Computing the $n \times m$ Shortest Paths Efficiently

Tetsuo SHIBUYA
IBM Tokyo Research Laboratory

Computation of all the shortest paths between multiple sources and multiple destinations on various networks is required in many problems, such as the traveling salesperson problem (TSP) and the vehicle routing problem (VRP). This paper proposes new algorithms that compute the set of shortest paths efficiently by using the A* algorithm. The efficiency and properties of these algorithms are examined by using the results of experiments on an actual road network.

Categories and Subject Descriptors: F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Path and circuit problems*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: $n \times m$ shortest paths, A* algorithm, GIS

1. INTRODUCTION

Computation of all the shortest paths between multiple sources and multiple destinations on various networks is required in many problems, such as the traveling salesperson problem (TSP), the vehicle routing problem (VRP), the warehouse location problem (WLP), and the quadratic assignment problem (QAP). Accordingly, a function for performing such computation is required in geographical information systems (GISs), logistics tools, and so on. There are many fast heuristic algorithms for solving such problems as TSP, and the computation time needed to find all the shortest paths sometimes occupies a large percentage of the total computation time. A more efficient way of computing the set of shortest paths is therefore desired.

The Dijkstra method [Dijkstra 1959] is the most traditional and widely-used algorithm for this kind of problem. It can compute the shortest paths from one source to n destinations in $O(|E| + |V|\log(|V|)$ time on a directed graph $G = (V, E)$ with no negative edges [Fredman and Tarjan 1987]. Note that, assuming integer edge lengths stored in words or in one word, this bound can be improved [Cherkassky et al. 1997; Fredman and Willard 1994; Raman 1996a; Raman 1996b; Thorup 1996]. For a long time, this algorithm has been believed to be the best for computing all the shortest paths between two sets of vertices, especially on sparse

Address: 1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan.

graphs such as actual road networks. We call this problem the $n \times m$-pairs shortest paths problem or simply the $n \times m$ shortest paths problem. Using the Dijkstra method, it takes $O(\min(n, m) \cdot (|E| + |V| \log(|V|))$ time to obtain all the shortest paths between n sources and m destinations.

Much work has been done on improving the method's efficiency in solving the 2-terminal shortest path problem. The most famous example is the A* algorithm [Barr and Feigenbaum 1981; Dechter and Pearl 1985; Gelperin 1977; Hart et al. 1968; Nilsson 1971; Nilsson 1980; Shirai and Tsuji 1982], which improves the efficiency of the Dijkstra algorithm by using a heuristic estimator. Another famous technique is the bidirectional search algorithm [Champeaus 1983; Hiraga et al. 1993; Ikeda et al. 1994; Luby and Ragde 1985; Pohl 1971], which searches from both the source and the destination. But these techniques have been believed to be inapplicable to the $n \times m$ shortest paths problem.

This paper proposes two new algorithms based on the A* algorithm for solving the $n \times m$ shortest paths problem, and examines their efficiency and properties by using the results of experiments on an actual digital network of roads in the Kanto area of Japan (around Tokyo). We also discuss how to apply these algorithms to dynamic networks.

One of the algorithms uses an estimator based on those used for the 2-terminal A* algorithm. In the case of digital road networks, we can use an estimator based on Euclidean distance, and computing the estimates takes $O(|V| \log \max(n, m))$ time. This algorithm does not improve the computational bound of the Dijkstra method, but according to the experiments, it reduces the time for loading data, and in some cases, the total computing time.

The other algorithm uses the concept of the network Voronoi diagram (which is a division of V similar to the Voronoi diagram). Although it is based on the same principle as the first algorithm, it is very closely related to the bidirectional search method, so we call it the bidirectional-method-based A* algorithm, or simply the bidirectional method. This algorithm can be used on networks that do not have any appropriate estimator for 2-terminal A* algorithm (Euclidean distance, etc.), which is also a feature of the bidirectional method for 2-terminal problems. It takes $O(|E| + |V| \log(|V|))$ time to compute the estimates; this is also the time taken to construct the network Voronoi diagram. This algorithm also does not improve the computational bound of the Dijkstra method, but according to the experiments, it reduces the computing time by 30%-70% in most cases, compared with the Dijkstra method.

2. PRELIMINARIES

2.1 The Dijkstra Method

The Dijkstra method [Dijkstra 1959] is the most basic algorithm for the shortest path problem. The original algorithm computes the shortest paths from one source to all the other vertices in the graph, but it can be easily modified for the problem of computing the shortest paths from one source to several specified other vertices.

Let $G = (V, E)$ be a directed graph with no negative edges, $s \in V$ be the source, $T = \{t_1, t_2, \ldots, t_m\}$ be the set of destinations, and $l(v, w)$ be the length of an edge $(v, w) \in E$. The outline of the algorithm is then as follows:

ALGORITHM 1.

(1) Let U be an empty set, and the potential $p(v)$ be $+\infty$ for each vertex $v \in V$ except for $p(s) = 0$.
(2) Add to U the vertex v_0 that has the smallest potential in $V - U$. If $T \subseteq U$, halt.
(3) For each vertex $v \in V$ such that $(v_0, v) \in E$, if $p(v_0) + l(v_0, v) < p(v)$, update $p(v)$ with $p(v_0) + l(v_0, v)$ and let $previous(v)$ be v_0.
(4) Goto step 2.

The s-t_i shortest path is obtained by tracing $previous(v)$ from t_i to s, and its length is stored in $p(t_i)$. Note that the overall required time for this algorithm is $O(|E| + |V| \log(|V|)$ [Fredman and Tarjan 1987].

The most traditional and widely used method for solving the $n \times m$ shortest paths problem is repeating this procedure n times. Note that, if $m \leq n$, it is better to repeat m times the same kind of procedure in which we search from the destinations. Thus the required time for this problem is $O(\min(n, m) \cdot (|E| + |V| \log(|V|))$. The present paper focuses on how to improve this method.

2.2 The A* Algorithm

The A algorithm, an extension of the Dijkstra method, is a heuristic algorithm for the 2-terminal shortest path problem. It uses a heuristic estimator for the shortest path length from every vertex in the graph to the destination. Let $h(u, v)$ be the estimate for the u-v shortest path length, and $h^*(u, v)$ be the actual u-v shortest path length. Then the algorithm is as follows, letting t be the destination:

ALGORITHM 2.

(1) Let U be an empty set, and let the potential $p(v)$ be $+\infty$ for each vertex $v \in V$ except for $p(s) = 0$.
(2) Add to U the vertex v_0 that has the smallest value of $p(v) + h(v_0, t)$ in $V - U$. If $v_0 = t$, halt.
(3) For each vertex $v \in V$ such that $(v_0, v) \in E$, if $p(v_0) + l(v_0, v) < p(v)$, update $p(v)$ with $p(v_0) + l(v_0, v)$, let $previous(v)$ be v_0, and remove v from U if $v \in U$.
(4) Goto step 2.

If $h(v, t)$ satisfies the following constraint, which means that $h(v, t)$ is a lower bound of $h^*(v, t)$, the obtained path is guaranteed to be the optimal shortest path and the algorithm is called the A* algorithm [Barr and Feigenbaum 1981; Dechter and Pearl 1985; Gelperin 1977; Hart et al. 1968; Nilsson 1971; Nilsson 1980; Shirai and Tsuji 1982].

$$\forall v \in V \quad h(v, t) \leq h^*(v, t) \tag{1}$$

Note that if $h(v, t)$ equals $h^*(v, t)$ for all $v \in V$, the A* algorithm is known to search only the edges on the s-t shortest path. Moreover, the removal of vertices from U in step 3 can be omitted if the estimator satisfies the following constraint, which is called monotone restriction:

$$\forall (u, v) \in E \quad l(u, v) + h(v, t) \geq h(u, t). \tag{2}$$

An estimator under this constraint is called a dual feasible estimator. For example, the Euclidean distance on a road network is a dual feasible estimator. Obviously, $h^*(v,t)$ also satisfies the above constraint. Note that the number of vertices visited in this case is never larger than the number of those visited by the Dijkstra method.

2.3 The Bidirectional Method

The bidirectional method [Champeaus 1983; Hiraga et al. 1993; Ikeda et al. 1994; Luby and Ragde 1985; Pohl 1971] is also considered for the 2-terminal shortest path problem. It does not require any heuristic estimator, but can reduce the number of vertices visited in many cases. In this algorithm, the searches are carried out not only from the source but also from the destination. The algorithm is as follows:

ALGORITHM 3.

```
(1) Let $U$ and $W$ be empty sets, and let the potentials $p_s(v)$ and $p_t(v)$ be $+\infty$
    for each vertex $v \in V$ except for $p_s(s) = 0$ and $p_t(t) = 0$.
(2) Add to $U$ the vertex $v_0$ that has the smallest potential $p_s(v)$ in $V - U$. If
    $v_0 \in W$, goto step 7.
(3) For each vertex $v \in V$ such that $(v_0, v) \in E$, if $p_s(v_0) + l(v_0, v) < p_s(v)$,
    update $p_s(v)$ with $p_s(v_0) + l(v_0, v)$ and let $previous_s(v)$ be $v_0$.
(4) Add to $W$ the vertex $v_0$ that has the smallest potential $p_t(v)$ in $V - W$. If
    $v_0 \in U$, goto step 7.
(5) For each vertex $v \in V$ such that $(v, v_0) \in E$, if $p_t(v_0) + l(v, v_0) < p_t(v)$,
    update $p_t(v)$ with $p_t(v_0) + l(v, v_0)$ and let $previous_t(v)$ be $v_0$.
(6) Goto step 2.
(7) Find the edge $(u_0, w_0) \in E$ that has the smallest value of $p_s(u) + l(u,w) +$
    $p_t(w)$. The $s$-$t$ shortest path consists of the $s$-$u_0$ shortest path, the edge $(u_0, w_0)$,
    and the $w_0$-$t$ shortest path.
```

3. NEW APPROACHES FOR COMPUTING THE $N \times M$ SHORTEST PATHS

3.1 The Basic Principle

We discuss in this section how to compute all the shortest paths between two sets of vertices efficiently. Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of sources, and $T = \{t_1, t_2, \ldots, t_m\}$ be the set of destinations. It does not matter if some of the vertices are in both S and T.

The basic idea of our approach is to find an estimator that can be used in every search between two vertices s_i and t_j. Let $h(v, t_i)$ be an estimator for t_i. Then consider the following estimator:

$$h(v) = \min_i h(v, t_i). \tag{3}$$

Can this estimator be used in the search from any source to any destination? To answer this question, we present the following theorems:

THEOREM 1. *The estimator h as in expression (3) can be used as an A^* estimator for any t_i, if $h(v, t_j)$ is a lower bound of $h^*(v, t_j)$ for each j.*

Proof. Consider the case of searching the shortest path to t_k.

$$h(v) = \min_i h(v, t_i) \leq h(v, t_k) \leq h^*(v, t_k) \tag{4}$$

This inequality means that $h(v)$ is also a lower bound. Thus we can use it for an A* estimator for any t_i. □

Theorem 2. *The estimator h as in expression (3) is a dual feasible estimator for any t_i if, for any j, $h(v, t_j)$ is a dual feasible estimator for t_j.*

Proof. Consider the dual feasibility around some arbitrary edge (u, v). There must be some k such that $h(v) = h(v, t_k)$, and the following inequality is derived from the dual feasibility of $h(v, t_k)$:

$$l(u, v) + h(v, t_k) \geq h(u, t_k). \tag{5}$$

Thus we can obtain the following inequality:

$$l(u, v) + h(v) = l(u, v) + h(v, t_k) \geq h(u, t_k) \geq \min_i h(u, t_i) = h(u). \tag{6}$$

This means that $h(v)$ is a dual feasible estimator. □

Using this dual feasible estimator, we can solve the $n \times m$ shortest paths problem as follows:

Algorithm 4. *For each i, do the following:*

```
(1) Let $U$ be an empty set, and let the potential $p(v)$ be $+\infty$ for each vertex $v \in V$ except for $p(s_i) = 0$.
(2) Add to $U$ the vertex $v_0$ that has the smallest value of $p(v) + h(v)$ in $V - U$. If $T \subseteq U$, halt.
(3) For each vertex $v \in V$ such that $(v_0, v) \in E$, if $p(v_0) + l(v_0, v) < p(v)$, update $p(v)$ with $p(v_0) + l(v_0, v)$ and let previous$(v)$ be $v_0$.
(4) Goto step 2.
```

3.2 Techniques for a Road Network

For the 2-terminal problem in a road network, we often use the Euclidean distance $d(v, w)$ as a dual feasible estimator of the v-w shortest path length. Thus we can consider the following estimator for the $n \times m$ shortest paths problem:

$$h(v) = \min_i d(v, t_i). \tag{7}$$

This estimate is the Euclidean distance to the nearest vertex in the destination set T.

A *k-d* tree [Bentley 1975] is a very efficient data structure for coping with this nearest neighbor problem, especially in two-dimensional space. In k-dimensional Euclidean space, the time taken to build a *k-d* tree for m points is $O(m \log m)$, and the time taken to query the nearest neighbor of some other point is $O(\log m)$.

We have to compute the nearest neighbor of a particular point only once, because we use the same estimator in each search from each source. Thus, the extra time needed to compute all the required estimates is $O(|V| \log m)$, because we can ignore the time $O(m \log m)$ for building the tree. We call this algorithm the Euclidean-distance-based A* algorithm.

3.3 The Bidirectional Method

In this subsection, we discuss how we can solve the $n \times m$ shortest paths problem efficiently even if we do not have any appropriate estimator, as when we use the bidirectional method in the 2-terminal case.

Consider the following estimator in the 2-terminal shortest path problem:

$$h(v,t) = \min(c, h^*(v,t)), \quad c : constant. \tag{8}$$

The following corollary of Theorem 2 shows that this estimator is dual feasible:

Corollary 1. *The estimator $h'(v,t) = \min(c, h(v,t))$ is dual feasible if $h(v,t)$ is a dual feasible estimator and c is a constant.*

How does the algorithm behave if we use this estimator? First, we must search from the destination t to obtain the estimates until we find some vertex from which the shortest path length to the destination is larger than c. Let T' be the set of vertices covered by this backward search. The search will be done from the source s until it encounters some vertex in T'. Let S' be the set of vertices visited by this forward search at the time, and let E' be the set of edges $(u,v) \in E$ such that $u \in S'$ and $v \in T'$. Let v_0 be the vertex such that $e = (u_0, v_0) \in E'$ for some $u_0 \in S'$ and the s-t shortest path includes the vertex v_0. After the encounter, the algorithm searches only edges in E and on the v_0-t shortest path. Thus, its behavior is very similar to that of the bidirectional algorithm (Algorithm 3). If we let c be the largest value of $p_t(v)$ in the bidirectional method except for $+\infty$, the region searched by this algorithm is almost the same as that searched by the bidirectional method. Thus, the bidirectional method can be said to be a variation of the A* algorithm.

On this assumption, the bidirectional method can be extended for the $n \times m$ shortest paths problem to the A* algorithm, which uses the following estimator:

$$h(v) = \min_i h^*(v, t_i). \tag{9}$$

This estimator gives the shortest path length to the set of destinations. According to the theorems in the last subsection, it is a dual feasible estimator.

We can obtain this estimator by a variation of the backward Dijkstra method as follows. Let U be the set of vertices for which we want to know the value $h(v)$.

Algorithm 5.

```
(1) Let $W$ be an empty set. Let the potential $p(v)$ be $+\infty$ for each vertex $v \in V - T$, and $p(v)$ be 0 for each vertex $v \in T$.
(2) Add to $W$ the vertex $v_0$ that has the smallest potential in $V - W$, and set $h(v_0)$ with $p(v_0)$. If $U \subseteq W$, halt.
(3) For each vertex $v \in V$ such that $(v, v_0) \in E$, update $p(v)$ with $p(v_0) + l(v, v_0)$ if $p(v_0) + l(v, v_0) < p(v)$.
(4) Goto step 2.
```

Thus, the extra time taken to compute estimates as in (9) for all the nodes is $O(|E| + |V|\log(|V|)$, which is the same as the time taken by the ordinary Dijkstra method. Note that we here construct the network Voronoi diagram of the destination set T. The network Voronoi diagram of T is a subdivision of V to $\{V_i\}$

where the shortest path length from $v \in V_i$ to t_i is not larger than those to any other vertices in T. Note also that we do not have to compute these estimates twice or more, because we use the same estimator in each search from each source. We call this algorithm the bidirectional-method-based A* algorithm, or simply the bidirectional method.

This backward search should be performed as required at the same time as the forward search. In this way, we can, in most cases, reduce the number of vertices covered by the backward search. But if there are vertices from which there is no path to any of the destinations, the backward search may continue throughout the graph without stopping. Thus, if the graph has such vertices and is very large compared with the regions searched by the forward searches, we should modify the estimator as follows, using some appropriate constant c:

$$h(v) = \min(c, \min_i h^*(v, t_i)). \tag{10}$$

According to corollary 1, this is also a dual feasible estimator. To compute this kind of estimate for all the vertices, we only have to let $p(v)$ be c in step 1 of the algorithm 5. Note that this estimator is more similar to the estimator in expression (8) than that in expression (9). If we use this estimator, we do not have to search the whole graph to obtain this estimate for any vertex, but it may be difficult to decide an appropriate c. A good way to set c is to set some appropriate value larger than $\max_i \min_j h^*(s_i, t_j)$, which means that we do not set c until the estimates for all the sources have been computed.

3.4 Computation on Dynamic Networks

Computation of the $n \times m$ shortest paths on dynamic networks is also important. For example, an actual road network varies continuously because of traffic jams, road repairs, and so on.

Related to this, a heuristic estimators for the A* algorithm has the following properties. Let $G' = (V, E')$ be a modified form of the graph $G = (V, E)$ obtained by increasing some of the edge lengths. Note that this modification includes deletion of edges: we only let these edge lengths be $+\infty$.

Theorem 3. *If a heuristic estimator h satisfies inequality (1) on graph G, it also does so on graph G'.*

Proof. Let $h^*_G(v, w)$ be the shortest path length on graph G. It is obvious that the shortest path on G' is longer than that on G. Thus if the estimator satisfies inequality (1) on G, then the following inequality is satisfied:

$$\forall v \in V \quad h(v, t) \leq h^*_G(v, t) \leq h^*_{G'}(v, t). \tag{11}$$

This means that h also satisfies the inequality on G'. □

Theorem 4. *If a heuristic estimator h satisfies inequality (2) on graph G, it also does so on graph G'.*

Proof. Let $l'(v, w)$ be the length of edge (v, w) on graph G'. According to the definition of G', $l'(v, w)$ is not smaller than $l(v, w)$. Thus the following inequality is satisfied if h satisfies inequality (2) on G:

$$\forall (u, v) \in E \quad l'(u, v) + h(v, t) \geq l(u, v) + h(v, t) \geq h(u, t) \tag{12}$$

This means that h also satisfies the inequality on G'. □

According to these theorems, we can use the same estimator on the dynamic graph and do not have to recompute the estimates, if the only change is an increase of the edge lengths or deletion of edges. All the estimators that we proposed in this paper satisfy inequalities (1) and (2), and we can efficiently use them for such dynamic graphs.

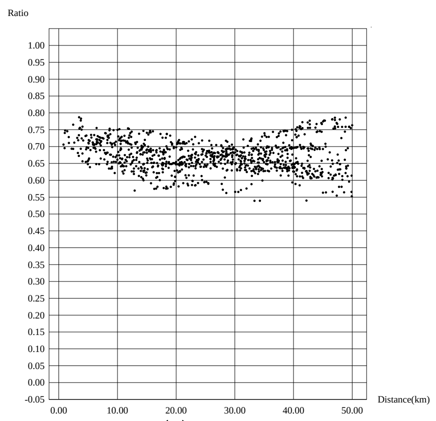
4. COMPUTATIONAL EXPERIMENTS ON A ROAD NETWORK

In this section, we investigate the efficiency of our algorithms by using actual digital road network data. The network covers a square region of 200 km $\times$ 200 km in the Kanto area of Japan, which contains several large cities such as Tokyo and Yokohama. There are $387,199$ vertices and $782,073$ edges in the network. We did all the experiments on an IBM RS/6000 Model 7015-990 with 512M bytes of memory. We use the time taken to traverse an edge as the length of that edge. Thus we compute the Euclidean-distance-based estimator using the value of the Euclidean distance divided by the maximum speed. In this section, we call the Euclidean-distance-based A* algorithm simply "the A* algorithm," and the bidirectional-method-based A* algorithm "the bidirectional method."

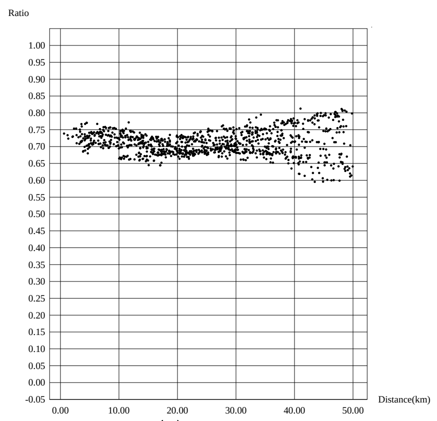
In our algorithms, we compute estimates at most once for any node regardless of the number of sources and destinations, and the time for computing estimates is relatively small if the number of sources and destinations is large. Therefore it is very important to analyze results of experiments for searching time without the time for computing estimates. Figures 1 and 2 show the ratios of the number of nodes visited by our algorithms to the number visited by the Dijkstra method. In the experiments, we first randomly chose 1000 points within a circle of diameter 100 km, as the starting points of searches. For the destination set, we also randomly chose (1) 5, (2) 10, (3) 20, (4) 50, (5) 100, and (6) 200 points within a circle of diameter 20 km and concentric with the circle containing the starting points. Note that the center is located near Shibuya, Tokyo. We then searched from these 1000 points to the destination sets by using various algorithms, and plotted the ratio of the number of nodes visited by our algorithms to the number visited by the Dijkstra method, and the distance from the center to the starting point.

For the A* algorithm, the ratio is about 50%-80% in cases where the number of destinations is small, and 60%-90% in cases where the number of destinations is large, but the difference between these ratios is not so remarkable. The ratio improves if the starting point is moved away from the center up to about 20 km, twice the radius of distribution of the destination set. For points at distances of more than 20 km, the average ratio does not change remarkably as the distance increases, but some have much better or much worse ratios than those nearer to the center.

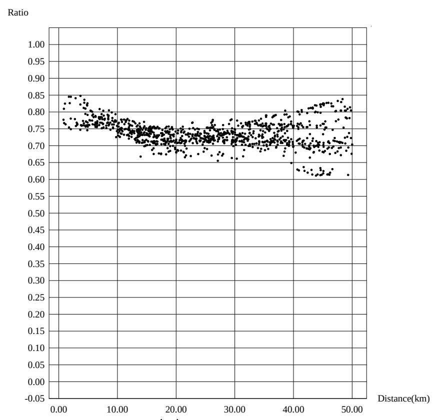
For the bidirectional method, the ratio is about 20%-50% in cases where the number of destinations is small, and 40%-70% in cases where the number of destinations is large. In this case, the number of destinations strongly influences the ratio. On the other hand, the influence of the distance from the center is very similar to that in the case of the A* algorithm, but in this case, the ratio decreases as the distance increases up to about 30 km, which is more than in the previous case.



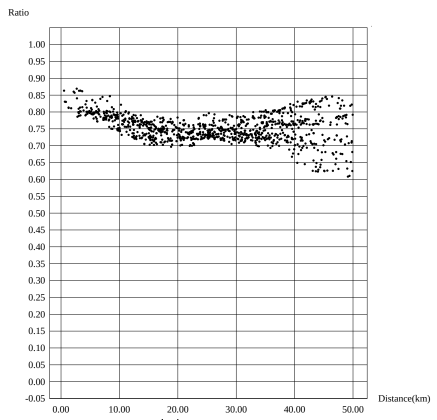
(1) 5 points



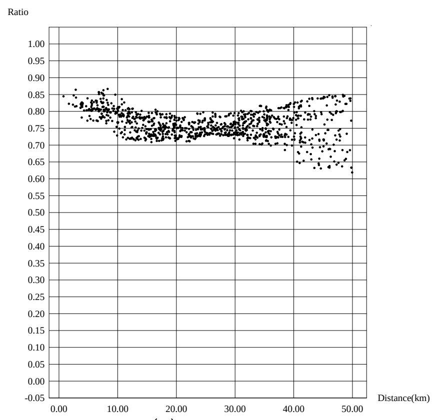
(2) 10 points



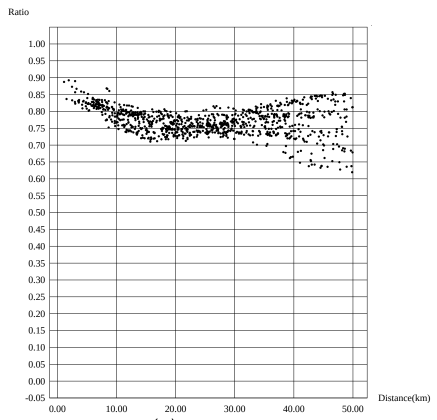
(3) 20 points



(4) 50 points



(5) 100 points



(6) 200 points

Fig. 1. Ratios of the number of nodes visited by the A* Algorithm to the number visited by the Dijkstra method.

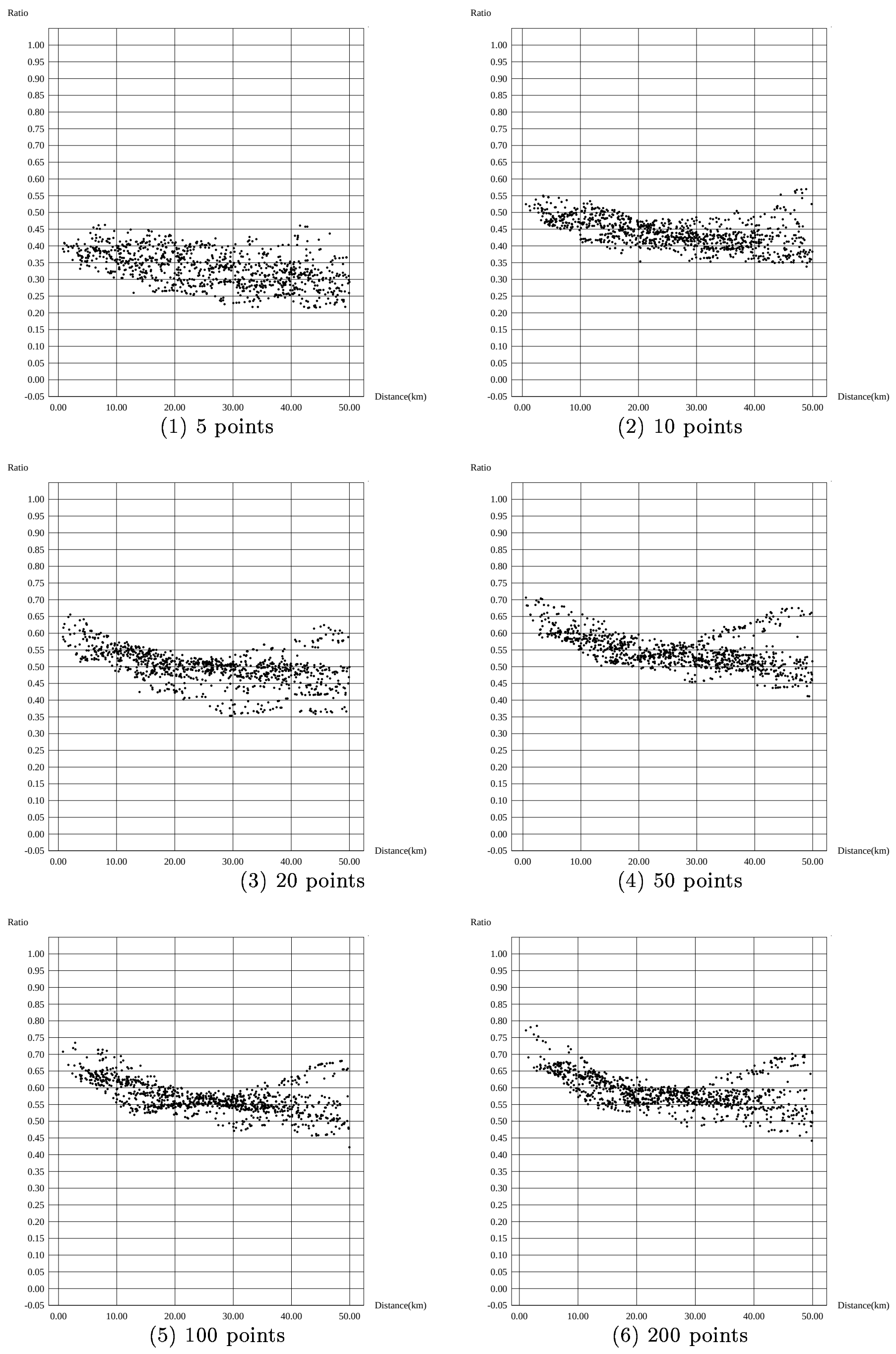
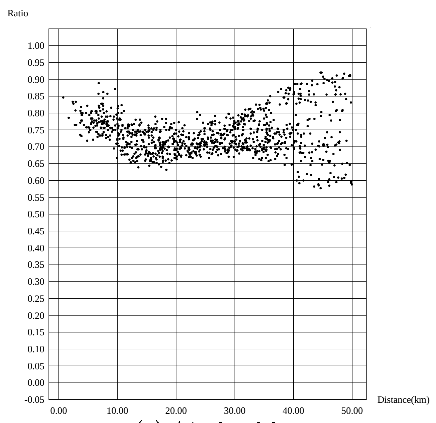
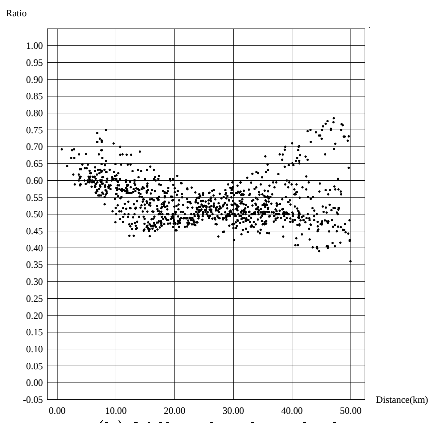


Fig. 2. Ratios of the number of nodes visited by the bidirectional method to the number visited by the Dijkstra method.



(a) A* algorithm



(b) bidirectional method

Fig. 3. Ratios of the time taken by our algorithms to the time taken by the Dijkstra method.

The above analyses are based not on the actual computing time, but on the number of visited nodes, because the actual computing time is influenced by the method of implementation, the machine type, and so on. Figure 3 shows the results of the same experiment using destination set (5), but it shows the ratio of time (which does not include the time for computing estimates). We can easily see that the actual computing time clearly reflects the number of visited nodes.

Table 1 shows the time for computing estimates for all 387,199 nodes in the network for the same destination sets as in the previous experiments. Hence the actual time for computing those estimates cannot be larger than this. In the case of the A* algorithm, this time increases as the number of destinations becomes larger. On the other hand, the number of destinations does not influence the time at all in the case of the bidirectional method, because the computation is always done by using a single Dijkstra method search in the case of the bidirectional method.

According to the number of searched nodes, and the time for computing estimates, the bidirectional method seems far better than the A* algorithm. Table 2 shows the results of the experiments in several actual cases. In the table, #Searched means the total number of vertices searched by all the n searches, #Loaded means the number of vertices loaded to memory, T_{total} means the total computing time (in seconds), and $T_{estimate}$ means the time taken to compute estimates. Note that T_{total} includes $T_{estimate}$. In case 1, we compute all the shortest paths among 50 points around Tokyo. The problem in case 2 is to compute the shortest paths between 20 sources and 150 destinations, both of which are distributed around Tokyo, while in case 3, the problem is to compute the shortest paths between 30 sources

Table 1. Time (sec) for computing estimates for all nodes in the network.

#destinations	5	10	20	50	100	200
A* algorithm	6.33	8.15	9.32	13.88	16.78	24.75
bidirectional method	3.78	3.78	3.78	3.78	3.80	3.82

Table 2. Computation Results in Actual Cases

case	Method	#Searched	#Loaded	T_{total}	$T_{estimate}$
1 (50 × 50)	Dijkstra	5671181	232117	65.58	-
	A*	4793515	180913	58.50	5.38
	Bidirectional	3860605	247245	42.98	1.98
2 (20 × 150)	Dijkstra	2636279	215130	30.10	-
	A*	2219739	170147	31.20	6.12
	Bidirectional	1868263	228316	21.83	2.43
3 (30 × 40)	Dijkstra	4818566	193723	56.00	-
	A*	2901990	125212	35.13	4.06
	Bidirectional	1559049	135635	17.37	0.84

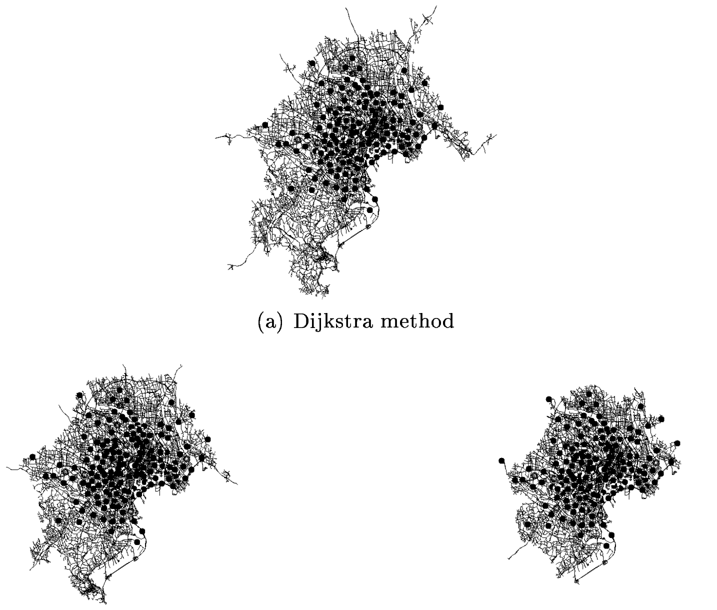
around Tokyo and 40 destinations around Yokohama. Note that the situations in all of these cases are very common in real problems. Note also that our algorithms are very advantageous in a situation such as case 3.

According to the table, the bidirectional method shows the best performance in all cases. It reduces the computing time by about 30% compared with the simple Dijkstra method in normal cases (1 and 2). If the sources and destinations are located in two distant clusters, as in case 3, the ratio becomes almost 70%, because the number of searched vertices is dramatically reduced. Note that both the original A* algorithm and the original bidirectional method reduce the computing time by 40% to 60% compared with the Dijkstra method in the 2-terminal case on such a road network [Ikeda et al. 1994]. The A* algorithm also reduces the number of searched vertices, but takes a long time to compute the estimates, even though we use a 2-d tree as in section 3.2. Thus it is not efficient, especially when the number of destinations (sources if the searches are done from the destinations) is large, as in case 2. However, it performs well compared with the Dijkstra method in situations like case 3, because the searching is done mainly in the direction of the destinations by using the Euclidean-distance-based estimator. Figure 4 shows the regions searched from the same source as in case 2. We can easily see that the bidirectional method visits the fewest vertices.

The A* algorithm is not fast as the bidirectional method on our system, but experiments reveal that it may be useful on some other systems. Table 2 shows that the number of vertices loaded to memory is small if we use the A* algorithm. Figure 5 shows the region loaded to memory in case 2. We can easily see that, among the three algorithms, the A* algorithm loads to memory the fewest vertices. To compute the estimates, the bidirectional method must load more vertices to memory than the A* algorithm. This means that the A* algorithm is one of the choices if the data storage device on the system is very slow.

5. CONCLUDING REMARKS

We have proposed new algorithms for the $n \times m$ shortest paths problem. We showed what kind of estimators for the A* algorithm could deal with this $n \times m$ shortest paths problem. As examples, we proposed two kinds of estimators, one based on Euclidean distance, and the other on the bidirectional method. We examined the efficiency of the algorithms using these estimators through experiments on an actual digital road network. The experiments revealed that the bidirectional-method-


(a) Dijkstra method

(b) A* algorithm

(c) Bidirectional method

Fig. 4. Regions searched from one of the sources by various algorithms

based A* algorithm is the best, and that it reduces the computing time by 30%-70% compared with the simple Dijkstra method. They also implied that the Euclidean-distance-based A* algorithm is useful on systems with very slow storage devices.

REFERENCES

Barr, A. and Feigenbaum, E. A. 1981. *Handbook of artificial intelligence*, William Kaufman, Inc., Los Altos, Calif.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9, 509–517.

Champeaus, D. 1983. Bidirectional heuristic search again. *J. ACM 30*, 22–32.

Cherkassky, B. V., Goldberg, A. V., and Silverstein, C. 1997. Buckets, heaps, lists and monotone priority queues. *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, 83–92.

Dechter, R. and Pearl, J. 1985. Generalized best-first search strategies and the optimality of a*. *J. ACM 32*, 3, 505–536.

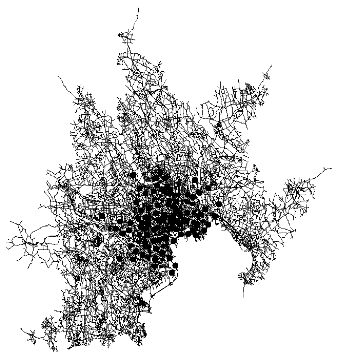
Dijkstra, E. 1959. A note on two problems in connection with graphs. *Numerical Mathematics 1*, 395–412.

Fredman, M. L. and Tarjan, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM 34*, 3, 596–615.

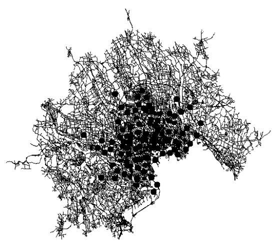
Fredman, M. L. and Willard, D. E. 1994. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comp. Syst. Sc. 48*, 533–551.

Gelperin, D. 1977. On the optimality of A*. *Artif. Intell. 8*, 1, 69–76.

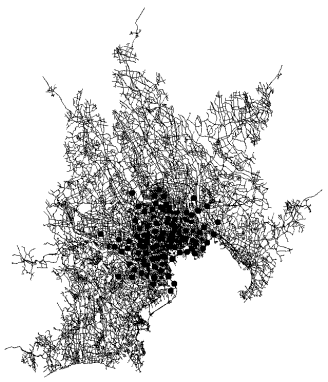
Hart, P. E., Nillson, N. J., and Rafael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. and Cyb. 4*, 100–107.



(a) Dijkstra method



(b) A* algorithm



(c) Bidirectional method

Fig. 5. Regions of vertices loaded by various algorithms

HIRAGA, T., KOSEKI, Y., KAJITANI, Y., AND TAKAHASHI, A. 1993. An improved bidirectional search algorithm for the 2 terminal shortest path. *The 6th Karuizawa Workshop on Circuits and Systems*, 249–254.

IKEDA, T., HSU, M. Y., IMAI, H., NISHIMURA, S., SHIMOURA, H., TENMOKU, K., AND MITOH, K. 1994. A fast algorithm for finding better routes by AI search techniques. *IEEE VNIS*, 90–99.

LUBY, M. AND RAGDE, P. 1985. A bidirectional shortest-path algorithm with good average-case behavior. *Proc. 12th International Colloquium on Automata, Languages and Programming, LNCS 194*, 394–403.

NILSSON, N. J. 1971. *Problem-solving methods in artificial intelligence*, McGraw-Hill, New York.

NILSSON, N. J. 1980. *Principles of artificial intelligence*, Tioga, Palo Alto, Calif.

POHL, I. 1971. Bi-directional search. *Machine Intelligence 6*, 127–140.

RAMAN, R. 1996a. Priority queues: small monotone, and trans-dichotomous. *Proc. ESA'96, LNCS 1136*, 121–137.

RAMAN, R. 1996b. A summary of shortest path results. *Technical Report TR 96-13*, Kings College, London.

SHIRAI, Y. AND TSUJI, J. 1982. *Artificial intelligence, Iwanami course: Information science*, *22.* Iwanami, Japan (in Japanese).

THORUP, M. 1996. On RAM priority queues. *Proc. 7th ACM-SIAM Symposium on Discrete Algorithms*, 59–67.