/
temp_cont[i][j]=0;
}
}

}

/*****
/*          generate an event with a focal location          */
/*****
event_generator()
{

```

```

int i,ero;
double m,s,x,temp1,temp2,temp3,temp,f;
int ff,temp_event;

/*****event demand value taking from normal distribution*****/

if ((year-syear[location]>=0)&&(year-syear[location]<=EYEAR)){ /*checking
starting year for the individual event*/

    x=year-syear[location];
    eyear[location]=ceil(x); /*take the integer value*/

    m=EYEAR/2.0; /*mean value*/ /*make each event lasting 10 years*/
    s=EYEAR/4.0; /*standard deviation*/

    temp1=1/sqrt(2*3.1416*s*s);
    temp2=-0.5*(x-m)*(x-m)/(s*s);
    temp3=exp(temp2);
    temp=temp1*temp3;

    f=500*temp; /*ranging from about 0 to 20 in values*/
    ff=ceil(f); /*round them to the next integer*/

    for(ero=1;ero<=ERO;ero++){ /*event resource dimensions and demand*/
        temp_event=ff; /*same amount of resource demands; (random()%ff);*/ /*could be
zero?*/
        event[location][ero]=random()%temp_event; /*with own event year dimension*/
    }
}

else { /*not in event year yet*/
    for(ero=1;ero<=ERO;ero++){ /*event resource dimensions and demand*/
        event[location][ero]=0;
    }
}

/*****
/*          check focal nodes resources again the event          */
/*****
resource_check() /*check at each time unit*/
{
int i,ro;
int res_avail[NM+1];

differs[location]=0; /*difference between the focal node and the event*/
ownres[location]=0; /*focal node total available output resources*/
event_dmd[location]=0;

for (ro=1;ro<=RO;ro++){
    differs[location]+=(event[location][ro]-resource_out[location][ro]);
    ownres[location]+=(resource_out[location][ro]);
}
}

```

```

for (ro=1;ro<=ERO;ro++){ /*external event demand*/
  event_dmd[location]+=event[location][ro]; /*resource demand by event*/
}

if (event_dmd[location]>0){
  if (ownres[location]>=inires_out[location]/2.0){
    trigger=0; /*no need for external search*/

    for (ro=1;ro<=RO;ro++){
      if (resource_out[location][ro]>0){
        event[location][ro]=event[location][ro]-1;
        event_dmd[location]=event_dmd[location]-1;
        resource_out[location][ro]=resource_out[location][ro]-1;
        ownres[location]=ownres[location]-1;
      }
    }
  }
  else{/*own resources below half of original amount*/
    shortage[location]=1; /*need external help*/
    trigger=1;
  }
}
else{
  shortage[location]=0; /*no event resource demand*/
  trigger=0;
}

res_avail[location]=0; /*resource availability signal*/

for (i=1;i<=NM;i++)
  if ((i!=location)&&(rem_resout[i]>0))
    res_avail[location]=1; /*some resources available in the network*/

if (res_avail[location]==0)
  curr_partner[node]=location; /*can only rely on self*/

if (ownres[location]<=0) /*no more resources by the focal node either*/
  trigger=0;

/*what if resource needs are all met after several time units*/
}

/*****
/*      partner search based on past interactions      */
/*****
behavioral_search()
{
int i,j,ri,ro;
int max_match,max_partner;
int temp_sig;

/*assuming that within one normal time period, a node can locate one node and
exchange resources with that node*/

```

```

/*locate a potential partner that has the most previous contacts and also has
available output resources to offer*/

/*during next time unit, a next potential partner with the most contacts at that
time will be chosen, which may or may not be the same partner*/

max_partner=random()%NM+1;
max_match=(-10);

temp_sig=0; /*check to see if there are potential partners with resources*/
for (j=1;j<=NM;j++)
    if ((j!=location)&&(rem_resout[j]>0))
        temp_sig=1;

if (temp_sig==0)
    max_partner=location; /*rely on self*/
else {
    for (j=1;j<=NM;j++){ /*find the most contacted and exchange partner*/

        if ((j!=location)&&(rem_resout[j]<=0)){/*partner resources depleted*/
            temp_cont[location][j]=(-10); /*dropped to the end of list*/
            max_partner=random()%NM+1; /*choose a random one*/
        }
        else
            if ((j!=location)&&(temp_cont[location][j]>=max_match)){
                max_partner=j;
                max_match=temp_cont[location][j];
            }
    }
}

curr_partner[location]=max_partner;

if (curr_partner[location]!=last_partner[location])
    diff_partner[location]+=1;

last_partner[location]=curr_partner[location]; /*make current partner as last
partner for next period*/
}

/*****
/*      partner search based on resource matches      */
*****/
economic_search()
{
int i,j,ri,ro;
int max_match,max_partner;
int temp_sig;

/*assuming that within one normal time period, a node can locate one node and
exchange resources with that node*/

/*locate a potential partner that has the highest resource match and also has
available output resources to offer*/

```

```

/*during next time unit, a next potential partner with the most resource match
will be chosen, which may or may not be the same partner*/

max_partner=random()%NM+1;
max_match=(-10);

temp_sig=0; /*check to see if there are potential partners with resources*/
for (j=1;j<=NM;j++)

    if ((j!=location)&&(rem_resout[j]>0))
        temp_sig=1;

if (temp_sig==0)
    max_partner=random()%NM+1; /*choose another partner*/
else {
    for (j=1;j<=NM;j++){ /*find the next most resource matched partner*/
        if ((j!=location)&&(rem_resout[j]<=0)){/*partner resources depleted*/
            temp_resmatch[location][j]=(-10); /*dropped to the end of list*/
            max_partner=random()%NM+1; /*choose a random one*/
        }
        else
            if ((j!=location)&&(temp_resmatch[location][j]>=max_match)){
                max_partner=j;
                max_match=temp_resmatch[location][j];
            }
        }
    }
}

curr_partner[location]=max_partner;

if (curr_partner[location]!=last_partner[location])
    diff_partner[location]+=1;

last_partner[location]=curr_partner[location]; /*make current partner as last
partner for next period*/
}

/*****
/*      partner search based on minimal resource requirement      */
/*****
satisficing_search()
{
int i,j,ri,ro;
int max_match,max_partner;
int temp_sig;

/*assuming that within one normal time period, a node can locate one node and
exchange resources with that node*/

/*locate a random partner.  If the partner has the minimal resources, take that
one*/

max_partner=random()%NM+1;

```

```

for (i=1;i<=NM;i++){
  if (max_partner!=location){
    if (rem_resout[max_partner]>0)
      i=NM+1; /*stop the search process*/
    else
      max_partner=random()%NM+1;
  }
  else
    max_partner=random()%NM+1;
}

curr_partner[location]=max_partner;

if (curr_partner[location]!=last_partner[location])
  diff_partner[location]+=1;

last_partner[location]=curr_partner[location]; /*make current partner as last
partner for next period*/
}

/*****
/*      partner search based on both past interactions and resource match      */
*****/
integrated_search()
{
int i,j,ri,ro;
int max_match,max_partner;
int temp_sig;

/*assuming that within one normal time period, a node can locate one node and
exchange resources with that node*/

/*locate a potential partner that has the most integrated score based on
contacts and resource match, and also has available output resources to offer*/

/*during next time unit, a next potential partner with the most integrated score
at that time will be chosen, which may or may not be the same partner*/

  max_partner=random()%NM+1;
  max_match=(-10);

temp_sig=0; /*check to see if there are potential partners with resources*/
for (j=1;j<=NM;j++)
  if ((j!=location)&&(rem_resout[j]>0))
    temp_sig=1;

if (temp_sig==0)
  max_partner=location; /*rely on self*/
else {
  for (j=1;j<=NM;j++){ /*find the one with highest score*/

    if ((j!=location)&&(rem_resout[j]<=0)){/*partner resources depleted*/

```



```

temp_trans=2*temp_inoutratio;

if (event[location][ro]-temp_trans<0){ /*event has only little demand left, keep
the extra resources with the firm*/
  resource_out[location][ro]=resource_out[location][ro]-event[location][ro];
  event[location][ro]=0;
}
else{ /*event has more demand left and transfer all to the event*/
  resource_out[location][ro]= resource_out[location][ro]-temp_trans;
  event[location][ro]=event[location][ro]-temp_trans; /*event has more demand
left and transfer both units to the event*/
}

  rem_resout[partner]=rem_resout[partner]-2; /*adjust remaining output resources
for the partner*/

  netcontact[location][partner]+=2; /*record resources provided by the partner*/

  ex_resource[location]+=2; /*record resources accumulated by the focal node*/

  resource_out[partner][ro]=resource_out[partner][ro]-2; /*partner resource
reduced*/

  contact[location][partner]+=2; /*update contact matrix, based on dimensions*/
/*means more units exchanged, more contacts recorded*/

  contact[partner][location]+=2; /*when the partner later searches for resources,
the current focal node will have a higher chance of being selected*/

}
else{
  resource_in[location][ro]+=1; /*transfer to the input dimension of focal node*/

  resource_out[location][ro]+=(temp_inoutratio);

temp_trans=1*temp_inoutratio;

if (event[location][ro]-temp_trans<0){ /*event has only little demand left, keep
the extra resources with the firm*/
  resource_out[location][ro]= resource_out[location][ro]-event[location][ro];
  event[location][ro]=0;
}
else{ /*event has more demand left and transfer all to the event*/
  resource_out[location][ro]= resource_out[location][ro]-temp_trans;
  event[location][ro]=event[location][ro]-temp_trans; /*event has more demand
left and transfer both units to the event*/
}

  rem_resout[partner]=rem_resout[partner]-1; /*adjust remaining output resources
for the partner*/

  netcontact[location][partner]+=1; /*record resources provided by the partner*/

  ex_resource[location]+=1; /*record resources accumulated by the focal node*/

```

```

resource_out[partner][ro]=resource_out[partner][ro]-1; /*partner resource
reduced*/

contact[location][partner]+=1; /*update contact matrix, based on dimensions*/
/*means more units exchanged, more contacts recorded*/

contact[partner][location]+=1; /*when the partner later searches for resources,
the current focal node will have a higher chance of being selected*/
}

if (partner!=location){
    temp_cont[location][partner]=contact[location][partner];

temp_intcont[location][partner]=resource_match[location][partner]+contact[locati
on][partner];
    temp_cont[partner][location]=contact[partner][location];

temp_intcont[partner][location]=resource_match[partner][partner]+contact[partner
][location];
}
}
}

}

/*****
/*      resource growth and additions after each macro period: event year      */
/*****
industry_growth()
{
int i,ri,ro;
double temp_inoutratio,temp_growth;

/*for transform into output*/
if (inires_in[location]>0)
    temp_inoutratio=inires_out[location]/inires_out[location];
else
    temp_inoutratio=0;

for(i=1;i<=NM;i++){
    for(ri=1;ri<=RI;ri++){ /*input resource dimensions and amount*/
        resource_in[i][ri]+=GROWTH*(inires_in[i]/(1.0*RI));/*same type growth*/
        temp_resin[i][ri]=resource_in[i][ri];
    }

    for(ro=1;ro<=RO;ro++){ /*output resource dimensions and amount*/
        temp_growth=temp_inoutratio*GROWTH*(inires_in[i]/(1.0*RI));
        resource_out[i][ro]+=temp_growth; /*same type growth*/
        temp_resout[i][ro]=resource_out[i][ro];
    }
}

for (i=1;i<=NM;i++){ /*availability of output resources*/
    rem_resout[i]=0; /*recalculate */
    for(ro=1;ro<=RO;ro++) /*output resource dimensions and amount*/
        rem_resout[i]+=resource_out[i][ro];
}

```

```
}  
}
```

```
/*  
*****  
output after each micro step: time period  
*****  
*/
```

```
individual_output()  
{
```

```
int i;
```

```
op=fopen("evr7_all6.out","a");
```

```
if (curr_partner[location]==0)  
curr_partner[location]=node;
```

```
if (!(event_dmd[location]>0))  
event_dmd[location]=0;
```

```
if (!(ownres[location]>0))  
ownres[location]=0;
```

```
if (rem_resout[location]<0)  
rem_resout[location]=0;
```

```
if (ex_resource[location]<0)  
ex_resource[location]=0;
```

```
fprintf(op,"\n%3d %3d %3d %3d %3d %3d %3d %6.2f %6.2f %3d %3d %3d %10.2f %10.2f  
%6.2f %3d %3d %4d %10.2f %10.2f %8.2f %8.2f",  
SEARCH,run,NM,RI,RO,MAXIN,MAXOUT,THRESHOLD,GROWTH,year,eyear[location],temp_t,ev  
ent_dmd[location],ownres[location],shortage[location],location,curr_partner[loca  
tion],tot_partner[location],ex_resource[location],rem_resout[location],inires_in  
[location],inires_out[location]);
```

```
fclose(op);
```

```
}
```

```
/*  
*****  
output after each macro step: event year  
*****  
*/
```

```
macro_output()  
{
```

```
int i;
```

```
op=fopen("evr7_yall6.out","a");
```

```
if (curr_partner[location]==0)  
curr_partner[location]=node;
```

```
new_exres[location]=ex_resource[location]-last_exres[location];  
last_exres[location]=ex_resource[location]; /*make it last year resource for  
next event year*/
```

```
if (!(event_dmd[location]>0))
```

```

    event_dmd[location]=0;
if (!(ownres[location]>0))
    ownres[location]=0;
if (new_exres[location]<0)
    new_exres[location]=0;
if (ex_resource[location]<0)
    ex_resource[location]=0;

fprintf(op,"\n%3d %3d %3d %3d %3d %3d %3d %6.2f %6.2f %3d %3d %4d %10.2f %10.2f
%3d %3d %10.2f %8.2f %8.2f %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f",
SEARCH,run,NM,RI,RO,MAXIN,MAXOUT,THRESHOLD,GROWTH,year,eyear[location],tot_partn
er[location],event_dmd[location],ownres[location],location,diff_partner[location
],new_exres[location],ex_resource[location],inires_in[location],inires_out[locat
ion],shole[location],centr[location],g_cenmin,g_cenmax,g_cenmm,g_stderr);

/*notice eyear[location]*/

fclose(op);

}

/*****
/*          network measures after each macro step: event year          */
*****/
net_output()
{
int ii,jj,kk;
double pcentr,max_centr,min_centr,sum_centr_num,sum_centr_den;
double pcentr2,max_centr2,min_centr2,sum_centr_num2,sum_centr_den2;
double pcentr3,max_centr3,min_centr3,sum_centr_num3,sum_centr_den3;
double sum_diff_sq,ave_centr,sum_diff_sqrt;

/*number of structural holes*/
for (ii=1;ii<=NM;ii++){
    shole[ii]=0;
    for (jj=1;jj<=NM;jj++){
        if (jj!=ii){

            for (kk=1;kk<=NM;kk++){
                if ((kk!=ii)&&(netcontact[jj][kk]<=0)){
                    /*jj & kk are not directly contacted*/
                    if ((netcontact[ii][jj]>=1)&&(netcontact[ii][kk]>=1))
                        shole[ii]+=1;
                }
            }
        }
    }
}

/*point centrality based on weighted average contacts*/
for (ii=1;ii<=NM;ii++){
    pcentr=0;

```

```

    for (jj=1;jj<=NM;jj++){
        pcentr+=netcontact[jj][ii]; /*may use other measure for tie*/
    }
    centr[ii]=pcentr/(NM-1);
}

/*graph centralities based on minimal, maximum, or both point centralities*/
ave_centr=0;

for (ii=1;ii<=NM;ii++){
    ave_centr+=centr[ii];
}
ave_centr=ave_centr/(1.0*NM);

max_centr=-20.0;
for (ii=1;ii<=NM;ii++){
    if (max_centr<centr[ii])
        max_centr=centr[ii];
}

min_centr=20.0;
for (ii=1;ii<=NM;ii++){
    if (min_centr>centr[ii])
        min_centr=centr[ii];
}

sum_centr_num=0;
sum_centr_den=0;

sum_centr_num2=0;
sum_centr_den2=0;

sum_centr_num3=0;
sum_centr_den3=0;

sum_diff_sq=0;
sum_diff_sqrt=0;

for (ii=1;ii<=NM;ii++){
    sum_centr_num+=(centr[ii]-min_centr);
    sum_centr_den+=((NM-1)*1.0);

    sum_centr_num2+=(max_centr-centr[ii]);
    sum_centr_den2+=((NM-1)*1.0);

    sum_centr_num3+=(max_centr-centr[ii]);
    sum_centr_den3+=(max_centr-min_centr);

    sum_diff_sq+=(centr[ii]-ave_centr)*(centr[ii]-ave_centr);
    sum_diff_sqrt=sqrt(sum_diff_sq);
}
/*graph centrality based on relations to minimum point centrality*/
if (sum_centr_den>0)
    g_cenmin=sum_centr_num/sum_centr_den;
else
    g_cenmin=0;

```

```
/*graph centrality based on relations to maximum point centrality*/
if (sum_centr_den2>0)
  g_cenmax=sum_centr_num2/sum_centr_den2;
else
  g_cenmax=0;

/*graph centrality based on relations to minimum and maximum point
centralities*/
if (sum_centr_den3>0)
  g_cenmm=sum_centr_num3/sum_centr_den3;
else
  g_cenmm=0;

/*standard errors for graph centrality*/
if (sqrt(NM-1)>0)
  g_stderr=sum_diff_sqrt/sqrt(NM-1);
else
  g_stderr=0;

}
```