I/O Efficieny of Highway Hierarchies

Riko Jacob Sushant Sachdeva

Departement of Computer Science ETH Zurich,
Technical Report 531, September 2006

Abstract

Recently, Sanders and Schultes presented a shortest path algorithm, named Highway Hierarchies, for fast point-to-point shortest path queries. They report extremely quick average queries on the road network of USA. We consider the I/O efficiency of the algorithm and investigate how a good graph layout affects the average query time. We experiment with a few layout heuristics and obtain a speed-up factor of around 1.3 as compared to the default layout and around 1.7 as compared to a random layout.

1 Introduction

The shortest path problem has been a heavily researched. Formally the problem is stated as below:

> Given a *weighted, directed* graph $\mathcal{G}$, and two vertices s and t, find a path from s to t with the least weight.

The problem as stated above, in its general form, was solved by Dijkstra around 50 years ago. Lately, many *practical* algorithms have been developed for large graphs, keeping route planning in mind. These algorithms assume that $\mathcal{G}$ has a certain structure and doesn't change too often. Hence, it can be worthwhile to invest in a preprocessing step. This preprocessing step is usually linear or slightly superlinear in the size of G and generates some extra information. This extra information is then utilized to accelerate Dijkstra-like shortest path queries by reducing the number of nodes that need to be explored.

It also has been recognized that the performance of modern computers heavily depends on the way an algorithm uses the different types of memory, i.e., the registers of the CPU, level 1 and level 2 cache, the main memory, and the disk, the so-called memory hierarchy. This lead to the development of algorithms in the so-called I/O-model, where the performance measurement is the number of times a memory-page is transferred between two levels of the memory hierarchy. The shortest path problem has been investigated in this model, one of the latest algorithms is by Meyer and Zeh [MZ06]. The focus of these algorithms is on asymptotic worst case performance. Among other things, this leads to the assumption that the graph (that needs to be explored completely in the worst case) is huge in comparison to the main memory. The solution have as the main components an efficient way of handling the fringe set and a clever clustering of the graph.

The idea we pursue here is to combine the two algorithmic paradigms: By storing the graph in a good layout, as inspired by the I/O-efficient shortest path algorithms, we try to increase the efficiency of the accelerated shortest path algorithms. The intuition with this is that the pruning of the shortest path search allows the explored nodes to remain in main memory, such that some of the more complicated aspects of the I/O-efficient algorithms are not needed. This is similar to what is known as a semi-external memory setting, where the nodes can be stored in main memory, but the edges cannot.

Out of several *practical* algorithms, we chose to work with Highway Hierarchies from the paper by Peter Sanders and Dominik Schultes [SS05, SS06] because of the small pre-processing time and the in-built hierarchial structure. We tried to see the effect of the layout of the graph on the average query time. The idea was that a good layout can help to reduce the average number of cache misses during a query and can hence help to reduce the query time. We wanted to have an idea of how much room is there for improvement in query times of such algorithms by just being I/O efficient. By the layout of the graph, we shall mean the order in which the nodes are numbered and stored in the memory. Our experiments focus on the cache-efficiency of the algorithms, not on their performance if the graph must be stored on disk.

2 Layout Heuristics

We tried out a heuristic in order to obtain a *good* layout of the graph. The main idea in our heuristic is construction of connected blocks. The blocks are constructed in the graph in a depth first search sort of way. Each block itself is constructed like a minimum spanning tree as in Prim's algorithm until it has a desired size. The steps in the heuristic are as follows.

1. **Select an unsettled boundary vertex :** We choose an arbitrary vertex and take the unsettled vertex farthest from it as the starting vertex v.
2. **Construct a block :** We start constructing an MST using Prim's algorithm, starting with v and adding unsettled vertices to the tree until the no of verices in this block becomes B. At the end, we push all the labelled vertices into a stack.
3. **Select a new starting point :** If the stack is non-empty, we pop the topmost unsettled vertex from the stack and it becomes the new v and we repeat from step 2 onwards. Otherwise, if there are still vertices that have not yet been settled, we repeat from step 1 onwards.

3 Experiments

We performed several experiments in order to find out the mileage that can be obtained by using a good layout. In order to have a reference to compare our results to, we also ran our implementation of Highway Hierarchies on the same graphs with their layout randomized. It should be noted that we performed equivalent queries on the various graphs, e.g., if the nodes v and u in $\mathcal{G}_1$ are indexed as $\pi(v)$ and $\pi(u)$ in $\mathcal{G}_2$ and we ask for the shortest path from v to u in $\mathcal{G}_1$, we ask for the shortest distance from $\pi(v)$ to $\pi(u)$ in $\mathcal{G}_2$.

The Block Chain experiment was done using a one-directional chain graph with 400,000 vertices (and consequently 399,999 edges). All the other experiments were done using the graph of the road network of the USA Bay area (Source : `http://www.dis.uniroma1.it/~challenge9/`) with distances as edge lenghts, having 321269 nodes and 801264 edges.

3.1 Preliminary tests

We did a preliminary test with randomized block chains. We take a directed chain graph and divide it into a number of blocks, each consisting of the same number of vertices. The vertices within a block are numbered contiguously and hence stored in contiguous locations in memory, whereas the blocks themselves are randomized. We vary the block size from 1 to the length of the chain and measure average query time using our implementation of Highway Hierarchies. These tests indicate the best speed-up we can ideally expect to obtain just by giving the graph a *good* layout.

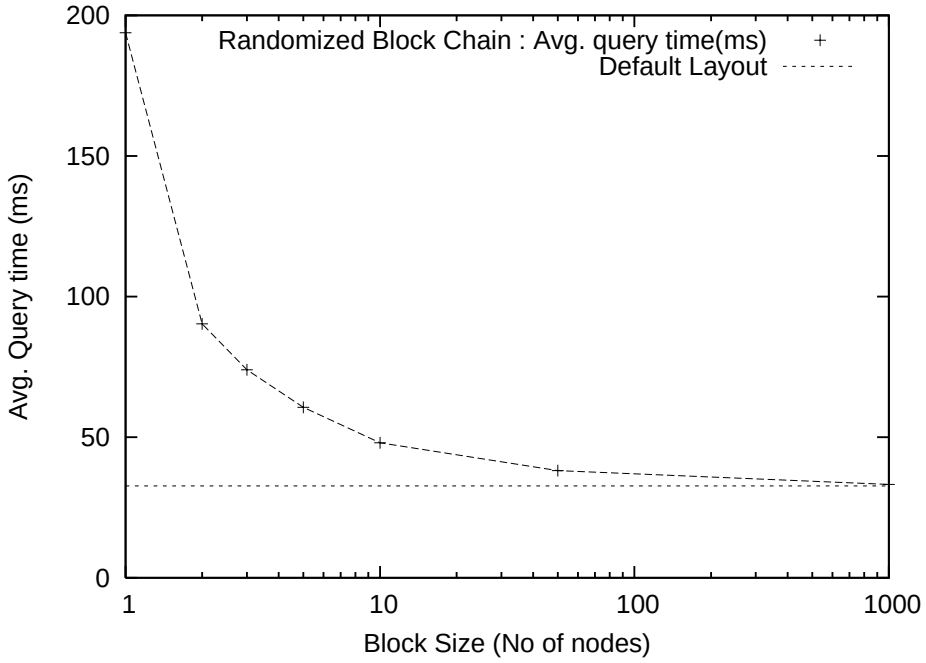

Figure 1: Results - Randomized Block Chain test

3.2 Simple Clustering

We use the heuristic described above with various values of the parameter B, find out the average query time and compare it to that with the randomized layout.

First, we run Dijkstra's Algorithm on the graph. We see that the default layout is much better than the randomized one, and that the simple clustering gives some further improvement, which does not depend very much on the choice of the parameter.

Next, we performed the same experiment using HH as the query algorithm. It is clear from our

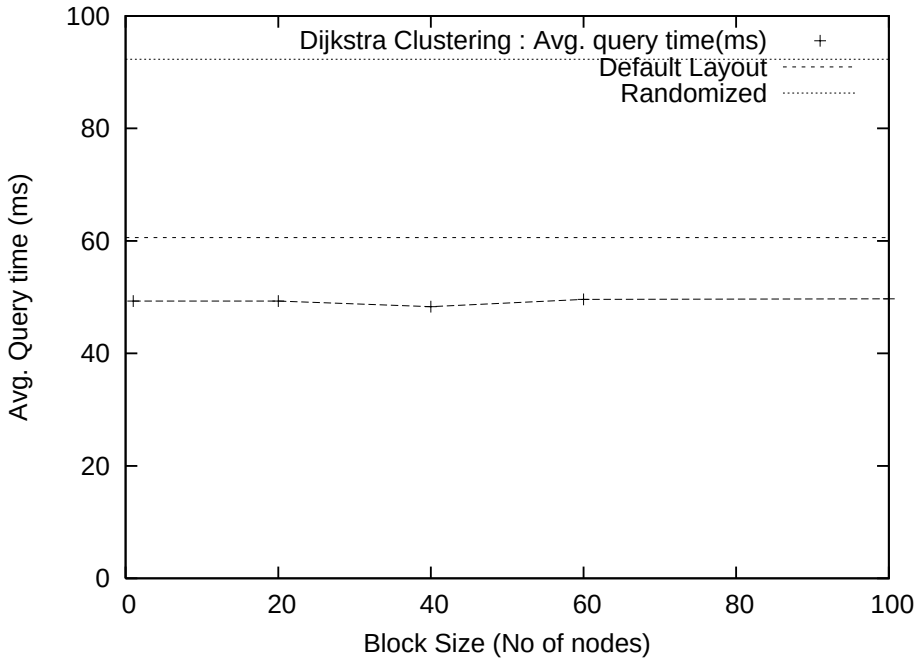

Figure 2: Results - Dijkstra Clustering test

results that our heuristic provided slightly better performance than even the default layout and gives approximately a 20% gain as compared to the randomized layout.

3.3 Level Clustering

We noticed that the query algorithm in [SS06] always moves upwards in the hierarchy. So once it settles a node on level l, it will not access any node on a level less than l. Thus, we expected clustering the nodes in one level together to help. So we modified the above heuristic and clustered the nodes according to the highest level that they belonged to. So all the nodes belonging to level l and not to $l + 1$ were clustered together. The blocks at each level were still constructed in the manner described above. As a comparison, we also determined the running time for a layout, where the nodes of a level are stored together, but within the level in a random order. This already gives a slight improvement over the default layout.

4 Conclusion

The above experiments indicate that average query time can be reduced by being I/O efficient, both in Dijkstra's algorithm and with Highway Hierarchy, presumable also with similar speed-up techniques. We conclude this from the observation that a randomized layout hurts in both cases in comparison to the natural layout in which the graphs are given. Furthermore, even a simple clustering idea can improve over this natural layout. Because the query algorithm is independent of the layout and performs the same calculations, we know that the speed difference must stem from different memory access patterns.

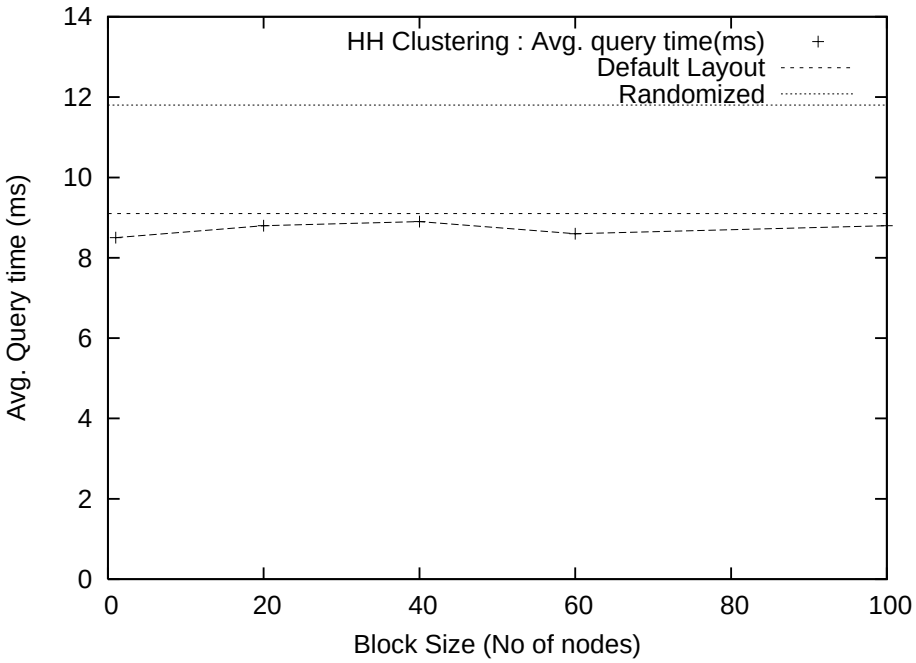

Figure 3: Results - HH Clustering test

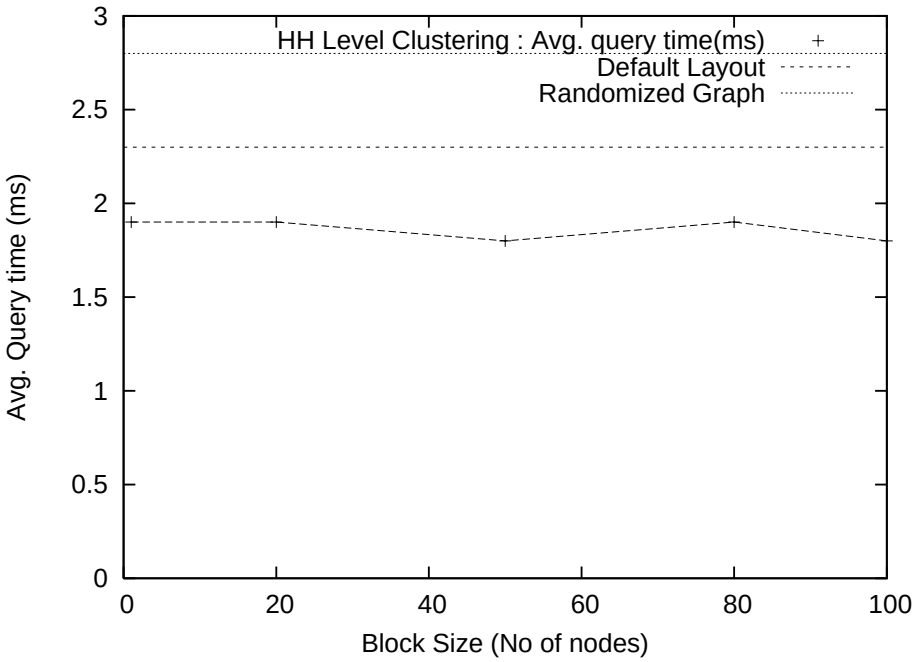

Figure 4: Results - HH Level Clustering test

Hence, the observed acceleration must be due to a decrease in the number of cache misses.

We expect the timing difference to be much more pronounced in the setting where the graph does not fit in main memory and needs to be loaded from disk (the level of the memory hierarchy we investigate is memory-disk instead of cache-memory).

Obviously, the simple clustering we investigated here is not necessarily the best possible memory layout. It is actually a well defined optimization problem to ask for the layout of the graph in blocks, such that a Dijkstra or HH query needs in average the least possible number of I/O. It is a topic of future research to better understand this optimization problem.

Further, it seems reasonable to consider modifications of HH that would unify the region of the graph explored (on all levels). This would make it easier to come up with a layout that is good for many source-destination pairs.

References

[AE06] Yossi Azar and Thomas Erlebach, editors. *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*. Springer, 2006.

[MZ06] Ulrich Meyer and Norbert Zeh. I/O-efficient undirected shortest paths with unbounded edge lengths. In Azar and Erlebach [AE06], pages 540–551.

[SS05] Peter Sanders and Dominik Schultes. Highway hierarchies hasten exact shortest path queries. In Gerth Stølting Brodal and Stefano Leonardi, editors, *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 568–579. Springer, 2005.

[SS06] Peter Sanders and Dominik Schultes. Engineering highway hierarchies. In Azar and Erlebach [AE06].