Computing the shortest path

Bidirectional search and Goal-directed Dijkstra

Alexander Kozyntsev

October 18, 2010

Abstract

We study the problem of finding a shortest path between two vertices in a directed graph. This is an important problem with many applications, including that of computing driving directions. In this work we present a bidirectional search algorithm that has expected running time $(\sqrt{n}\log n)$ instead of Dijkstra's unidirectional search algorithm which has the expected running time $(n \log n)$. Also, $A^\star$ search in combination with a new graph-theoretic lower-bounding technique based on landmarks and the triangular inequality is discussed. We also present bidirectional variants of $A^\star$ search.

Contents

1 Introduction

The shortest path problem is a fundamental problem with numerous applications. In this paper we study one of the most common variants of the problem, where the goal is to find a point-to-point shortest path in a directed graph. The fastest algorithm known due to Dijkstra, it performs a unidirectional search outward from the source node s toward the destination node d. A natural alternative, termed bidirectional search, is to search simultaneously forward from s and backward from d. In the worst case this is roughly twice as slow as unidirectional search, but empirical results suggest that it is faster in practice [5] and [4].

1.1 Definition of a Random Graph

For expository purposes we use the following probabilistic model. We assume that the graph is a complete directed graph with n nodes. An exponential distribution is used to choose the length of each edge: the mean length of an edge is $1/\lambda$ for all n. Then $Pr[\ell_e \leq x] = 1 - \exp^{-\lambda x}$ for some fixed $\lambda > 0$. The important property of exponential distribution that we will use later is that it is memoryless, that is, $Pr[\ell_e \leq x + y | \ell_e \geq x] = Pr[\ell_e \leq y]$ for all $x, y \geq 0$.

2 Analysis of Unidirectional search

We describe the procedure which will be used to simulate the behaviour of the unidirectional search algorithm on a random graph. The edge-lengths are chosen from the exponential distribution, but are unknown to the procedure. All edges are initially *inactive.* Associated with the procedure is a real-valued variable L, which starts at zero and increases in a continous manner. At the beginning of the procedure, s is added to S and all edges out of s are activated. If the procedure makes an edge e *active*, when $L = x$, then, when $L = x + \ell_e$, the edge is *discovered* by the procedure. An edge that has been discovered is no longer considered active.

Theorem 1. *Let the random variable X be the number of nodes in S at the end of the algorithm. Then*

$$Pr(X = k) = \frac{1}{(n-1)},\ for\ 2 \leq k \leq n \tag{1}$$

$$E(X) = \Theta(n) \tag{2}$$

Internal edges may also be discovered during the course of the algorithm. These discoveries do not terminate a stage; but they correspond to queue operations.

Theorem 2. *The expected number of edge discoveries in unidirectional search, both internal and external, is $\Theta(n)$.*

3 Bidirectional search

3.1 Definitions

The bidirectional search proceeds in two phases (cf. Figure 1). In the first phase we alternate between two unidirectional searches: one forward from s, growing a tree spanning a set of nodes S for which the minimum distance from s is known, and the other backward from d spanning a set D of nodes for which the minimum distance to d is known. Phase I alternately adds one node to S and one node to D, continuing until an edge crossing from S to D is drawn. At this point, the shortest path is known to lie within the search trees associated with S and D except for at most one edge from S to D.

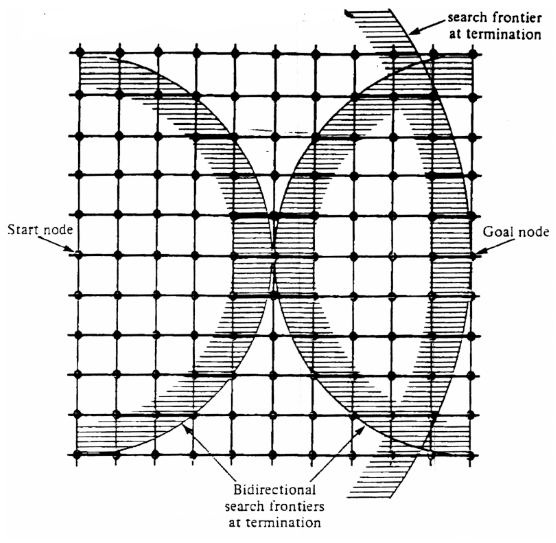

Figure 1: Bidirectional search

3.1.1 Phase I

Let X be the number of stages in Phase I. Then $E[X] = \Theta(\sqrt{n})$ and the total number of edges discovered in phase I is bounded by $2X$ plus the number of internal edges that are discovered.

Theorem 3. *The expected number of internal edges discovered in Phase I is $O(1)$*

The significance of Theorem 3 is that the expected overhead due to drawing internal edges from the queue is only $O(1)$ queue operations.

3.1.2 Phase II

In general, the $s-d$ path P found at the end of phase I is not necessarily the shortest $s-d$ path. However, by the following argument, there is no $s-d$ path P' which is shorter than P such that P' contains a node x not in $S \cup D$. It is not hard to see that the length of P is at most $L_S + L_D$. On the other hand, by the correctness of Dijkstra's algorithm, the distance from s to x along P' is at least L_S; similarly, the distance from x to d along P' is at least L_D. Thus P' is at least as long as P.
As a consequence, the shortest $s-d$ path lies entirely within the search trees associated with S and D except for at most one cross-edge. The aim of phase II is to find the shortest path among this resricted set of paths.

The following theorem shows that the expected number of queue operations in bidirectional search is $\Theta(\sqrt{n})$.

Theorem 4. *The expected number of edges discovered during Phase II is $O(\sqrt{n})$.*

3.2 Comparison

In an implementation of unidirectional or bidirectional search using priority queues, there are at most n edges in a queue at any one time, one for each node in the S or D sets. Hence each queue operation takes $O(\log n)$ time.
Unidirectional Search

1. Each queue operation takes $O(\log n)$ time

2. The expected running time is $O(n \log n)$

Bidirectional Search

1. Each queue operation takes $O(\log n)$ time

2. The expected running time is $O(\sqrt{n} \log n)$

4 Goal-directed search

4.1 Definitions

This technique, originating from [2], modifies the priority of active nodes to change the order in which the nodes are processed. More precisely, a goal-directed search adds to the priority $dist(u)$ a potential ρ_t depending on the target t of the search.

Definition 4.1. *A potential function is a function from vertices to reals. Given a potential function ρ, we define the reduced cost of an edge by*

$$\ell'(u, v) := \ell(u, v) - \rho_t(u) + \rho_t(v) \tag{3}$$

With a suited potential, the search can be pushed towards the target thereby reducing the running time while the algorithm still returns a shortest path. Intuitively speaking, one can compare a path in traffic network with a walk in a landscape. If you add a potential, the affected region is raised. If the added potential is small next to the target , you create a valley around the target. As walking downhill is easier than uphill, you are likely to hit the target sooner than without the potential added.

Definition 4.2. *Given a weighted graph $G = (V, E)$, $\ell : V \to \Re_0^+$, a potential $\rho : V \to \Re$ is called feasible, if $\ell(u, v) - \rho_t(u) + \rho_t(v) \geq 0$ for all edges $e \in E$.*

It can be shown that a feasible potential ρ is a lower bound of the distance to the target t if $\rho(t) \leq 0$. The tighter the bound is, the more the search is attracted to the target. In particular, a goal-directed search visits only nodes on the shortest path, if the potential is the distance to tha target.

We will now present three scenarios and how to obtain feasible potentials in these cases:

1. **Euclidean distances:** Assume a layout $L : V \to \Re^2$ of the graph is available where the length of an edge is somehow correlated with Euclidean distance of its end nodes. Then a feasible potential for a node v can be obtained using the Euclidean distance $||L(v) - L(t)||$ to the target t.
2. **Landmarks:** With preprocessing, it is possible to gather information about the graph that can be used to obtain improved lower bounds. In [1], a small fixed-sized subset $L \in V$ of "landmarks" is chosen. Then, for all nodes $v \in V$, the distance $dist(v, \ell)$ to all nodes $\ell \in L$ is precomputed and stored. These distances can be used to determine a feasible potential ([6]).
3. **Distances from Graph Condensation:** One can run Dijkstra's algorithm for a condensed graph and get the distances for all v to the target t. These distances provide a feasible potential for the time-expanded graph ([1]).

4.2 Landmark selection

Finding good landmarks is critical for the overall performance of lower-bounding algorithms. We will now present a few methods of selecting landmarks:

- **Random landmark selection:** The simplest way of selecting landmarks is to select k landmark vertices at random. This works reasonably well, but one can do better.
- **Farthest landmark selection:** Pick a start vertex and find a vertex v_1 that is farthest away from it. Add v_1 to the set of landmarks. Proceed in iterations, at each iteration finding a vertex that is farthest away from the current set of landmarks and adding the vertex to the set.
- **Planar landmark selection:** Find a vertex c closest to the center of the embedding. Divide the embedding into k pie-clice sectors centerd at c, each containing approximately the same number of vertices. For each sector pick a vertex farthest away from the center. To avoid having two landmarks close to each other, if we processed sector A and are processing sector B such that the landmark for A is close to the border of A and B, we skip the vertices of B close to the border.

The above three selection rules are relatively fast, and one can optimize them various ways.

5 Bidirectional lower-bounding algorithms

In this section we show how to combine the ideas of bidirectional search and $A^{\star}$ search. This seems trivial: just run the forward and the reverse searches and stop as soon as they meet. This doesn't work, however.

Definition 5.1. *We say that ρ_t and ρ_s are consistent if for all arcs (v, w), $\ell_{\rho_t}(v, w)$ in the original graph is equal to $\ell_{\rho_s}(w, v)$ in the reverse graph. In other words $\rho_t + \rho_s = const$*

It is easy to come up with lower-bounding schemes for which ρ_t and ρ_s are not consistent. If they are not, the forward and the reverse searches use different length functions. Therefore when the searches meet, we have no guarantee that the shortest path has been found.
One can overcome this difficulty in two ways: develop a new termination condition, or use consistent potential functions.

5.1 Symmetric approach

The following symmetric algorithm is due to [5]. Run the forward and the reverse searches, alternating in some way. Each time a forward search scans an arc (v, w) such that w has been scanned by the reverse search, see if the concatenation of the $s - t$ path formed by concatenating the shortest $s - v$ path found by the forward search, (v, w), and the shortest $w - t$ path found by the reverse search, is shorter than best $s - t$ path found so far, and update the best path and its length, μ, if needed. Also do the corresponding updates during the reverse search. Stop when one of the searches is about to scan a vertex v with $k(v) \geq \mu$ or when both searches have no labeled vertices. The algorithm is correct because the search must have found the shortest path by then.

5.2 Consistent approach

Given a potential function ρ, a consistent algorithm uses ρ for the forward computation and $-\rho$ (or its shift by a constant, which is equivalent correctness-wise) for the reverse one. These two potential functions are consistent; the difficulty is to select a function ρ that works well.

Let Π_t and Π_s be feasible potential functions giving lower bounds to the source and from the sink. Ikeda at [3] use $\rho_t(v) = \frac{\Pi_t(v)-\Pi_s(v)}{2}$ as the potential function for forward computation and $\rho_s(v) = \frac{\Pi_s(v)-\Pi_t(v)}{2}$ for the reverse one. Feasibility of the average of ρ_t and $-\rho_s$ is obvious.

6 Conclusion

Two methods for finding a shortest path have been presented. The first one, bidirectional search algorithm is a multiplicative factor of $\sqrt{n}$ faster than the unidirectional algorithm in the average case, where edge length are independent random variables. Presented results hold for a large class of probability distributions on random graphs, including both directed and undirected graphs, sparse as well as dense graphs, and graphs where the length of each edge is drawn from a different probability distribution. Besides, we has shown that unidirectional algorithm searches a ball with s in the center and d on the boundary, instead of bidirectional algorithm which searches two touching balls centered at s and d. And the second one, goal-directed search algorithm is based on landmarks and triangular inequality, for which several landmark selections were presented. Also, we have shown how to combine these two techiques - the ideas of bidirectional search and $A^\star$ search.

References

[1] A.V. Galdberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, page 156 165, 2005.

[2] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis or the heuristic determination of minimum cost paths. *IEEE transactions on systems science and cybernetics*, 4:100 107, 1968.

[3] T. Ikeda, Min-Yao Hsu, H. Imai, S. Nishimura, and other. A fast algorithm for finding better routes by AI search techniques. *Vehicle navigation and information systems conference*, 1994.

[4] Y. Ma. A shortest path algorithm with expected running time $o(\sqrt{V}\log v)$. *Master's project report, University of California, Berkeley.*

[5] I. Pohl. Bidirectional search. *Machine intelligence*, 6, 1971.

[6] D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. *Institut fur Theoretische Informatik*, 2005.