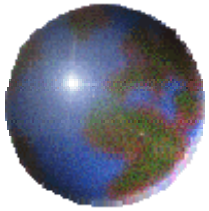
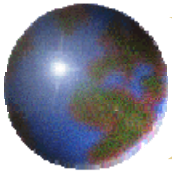


Fast Shortest Path Algorithms for Large Road Networks

廖明傑

2005.08.18

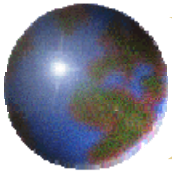


Reference

- Faramroze Engineer
- Fast Shortest Path Algorithms for Large Road Networks
- ORSNZ Conference Twenty Naught One 30 November to 1 December, 2001
  University of Canterbury, Christchurch, NZ

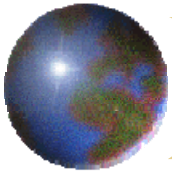
Outline

- Introduction
- Shortest Path Algorithms
- Search Performance Analysis
- Conclusion



Introduction

- Due to the nature of routing applications, we need flexible and efficient shortest path procedures, both from a processing time point of view and also in terms of the memory requirements.
- Since no “best” algorithm currently exists for every kind of transportation problem, research in this field has recently moved to the design and implementation of “heuristic” shortest path procedures.

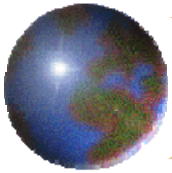


Goal

- As it is impossible to cover all search implementations, we use Dijkstra's algorithms a building block to create an efficient search algorithm.
- It may not guarantee optimal results but gives significant savings in terms of memory requirements and processing speed.

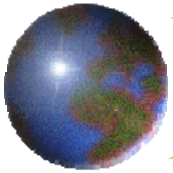
Shortest Path Algorithms

- Dijkstra's Algorithm
- Symmetrical Dijkstra's Algorithm
- A* Search
- Radius Search



Dijkstra's Algorithm

- It functions by constructing a shortest-path tree from the initial vertex to every other vertex in the graph .(Example)



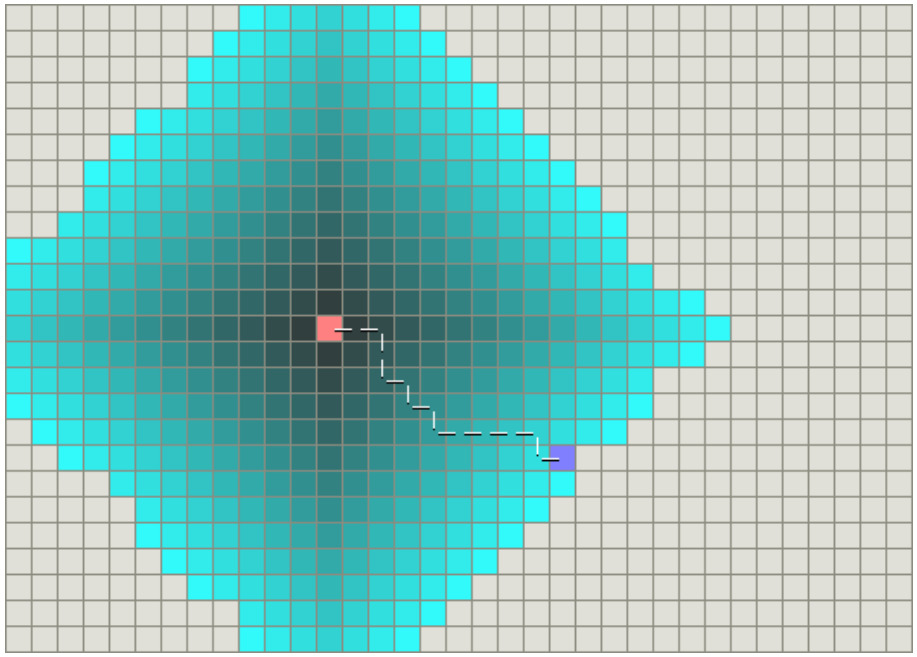
Symmetrical Dijkstra's Algorithm

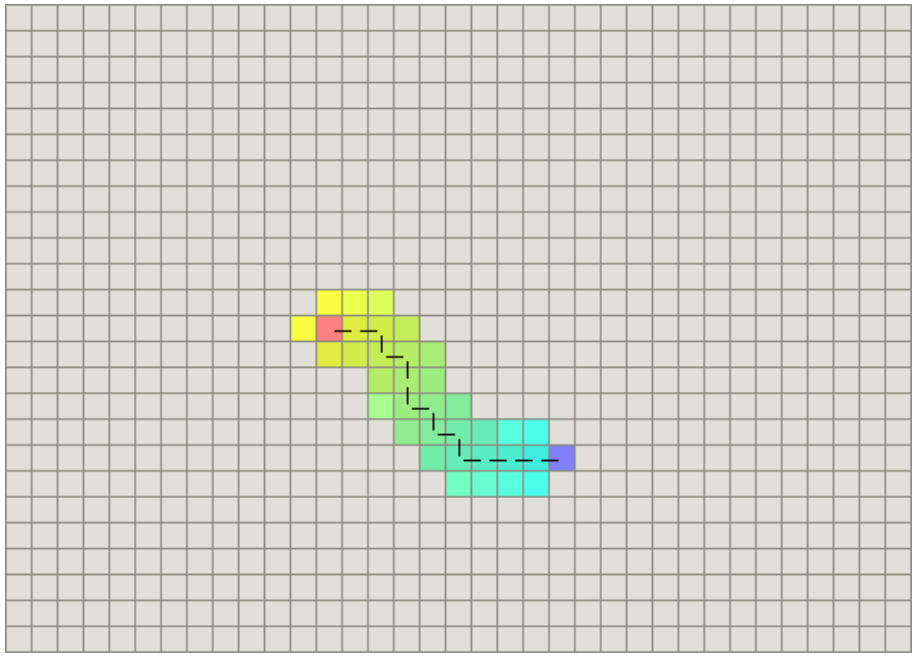
- The Symmetrical algorithm grows two search trees, one from the origin (L_F), and the other from the destination (L_B).
- We iteratively add one node to either L_F or L_B until there exists an arc crossing from L_F to L_B .

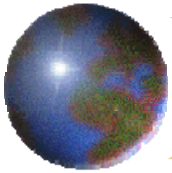
A* Search

- A* typically searches outward from the starting node until it reaches the goal node, always expanding the current fringe node that looks like it is along the best path from the start node to the goal node.
- The best node is the current fringe node with the minimum cost from the start node to the fringe node, plus the expected remaining cost (the heuristic cost) to get from the fringe node to the goal node.
- The heuristic used to estimate the remaining cost from any node to the goal plays a key role in A*.

The Difference between Dijkstra's and A*

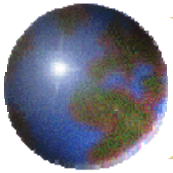






Radius Search

- A *Radius* search is a hierarchical search with a continuous range of hierarchy levels.
- When looking for a shortest path from *s* to *t*, a node *i* is considered as a possible node to include in the search only if *s* or *t* lies inside a circle of radius *r(i)* centered at node *i*.
- If both distances are greater than the node radius, the node is simply ignored.



The Optimal Radius for a Node *i*

- Let $R = (R_{[1]}, R_{[2]}.....R_{[|p|]})$ be an optimal path (sequence of nodes) from an origin node $R_{[1]}$ to a destination node $R_{[|p|]}$.
- Let $R = (R_1, R_2.....R_R)$ be the set of all optimal paths on G.
- Let $R(i)$ be the set of all optimal paths that use node i, $R(i) = \{R \hat{I} R : \$h ? \{1, 2..... | R | \quad 1 : R_{[h]} = i\}\}$
- $r(i) = \max_{R \hat{I} R(i)} \{\min\{h_E(i, R_{[1]}), h_E(i, R_{[R]})\}\}$

Radius Search-phase 1

Divide Network into grids of approximately 2000 nodes $N_{SUB1} \hat{I} N$

Initial Radii of all nodes $n \hat{I} N$ *to 0*

while *time permits*

Select a random node $s \hat{I} N$

Select random node t within the same grid as s

Solve the shortest path R from s to t

UpdateNodeRadii(R)

loop

UpdateNodeRadii(R)

{

for *all* $n \hat{I} R$

$r(n) = \max(r(n), \min(h_E(n, R_{[1]}), h_E(n, R_{[R]}))$

next *n*

}

Radius Search-phase 2

for all nodes do

if *r(n) = 0 and* ***ClosedNode(n) = false then***

Define a sub-graph G_{SUB2} *containing:*

- *200 of the closest nodes* N_{SUB2} *to n*
- *All shortest paths* R_{SUB2} *defined on* G_{SUB2}

for *every shortest path* $R \hat{I} R_{SUB2}$ *do*

UpdateNodeRadii(R)

next *R*

end *if*

next *n*

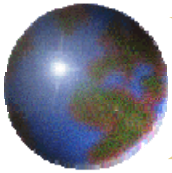
Radius Search-phase 3

Define extreme nodes $N_{EXTREME}$

for *all shortest paths* R *from* $s \hat{I} N_{EXTREME}$ *to* $t \hat{I} N_{EXTREME}$

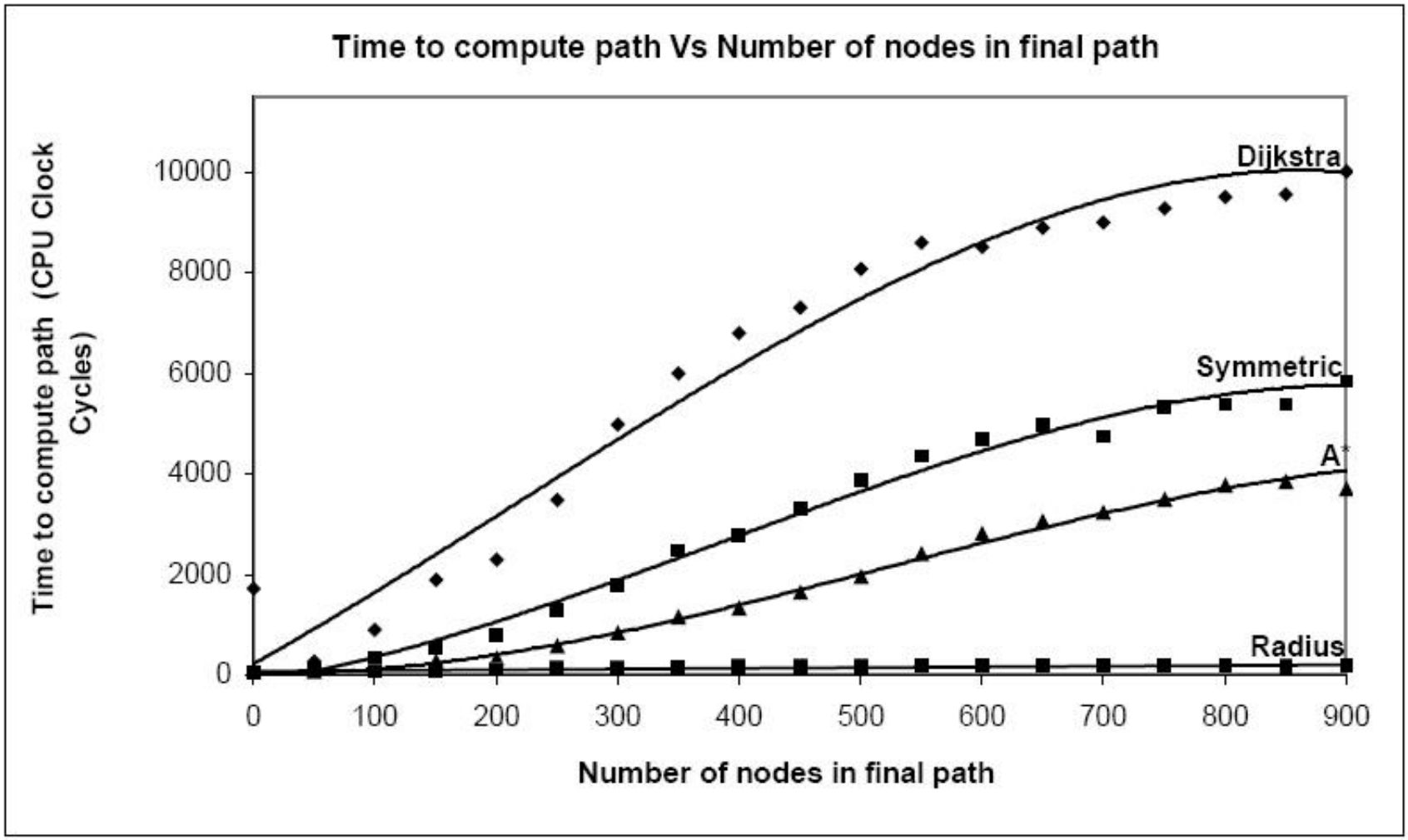
UpdateNodeRadii(R)

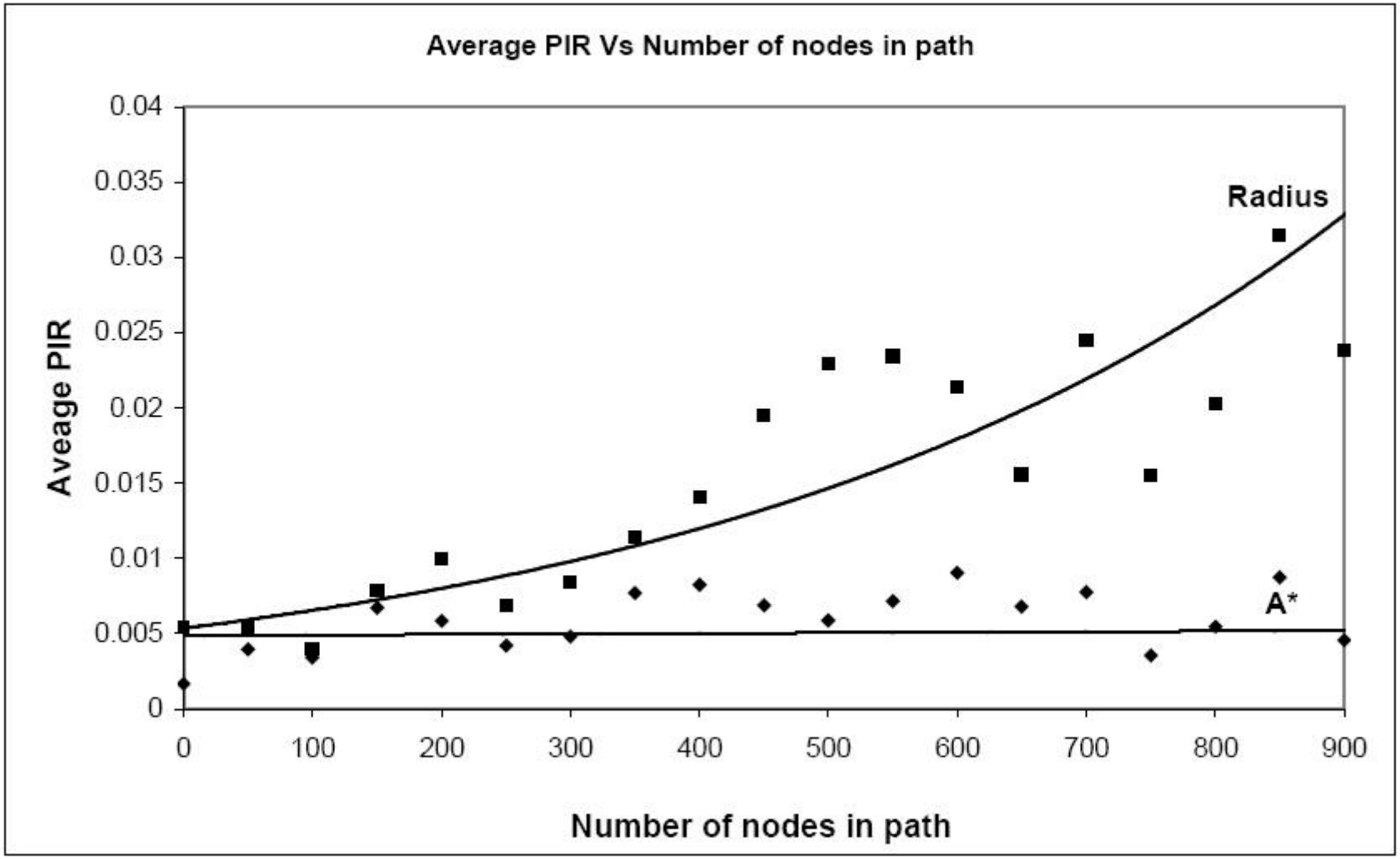
next R



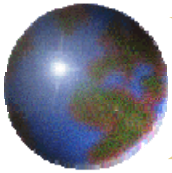
Search Performance Analysis

- To test these algorithms we used parts of the London road network(22,500 km^2, has approximately 140,000 nodes and 298,000 directed arcs)
- 4 search algorithms:
  - Dijkstra's(Dijkstra's)
  - Dijkstra's Bi-directional (Symmetric)
  - A* algorithm (A*)
  - Dijkstra's Bi-directional algorithm with radius restriction (Radius)


Time to compute path Vs Number of nodes in final path
Time to compute path (CPU Clock Cycles)
10000
8000
6000
4000
2000
0
0
100
200
300
400
500
600
700
800
900
Number of nodes in final path
Dijkstra
Symmetric
A*
Radius



Note: (PIR) Path Inaccuracy Rate



Conclusion

- By exploiting the physical structure of road networks, the A* algorithm is able to bias its search towards a goal and reduce the search space.
- By using the concept of radii as a measure of importance of nodes, we are able to incorporate pre-processing within our shortest path algorithm to further restrict the search space.
- This dramatically reduces the search complexity in terms of the run time performance while still maintaining an acceptable level of inaccuracy.