

Terracost: A Versatile and Scalable Approach to Computing Least-Cost-Path Surfaces for Massive Grid-Based Terrains

Thomas Hazel

Bowdoin College

Laura Toma

Bowdoin College

Jan Vahrenhold

U. Muenster

Rajiv Wickremesinghe

Duke University

ACM SAC April 2006

Dijon, France

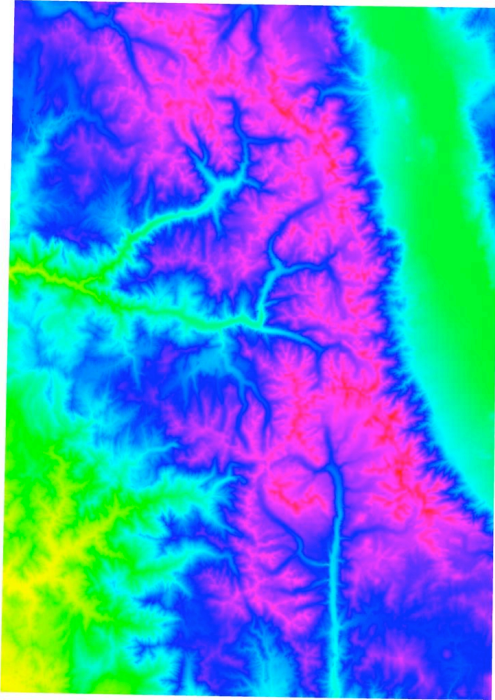
Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Least-cost path surfaces

- Problem
 - Input: A **cost surface** of a grid terrain and a set of **sources**
 - Output: A **least-cost path surface**: each point represents the shortest distance to a source
- Applications
 - Spread of fires from different sources
 - Distance from streams or roads
 - Cost of building pipelines or roads

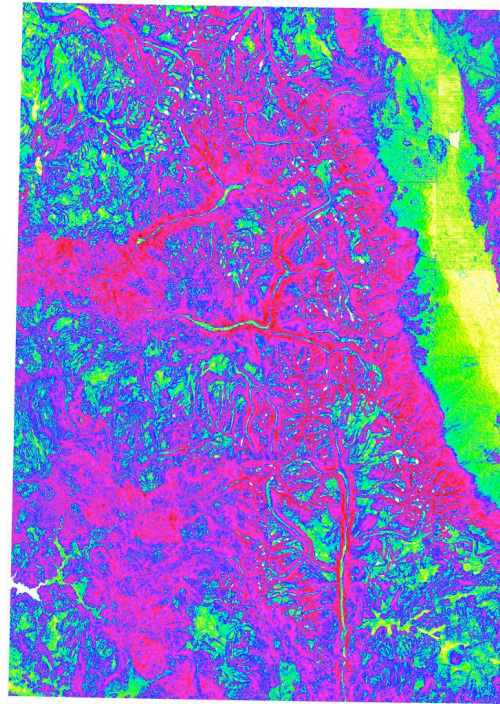
Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Grid terrains



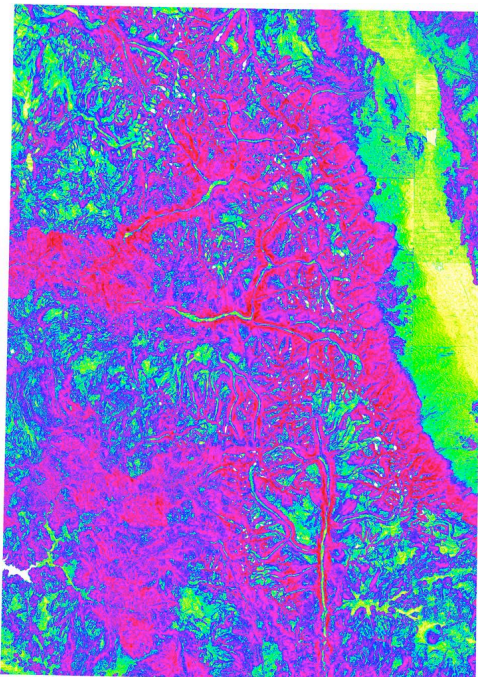
Sierra Nevada, 30m resolution

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

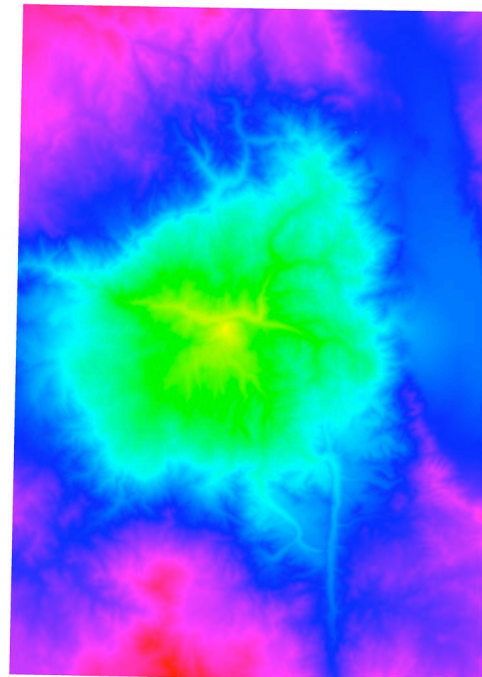


Sierra Nevada, a cost surface

Least-Cost Surface with one source



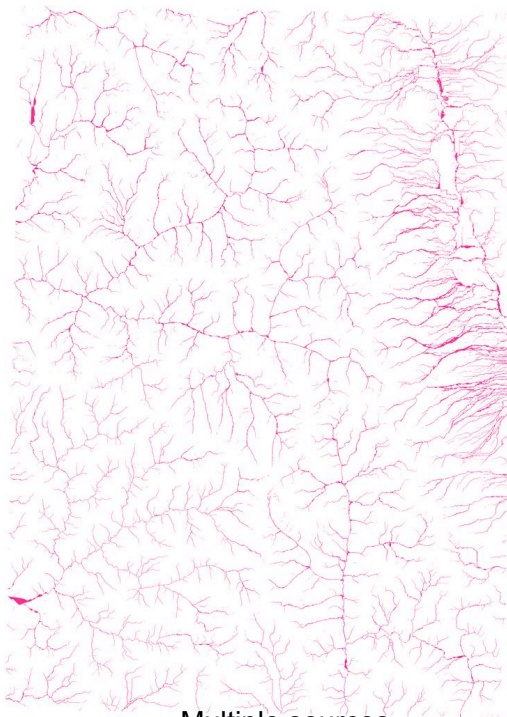
Cost surface



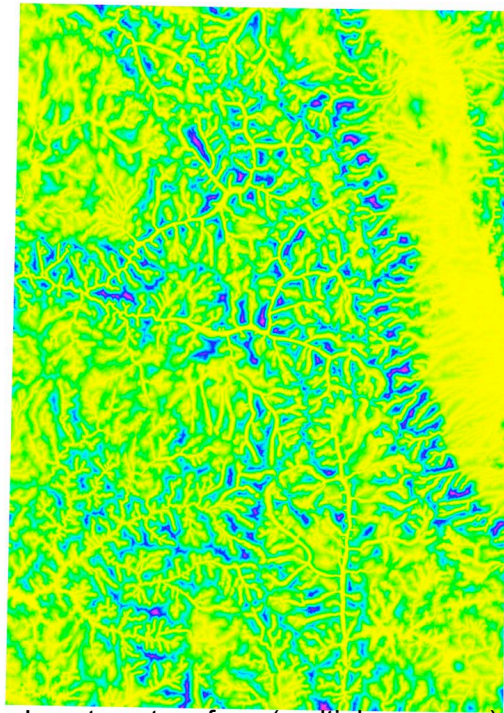
Least-cost path surface (1 source)

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Least-cost surface with Multiple Sources



Multiple sources



Least-cost surface (multiple sources)

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Least-cost path surfaces on massive terrains

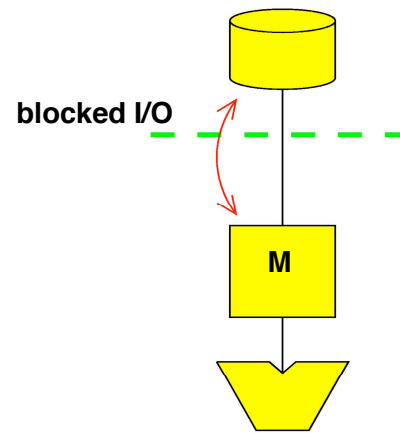
- Why massive terrains?
 - Large amounts of data are becoming available
 - NASA SRTM project: 30m resolution over the entire globe (~10TB)
 - LIDAR data: sub-meter resolution
 - Traditional algorithms designed in RAM model don't scale
 - Buy more RAM?
 - Data grows faster than memory
 - Data does not fit in memory, sits on disk
 - Random I/O + Virtual memory => swapping
- => I/O-bottleneck



Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

I/O-Efficient Algorithms

- Input data (grid) stored on disk
- I/O-model [Agarwal and Vitter, 88]
 - N = size of grid
 - M = size of main memory
 - B = size of disk block
 - I/O-operation (I/O): Reading/Writing one block of data from/to disk
- I/O efficiency
 - Number of I/Os performed by the algorithm
- Basic I/O bounds
 - Scanning: $\text{Scan}(N) = \theta(N/B)$ I/Os
 - Sorting: $\text{Sort}(N) = \theta(N/B \log_{M/B} N/B)$ I/Os
 - In practice M and B are big:
 - $\text{Scan}(N) < \text{Sort}(N) \ll N$ I/Os



Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Terracost

- Scalable approach to computing least-cost path surfaces on massive terrains
 - Based on optimal I/O-efficient algorithm: $O(\text{Sort}(N))$ I/Os
- Experimental analysis on real-life data
 - Can handle bigger grids
 - Can handle more sources
- Versatile: Interpolate between versions optimized for I/O or CPU
- Parallelization on a cluster

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Outline

- Background
 - Shortest paths
 - Related Work
 - Shortest paths in the I/O-Model
- Terracost
 - Algorithm
 - Experimental analysis
 - Cluster implementation
- Conclusions

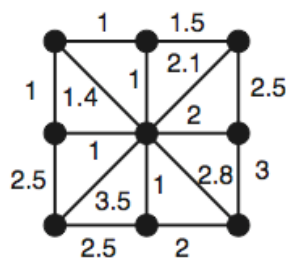
Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Shortest Paths

- Least-cost path surfaces correspond to computing shortest paths
- Shortest paths
 - Ubiquitous graph problem
 - Variations
 - SSSP: Single source shortest path
 - MSSP: Multiple source shortest path
- Grid terrains --> graphs

1	1	2
1	1	3
4	1	3

Cost grid



Corresponding graph

1.4	1	2.1
1	0	2
3.5	1	2.8

Shortest-distance
from center point

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Related Work

- Dijkstra's Algorithm
 - Best known for SSSP/MSSP on general graphs, non-negative weights
- Recent variations on the SP algorithm
 - Goldberg et al SODA 200, 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
 - Route planning graphs
 - Use RAM model
 - When dealing with massive graphs ==> I/O bottleneck

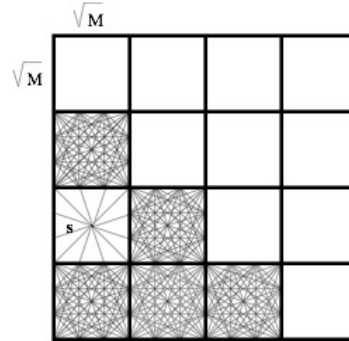
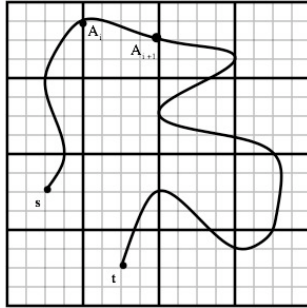
Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Dijkstra's Algorithm

- 1: Insert sources in a priority queue(PQ)
 - 2: While PQ is not empty
 - 3: DeleteMin vertex u with the least cost from PQ
 - 4: Relax all edges incident to u
- In external memory
 - Dijkstra's algorithm requires 3 main data structures:
 - 1: Priority queue (can be implemented I/O-efficiently)
 - 2: Grid of costs (size = $N \gg M$)
 - 3: Grid of current shortest path (size = $N \gg M$)
 - Each time we DeleteMin from PQ, for every adjacent edge (u,v) we must do a lookup in both grids.
 - To check whether v can be relaxed
 - These lookups can cost $O(1)$ I/Os each in the worst case
- ==> Total $O(N)$ I/Os

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

I/O-Efficient SSSP on Grids [ATV'01]

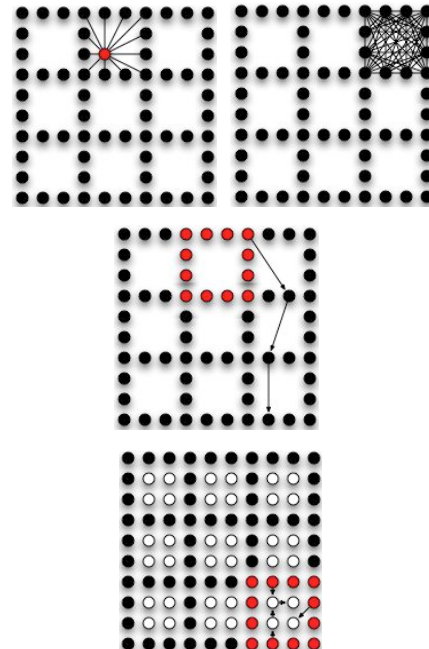


- 0: Divide grid G into subgrids(G_i) of size $O(M)$
- 1: Construct a substitute graph S on the boundary vertices
 - Replace each subgrid with a complete graph on its boundary
 - For any u, v on the boundary of G_i , the weight of edge (u, v) in S is $SP_{G_i}(u, v)$
- Lemma: S has $O(N/\sqrt{M})$ vertices, $O(N)$ edges and it preserves the SP in G between any two boundary vertices u, v .
- 2: Solve SP in S
 - Gives SP for all boundary vertices in G
- 3: Compute SP to vertices inside subgrids

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Terracost

- **Step 1: (intra-tile Dijkstra)**
 - Partition into tiles of size R
 - Compute an edge-list representation of substitute graph S
 - Dijkstra from each boundary to tile boundaries
 - Dijkstra from sources to tile boundaries
- **Step 2:**
 - Sort boundary-to-boundary stream
- **Step 3: (inter-tile Dijkstra)**
 - Dijkstra on S
- **Step 4: (final Dijkstra)**
 - MSSP for each tile



Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Experimental Analysis

Dataset	Grid Size (million elements)	MB (Grid Only)
Kaweah	1.6	6
Puerto Rico	5.9	24
Hawaii	28.2	112
Sierra Nevada	9.5	38
Cumberlands	67	268
Lower New England	77.8	312
Midwest USA	280	1100

Experimental Platform

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

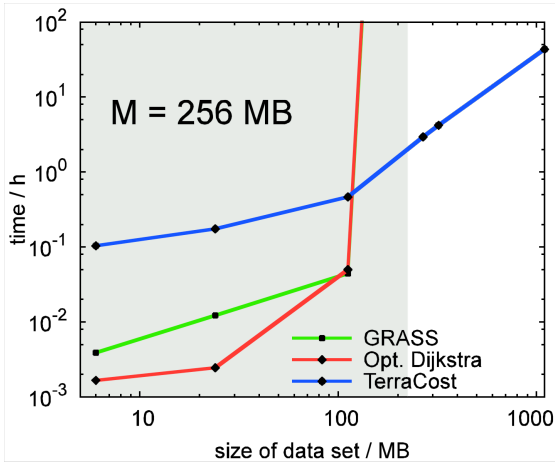
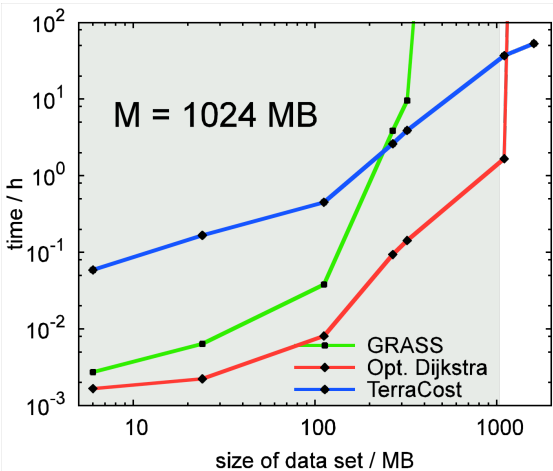
Compare Terracost with r.cost in GRASS

- r.cost has same functionality
- GRASS users have complained it is very slow for large terrains

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Experimental Analysis

- GRASS: r.cost
- Opt Dijkstra: internal memory version of Terracost (num tiles = 1)
- Terracost: I/O-efficient version of Terracost

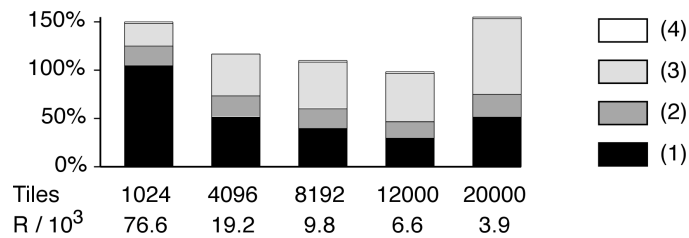


Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

CPU-I/O Tradeoff

R = tile size

- I/O-complexity = $O(N / \sqrt{R} + \text{sort}(N))$
 - Dominated by Step 3 (inter-tile Dijkstra)
- CPU-complexity = $O(N \sqrt{R} \log R)$
 - Dominated by Step 1 (intra-tile Dijkstra)
- So, to optimize I/Os, we want a large R .
- But, to optimize CPU, we want a small R .
- Optimal performance: balance I/O-CPU

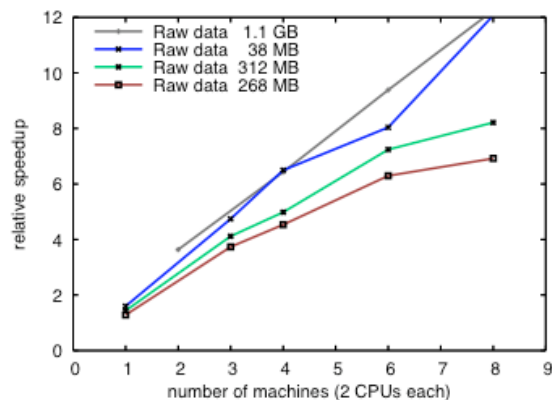
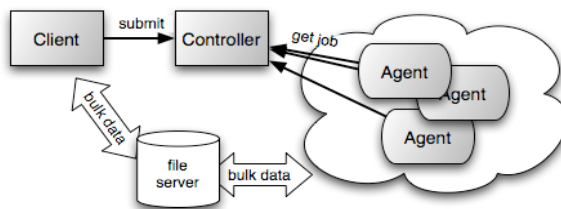


Lower NE, Cost = Slope, 1GB RAM, Single source

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Terracost on Clusters

- Terracost lends itself to parallelization
- We parallelized the most CPU-intensive part
 - Computing the substitute graph (Step 1)
- **Hgrid**
 - Cluster management tool
 - Clients submit requests (run jobs, query status); agents get jobs and run them
- Near-linear speedup



Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Conclusions and Future Work

Key Points

- Dijkstra's algorithm is I/O-inefficient on large data sets
- Terracost restructures the input grid to run I/O-efficiently
 - But we can't ignore CPU-complexity completely
- I/O-bottleneck increases with number of sources for MSSP
- Tiling in Terracost allows for parallelization

Future Work

- Determine the optimal tile size analytically
- Find I/O-efficient SSSP/MSSP w/o increase of CPU-efficiency

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

Thank you.

Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe