MIDAS Programming Project
Using Landmarks to Estimate Distances in a Graph

Andrew V. Goldberg and Renato F. Werneck
Microsoft Research Silicon Valley

June 4, 2010

1 Introduction

The goal of the programming project is to give the students a taste of algorithm design and engineering by developing and implementing an algorithm for a well-defined problem. Another goal is to get the students acquainted with one another before the school starts by working in teams. We hope that many students will find the project interesting and will have fun working on it. The project is open-ended in the sense that we do not know what the best solution is. We do know that, with moderate effort, one can improve upon the naive solution we give as an example. Investing more time may lead to bigger improvements.

The programming projects counts 10% of the total score, same as a homework problem. Although this is as much as one homework problem, the project offers a higher potential for learning. To provide additional incentive, the best team will be given the opportunity to present their solution during the last day of the conference.

2 Background

Suppose we are given an undirected graph $G = (V, E)$ with a length function $\ell : E \to \mathcal{N}$. Let $\text{dist}(v, w)$ denote the distance between vertices v and w. Note that in an undirected graph, $\text{dist}(v, w) = \text{dist}(w, v)$. For the project, graphs are road networks, with vertices representing intersections and edges representing road segments. Edge lengths represent transit times.

2.1 Landmark Triangulation

A *landmark* is a vertex x such that for every vertex v, we know (precompute) the distance $\text{dist}(x, v)$. In combination with the triangle inequality[1], landmarks can be used to estimate the distance between v and w, for any pair v, w, as follows:

- $\text{dist}(v, w) \le \text{dist}(v, x) + \text{dist}(w, x)$ is an upper bound on $\text{dist}(v, w)$; and
- $\text{dist}(v, w) \ge |\text{dist}(v, x) - \text{dist}(w, x)|$ is a lower bound on $\text{dist}(v, w)$.

[1] Note that triangle inequality always applies to *distances* in a graph, even if *edge lengths* do not satisfy the inequality.

Generally speaking, the upper bounds given by x are better when x is "between" v and w. Similarly, the lower bounds are better when x is "behind" v or w. When multiple landmarks are available, we can get more accurate bounds by taking the maximum lower bound and the minimum upper bound they give.

See Figure 1 for an example of how these bounds can be applied.

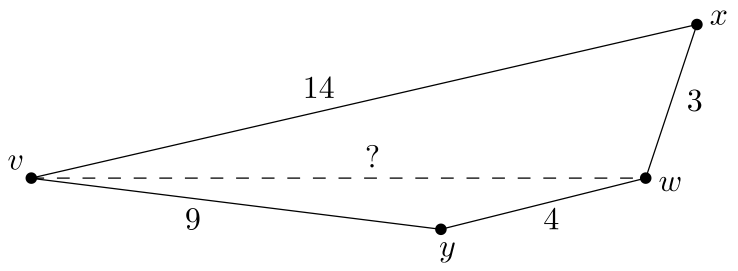

Figure 1: One can use landmarks x and y, together with triangle inequality, to compute bounds on $\text{dist}(v, w)$. Landmark x gives an upper bound of $\text{dist}(v, x) + \text{dist}(x, w) = 17$ and a lower bound of $\text{dist}(v, x) - \text{dist}(x, w) = 11$. Similarly, landmark y gives an upper bound of $\text{dist}(v, y) + \text{dist}(y, w) = 13$ and a lower bound of $\text{dist}(v, y) - \text{dist}(y, w) = 5$. Using both landmarks, we conclude that $11 \leq \text{dist}(v, w) \leq 13$.

The idea of using landmarks to compute distance lower bounds has been introduced in

> A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2005. `http://portal.acm.org/citation.cfm?doid=1070432.1070455`.[2]

and further discussed in

> A. V. Goldberg and R F. Werneck. Computing Point-to-Point Shortest Paths from External Memory. In *Proc. 7th Workshop on Algorithm Engineering*, pages 26-40, 2005. `http://www.siam.org/meetings/alenex05/papers/03agoldberg.pdf`.

You may find Section 7 of the first paper and Section 6.3 of the second paper good starting points for the project. You may also search the web for related work.

2.2 Landmark Generation

Given the graph and the desired number of landmarks, the *landmark generation problem* is to generate a set of landmarks that gives the best bounds for a particular input distribution. Your task in this project is to design and implement your own landmark generation module.

For the project, we set the number of landmarks to 20 and generate input vertex pairs v, w by choosing $v \neq w$ uniformly at random. For a given pair v, w, we compute the minimum upper bound U and the maximum lower bound L (over all 20 landmarks). The *score* for this query (pair) is given by:

$$\text{score} = 101^{L/U} - 1.$$

[2]See also `http://www.avglab.com/andrew/pub/soda05.pdf`.

Note that the ratio L/U is always a real number between 0 and 1, which means the score is a real number between 0 and 100. Higher scores indicate the bounds are more accurate (i.e., are closer to each other). The score for the entire set of input pairs is simply the average of all individual scores.

For a given pair of vertices, landmarks that give good lower bounds tend to give bad upper bounds. You may consider generating some landmarks aimed at lower bounds and some aimed at upper bounds.

3 The Project

We provide a complete solution to the landmark generation problem: a program that reads in a graph, generates landmarks, and evaluates their quality on random pairs of vertices. Your task is to design and implement your own landmark generation module to replace ours. (We give more details about our code in Section 4.)

As examples, we give two simple landmark generation procedures, one that generates random landmarks and one that generates landmarks using the *farthest* heuristic (see the above reference, Section 7). Although the latter produces better lower bounds, its score is actually worse than that of random selection because of the low quality of its upper bounds. You can look at these sample implementations as a starting point for your own implementation, and you can also compare your code to random landmark selection to see how well you are doing. You should be able to do better than random selection!

3.1 Basic Rules

- We will form teams of 3–4 students at random. A document with suggestions on how to work in teams (in Russian) can be found on MIDAS website (`http://logic.pdmi.ras.ru/midas/?q=workinginteams`).
- Every team must submit one program.
- We expect the teams to do all the work)except testing and calibration on the actual machines we will use for grading) before the school starts.
- Your code should be written in C++ and compile on Visual Studio Express 2010 on Windows XP.
- Any code you submit should be your own. You cannot use third-party libraries (like Boost). You may, however, call functions that we provide (e.g., Dijkstra's algorithm).
- You may search the relevant literature and use any ideas you find.
- In addition to the code, you must submit a brief write-up (in English) describing how your landmark generation algorithm works, making it clear which ideas are original and citing those from the literature.
- Your score will depend on the quality of your landmarks and on your write-up. We will not penalize you for using ideas from the literature.

- For each graph, your landmark generation must take at most 1000 seconds on the machine we use for testing.

3.2 Testing Environment

We will evaluate your landmark generation algorithm on a machine with 2GB of RAM and DualCore E2180 2.0 Ghz processor running Windows XP Professional SP3. We will use Visual C++ Express 2010 (which can be downloaded at no cost from `http://www.microsoft.com/express/Downloads/#2010-Visual-CPP`) to compile the code.

You can use any system for development, but we can only provide help with Visual Studio Express questions. Your submission must compile on the test machines, and you will have to make sure your program takes at most 1000 seconds on road network graphs with fewer than 2M vertices. From the first day of MIDAS, you will have access to the test machines to test and calibrate your code.

3.3 Grading

We expect you to develop your solution before the school starts and test it on the suggested graphs. During the school, you will have to make sure that your code compiles and runs on the (identical) machines we will be using for testing, and that your landmark generation procedure takes at most 1000 seconds on each graph. The latter may require some parameter adjustment, but if you completed the development before the school as you are supposed to, testing and tuning should not take long.

You must submit your source code by 12 noon on Wednesday, August 11 (Moscow time). You also need to submit a write-up describing how your landmark generation works (in English). The write-up should briefly describe the main ideas of your algorithm. Although there is no page limit, a brief summary with up to 2 or 3 pages should be sufficient. Please use word-processing software such as Word or Latex to make your write-up easy to read, and submit your document in a common format (such as `doc` or `pdf`).

We will test your code on three subgraphs of the `USA-road-t.USA.gr` graph with one to two million vertices (more details in Section 4.1). We will not make these subgraphs available until the project is due, so you should not tune your algorithm to a specific graph.

On each graph, to test your code we first run your landmark generation routine to generate 20 landmarks. We then compute the average score of your landmarks (as defined in Section 2.2) on a large number of pairs of vertices chosen uniformly at random. If the generation process fails or takes more than 1000 seconds, we will consider the score for this graph to be zero.

Most of your grade will depend on how good (high) the average score is (considering all three graphs). In addition, we will award a small number of points for the quality of your write-up.

4 Code

We provide the `midas` program, which is used to both generate and evaluate landmarks. Our distribution provides the source code in C++ together with a Visual C++ Express 2010 solution. [3] Download the distribution archive from

[3]The solution file may not work with other versions of Visual Studio.

`http://research.microsoft.com/en-us/downloads/624e5dad-4323-4f50-8531-9816222a187d/default.aspx` and unpack it; switch to the `prog_project` folder. Double-click on `midas.sln` to open it on Visual C++ Express 2010. You can then compile (build) the code by pressing `ctrl+shift+B`, for example. The resulting executable is `Release/midas.exe`. The source files are available in the `src` folder.[4]

This section discusses our code in detail, including which parts you will need to modify. You are free to use any of the code we provide. We suggest that you study our sample landmark generation procedures (for random and farthest landmarks) before you write your own.

4.1 Graph Representation

The input to the landmark generation problem is an undirected, connected graph in DIMACS format. Sample graphs can be downloaded from `http://www.dis.uniroma1.it/~challenge9/download.shtml`. You should use "travel time" graphs, with lengths corresponding to travel times. The files are in DIMACS format, but you **do not need** to understand the format if you use our parser.[5]

To grade your project, we will use three graphs (each with 1 to 2 million vertices) which we will extract from the full USA "travel time" graph. Each graph will be the result of cutting a rectangular region of the graph (based on latitudes and longitudes) and taking its largest connected component.

You can use any of the available graphs to test and tune your program, and you can create your own problems by extracting graphs from the full USA graph if you feel you need more instances (but you will have to write an extractor). Note that the download site also has coordinate files for the road network graphs. You should not use coordinates for landmark generation (since we will not provide this information for our test graph), but they might be useful to visualize your landmarks.

Internally, our implementation uses simple array-based static graph data structures. In particular, if you need a variable associated with every vertex, you can implement this by creating an array and storing the variable for the vertex with ID i in position i of the array. Note that vertex IDs are from 1 to n (on an n-vertex graph). If desired, you can also create your own graph structure and convert our representation to yours.

4.1.1 Examples: Traversing Lists and Trees

The following code fragment traverses the adjacency list of vertex `v`:

```
Arc *start, *end;
g->GetBounds(v, start, end);
for (Arc *a = start; a < end; a++) {
   VertexId w = a->head; //edge is (v,w)
   TCost c = a->cost;    //its cost is c
   //do something with the edge here
}
```

[4]This folder also contains a simple makefile, which may be helpful (possibly with modifications) if you use some other system/compiler for development. Just remember that (1) we will not be able to answer questions regarding alternative systems and (2) the code you submit must compile under Visual C++ Express 2010.

[5]If you are curious, the format is described at `http://www.dis.uniroma1.it/~challenge9/format.shtml`.

Note that the function `GetBounds(v, start, end)` makes `start` point to the first arc out of `v` and `end` point to the arc after the last arc. All arcs incident to `v` appear in consecutive locations from `start` to `end-1` (both inclusive).

Besides the graph itself, we also supply code for Dijkstra's shortest path algorithm (`dijkstra.h`), which you may find useful. To build the shortest path tree from some vertex `r`, simply run:

```
Dijkstra *dij = new Dijkstra(g);
dij->RunDijkstra(r);
```

Note that `RunDijkstra` itself does not return anything. It just updates the internal state of object `dij`, which you can query. To retrieve the distance from the root `r` to any vertex `v`, call `dij->GetDistance(v)`. To determine the parent of `v` in the shortest path tree, call `dij->GetParent(v)` (by convention, the parent of the root is the root itself). Finally, to determine the ID of the `i`th vertex scanned by Dijkstra's algorithm, call `dij->GetScannedVertex(i)`, with `i` between 0 and $n-1$ (note that this function returns the root if `i`=0).

The `GetScannedVertex(i)` method is particularly useful to traverse the shortest path tree efficiently. For example, if you visit the vertices in decreasing order of i (from $n-1$ down to 0), each vertex is guaranteed to be processed only after its children are. To illustrate this, the following code fragment computes heights of vertices in the tree (after a call to `RunDijkstra(r)`).

```
VertexId i, v, p, n = g->VertexCount();
VertexId *height = new VertexId[n+1];

//initialize all heights to zero
for (v=1; v<=n; v++) {height[v] = 0;}

//traverse the tree bottom-up, updating heights of parents
for (i=n-1; i>=1; i--) {
   v = dij->GetScannedVertex(i);
   p = dij->GetParent(v);
   if (height[p] < height[v]+1) {
       height[p] = height[v]+1;
   }
}
```

Finally, we also provide a pseudorandom number generator (see `mt_random.h`). For instance, to generate an integer uniformly at random in the interval $[3, 9]$, call `MTRandom::GetInteger(3,9)`. To generate a random double-precision number in the range $[0, 1)$, call `MTRandom::GetDouble()`.

4.2 Landmark Generation

To see how our program works, assume you compiled `midas.exe`, downloaded the San Francisco Bay Area graph (`USA-road-t.BAY.gr`) and saved it locally as `bay.gr`. Assuming you are in a command prompt window in the same directory where the files are, you can generate 20 landmarks using your method as follows:

```
midas.exe -generate bay.gr > landmarks.lmk
```

A more complete example is as follows:

```
midas.exe -generate bay.gr -nl 32 -seed 14 -method 0 -timebound 60 > landmarks.lmk
```

This will use your method (0, the default) to generate 32 landmarks (the default is 20) using seed 14 (the default is 1) and setting the maximum allowed time to 60 seconds (the default is 1000).[6]. The other valid methods are 1 (random) and 2 (farthest), both implemented in `landfile.h`. Your method is implemented in `myfinder.h`. The function that implements it is

```
void GenerateMyLandmarks(Graph *g, VertexId k, VertexId *list, MIDASTimer *timer)
```

It takes as input the graph (`g`), the number of landmarks desired (`k`), an array (`list`) of size `k`, and a timer (`timer`). Your implementation must fill this array (from positions 0 to $\texttt{k} - 1$) with the IDs of the landmarks you choose.

You may want to use the timer `timer` to ensure your code runs within the allotted time (1000 seconds). To determine the number of seconds elapsed since your function was invoked, call `timer->GetElapsedTime()`. Alternatively, you can call `timer->GetTimeToExpire()` to determine how many seconds are left until the timer expires. Your function must return before the timer expires!

The only file you need to modify is `myfinder.h`. All code you submit must be in this file.

After calling your function, the program outputs a landmark file in the following format:

```
p land <k>
t <x>
l <landmark_0>
l <landmark_1>
...
l <landmark_k-1>
```

Here, `k` is the number of landmarks and `x` is the time taken by the landmark generation function. The last `k` lines list the vertex IDs of the landmarks.

4.3 Landmark Evaluation

We also provide an evaluation procedure to compute a score for a set of landmarks. The procedure reads a graph and a landmark file and generates a large number of pairs of distinct vertices picked uniformly at random. For each pair, the landmarks are used to compute an upper bound U, a lower bound L, and the score (given by $101^{L/U} - 1$). The program outputs the average score obtained by the algorithm. To test the landmarks generated in the example above, run:

```
midas.exe -evaluate bay.gr < landmarks.lmk
```

A more complete version is as follows:

```
midas.exe -evaluate bay.gr -seed 17 -np 123456 < landmarks.lmk
```

This will use 17 as the seed for the random number generator (the default is 1) and test 123 456 pairs of random vertices (the default is 1 000 000).

The method responsible for landmark evaluation is `LandmarkEvaluator::EvaluateLandmarks`, implemented in `evaluator.h`. If desired, you may call this and other methods of this class within your landmark generation routine.

[6] We will set the time bound to 1000 when testing your code

5 Resources

Questions about the programming assignment should be send to `midas2010-project@live.ru`. We also created a forum `http://logic.pdmi.ras.ru/midas/?q=en/programming_project_forum`. The forum can be used to discuss the project.

6 What and How to Submit

By 12 noon on Wednesday, August 11, you must submit your `myfinder.h` file with your code and a `writeup.[doc,pdf]` file with the description of your algorithm. Please put your team number and a list of the team members in your write-up and in your code (as a comment). Mail the files (as attachments) to `midas2010-project@live.ru`. Include your team number in the message subject.