第44卷第10期 中国科学技术大学学报 Vol. 44, No. 10
2014年10月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Oct. 2014

Article ID: 0253-2778(2014)10-0874-07

An efficient lower-bounding approach to point-to-point shortest path problem

ZHANG Zhong, LV Min, SUN Guangzhong, CHEN Guoliang

(*School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*)

Abstract: Querying for the shortest path from a source vertex to a sink vertex in real time is a fundamental problem in numerous applications. Several lower-bounding schemes have been proposed to solve the problem so far, such as A* search and ALT algorithm. But these schemes adopted loose valuations on distance so that there are great potentialities for improving the lower bounds. A novel two-stage goal directed lower-bounding approach, called ACT algorithm, was proposed, which combined A* search, centers and triangle inequality with no prior domain knowledge. Unlike previous schemes, the new algorithm can obtain excellent distance bounds by exploiting a large number of pre-computed data. The experimental results on real road networks show that ACT algorithm significantly outperforms all previous algorithms. In some instances, the vertices scanned by ACT algorithm are only about 25% more than those on optimal paths.

Key words: shortest path; lower bounds; preprocessing; A* search; centers; triangle inequality

CLC number: TP393 **Document code**: A doi: 10.3969/j. issn. 0253-2778. 2014. 10. 012

Citation: Zhang Zhong, Lv Min, Sun Guangzhong, et al. An efficient lower-bounding approach to point-to-point shortest path problem[J]. Journal of University of Science and Technology of China, 2014, 44(10): 874-880.

一种求解点到点的最短路径的高效下界算法

张　钟，吕　敏，孙广中，陈国良

（中国科学技术大学计算机科学与技术学院，安徽合肥 230027）

摘要：在许多应用中，实时计算一个源点到一个目的点的最短路径是一个非常重要的问题. 学术界已经提出若干下界算法求解点到点的最短路径问题，如 A* 算法，ALT 算法等. 这些算法所使用的距离估值比较松散，仍然有很大的提升潜力. ACT 算法是一种新的两阶段目标制导下界算法，它组合使用了 A* 搜索，中心点和三角不等式，并且不依赖于特定领域的先验知识. 新算法充分利用了预处理数据，可以获得非常好的距离下界. 在真实路网上的实验结果表明，新算法的性能明显优于以往的算法. 在某些实例下，最优版本的 ACT 算法所扩展的顶点数量仅仅比最短路径上的顶点数量多 25%左右.

关键词：最短路径；下界；预处理；A* 搜索；中心点；三角不等式

Received: 2014-04-24; **Revised**: 2014-05-28

Foundation item: Supported by National Natural Science Foundation of China (61033009, 61303047).

Biography: ZHANG Zhong, male, born in 1987. Research field: Routing algorithm. E-mail: bergdino@mail. ustc. edu. cn

Corresponding author: SUN Guangzhong, PhD/associate professor. E-mail: gzsun@ustc. edu. cn

0　Introduction

The shortest path problem is a fundamental problem with numerous applications. In this paper we study the following P2P problem: To find a point-to-point shortest path in a weighted directed graph.

To solve the P2P problem, many algorithms have been proposed (see Ref. [1]). Almost all the algorithms are two-stage algorithms including an offline preprocessing stage and an online query stage. The input to the preprocessing stage of these algorithms is a directed graph $G(V,E,w)$ with n vertices, m arcs, and nonnegative arc weight. The preprocessing algorithms have to be executed once for each graph, thus taking a lot of time and outputing some auxiliary data. Then, after preprocessing, online search algorithms get queries specified by a source vertex s and a sink vertex t. The answer to a query is the shortest path from s to t. At the query stage, for each pair of queries, online search algorithms should respond very fast (real time) by using preprocessed data.

These algorithms fit into several categories. In this paper, we focus on lower-bounding algorithms. Generally speaking, lower-bounding algorithms select a labeled vertex v with the smallest value $k(v)$, where $k(v)$ is an estimate on distance from source to destination via v. The better the quality of estimates, the smaller the search space. Several lower-bounding algorithms have been proposed so far, such as A* search, reach and ALT algorithm.

In this paper, we propose a novel two-stage goal directed approach to solve the P2P problem, called ACT algorithm for it is based on A* search, centers, and triangle inequality. Its fundamental idea is similar to ALT algorithm. At the preprocessing stage, it selects a small subset of vertices as centers, computes the shortest distances between each pair of centers, and, for each vertex v, computes the shortest distances to and from several centers which are correlated by v. At the query stage, it uses A* search. Lower bounds can be computed in constant time using these pre-computed distances in combination with the triangle inequality. It can also be directly combined with reaches.

Our first contribution is a new preprocessing-based technique for computing distance bounds. Unlike previous schemes, the new algorithm can obtain excellent distance bounds by exploiting a large number of pre-computed data. Here we are talking about the triangle inequality with respect to the shortest path distances in the graph, not an embedding in Euclidean space or some other metric, which need not be present. Our experimental results show that ACT algorithms are much more efficient than previous algorithms.

Proper center selection is important to the quality of the bounds. As our second contribution, we give a greedy algorithm for selecting centers with no domain knowledge required. We study the relationship between query performance and the choice of each parameter of center selection.

Our third contribution is an experimental study comparing the new and previously known algorithms on real road network graphs taken from 9th DIMACS Implementation Challenge. We study which variants of ACT algorithms perform best in practice, and show that they compare very well to previous algorithms. Our experiments give insight into how ACT algorithm efficiency depends on the number of centers, the number of correlated centers, and graph size. Some of the experimental methodology we use may prove helpful in future work in this area.

1　Related work

Usually, to solve the P2P problem, one can run Dijkstra's algorithm until the target is reached. However, its performance is very poor since the search space is huge.

Pohl[2] presented A* search, which only searches a small area by estimating distances to the destination to guide vertex selection in a search from the source. It selects a labeled vertex v with the smallest value $k(v)=\text{label}(v)+\pi_t(v)$, where $\pi_t(v)$ is a lower bound on the distance from v to t, to scan next vertices at each iteration. Original variant of A* search to the P2P problem uses information implicit in the domain, such as Euclidean distances for Euclidean graphs, to compute lower bounds with no preprocessing required.

There are some works on pre-computation processing. Gutman[3] introduced the notion of "vertex reach". Informally, the reach of a vertex v is large if v is close to the middle of some long shortest path and small otherwise. For example, on road networks, local intersections have small reach and highways have large reach. Gutman showed how to prune an s-t search based on (upper bounds on) reaches and (lower bounds on) vertex distances from s to t, using Euclidean distances as lower bounds. He also observed that efficiency improves further when reaches are combined with Euclidean-based A* search, which uses lower bounds of the distance to the destination to direct the search towards it.

Goldberg and Harrelson[4] presented a different two-stage approach called ALT algorithm. They have shown that A* search performs significantly better when landmark-based lower bounds are used. Their approach is named ALT algorithm. The preprocessing algorithm computes and stores the distances between every vertex and a small set of special vertices, called landmarks. Queries use the triangle inequality to obtain lower bounds on the distances between any two vertices in the graph. Furthermore, Goldberg etc.[5] have presented significant improvements to ALT algorithm which is combined with the reach-based pruning.

2 Preliminaries

We define a potential function $\pi: v\in V\rightarrow R$ is a function from vertices to reals. For a given potential function π, the reduced cost of an arc (u,v) is defined as $w_\pi(u,v)=w(u,v)-\pi(u)+\pi(v)$. We replace w by w_π. Then for any two vertices s and t, the length of any s-t path, denoted by $\delta(s,t)$, is modified by the same amount. Thus a path is a shortest path in $G(V,E,w)$ if and only if it is a shortest path in $G(V,E,w_\pi)$, that is, the two problems are equivalent.

We define that π is feasible if w_π is nonnegative for all arcs. The following facts are well-known for any feasible potential function:

Lemma 2.1 If π is feasible and for a vertex $t\in V$, we have $\pi(t)\leqslant 0$, then for any $v\in V$, $\pi(v)\leqslant\delta(v,t)$.

Lemma 2.2 If π_1 and π_2 are both feasible potential functions, then $\max(\pi_1,\pi_2)$ is feasible.

Lemma 2.1 implies that we can often think of $\pi(v)$ as a lower distance bound from v to t. Lemma 2.2 allows us to combine feasible lower bound functions into a function that is also feasible, whose value at any point is at least as high as any original one.

It is easy to see that A* search is equivalent to Dijkstra's algorithm on $G(V,E,w_\pi)$. So that, A* search scans vertices in nondecreasing order of their distances from the source and scans each vertex at most once if $\pi_t(v)$ is feasible.

Theorem 2.1 Let π_1 and π_2 be two feasible potential functions such that $\pi_1(t)=\pi_2(t)=0$ and $\pi_2(v)\geqslant\pi_1(v)$ for any vertex v. If ties are broken consistently when selecting the next vertex to scan, the following holds. The set of vertices scanned by A* search using π_2 is contained in the set of vertices scanned by A* search using π_1.

Proof Suppose vertex v isn't scanned by search using π_1, then:

$$\because \quad \delta(s,v)+\pi_1(v)\geqslant\delta(s,t)+\pi_1(t)$$
$$\therefore \quad \delta(s,v)+\pi_1(v)\geqslant\delta(s,t)$$

If $\delta(s,v)+\pi_1(v)>\delta(s,t)$, then $\delta(s,v)+\pi_2(v)>\delta(s,t)$. Obviously, v will not be scanned with π_2. Otherwise, $\delta(s,v)+\pi_1(v)=\delta(s,t)$, which means v lies on another shortest path from s to t. But the search has terminated before scanning v, since it has found a shortest path. So in this case, either $\delta(s,v)+\pi_2(v)>\delta(s,t)$ or $\delta(s,v)+\pi_2(v)=\delta(s,t)$, v will not be scanned with π_2.

3 A* search based on landmarks

At present, the best lower-bounding algorithm is ALT algorithm. Its basic idea is to use landmarks and triangle inequality to compute feasible lower bounds.

Theorem 3.1 (Triangular inequality) For any edge $(u,v)\in E$, we have $\delta(s,v)\leqslant\delta(s,u)+w(u,v)$.

At the preprocessing stage, it selects a small subset of vertices as landmarks and, for each vertex in the graph, pre-computes distances to and from every landmark. At the querying stage, it uses A* search. The lower bounds are given by pre-computed data and triangle inequality. For example, given a set $L\subseteq V$ of landmarks and pre-computed distances $\delta(v,l)$, $\delta(l,v)$ for all vertices $v\in V$ and landmarks $l\in L$, by the triangle inequality, the following expression holds:

$$\because\quad \delta(l,v)+\delta(v,t)\geqslant\delta(l,t)$$
$$\therefore\quad \pi_t^l(v)=\delta(l,t)-\delta(l,v)\leqslant\delta(v,t)$$
$$\because\quad \delta(v,t)+\delta(t,l)\geqslant\delta(v,l)$$
$$\therefore\quad \pi_t^l(v)=\delta(v,l)-\delta(t,l)\leqslant\delta(v,t)$$

Therefore, $\pi_t(v)=\max_{l\in L}\max\{\pi_t^l(v)\}$ provides a lower bound for the distance $\delta(v,t)$.

4 ACT algorithm

Although the distances bounds based on landmarks are significantly more efficient than those based on Euclidean distances, we still think there is great potential promotional space for the quality of the lower bounds. It is an intuition that, the more landmarks are preprocessed, the better are the lower bounds obtained. But in practice, the number of landmarks is very small, because, most of landmarks are non-helpful for a special s-t path. Goldberg stated that efficiency is best when selecting 16 landmarks for a graph and using a subset during a query. Despite the fact that using fewer landmarks may lead to more vertex scans, this increase is very small relative to the improved efficiency of the lower bound computations for a large set of landmarks and a small set. One downside of ALT algorithm seems to be that more landmarks do not automatically lead to greater efficiency in querying. Thus we propose an enhanced algorithm called ACT (based on A* search, centers and triangle inequality). Our goal is to obtain better query performance which is proportional to the amount of preprocessing computation.

The main idea of ACT algorithm is similar to that of ALT algorithm. The preprocessing procedures are described below. First, we select a large number of vertices as centers which are far more numerous than landmarks. Generally speaking, centers are distributed in interior while landmarks are located on border of graphs. Then, for a selected set of centers, we calculate the shortest distance between each pair of centers using Dijkstra's algorithm. Finally, for each vertex v, we find the nearest λ centers surrounding v, correlate v with these centers and compute the shortest distances to and from correlated centers. All pre-computed results are stored in the main memory.

The lower bounds are given by pre-computed data and triangle inequality. For example, for a given pair of vertices v and t, centers c_1 and c_2. are correlated by v and t, respectively. Let $\delta(v,c_1)$ be the distance from v to c_1 and $\delta(c_2,v)$ be the one from c_2 to t. Then by the triangle inequality, the following expression holds.

$$\left.\begin{aligned}&\delta(c_1,v)+\delta(v,t)+\delta(t,c_2)\geqslant\delta(c_1,c_2)\\&\pi_t(v)=\delta(c_1,c_2)-\delta(c_1,v)-\delta(t,c_2)\leqslant\delta(v,t)\end{aligned}\right\}\tag{1}$$

Therefore, it provides a lower bound for the distance $\delta(v,t)$. According to Lemma 2.2, one can take the maximum $\pi_t(v)$, over all correlated centers to v and t, to compute the tightest lower bound. It is easy to see that, the smaller the values of $\delta(c_1,v)$ and $\delta(t,c_2)$, the better the quality of lower bounds, and further, the smaller the search space.

4.1 Centers selection

As noted above, finding good centers is critical for the overall performance of online search algorithms. But finding the optimal set of centers is an NP-hard problem. In this subsection, we present a polynomial greedy scheme to select centers. At the initialization, all vertices are listed at random. For each vertex, one checks if it has been scanned. If not, one picks it as a center, and then scans vertices nearby until the radius of the scan area is greater than Δ, where Δ is an input parameter.

4.2 Search

The online search algorithm uses A* search. At each iteration it selects a labeled vertex v with the smallest key, defined as $k(v) = \text{label}(v) + \pi_t(v)$, to scan next. This guides the search towards t effectively.

4.3 Pruning

Pruning may reduce the time and memory requirements of search algorithms. In our algorithm, we use the reach method to pruning. Given a path P from s to t and a vertex v on P, the reach of v with respect to P is the minimum of the lengths of the s-v and v-t subpaths of P. The reach of v, denoted by $r(v)$, is the maximum of the reach of v with respect to P over all shortest paths P through v. In short, $r(v)$ encodes the lengths of shortest paths on which v lies. We combine online search and reaches in the following way: when scanning a vertex v, one can prune the search at v if $r(v) < \min(\text{label}[v], \pi_t(v))$ because v will certainly not lie on the shortest path from s to t in this case.

4.4 Complexity analysis

Basically, our ACT algorithm is divided into two phases: off-line and on-line. In the off-line phase, a center set C is found using $O(m\log n)$ time and $O(n\log n)$ space, where n is the size of vertice set V and m is the size of edge set E in $G(V,E)$. After that, all distances of pair vertices in C are computed by Dijkstra's algorithm with time complexity of $O(cn(m+n\log n))$ and space complexity of $O(c)$, where c is the size of center set C. Specifically, we can run this procedure on reverse graph so that we can find the correlated centers for each vertex.

The computation complexity of the online search algorithm is equal to that of the implementation variant of Dijkstra's algorithm. In spite of this, our algorithm outperforms previous algorithms in practice.

5 Experiment

5.1 Setup

We experimented with five real road networks extracted from 9th DIMACS implementation challenge[6]: New York City, San Francisco Bay, Colorado, Florida and Northwest USA. We use road segment distances as arc lengths. Tab. 1 reports the graph sizes.

Tab. 1 Road networks used in the experiments

graph	description	nodes	arcs
NA	New York City	264 346	733 846
BAY	San Francisco Bay	321 270	800 172
COL	Colorado	435 666	1 057 066
FLA	Florida	1 070 376	2 712 798
NW	Northwest USA	1 207 945	2 840 208

Our code was written in C and compiled under GCC 4.6.2 with O3 flag. All experiments were run under Debian Linux 7.0 on a workstation, which has 16 GB of memory and a 3.30 GHz Xeon E3-1230 processor.

5.2 Preprocessing

In this section, we attempt to evaluate the cost of the preprocessing stage. We ran ACT preprocessing algorithms on each of graphs. We set $\lambda=3$ because it has the best efficiency for next test cases. About the other parameter, Δ, we set an appropriate value for each graph to roughly maintain the number of centers at 55 000 is the maximum number that fits in memory for our experimental platform. Because of the huge amount of calculation the algorithm required, we pre-computed in parallel with 8 threads except serially selecting centers. As a contrast, similar computations were made for ALT algorithm. On each graph, we generated 16 landmarks via maxcover scheme because Goldberg stated that its efficiency was excellent[7]. Results for preprocessing are presented on Tab. 2.

Tab. 2 Experimental results for Preprocessing

graph	ACT algorithm					ALT algorithm			
	GB	minutes				GB	minutes		
		selection	compute-8x	correlate-8x	total		selection	compute	total
NY	12	0.000 25	3.93	0.13	4.06	0.033	0.38	0.03	0.41
BAY	12	0.000 29	4.55	0.54	5.09	0.041	0.40	0.04	0.44
COL	12	0.000 40	6.46	1.29	7.75	0.056	0.57	0.06	0.63
FLA	12	0.001 12	17.04	3.82	20.86	0.137	1.42	0.12	1.54
NW	12	0.001 31	20.97	5.11	26.08	0.150	1.80	0.22	2.02

Evidently, the cost of preprocessing of ACT algorithm is higher, both in terms of time and space. But note that, compared with ALT algorithm, the number of centers increases by roughly three orders of magnitude. However, there are only one or two orders of magnitude increases in terms of time and space requirements, respectively.

A further observation is that preprocessing of ACT algorithm is more efficient. All in all, despite having an increase in the cost of preprocessing, we think it is feasible and worthwhile.

In the following sections, we designed experiments to test several variants of the lower-bounding algorithms. For each graph, we picked a random set of 10 000 *s-t* pairs and ran the lower-bounding algorithms on it for all variants tested. For consistency, each graph is tested with the same set of 10 000 pairs on all experiments.

To parse the experimental results of querying, we use two measures of performance. One is running time and the other is efficiency that is a machine-independent measure. The efficiency of a run of a P2P algorithm is defined as the number of vertices on the shortest path divided by the number of vertices scanned by the algorithm, reported in percentages. Note that an optimal algorithm scans only the vertices on the shortest paths whose efficiency is 100%.

5.3 Design choices

Our algorithm has two parameters: Δ and λ. In this section, we study the relationship between query performance and the choice of each parameter.

Let $\lambda=4$, we ran experiments with $\Delta=3\ 000$, 4 500, 6 000, 7 500, 9 000, 10 500 and 12 000 on NY graph. Tab. 3 shows the results. From this table one can find that, when the value of Δ decreases, the number of centers and efficiency increase, and the running time decreases. It is easy to see that the quality of lower bounds becomes better and better with an increase in the amount of centers. Thus the performance of query is directly proportional to the quality of lower bounds. In other words, it is also directly proportional to the amount of centers. Experimental results coincide with Theorem 2.1.

Tab. 3 Experimental results with various Δ

Δ	ratio of centers	time/μs	Eff.
3 000	21.06%	289	62.52%
4 500	12.68%	350	48.12%
6 000	8.55%	423	38.35%
7 500	6.16%	514	30.97%
9 000	4.69%	589	25.73%
10 500	3.67%	680	21.63%
12 000	2.94%	759	18.61%

Next we discuss the other parameters. Let $\Delta=3\ 000$. We run the same experiments with $\lambda=1,2,3,4,5,6$ and 7. Results are shown in Fig. 1.

It is obvious that, as the value of λ increases, so does algorithm efficiency. The algorithm can choose a tightest one from more candidates when computing lower bounds. But when $\lambda>3$, the search offers little efficiency improvement but hurts the running time because more loops and comparison operations are executed for computing lower bounds. As shown in Fig. 1, the version of $\lambda=3$ outperforms all other versions.

In summary, choosing as many centers as possible and correlating with appropriate centers

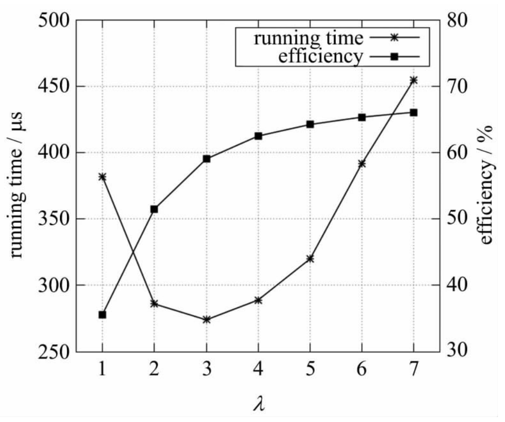

Fig. 1 Experimental results with various λ

will yield a good query performance.

5.4 Comparison

We compared four variants of lower-bounding algorithms: ALT algorithm (ALT), ACT algorithm (ACT), ALT algorithm combined with reach (REALT) and ACT algorithm combined with reach (REACT) in this section. Results are reported in Tab. 4.

Tab. 4 Experimental results for query

	NY	BAY	COL	FLA	NW
ratio of centers	21.06%	17.45%	12.97%	5.35%	4.55%
ALT	1 793 μs 12.35%	2 552 μs 10.08%	2 875 μs 13.01%	8 578 μs 7.24%	7 135 μs 8.64%
ACT	319 μs 59.06%	388 μs 54.14%	698 μs 34.41%	1 446 μs 25.98%	2 505 μs 17.80%
REALT	441 μs 32.63%	481 μs 36.71%	695 μs 41.88%	1 281 μs 37.34%	1 291 μs 37.43%
REACT	133 μs 80.68%	159 μs 81.48%	232 μs 74.04%	400 μs 72.54%	496 μs 63.68%

Comparing ACT with ALT, we note that ACT usually outperforms the other by more than a factor of two in efficiency. While in the term of running time, the reduction is more than three times. It is evident that the effect of ACT is unstable. This is to be expected as shown below. Due to the restriction on memory size, the number of centers is about 55 000 for each of the graphs. With the increasing of scale of graph, the ratio of centers to vertices decreases from 21.06% to 4.55%. Meanwhile, the quality of lower bounds becomes worse.

Combining lower-bounding algorithms with reaches yields a major performance improvement, both in REALT and REACT. REACT is the algorithm with the highest efficiency and performance. It usually outperforms ALT by one order of magnitude in running time. Even on the largest scale graph, its efficiency is still excellent. The vertices scanned are only about 25% more than those on the optimal path in some instances.

6 Conclusion

In this paper, we have presented a novel preprocessing lower-bounding algorithm. The proposed algorithm can obtain much tighter lower bounds than previous ones. And the performance with the amount of preprocessing is directly proportional. Despite the time and space requirements are larger, we think it is still feasible and worthwhile. Our experiments have confirmed the statements mentioned above.

References

[1] Delling D, Sanders P, Schultes D, et al. Engineering route planning algorithms[M]//Algorithmics of large and complex networks. Springer Berlin Heidelberg, 2009: 117-139.

[2] Pohl I. Bi-directional and heuristic search in path problems[M]. Department of Computer Science, Stanford University, 1969.

[3] Gutman R J. Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), 100-111.

[4] Goldberg A V, Harrelson C. Computing the shortest path: A search meets graph theory[C]//Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2005: 156-165.

[5] Goldberg A V, Kaplan H, Werneck R F. Reach for A*: Efficient point-to-point shortest path algorithms [C]//IN WORKSHOP ON ALGORITHM ENGINEERING & EXPERIMENTS. 2006.

[6] 9th DIMACS Implementation Challenge: http://www.dis.uniroma1.it/challenge9/download.shtml.

[7] Goldberg A V, Werneck R F F. Computing Point-to-Point Shortest Paths from External Memory[C]//ALENEX/ANALCO. 2005: 26-40.