情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-AL-137 No.7
2011/11/18

A*アルゴリズムのための実用的高速化手法

高　明　姫[†1]　　趙　　亮[†1]

A*は，二点間の最短路を計算するアルゴリズムのフレームワークである．特に (Dijkstra 法が有名な A*アルゴリズムの一つである．本論文は，A*アルゴリズムに対して，グラフの前処理を要しない節点の早期確定による高速化手法を提案する．

Some practical speed-up techniques for A* algorithms

Mingji Gao[†1] and Liang Zhao[†1]

A* is an algorithm framework for solving the Peer-to-Peer Shortest Path Problem. In particular, Dijkstra's algorithm is a well-known A* algorithm. This paper presents some practical speed-up techniques for A* algorithms based on Node Early-Fixing which does not require graph preprocessing.

1. A* framework

Let $G = (V, E, \ell)$ be a (directed or undirected) graph with a vertex set V, an edge set E and a nonnegative edge length function $\ell : E \to \mathbb{R}^+$. Given a *source* $s \in V$ and a *destination* $d \in V$, the *Peer-to-Peer Shortest Path Problem* asks to find a shortest s, d-path in G or its non-existence. For this problem, A* is a well-known framework of algorithms including the Dijkstra's algorithm, the ALT algorithm[1)] and others. The following pseudo-code describes the framework, where $h(v)$ denotes a *pre-defined* function for estimating the (shortest) distance from a vertex v to the destination d, and $g(v)$ denotes the shortest s, v-distance found so far in the algorithm.

†1 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University

```
Input: Graph $G = (V, E, \ell)$, source $s$ and destination $d$.
Output: a shortest $s, d$-path in $G$ with its length or the non-existence.
$g(s) := 0$;  $Q := \{s\}$;  $F := \emptyset$;
**while** $Q \neq \emptyset$ **do**
    $v := \text{argmin}\{g(u) + h(u) \mid u \in Q\}$; $Q := Q \setminus \{v\}$; $F := F \cup \{v\}$;
    **if** $v = d$ **break**;
    **for** all edges $e = (v, w) \in E$ **do**
        **if** $w \notin F \cup Q$
            $g(w) := g(v) + \ell(e)$;  $p(w) := v$;  $Q := Q \cup \{w\}$
        **else if** $w \in Q$ **and** $g(w) > g(v) + \ell(e)$
            $g(w) := g(v) + \ell(e)$;  $p(w) := v$
        **end if**
    **end for**
**end while**
**if** $d \in Q$ **output** path $s \to \ldots \to p(d) \to d$ and $g(d)$
**else output** "No $s, d$-path exits."
```

図 1 Pseudo-code of the A* framework considered in this study.

In this paper, we assume that h satisfies the *triangle inequality*, i.e.,

$$h(v) \ \leq \ h(w) \ + \ \ell(e), \quad \forall e = (v, w) \in E. \tag{1}$$

With this condition, it is easy to show that the above framework is correct (see, e.g., (1)). Notice that the Dijkstra's algorithm is an A* algorithm with $h \equiv 0$.

The performance of an A* algorithm depends on h. Recently Goldberg et al gave an elegant method for constructing a good h[1)] with graph preprocessing. For speed-up without preprocessing, Zhao et al proposed a practical method called SmartUpdate[2)]. The idea is to update (or initialize) the distance label $g(v)$ for the degree-1 and degree-2 vertices v without inserting them into Q. This can reduce the task for maintaining Q, and for calculating $h(v)$ which is sometimes *heavy*. This paper gives an extension of SmartUpdate, called Node Early-Fixing.

2. Node Early-Fixing for undirected graphs

Consider the framework in Fig. 1. Let us call a vertex in F a *fixed* vertex and the vertex v chosen in Line 3 the *active* vertex. Suppose that we are checking an edge (v, w) (Lines 6–10 in Fig. 1), where v is the active vertex and $w \notin F$. See Fig. 2.

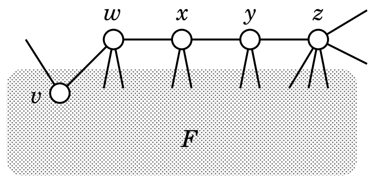

図 2 An illustration for Node Early-Fixing in undirected graph, where v is the active vertex.

We first update (or initialize) $g(w)$ like a normal A* algorithm (see Lines 6–10 in Fig. 1). Then we fix w (i.e., $F := F \cup \{w\}$ and $Q := Q \setminus \{w\}$) if w has only one unfixed neighbor x (i.e., $x \notin F$), and continue to update (or initialize) $g(x)$ as if w was the active vertex. We stop this process until a vertex $z \notin F$ with no or at least two unfixed neighbors is found (notice that if z has no unfixed neighbor, then its degree must be one). This finishes checking edge (v, w). Notice that SmartUpdate[2)] is a special case of Node Early-Fixing. The correctness proof is easy but omitted in this abstract version.

3. Node Early-Fixing for directed graphs

The argument for undirected graph can be applied to directed graph too. We just give an illustration to show that the situation is more complicated than undirected graphs. In particular, Fig. 3 shows that the number of skipped edges (i.e., edges that are not checked due to Node Early-Fixing) could be more than the case of undirected graph.

4. Experimental result

Node Early-Fixing cannot guarantee a speed-up for all kinds of graphs. Nevertheless, for sparse graphs with small average degrees, we can expect a good speed-up. For experimental study, we use a road network *BAY* (`http://www.dis.uniroma1.it/`

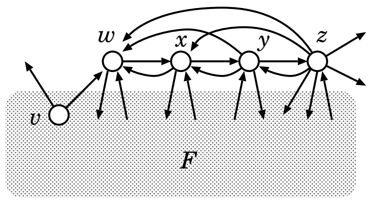

図 3 An illustration for Node Early-Fixing in directed graph, where v is the active vertex.

`~challenge9/`) with 321,270 vertices and (undirected) 400,086 edges, and a scale-free router network *ITDK* with 190,914 vertices, (undirected) 607,610 edges and random lengths in $[1, 100]$ (see `http://www.caida.org/` for the original unweighted graph).

We compare the performance of the normal Dijkstra's algorithm, the Dijkstra's algorithm with SmartUpdate and the Dijkstra's algorithm with Node Early-Fixing. All use a 6-heap as the priority queue that supports operations *deletemin, insert*, *update* and *delete*. For each instance, we randomly pick up 128 queries and for each query, we run the algorithm for 10 times. The results are shown in the next table.

表 1 Comparison of the average running time (in ms).

instance	Dijkstra	Dijkstra+SmartUpdate	Dijkstra+Node Early-Fixing
BAY	788	497	438
itdk	1100	817	770

From these results, we see that Node Early-Fixing is more efficient than SmartUpdate.

Acknowledgment

This work is supported by the Grant-in-Aid for Scientific Research no. 23700018.

参 考 文 献

1) A.V. Goldberg and C. Harrelson: Computing the shortest path: A* search meets graph theory, *Proc. SODA 2005*, 156–165 (2005).
2) L. Zhao, P. Eumthurapojn, and H. Nagamochi: A Practical Speed-up Technique for A* Algorithms (Abstract), in *AAAC* 2011, Taiwan (2011).