Flow on terrains
(I)

Laura Toma
Bowdoin College
Overview

- Flow on grid terrains
  - Flow direction
  - Flow accumulation
  - Flat areas
  - Watersheds and watershed hierarchy
• Where does the water go when it rains?
• What will happen when it rains (a lot)?
• What are the areas susceptible to flooding?
• What areas will flood first?
• What parts of the world will go under water when sea level rises by e.g. 10 ft?
• River data is expensive to collect. Is it possible to model and automatically compute rivers on a terrain?
• What area drains to a point?
• Suppose someone spilled some pollutant)at this point on the terrain—-what area is contaminated when it rains?
• ... and many more.
Flow on digital terrain models

river network, watersheds, flooding,
.....
Big data

• Massive amounts of terrain data available
  • e.g. NASA SRTM, acquired 80% of Earth at 30m resolution. Total 5TB !!
  • USGS: most USA at 10m resolution
  • LIDAR data: 1m resolution

==> need efficient algorithms!!

Example:

• Area if approx. 800 km x 800 km
• Sampled at:
  • 100 resolution: 64 million points (128MB)
  • 30m resolution: 640 (1.2GB)
  • 10m resolution: 6400 = 6.4 billion (12GB)
  • 1m resolution: 600.4 billion (1.2TB)
Flow on grid terrains

• Modeled by two basic concepts
  • **Flow direction (FD)**
    • the direction water flows at a point
  • **Flow accumulation (FA)**
    • total amount of water flowing through a point

• Based on this can define
  • watersheds, drainage areas, river network, flooding
  • (Pfafstteter) river and watershed hierarchy
Flow direction (FD)

- **FD(p)** = the direction water flows at p
- Generally,
  - FD is direction of gradient at p, i.e. direction of greatest decrease
  - FD can be approximated based on a neighborhood of p
- FD on grids:
  - discretized to eight directions (8 neighbors), multiple of 45°

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

SFD: Single flow direction
(steepest downslope)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

MFD: Multiple flow directions
(all downslope neighbors)
Flow direction

- FD can be computed in $O(n)$ time
- Issue: flat areas… later

$n = \text{nb. of cells in the grid}$

Point $(i,j)$ in FD grid stores $\text{FD}(i,j)$ values usually coded as

<table>
<thead>
<tr>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
Flow direction

n = nb. of cells in the grid

Elevation surface

Flow direction

Direction coding

The coding of the direction of flow
Flow accumulation (FA)

- FA(p) = how much water goes through point p

- FA grid:
  - Compute, for each cell (point) c, how much water passes through that cell.
  - Assume each cell starts with 1 unit of water
  - Assume each cell sends its initial as well as incoming water to the neighbor cell pointed to by its FD
Flow accumulation (FA)

- $\text{FA}(p)$ = how much water goes through point $p$

- FA grid:
  - Compute, for each point/cell $c$, how much water passes through that cell.
FD and FA

- Some observations
  - FD graph: forest of trees
  - each tree represents a separate “river tree”
  - points with small FA = ridges
  - points with high FA = channels (rivers)
  - FA: how many cells are upstream, or size of subtree of that cell, if viewing the tree upside down

- FA models rivers!
  - set an arbitrary threshold t
  - cell c is on a river if FA(c) >= t
Flow accumulation

FA 2D view
- high values: blue
- medium values: light blue
- low values: yellow

FA grid draped over elevation grid
Computing FA: naive algorithms

• Idea 1:
  • Scan row-by-row: for each cell add +1 to flow of all cells along its downstream path

• Idea 2:
  • Flow at cell c is the sum of the flows of the neighbors that flow into c
  • Use recursion
  • Do this for every cell

• Other ideas?
Computing FA: naive algorithms

• Idea 1:
  • Scan row-by-row: for each cell add +1 to flow of all cells along its downstream path
  • Analysis??

• Idea 2:
  • Flow at cell c is the sum of the flows of the neighbors that flow into c
  • Use recursion
  • Do this for every cell
  • Analysis??

• Other ideas?
  • Analysis??
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haerkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haerkort
Computing FA: naive algorithm (1)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (1)

\[ \text{worst-case running time} \Theta(n^2) \]

\( n = \text{nb. of cells in the grid} \)

thanks!!! to H. Haverkort
Computing FA: naive algorithm (2)

```c
//do it for all
for (i=0; i<nrows; i++)
  for (j=0; j<ncols; j++)
    flow[i][j] = compute_flow(i,j);

//return the flow of cell (i,j)
void compute_flow(i,j) {
  assert(inside_grid(i,j));
  int f = 0;
  //initial flow at (i,j)
  for (k=-1; k<= 1; k++) {
    for (l=-1; l<= 1; l++) {
      if flows_into(i+k, j+l, i,j)
        f += compute_flow(i+k, j+l);
    } //for l
  } //for k
  return f;
}

//return 1 if cell (a,b) flows into cell (x,y)
// that is, if (a,b)'s FD points towards (x,y)
int flows_into(a,b, x,y) {
  if (!inside_grid(a,b)) return 0;
  ...
}
```
Computing FA: naive algorithm (2)

```c
//do it for all
for (i=0; i<nrows; i++)
    for (j=0; j<ncols; j++)
        flow[i][j] = compute_flow(i, j);
```

```c
//return the flow of cell (i,j)
void compute_flow(i, j) {
    assert(inside_grid(i, j));
    int f = 0; //initial flow at (i,j)
    for (k=-1; k<=1; k++)
        for (l=-1; l<=1; l++)
            if (flows_into(i+k, j+l, i, j))
                f += compute_flow(i+k, j+l);
    return f;
}
```

```c
//return 1 if cell (a,b) flows into cell (x,y)
// that is, if (a,b)'s FD points towards (x,y)
int flows_into(a, b, x, y) {
    if (!inside_grid(a, b)) return 0;
    ...
}
```

Questions:
- What is the worst case running time?
  - Is it linear?
- What sort of FD graph would trigger worst-case?
Computing FA: naive algorithm (2)

//do it for all
for (i=0; i<nrows; i++)
    for (j=0; j<ncols; j++)
        flow[i][j] = compute_flow(i,j);

//return the flow of cell (i,j)
void compute_flow(i,j)
{
    assert(inside_grid(i,j));
    int f = 0;
    //initial flow at (i,j)
    for (k=-1; k<= 1; k++) {
        for (l=-1; l<= 1; l++) {
            if flows_into(i+k, j+l, i,j) {
                f += compute_flow(i+k, j+l);
            }
        }
    }
    return f;
}

//return 1 if cell (a,b) flows into cell (x,y)
// that is, if (a,b)'s FD points towards (x,y)
int flows_into(a,b, x,y) {
    if (!inside_grid(a,b)) return 0;
    ...
}

worst-case running time
Theta(n^2)
Flow accumulation: smarter algorithms?

• Ideas?
Flow accumulation: smarter algorithms?

• Use recursion, but once a value $\text{flow}(i,j)$ is computed, store it in a table. This avoids re-computation.
  • dynamic programming!

• To completely avoid recursion, compute $\text{flow}(i,j)$ in topological order of FD graph
  • topological order can be computed in linear time
  • or: sort by height, but that’s $O( n \lg n)$

• Analysis?

• Which one would you chose in practice?