Vol.2011-AL-134 No.5
2011/3/7

Accelerating A* algorithms by sweeping out small-degree nodes

Liang Zhao,[†1] Pipaporn Eumthurapojn[†1]
and Hiroshi Nagamochi[†1]

A* is an algorithm framework for calculating point-to-point shortest paths. This paper gives a simple method to accelerate A* algorithms in practice by sweeping out small-degree nodes from the priority queue, which can reduce the running time of the queue operations and the distance estimations. Experiments show that our method is efficient in practice, especially for A* algorithms with a heavy estimation function such as the ALT algorithm (Goldberg and Harrelson, SODA 2005) and its time-dependent generalizations.

1. Introduction

Let $G = (V, E)$ denote a directed graph with a set V of nodes and a set E of arcs. Let $\ell : E \times \mathbb{R} \to \mathbb{R}^+$ denote a nonnegative *transit time function* of arcs, i.e., $\ell(e,t)$ denotes the transit time of an arc e with a departure time t (at the tail of e). We assume that ℓ satisfies the *FIFO* (*First-In-First-Out*) *property*, i.e., $t_1 + \ell(e,t_1) \le t_2 + \ell(e,t_2)$ for all $e \in E$ and $t_1 \le t_2$. This paper considers the following *point-to-point* (PtoP) *time-dependent shortest path* (TDSP) problem.

Problem 1 (PtoP TDSP) *Given a directed graph $G = (V, E)$, a nonnegative and FIFO transit time function $\ell : E \times \mathbb{R} \to \mathbb{R}^+$, a source node $s \in V$, a destination node $d \in V$ and a departure time t_0, find an s,t-path P^*_{s,d,t_0} that minimizes the transit time from s to d with departure time t_0 at s.*

Remark 1 If $\ell(e,t) \equiv c_e$ is a *constant* for every arc e, it reduces to the classical shortest path problem (t_0 can be an arbitrary value), which is also referred as the *static* shortest path problem in this paper to distinguish from the more general time-dependent formulation as stated by Problem 1.

Remark 2 In general, we also call a directed network $N = (G, \ell)$ *undirected* if for all arcs $e = (v,w)$ in G, its reverse arc $e_r = (w,v)$ is also in G and the transit time functions are the same, i.e., $\ell(e,t) = \ell(e_r,t)$ for all e and t. For multiple arcs, e_r should be considered in pairs with e.

Remark 3 In general, we require the transit time function to satisfy the FIFO property since the problem without this requirement is known to be NP-hard[11)].

Shortest path problems have many applications and so far numerous algorithms have been proposed, see, e.g., 1), 5), 11). This paper focuses on the A* algorithm framework (described in Section 2) and algorithms that belong to this framework, which include the Dijkstra's algorithm[3)], the ALT algorithm[6)], and their time-dependent generalizations (e.g., 5), 8), 9), 11)), from a practical point of view.

For the idea, we observe that in practice, e.g. in road networks and scale-free networks, there can exist a large number of small-degree nodes (more precisely, nodes of degree 1 or 2). If we handle these nodes carefully and keep them out of the priority queue during the calculation, then it is possible to reduce the running time of queue operations and distance estimations, see Section 3. In our experiments with the time-dependent ALT algorithm[9)] and road networks, it can accelerate the calculation by a maximum of 30%, see Section 4 for details.

2. A* framework

The study of A* algorithm framework was started by Hart, Nilsson and Raphael[7)]. An A* algorithm is much like the well-known Dijkstra's algorithm except that it employs an *estimation function* h to guide the searching direction toward to the destination (Dijkstra's algorithm is a special A* algorithm with $h \equiv 0$). The performance depends on the estimation function h, and how to design a good h is an important issue in practice.

For general conditions for a good h, we refer the readers to 6) for the static problem and 9) for the general time-dependent problem. The idea in this paper does not depend on the estimation function, thus it works for any A* algorithm.

†1 Graduate School of Informatics, Kyoto University.
Corresponding author: Liang Zhao (E-mail: liang@i.kyoto-u.ac.jp)

To describe it, let us first show the A* framework using the following pseudo-code, where $h(v, t)$ denotes the estimation function for a node v at time t, and $g(v)$ denotes the temporary *arrival time* at v (from s). For simplicity, we assume that the destination d is reachable from the source s.

A* algorithm framework for the PtoP TDSP problem

Input : an instance $(G = (V, E), \ell, s, d, t_0)$.

Output : a shortest s, d-path P^*_{s,d,t_0} and its transit time.

```
1   $g(s) := t_0$; OpenSet := $\{s\}$; ClosedSet := $\emptyset$;
2   while OpenSet is not empty do
3     $v$ := a node in OpenSet with the smallest $f(v) := g(v) + h(v, g(v))$;
4     If there are multiple candidates, choose one with the smallest $g(v)$;
5     if $v = d$ break the while loop;
6     ClosedSet := ClosedSet $\cup \{v\}$; /* see Fig. 1 for an illustration */
7     for all arcs $e = (v, w) \in E$ do
8       $g'(w) := g(v) + \ell(e, g(v))$;
9       if $w \notin$ OpenSet $\cup$ ClosedSet then
10        $g(w) := g'(w)$; OpenSet := OpenSet $\cup \{w\}$; $p(w) := v$;
11      else if $w \in$ OpenSet and $g(w) > g'(w)$ then
12        $g(w) := g'(w)$; $p(w) := v$;
13      else if $g(w) > g'(w)$ then
14        $g(w) := g'(w)$; $p(w) := v$;
15        OpenSet := OpenSet $\cup \{w\}$; ClosedSet := ClosedSet - $\{w\}$;
16      end if
17    end for
18  end while
19  output path $s \to \ldots \to p(p(d)) \to p(d) \to d$ and time $g(d) - t_0$.
```

Remark 4 It is easy to see the correctness of the above framework if the estimation function h is nonnegative and is a lower bound of the minimum transit time from v to d with departure time t (at v). This framework generalizes the well-known A* framework for the *static* problem. If $h \equiv 0$, it reduces to the *generalized* or the *time-dependent* Dijkstra's algorithm[5),11)]. On the other hand,

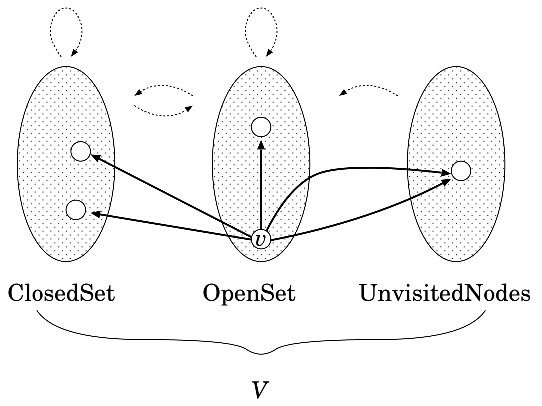

Fig. 1 An illustration of checking the incident arcs of node v in an A* algorithm (lines 7–17 in the pseudo-code), also called *relaxation* in the literature. In the figure, the dashed arrows show the state transition diagram of nodes.

if $h(v, t) \equiv h(v)$ is a constant for every node v, then line 4 is not required[9),10)]. We note that, if h satisfies certain conditions, then we can remove lines 13–15 to get a smaller searching area than the time-dependent Dijkstra's algorithm, see 9), 10) for the detail.

Remark 5 The algorithm requires to find a minimizer of $f(v)$ (and $g(v)$). For an efficient implementation, the OpenSet is usually maintained by a *priority queue* such as a heap. More precisely, the following operations of a priority queue are used: `insert`, `update` and `deletemin`. For the heap implementation, each operation can take $\Theta(\ln n_q)$ time, where n_q denotes the number of data in the heap. Therefore keeping a small priority queue is important in practice.

3. Method SmartUpdate for accelerating A* algorithms

Notice that how to design a good estimation function is not part of the A* framework. In 6), Goldberg and Harrelson gave an elegant method, called the ALT algorithm for this purpose. Later it is generalized to the time-dependent case. See 8)–10) for the details of these algorithms.

The motivation of this study is that, taking the ALT algorithm and its time-dependent generalization as examples, since calculating the values of h is a time-consuming task, it is important to find a way to avoid calculating function h (as well as keeping a small priority queue). This paper gives a method, called the *SmartUpdate* method, for that purpose by carefully handling small-degree nodes.

For simplicity, let us consider *connected* and *undirected* networks (see Remark 2) in the following. The degree $deg(v)$ of a node v is defined as the total number of arcs incident to v in the *undirected* sense, which equals to the in-degree and the out-degree in the *directed* sense. Notice that we allow multiple arcs.

Suppose we are checking an arc (v, w) in an A* algorithm (lines 7–17).

First we observe that, a degree-1 node w other than the destination d can be simply omitted because it can never be a node on an optimal path to d. Thus we never insert degree-1 node into the priority queue.

Next, we observe that the insertion of a degree-2 node w can also be avoided by considering its neighbour z that is not v. If such a neighbour z exists and is also of degree 2, we update (relax) it and repeat until a node f (called the *stop node*) of degree other than 2 or the destination d is found. We then insert the stop node f into the priority queue if it is d or is of degree 3 or more. See Fig. 2 for an illustration. To see the correctness, just consider a contraction of the path consisting of the skipped degree-2 nodes.

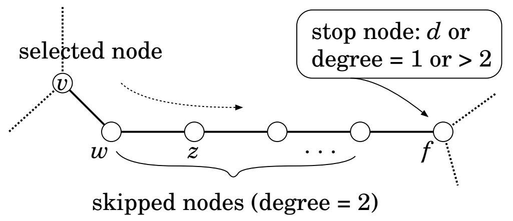

Fig. 2 An illustration of how to avoid insertion of degree 2 nodes into the priority queue by repeatedly relaxing its *next* neighbour. We insert the stop node f into the priority queue only if it is d or is of degree 3 or more.

So far we have shown our SmartUpdate method. Let us study its performance in practice by experiments in the next section.

4. Experiments

We show the instances and algorithms used in the experiments.

Table 1 Instance list (all are undirected and #arcs are in the undirected sense).

Name	network type	#nodes	#arcs	data source
BAY	static road network	321,270	400,086	4)
td-BAY	time-dependent BAY	32,1270	400,086	10)
itdk	static Internet network	190,914	607,610	the biggest component of 2) with randomly assigned lengths $\in [1, 100]$

Table 2 List of algorithms tested in the experiments. For ease of notation, the corresponding versions with SmartUpdate are not shown here.

Name	type	priority queue used	source
Dijkstra	static	binary heap	3)
td-Dijkstra	time-dependent	binary heap	5), 11)
ALT	static	binary heap	6)
td-ALT	time-dependent	binary heap	10)

For the ALT and the td-ALT algorithms which requires preprocessing, the parameters used in the calculation are shown in the next table.

Table 3 Parameters for the ALT and the td-ALT algorithms.

Name	#landmarks	#time_samples	method for choosing landmarks
ALT	16	–	farthest$^{6)}$
td-ALT	8	4	farthest$^{10)}$

Let us first show the degree distributions of each instances.

Table 4 Degree distribution (in %) of the instances.

Instance / Degree	1	2	3	4	5	6	7	≥ 8
BAY, td-BAY	22.7	19.2	44.8	13.1	0.2	0.0	0.0	0 (none)
itdk	21.7	22.5	12.4	9.0	6.0	4.6	3.5	20.3

From the degree distribution, we can see that over 40% of the nodes are of degree 1 or 2 even for different types of networks. We remark that the itdk instance obeys the power law, see 2).

In the test, for each instance, we randomly pick up 128 queries (s, d, t_0) and collect the total running time for each algorithm. This does not count the pre-processing time for ALT algorithms. Finally we compare the total running time with and without the proposed SmartUpdate method. The experimental results are shown in the following tables.

Table 5 Total running time of the experiments for the static problem, where SU stands for the proposed SmartUpdate method.

instance	Dijkstra	Dijkstra + SU	ALT	ALT + SU
BAY	7,880	5,160	1,280	950
itdk	11,790	9,070	1,360	1,170

Table 6 Total running time of the experiments for the time-dependent problem, where SU stands for the proposed SmartUpdate method.

instance	td-Dijkstra	td-Dijkstra + SU	td-ALT	td-ALT + SU
td-BAY	13,850ms	10,030ms	7,090ms	5,010ms

From the above tables, we observe that the proposed SmartUpdate method is efficient in practice. For a more detailed study, we show some results of the time-dependent instance td-BAY, which shows SmartUpdate reduces the number of priority queue inserts and the distance estimations. This explains why a speed-up can be obtained. We note that the speed-up ratio is about 30%. Considering the overhead, this matches the ratio of degree-1 and degree-2 nodes in the graph.

Table 7 Detailed results for the time-dependent experiment. All the values are the average of 128 random queries, where SU stands for the proposed SmartUpdate method.

	td-Dijkstra	td-Dijkstra + SU	td-ALT	td-ALT + SU
#selected_nodes	164,047	97,350	54,918	32,584
#queue_inserts	164,668	97,796	55,224	32,858
#queue_updates	13,644	13,745	5,140	5,173
#estimations	0	0	60,364	38,031

5. Conclusion

In this paper, we have proposed a simple yet practical method to accelerate the calculation of A* algorithms by keeping nodes of degree 1 or 2 out of the priority queue. This can reduce the running time of queue operations and distance estimations. Experiments on real road network and Internet router network show that the SmartUpdate method is efficient in practice. As a future work, it would be interesting to find a way to reduce the searching space.

References

1) Ahuja R.K., Magnanti T.L. and Orlin J.B.: *Network flows: theory, algorithms, and applications*, Prentice-Hall (1993).
2) CAIDA's router-level topology measurements, `http://www.caida.org/tools/measurement/skitter/router_topology/` (`itdk0304_rlinks_undirected.gz`)
3) Dijkstra E.W.: A note on two problems in connexion with graphs, *Numerische Mathematik*, 1, 269–271 (1959).
4) 9th DIMACS implementation challenge: Shortest paths, `http://www.dis.uniroma1.it/~challenge9/`
5) Dreyfus S.E.: An appraisal of some shortest-path algorithm, *Operations Research*, 17 (3), 395–412 (1969).
6) Goldberg A.V. and Harrelson C.: Computing the shortest path: A* search meets graph theory, *Proc. SODA 2005*, 156–165 (2005).
7) Hart P. E., Nilsson N. J., and Raphael B.: A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Systems Science and Cybernetics*, 4 (2), 100–107 (1968).
8) Nannicini G., Delling D., Liberti L. and Schultes D.: Bidirectional A* search for time-Dependent fast paths, *Proc. WEA 2008, LNCS* 5038, 334–346 (2008).
9) Ohshima T.: *A landmark algorithm for the time-dependent shortest path problem*, Master's thesis, Graduate School of Informatics, Kyoto University (2008).
10) Ohshima T., Eumthurapojn P., Zhao L. and Nagamochi H.: An A* Algorithm framework for the point-to-point time-dependent shortest path problem, *China-Japan Joint Conference on Computational Geometry, Graphs and Applications* (CGGA) (2010).
11) Orda A. and Rom R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length, *J. ACM*, 37 (3), 607–625 (1990).