r.terracost:
Computing Least-Cost Path Surfaces
for Massive Rasters

Thomas Hazel    Laura Toma    Jan Vahrenhold    Rajiv Wickremesinghe

Bowdoin College    U. Muenster    Oracle
USA    Germany    USA

FOSS4G 2006
Lausanne, Switzerland
Least-Cost Path Surfaces

- **Problem**
  - **Input**
    - a cost surface of a terrain
    - a set of sources
  - **Output**
    - a least-cost path surface: each point represents the shortest distance to a source

- **Cost surfaces**
  - Can be correlated elevation, slope, or simply constant (uniform cost)

- **Applications**
  - Spread of fires from different sources
  - Distance from streams or roads
  - Cost of building pipelines or roads
Example

Sierra Nevada, 30m resolution

Sierra Nevada, cost surface = slope
Example (One Source)

source

Least-cost path surface
Example (Many Sources)

Multiple sources

Least-cost path surface
Least-Cost Surfaces in GRASS

r.cost

Description: Outputs a raster map layer showing the cumulative cost of moving between different geographic locations on an input raster map layer whose cell category values represent cost.

Usage:
r.cost [-vkn] input=name output=name [start_sites=name] [stop_sites=name] [start_rast=name] [coordinate=x,y[,x,y,...]] [stop_coordinate=x,y[,x,y,...]] [max_cost=cost] [null_cost=null cost]

Flags
- v Run verbosely
- k Use the 'Knight’s move'; slower, but more accurate
- n Keep null values in output map

Parameters:
input Name of raster map containing grid cell cost information
output Name of raster map to contain results
start_sites Starting points site file
stop_sites Stop points site file
start_rast Starting points raster file coordinate
coordinate The map E and N grid coordinates of a starting point (E,N)
stop_coordinate The map E and N grid coordinates of a stopping point (E,N)
max_cost An optional maximum cumulative cost. default:
null_cost Cost assigned to null cells. By default, null cells are excluded
Massive Terrains

Why massive terrains?

- Large amounts of data are becoming available
  - NASA SRTM project: 30m resolution over the entire globe (~10TB)
  - LIDAR data: sub-meter resolution
- Traditional algorithms designed that assume that data fits in memory and has uniform access cost don’t scale
  - Buy more RAM?
    - Data grows faster than memory
    - Data does not fit in memory, sits on disk
    - Disks are MUCH slower than memory
- \( \Rightarrow \) I/O-bottleneck
Performance of r.cost

GRASS users have complained it is very slow for large grids.
What To Do?

- Massive data ⇒ needs efficient algorithms
  - small data: 1 sec vs 3 sec
  - large data: 1 hour vs 1 day (or worse)
- Massive data: bottleneck is the I/O
  ⇒ Design algorithms that specifically minimize I/O
  ⇒ I/O-efficient algorithms
- Idea:
  - Do not rely on virtual memory!
  - Instead, change the data access pattern of the algorithm to increase spatial locality and minimize the number of blocks transferred between main memory and disk
This project: r.terracost

- has same functionality as r.cost
- based on an I/O-efficient algorithm
- is scalable
  - can process grids that are out of scope with r.cost
- parallelizable on a cluster
Outline

- Background
  - Least-cost path surfaces and shortest paths in graphs
  - Dijkstra’s algorithm for SP
  - Dijkstra’s algorithm on large grids

- r.terracost
  - Algorithm
  - Experimental results
  - Cluster implementation

- Conclusions and current/future work
Least-Cost Path Surfaces and Shortest Paths in Graphs

- Raster terrains → graphs
- Least-cost path surfaces correspond to computing shortest paths on (raster) graphs

Cost raster

Corresponding graph

Shortest-distance from center point
Related Work on Shortest Paths

- **Dijkstra’s Algorithm**
  - Best known for SSSP on general graphs, non-negative weights

- **Recent variations on the SP algorithm**
  - Goldberg et al SODA 2000, WAE 2005
  - Kohler, Mohring, Schilling WEA 2005
  - Gutman WEA 2004
  - Lauther 2004

- **Different setting**
  - Point-to-point SP
    - E.g. Route planning, navigation systems
  - Exploit geometric characteristics of graph to narrow down search space
Dijkstra’s SP Algorithm

Greedy algorithm
Dijkstra’s SP Algorithm

Greedy algorithm
Dijkstra’s SP Algorithm
Dijkstra’s SP Algorithm
Dijkstra’s SP Algorithm
Dijkstra’s SP Algorithm
Dijkstra’s SP Algorithm
Dijkstra's SP Algorithm
Dijkstra’s SP Algorithm
SP (one source)
SP (many sources)
Dijkstra’s Algorithm on Large Grids

- Dijkstra’s algorithm requires 3 data structures:
  1. Cost grid
  2. Least-cost grid
  3. Priority queue

- If grids do not fit in main memory ⇒ stored on disk.
- For each vertex that we settle, we must do a lookup in both grids.
- These lookups can cost one I/O each in the worst case.
- ⇒ One I/O per element in the grid
GRASS Segment Library

- If data does not fit in memory
  - default: use the virtual memory system (VMS)
  - program may abort because of malloc() fail

- use GRASS segment library
  - bypass the VMS
  - manage data allocation and de-allocation in segments on disk
  - program will always run
  - but... may be slow

- GRASS segment library cannot change the data access pattern of the algorithm, and thus cannot optimize block transfer
Performance of r.cost

uses segment library
Step 1 (intra-tile Dijkstra)
- Divide grid $G$ into tiles of size $R$
- Compute boundary-to-boundary graph: Replace each tile with a complete graph on its boundary

Step 2 (Inter-tile Dijkstra)
- Dijkstra on boundary-to-boundary graph
- Gives SP for all boundary vertices in $G$

Step 3 (Final-Dijkstra)
- Dijkstra inside each tile
- Gives SP to vertices inside tiles
r.terracost

- Optimized for internal or external memory by setting numtiles
  - numtiles=1
    - r.terracost runs Dijkstra in memory
  - numtiles = xxx
    - Use xxx tiles
  - if numtiles is not specified
    - if computation is in memory, use numtiles = 1
    - otherwise, numtiles is set to an optimal value
r.terracost

Synopsis:
r.terracost computes a least-cost surface for a given cost grid and a set of start points. See “Terracost: a versatile and scalable approach for computing shortest paths on massive terrains” by Hazel, Toma, Vahrenhold and Wickremesinghe (2005)

Usage:
r.terracost [-hqdli0123] [cost=name] [start_raster=name] [distance=name] [memory=value] [STREAM_DIR=name] [VTMPDIR=name] [numtiles=value]

Flags:
-h Help
-q Quiet (suppress messages)
-d Debug (for developer use)
-i Info (prints useful information and exits)

Parameters:
  cost  Input cost grid
  start_raster  Input raster of source points
  distance  Output distance grid
  memory  Main memory size (in MB) default: 400
  STREAM_DIR  Location of temporary STREAM default: /var/tmp
  VTMPDIR  Location of intermediate STREAM default: /var/tmp/ltoma
  numtiles  Number of tiles (-h for info)
Example

```bash
GRASS:~ > r.terracost cost=elev start_rast=accu1000 dist=lcs numtiles=1
```

STREAM temporary files in /var/tmp (THESE INTERMEDIATE STREAMS WILL NOT BE DELETED IN CASE OF ABNORMAL TERMINATION OF THE PROGRAM. TO SAVE SPACE PLEASE DELETE THESE FILES MANUALLY!)
intermediate files in /var/tmp/ltoma
region size is 472 x 391
file set1-stats.out exists - renaming.
memory size: 400.00M (419430400) bytes
Memory manager registering memory in MM_WARN_ON_MEMORY_EXCEEDED mode.
Using normal Dijkstra
Using normal Dijkstra
99%
Opened raster file lcs for writing!

cleaning up...
```

GRASS:~ >
```
GRASS:~/nfs-gis > r.terracost  cost=elev start_rast=accu1000  dist=lcs numtiles=10
STREAM temporary files in /var/tmp (THESE INTERMEDIATE STREAMS WILL NOT BE DELETED IN CASE OF ABNORMAL TERMINATION OF THE PROGRAM. TO SAVE SPACE PLEASE DELETE THESE FILES MANUALLY!)
intermediate files in /var/tmp/ltoma
region size is 472 x 391
memory size: 400.00M (419430400) bytes
----------------------------------------
STEP 0:  COMPUTE SUBSTITUTE GRAPH
Grid size is: 184552 Tile size is: 18360  TF #Tiles: 12
----------------------------------------
STEP 1
TileFactory: Sorting internalstr...
----------------------------------------
STEP 2
Sorting b2b stream
----------------------------------------
STEP 3
----------------------------------------
INTER TILE DIJKSTRA
----------------------------------------
IN-TILE FINAL DIJKSTRA
r.terracost done
GRASS:~/nfs-gis >
## Experimental Results

### Experimental Platform
- Apple Power Macintosh G5
- Dual 2.5 GHz processors
- 512 KB L2 cache
- 1 GB RAM

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Grid Size (million elements)</th>
<th>MB (Grid Only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaweah</td>
<td>1.6</td>
<td>6</td>
</tr>
<tr>
<td>Puerto Rico</td>
<td>5.9</td>
<td>24</td>
</tr>
<tr>
<td>Hawaii</td>
<td>28.2</td>
<td>112</td>
</tr>
<tr>
<td>Sierra Nevada</td>
<td>9.5</td>
<td>38</td>
</tr>
<tr>
<td>Cumberlands</td>
<td>67</td>
<td>268</td>
</tr>
<tr>
<td>Lower New England</td>
<td>77.8</td>
<td>312</td>
</tr>
<tr>
<td>Midwest USA</td>
<td>280</td>
<td>1100</td>
</tr>
</tbody>
</table>
Experimental Results

- `r.cost`
- `Opt Dijkstra (r.terracost numtiles=1: internal memory version of Terracost)`
- `TerraCost (r.terracost numtiles=optimal: I/O-efficient version of Terracost)`
r.terracost on Clusters

- We parallelized the most CPU-intensive part (Step 1)
- Hgrid: Cluster management tool
  - Clients submit requests (run jobs, query status); agents get jobs and run them
  - Near-linear speedup
Results

elevation
cost=elevation, 1 source
cost=elevation, many src
flow accumulation
if(flowaccumulation>1000, 1, null())
cost=elevation, sources=flowaccu>1000
Conclusion

r.terracost
- has same functionality as r.cost
- based on an I/O-efficient algorithm
- is scalable
  - can process grids that are out of scope with r.cost
- parallelizable
Current/Future Work

- Scalable viewshed computation
  - GRASS: r.los
  - New: r.viewshed
r.viewshed

(.1M)
- r.los: 3 sec
- r.viewshed: 1 sec

Sierra (10M)
- r.los: 4.5 hours
- r.viewshed: 1 min

Washington (1000M)
- r.viewshed: 4.5 hours
Thank you.

Laura Toma
Bowdoin College
Maine, USA
ltoma@bowdoin.edu
http://www.bowdoin.edu/~ltoma/