

11.3 Movement of Ants

File: *Ants.nb*

Introduction to Computational Science: Modeling and Simulation for the Sciences
Angela B. Shiflet and George W. Shiflet
Wofford College
© 2006 by Princeton University Press

Based on Gaylord, Richard J., and Kazume Nishidate. 1996. "Chemotaxis." *Modeling Nature: Cellular Automata Simulations with Mathematica*. New York: TELOS/Springer-Verlag: chap. 12, 121-130.

Development of Ants Program

Lattice (matrix, grid)

Each lattice site has value of an ordered pair of intergers

1st component - level of chemical attractant, nonnegative integer

2nd component -

EMPTY = "T" - empty site

NORTH = "N" - N-facing ant

EAST = "E" - E-facing ant

SOUTH = "S" - S-facing ant

WEST = "W" - W-facing ant

Initially, 0 level of chemical attractant at all sites and random direction.

probAnt = probability of cell being occupied

```
(* Constants *)  
EMPTY = "T"; NORTH = "N"; EAST = "E"; SOUTH = "S";  
WEST = "W";
```

Update rules

1. Apply rules for orienting ant

sense[site, N, E, S, W]

Empty site remains unchanged

Ant orients itself so as to face nearest neighbor site that has highest chemical attractant level.

If two or more nearest neighbor sites have equal amount, pick at random.

2. Apply rules for diffusing ant on lattice

walk[site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww]

When ant moves, leaves behind chemical scent to attract other ants. (increase scent value by 1)

In absence of ant, chemical scent that has been laid down dissipates. (Decrease by 1)

Sensing rules

■ Orientation rule - *sense* function:

Empty site remains unchanged

```
sense[{a_, EMPTY}, _, _, _, _] := {a, EMPTY};
```

■ Orientation rule - *sense* function:

Ant turns to highest scent in N,E,S,W direction; at random to equal maximum locations

```
sense[{a_, _}, {n_, _}, {e_, _}, {s_, _}, {w_, _}] :=
Module[{lst, mx, firstLst, posList, lng, rndPos},
  lst = {n, e, s, w};
  mx = Max[lst];
  posList = Flatten[Position[lst, mx]];
  lng = Length[posList];
  rndPos = Random[Integer, {1, lng}];
  dir = {NORTH, EAST, SOUTH, WEST};
  {a, dir[[posList[[rndPos]] ]]}
];
```

Walking rules

■ Rules for movement of ant - *walk* function:

If no ant, decrease strength of scent by 1, but not lower than 0.

```
walk[{a_, EMPTY}, _, _, _, _, _, _, _, _, _, _, _] := {Max[a - 1, 0], EMPTY};
```

■ Have rules for facing each direction.

If move, increase by 1 the amount of chemical in site vacated.

```
walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, _, _, _, _] :=  
  {If[a > 0, a + 1, 0], EMPTY};  
  
walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, _, _, _, _, _] :=  
  {If[a > 0, a + 1, 0], EMPTY};  
  
walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, _, _, _, _] :=  
  {If[a > 0, a + 1, 0], EMPTY};  
  
walk[{a_, WEST}, _, _, _, {_, EMPTY}, _, _, _, _, _, _] :=  
  {If[a > 0, a + 1, 0], EMPTY};
```

- If movement would cause a collision, do not vacate site and do not change amount of chemical at site.

```

walk[{a_, NORTH}, {_, EMPTY}, _, _, _, {_, WEST}, _, _, _, _, _] :=
  {a, NORTH};

walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, {_, EAST}, _, _, _, _] :=
  {a, NORTH};

walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, _, {_, SOUTH}, _, _, _] :=
  {a, NORTH};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, {_, SOUTH}, _, _, _, _, _] :=
  {a, EAST};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, {_, NORTH}, _, _, _, _, _] :=
  {a, EAST};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, _, _, {_, WEST}, _, _] :=
  {a, EAST};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, {_, WEST}, _, _, _, _, _] :=
  {a, SOUTH};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, {_, EAST}, _, _, _, _, _] :=
  {a, SOUTH};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, _, _, {_, NORTH}, _] :=
  {a, SOUTH};

walk[{a_, WEST}, _, _, {_, EMPTY}, _, _, {_, NORTH}, _, _, _, _, _] :=
  {a, WEST};

walk[{a_, WEST}, _, _, {_, EMPTY}, _, _, {_, SOUTH}, _, _, _, _] :=
  {a, WEST};

walk[{a_, WEST}, _, _, {_, EMPTY}, _, _, _, _, _, {_, EAST}] :=
  {a, WEST};

```

- For other collisions, stay put with no increase in chemical.

```

walk[{a_, b_}, _, _, _, _, _, _, _, _, _, _] := {a, b};

```

- If site is not occupied and an ant to the N, E, S, or W wants to move into the site, let it do so. Keep the current level of chemical at site.

```
walk[{a_, EMPTY}, {_, SOUTH}, _, _, _, _, _, _, _, _, _, _] := {a, SOUTH};
walk[{a_, EMPTY}, _, {_, WEST}, _, _, _, _, _, _, _, _, _] := {a, WEST};
walk[{a_, EMPTY}, _, _, {_, NORTH}, _, _, _, _, _, _, _, _] := {a, NORTH};
walk[{a_, EMPTY}, _, _, _, {_, EAST}, _, _, _, _, _, _, _] := {a, EAST};
```

- If two ants want to move into site, do not allow either to do so and decrease the value of the chemical scent by 1 to minimum of 0.

```
walk[{a_, EMPTY}, {_, SOUTH}, {_, WEST}, _, _, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};

walk[{a_, EMPTY}, {_, SOUTH}, _, {_, NORTH}, _, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};

walk[{a_, EMPTY}, {_, SOUTH}, _, _, {_, EAST}, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};

walk[{a_, EMPTY}, _, {_, WEST}, {_, NORTH}, _, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};

walk[{a_, EMPTY}, _, {_, WEST}, _, {_, EAST}, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};

walk[{a_, EMPTY}, _, _, {_, NORTH}, {_, EAST}, _, _, _, _, _, _, _] :=
  {Max[a - 1, 0], EMPTY};
```

Boundaries

Periodic boundary conditions in this case

Infinite system

Extend boundary by 1 for *sense* and by 2 for *walk*

■ Function to return an $(n + 2)$ -by- $(n + 2)$ matrix for periodic boundaries

mat is an n -by- n matrix.

```
Clear[extendLat1]
extendLat1[mat_] := Module[{matNS, trans, transEW},
  (* glue on wrap of N & S rows *)
  matNS = Join[Take[mat, -1], mat, Take[mat, 1]];

  (* glue on wrap of E & W columns assuming original matrix is square *)
  trans = Transpose[matNS];
  transEW = Join[Take[trans, -1], trans, Take[trans, 1]];

  Transpose[transEW]
];
```

■ Function to return an $(n + 4)$ -by- $(n + 4)$ matrix for periodic boundaries

mat is an n -by- n matrix.

```
Clear[n, mat, extendLat2]

(* Function to return an (n + 4)-by-(n + 4) matrix *)
(* for periodic boundaries mat - n-by-n matrix *)
extendLat2[mat_] := Module[{matNS, trans, transEW},
  (* glue on wrap of N & S rows by 2 *)
  matNS = Join[Take[mat, -2], mat, Take[mat, 2]];

  (* glue on wrap of E & W columns by 2 assuming original matrix
  is square *)
  trans = Transpose[matNS];
  transEW = Join[Take[trans, -2], trans, Take[trans, 2]];

  Transpose[transEW]
];
```

Functions to apply a function parameter to extended matrix

■ Function to apply a function parameter to a matrix extended by one cell in each direction

Function returns a matrix of applications of function *fnc* (*fnc*[*site*, *N*, *E*, *S*, *W*]) to each element (*site*) of extended matrix *matExt* except for the first and last rows and columns

```
applyExtended1[fnc_, matExt_] :=  
  Module[{n, site, N, E, S, W, i, j},  
    n = Length[matExt] - 2;  
  
    Table[  
      site = matExt[[i, j]];  
      N = matExt[[i - 1, j]];  
      E = matExt[[i, j + 1]];  
      S = matExt[[i + 1, j]];  
      W = matExt[[i, j - 1]];  
      fnc[site, N, E, S, W],  
  
      {i, 2, n + 1}, {j, 2, n + 1}]  
    ];
```

■ Function to apply a function parameter to a matrix extended by two cells in each direction

Function returns a matrix of applications of function *fnc* (*fnc*[*site*, *N*, *E*, *S*, *W*, *NE*, *SE*, *SW*, *NW*, *Nn*, *Ee*, *Ss*, *Ww*]) to each element (*site*) of extended matrix *matExt* except for the first two and last two rows and columns

```

applyExtended2[fnc_, matExt_] :=
Module[{n, site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww, i, j},
  n = Length[matExt] - 4;

  Table[
    site = matExt[[i, j]];
    N = matExt[[i - 1, j]];
    E = matExt[[i, j + 1]];
    S = matExt[[i + 1, j]];
    W = matExt[[i, j - 1]];
    NE = matExt[[i - 1, j + 1]];
    SE = matExt[[i + 1, j + 1]];
    SW = matExt[[i + 1, j - 1]];
    NW = matExt[[i - 1, j - 1]];
    Nn = matExt[[i - 2, j]];
    Ee = matExt[[i, j + 2]];
    Ss = matExt[[i + 2, j]];
    Ww = matExt[[i, j - 2]];

    fnc[site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww],

    {i, 3, n + 2}, {j, 3, n + 2}
  ];

```

Ants simulation

```

Clear[ants, grid, t, gridList, emat1, gridSense, emat2]
ants[initialGrid_, t_] :=
Module[{grid, gridList, emat1, gridSense, emat2},
  grid = initialGrid;
  gridList = {grid};
  Do[
    emat1 = extendLat1[grid];
    gridSense = applyExtended1[sense, emat1];
    emat2 = extendLat2[gridSense];
    grid = applyExtended2[walk, emat2];
    gridList = Append[gridList, grid],
    {t}
  ];
  gridList
];

```

Animation of simulation

- Function to display a list of grids; show ant as red color with degree of color depending on amount of chemical in cell; show empty cell with shade of gray

```

Clear[showGraphs]
showGraphs[graphList_, maxChem_] :=
Module[{chemAmounts, k, g, siteColorAmt, lng},
  lng = Length[graphList[[1]]];
  Do[
    g = graphList[[k]];
    Show[Graphics[RasterArray[Table[
      siteColorAmt = 1.0 - (g[[i, j, 1]] * 0.5) / maxChem;
      If[g[[i, j, 2]] == EMPTY,
        RGBColor[siteColorAmt, siteColorAmt, siteColorAmt],
        RGBColor[siteColorAmt, 0, 0]],
      {i, lng}, {j, lng}
    ], AspectRatio → Automatic]],
    {k, 1, Length[graphList]}
  ]
];

```

Run Program

■ Initialization of grid

```
n = 17;
probAnt = 0.05; (* Probability of an ant in a cell *)
dir = {NORTH, EAST, SOUTH, WEST};
maxChem = 50;

SeedRandom[9];
initialGrid =
  Table[{0, If[Random[] < probAnt, dir[[Random[Integer, {1, 4}]]], EMPTY]},
    {n}, {n}];
Do[
  initialGrid[[IntegerPart[n/2] + 1, j]] =
    {IntegerPart[maxChem * j / n], EMPTY},
  {j, 1, n}]
```

■ Perform simulation and animation

```
graphList = ants[initialGrid, 20];
```

```
showGraphs[graphList, maxChem];
```

Ants Functions Together

■ Program

```
Clear[sense, walk, extendLat1, extendLat2, applyExtended1,
      applyExtended2, n, mat, ants, EMPTY, NORTH, EAST, SOUTH, WEST];
Clear[ants, grid, t, gridList, emat1, gridSense, emat2, showGraphs];
```

```
(* Constants *)
```

```
EMPTY = "T"; NORTH = "N"; EAST = "E"; SOUTH = "S";
WEST = "W";
```

```
(*****)
(***) Senseing Rules (***)
(* Empty site unchanged *)
```

```
sense[{a_, EMPTY}, _, _, _, _] := {a, EMPTY};
```

```
(* Ant turns to highest scent in N,E,S,W direction; *)
(* at random to equal maximum locations *)
```

```
sense[{a_, _}, {n_, _}, {e_, _}, {s_, _}, {w_, _}] :=
Module[{lst, mx, firstLst, posList, lng, rndPos},
  lst = {n, e, s, w};
  mx = Max[lst];
  posList = Flatten[Position[lst, mx]];
  lng = Length[posList];
  rndPos = Random[Integer, {1, lng}];
  dir = {NORTH, EAST, SOUTH, WEST};
  {a, dir[[posList[[rndPos]]]]}
];
```

```
(*****)
(***) Walking Rules (***)
(* If no ant, decrease strength of scent by 1, but not lower than 0. *)
```

```
walk[{a_, EMPTY}, _, _, _, _, _, _, _, _, _, _] := {Max[a - 1, 0], EMPTY};
```

```
(* If move and chemical at site, increase the amount of chemical *)
(* in site vacated by 1. Rules for facing each direction *)
```

```
walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, _, _, _] :=
{a + 1, EMPTY};
```

```

walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, _, _, _, _, _] := {a + 1, EMPTY};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, _, _, _, _] :=
  {a + 1, EMPTY};

walk[{a_, WEST}, _, _, _, {_, EMPTY}, _, _, _, _, _, _] := {a + 1, EMPTY};

(* If movement would cause a collision, do not vacate site and *)
(* do not change amount of chemical at site. *)
walk[{a_, NORTH}, {_, EMPTY}, _, _, _, {_, WEST}, _, _, _, _, _] :=
  {a, NORTH};

walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, {_, EAST}, _, _, _] :=
  {a, NORTH};

walk[{a_, NORTH}, {_, EMPTY}, _, _, _, _, _, _, {_, SOUTH}, _, _] :=
  {a, NORTH};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, {_, SOUTH}, _, _, _, _, _] :=
  {a, EAST};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, {_, NORTH}, _, _, _, _] :=
  {a, EAST};

walk[{a_, EAST}, _, {_, EMPTY}, _, _, _, _, _, {_, WEST}, _, _] :=
  {a, EAST};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, {_, WEST}, _, _, _, _] :=
  {a, SOUTH};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, {_, EAST}, _, _, _] :=
  {a, SOUTH};

walk[{a_, SOUTH}, _, _, {_, EMPTY}, _, _, _, _, _, {_, NORTH}, _] :=
  {a, SOUTH};

walk[{a_, WEST}, _, _, _, {_, EMPTY}, _, _, {_, NORTH}, _, _, _] :=
  {a, WEST};

walk[{a_, WEST}, _, _, _, {_, EMPTY}, _, _, _, {_, SOUTH}, _, _] :=
  {a, WEST};

walk[{a_, WEST}, _, _, _, {_, EMPTY}, _, _, _, _, _, {_, EAST}] :=
  {a, WEST};

(* For other collisions, stay put with no increase in chemical. *)
walk[{a_, b_}, _, _, _, _, _, _, _, _, _] := {a, b};

(* If site is not occupied and an ant to the N, E, S, or W wants to *)
(* move into the site, let it do so. Keep the current level of chemical,

```

```

    at site.*)
    walk[{a_, EMPTY}, {_, SOUTH}, _, _, _, _, _, _, _, _, _, _] := {a, SOUTH};

    walk[{a_, EMPTY}, _, {_, WEST}, _, _, _, _, _, _, _, _, _] := {a, WEST};

    walk[{a_, EMPTY}, _, _, {_, NORTH}, _, _, _, _, _, _, _, _] := {a, NORTH};

    walk[{a_, EMPTY}, _, _, _, {_, EAST}, _, _, _, _, _, _, _] := {a, EAST};

    (* If 2 ants want to move into site, do not allow either to do so *)
    (* and decrease the value of the chemical scent by 1 to minimum
       of 0. *)
    walk[{a_, EMPTY}, {_, SOUTH}, {_, WEST}, _, _, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

    walk[{a_, EMPTY}, {_, SOUTH}, _, {_, NORTH}, _, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

    walk[{a_, EMPTY}, {_, SOUTH}, _, _, {_, EAST}, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

    walk[{a_, EMPTY}, _, {_, WEST}, {_, NORTH}, _, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

    walk[{a_, EMPTY}, _, {_, WEST}, _, {_, EAST}, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

    walk[{a_, EMPTY}, _, _, {_, NORTH}, {_, EAST}, _, _, _, _, _, _] :=
      {Max[a - 1, 0], EMPTY};

```

```

(*****)
(** Extension functions **)
(* Function to return an (n + 2)-by-
   (n + 2) matrix with periodic boundaries for mat, an n-by-n matrix *)

extendLat1[mat_] := Module[{matNS, trans, transeW},
  (* glue on wrap of N & S rows *)
  matNS = Join[Take[mat, -1], mat, Take[mat, 1]];

  (* glue on wrap of E & W columns assuming original matrix is square *)
  trans = Transpose[matNS];
  transeW = Join[Take[trans, -1], trans, Take[trans, 1]];

  Transpose[transeW]
];

```

```

(* Function to return an (n + 4)-by-
(n + 4) matrix with periodic boundaries for mat, an n-by-n matrix *)

extendLat2[mat_] := Module[{matNS, trans, transEW},
  (* glue on wrap of N & S rows by 2 *)
  matNS = Join[Take[mat, -2], mat, Take[mat, 2]];

  (* glue on wrap of E & W columns by 2 assuming original matrix
  is square *)
  trans = Transpose[matNS];
  transEW = Join[Take[trans, -2], trans, Take[trans, 2]];

  Transpose[transEW]
];

```

```

(*****)
(* Function to return a matrix of applications of function fnc *)
(* (fnc[site, N, E, S, W]) to each element (site) of extended matrix *)
(* matExt except for the first and last rows and columns *)

applyExtended1[fnc_, matExt_] :=
  Module[{n, site, N, E, S, W, i, j},
    n = Length[matExt] - 2;

    Table[
      site = matExt[[i, j]];
      N = matExt[[i - 1, j]];
      E = matExt[[i, j + 1]];
      S = matExt[[i + 1, j]];
      W = matExt[[i, j - 1]];
      fnc[site, N, E, S, W],

      {i, 2, n + 1}, {j, 2, n + 1}
    ];

```

```
(* Function to return a matrix of applications of function fnc *)
(* (fnc[site, N, E, S, W, NE,SE,SW,NW,Nn,Ee,Ss,Ww]) to each
element (site) of *) (* extended matrix matExt except for the
first and last rows and columns *)

applyExtended2[fnc_, matExt_] :=
Module[{n, site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww, i, j},
  n = Length[matExt] - 4;

  Table[
    site = matExt[[i, j]];
    N = matExt[[i - 1, j]];
    E = matExt[[i, j + 1]];
    S = matExt[[i + 1, j]];
    W = matExt[[i, j - 1]];
    NE = matExt[[i - 1, j + 1]];
    SE = matExt[[i + 1, j + 1]];
    SW = matExt[[i + 1, j - 1]];
    NW = matExt[[i - 1, j - 1]];
    Nn = matExt[[i - 2, j]];
    Ee = matExt[[i, j + 2]];
    Ss = matExt[[i + 2, j]];
    Ww = matExt[[i, j - 2]];

    fnc[site, N, E, S, W, NE, SE, SW, NW, Nn, Ee, Ss, Ww],

    {i, 3, n + 2}, {j, 3, n + 2}
  ];
```

```

(*****)
(** Function to drive ant simulation **)
(**** Note: Cannot place a parameter on the left side of an
    equal sign ****)

ants[initialGrid_, t_] :=
  Module[{grid, gridList, emat1, gridSense, emat2},
    grid = initialGrid;
    gridList = {grid};
    Do[
      emat1 = extendLat1[grid];
      gridSense = applyExtended1[sense, emat1];
      emat2 = extendLat2[gridSense];
      grid = applyExtended2[walk, emat2];
      gridList = Append[gridList, grid],
      {t}
    ];
    gridList
  ];

```

```

(*****)
(* Function to display a list of grids show ant as red color *)
(* with degree of color depending on amount of chemical in cell; *)
(* show empty cell with shade of gray *)

showGraphs[ graphList_, maxChem_] :=
  Module[{chemAmounts, k, g, siteColorAmt, lng},
    lng = Length[graphList[[1]]];
    Do[
      g = graphList[[k]];
      Show[Graphics[RasterArray[Table[
        siteColorAmt = 1.0 - (g[[i, j, 1]] * 0.5) / maxChem;
        If[g[[i, j, 2]] == EMPTY,
          RGBColor[siteColorAmt, siteColorAmt, siteColorAmt],
          RGBColor[siteColorAmt, 0, 0]],
        {i, lng}, {j, lng}
      ], AspectRatio -> Automatic]],
      {k, 1, Length[graphList]}
    ]
  ];

```

■ Initialization of grid

```
n = 17;
probAnt = 0.05;
dir = {NORTH, EAST, SOUTH, WEST};
maxChem = 50;

SeedRandom[9];
initialGrid =
  Table[{0, If[Random[] < probAnt, dir[[Random[Integer, {1, 4}]]], EMPTY]},
    {n}, {n}];
Do[
  initialGrid[[IntegerPart[n/2] + 1, j]] =
    {IntegerPart[maxChem * j / n], EMPTY},
    {j, 2, n}];
initialGrid[[IntegerPart[n/2] + 1, j]] = {0, EMPTY};
```

■ Perform simulation and animation

```
graphList = ants[initialGrid, 20];
```

```
showGraphs[graphList, maxChem]
```