アルゴリズム 62－2
（1998．5．20）

A*アルゴリズムを用いた $n \times m$ 最短路問題の効率的解法

渋谷 哲朗

日本アイ・ビー・エム（株） 東京基礎研究所

概要: いろいろなネットワークにおいて、複数の始点集合と複数の終点集合がある場合に、すべての始点終点の組に対して最短路を求めることは、多くの問題を解く際に非常によく必要とされるものである。例えば、巡回セールス問題や車両配送問題などの多くの重要な問題において、そのような最短路問題を解くことが求められることが非常に多い。本論文では、そのような $n \times m$ 最短路問題を解くのに、A*アルゴリズムの概念を用いてダイクストラ法を２通りの方法で効率化している。更に、実際の道路ネットワークを用いた実験によって、それらのアルゴリズムの性能と性質を検証している。

Computing the $n \times m$ Shortest Paths Efficiently By Using the A* Algorithm

Tetsuo Shibuya

IBM Tokyo Research Laboratory

Abstract: 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 improves the Dijkstra method in two ways for this kind of shortest paths problems by using the concept of the A* algorithm. The efficiency and properties of these algorithms are also examined by using the results of experiments on an actual road network.

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 these problems, and the computation time needed to find all the shortest paths in the first step is sometimes much longer than that of the main routine. A more efficient way of computing the set of shortest paths is therefore desired.

The Dijkstra method [5] 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 [6]. For a long time, this algorithm has been believed to be the best for computing all the shortest paths between two sets of vertices. 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.

On the other hand, much work has been done on improving its efficiency in solving the 2-terminal shortest path problem. The most famous exam-

ple is the A* algorithm [1, 4, 7, 8, 12, 13, 15], which improves the efficiency of the Dijkstra algorithm using a heuristic estimator. Another famous technique is the bidirectional search algorithm [3, 9, 10, 11, 14], 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 the efficiency and properties of these algorithms by using the result of experiments on an actual digital road network in the Kanto area of Japan (around Tokyo).

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. According to the experiments, this algorithm 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 other one, 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 which does not have appropriate estimator for 2-terminal A* algorithm (Euclidian distance, etc), which is also the 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. According to the experiments, this algorithm is 30%-70% faster than the simple Dijkstra method in most cases.

2 Preliminaries

2.1 The Dijkstra Method

The Dijkstra method [5] 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 $(v, w) \in E$. Then the outline of the algorithm is 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 $O(|E| + |V| \log(|V|)$ [6].

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

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 $h(v,t)$ 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 [1, 4, 7, 8, 12, 13, 15].

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

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 searched in this case is always not larger than the number searched by the Dijkstra method.

2.3 The Bidirectional Method

The bidirectional method [3, 9, 10, 11, 14] is also considered for the 2-terminal shortest path problem. It does not require any heuristic estimator, but can reduce the number of searched vertices in many cases. In this algorithm, the searches are done 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 obtain 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.

$$h(v) = \min_i h(v,t_i) \leq h(v,t_k) \leq h^*(v,t_k) \quad (4)$$

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 $h(v,t_k)$:

$$l(u,v) + h(v,t_k) \geq h(u,t_k) \quad (5)$$

Thus we can obtain the following inequality:

$$\begin{aligned} l(u,v) + h(v) &= l(u,v) + h(v,t_k) \\ &\geq h(u,t_k) \\ &\geq \min_i h(u,t_i) \\ &= h(u) \end{aligned} \quad (6)$$

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:

$$h(v) = \min_i d(v,t_i) \quad (7)$$

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

k-d tree [2] is a very efficient data structure for coping with this nearest neighbor problem. In k-dimensional Euclidean space, the time for building a *k-d* tree for m points is $O(m \log m)$, and the time for querying the nearest neighbor of some other point is $O(\log m)$, for any k.

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:

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

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 searched 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:

$$h(v) = \min_i h^*(v, t_i) \qquad (9)$$

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 $p(v_0) + l(v, v_0) < p(v)$.*

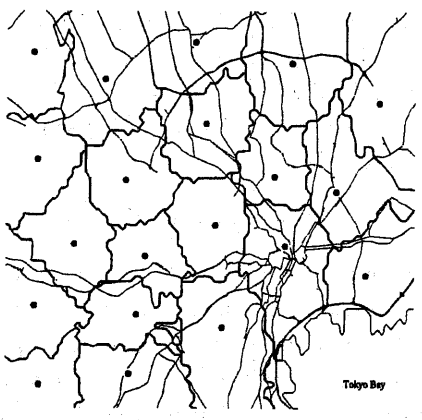


図 1: An example of the network Voronoi diagram on a digital road network

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. Figure 1 shows an example of this network Voronoi diagram of 20 points on an actual digital road network around Tokyo. In the figure, we show the boundary between the regions in which most (not all) of the nearest neighbors of the nodes are the same point. 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 of 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 with-

out 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:

$$h(v) = \min(c, \min_i h^*(v, t_i)) \qquad (10)$$

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 are computed.

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 kilometers × 200 kilometers in the Kanto area in 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."

Table 1 shows the results of the experiments in several 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 around Tokyo and 40 destinations around Yokohama.

According to the table, the bidirectional method shows the best performance in all cases. It is about 30% faster than 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, it is almost 70% faster than the Dijkstra method, because the number of searched vertices is dramatically reduced. Note that both the original A* algorithm and the original bidirectional method are about 40% to 60% faster than the normal Dijkstra method in the 2-terminal case on such a road network. 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 the destinations (sources if the searches are done from the destinations) is large, as in the 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 2 shows the regions searched from the same source as in case 2. We can easily see that the bidirectional method searches the fewest vertices.

The A* algorithm is not fast as the bidirectional method on our system, but the experiments reveal that it may be useful on some other systems. Table 1 shows that the number of vertices loaded to memory is small if we use the A* algorithm. Figure 3 shows the region loaded to memory in case 2. We can easily see that the A*

algorithm loads to memory the fewest vertices among the three algorithms. 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-based A* algorithm is the best, and that it is about 30%-70% more efficient than the simple Dijkstra method. They also implied that the Euclidean-distance-based A* algorithm is useful on systems with very slow storage devices.

参考文献

[1] A. Barr and E. A. Feigenbaum, *Handbook of Artificial Intelligence*, William Kaufman, Inc., Los Altos, Calif., 1981.

[2] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM, vol. 18, no. 9*, 1975, pp. 509-517.

[3] D. Champeaus, "Bidirectional Heuristic Search Again," *J. ACM, vol. 30*, 1983, pp.22-32.

[4] R. Dechter and J. Pearl, "Generalized Best-First Search Strategies and the Optimality of A*," *J. ACM, vol. 32, no. 3*, 1985, pp. 505-536.

[5] E. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerical Mathematics, vol. 1*, 1959, pp. 395-412.

[6] M. L. Fredman and R. E. Tarjan, "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms," *J. ACM, vol. 34, no. 3*, 1987, pp. 596-615.

[7] D. Gelperin, "On the Optimality of A*," *Artif. Intell. vol. 8, no. 1*, 1977, pp. 69-76.

[8] P. E. Hart, N. J. Nillson, and B. Rafael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Sys. Sci. and Cyb. SSC-4*, 1968, pp. 100-107.

[9] T. Hiraga, Y. Koseki, Y. Kajitani, and A. Takahashi, "An Improved Bidirectional Search Algorithm for the 2 Terminal Shortest Path," *The 6th Karuizawa Workshop on Circuits and Systems*, 1993, pp. 249-254 (in Japanese).

[10] T. Ikeda, M. Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, K. Tenmoku, and K. Mitoh, "A Fast Algorithm For Finding Better Routes By AI Search Techniques," *IEEE VNIS'94*, 1994, pp. 90-99.

[11] M. Luby and P. Ragde, "A Bidirectional Shortest-Path Algorithm With Good Average-Case Behavior," *Proc. 12th International Colloquium on Automata, Languages and Programming, LNCS 194*, 1985, pp. 394-403.

[12] N. J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[13] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga, Palo Alto, Calif., 1980.

[14] I. Pohl, "Bi-Directional Search," *Machine Intelligence, vol. 6*, pp. 127-140, 1971.

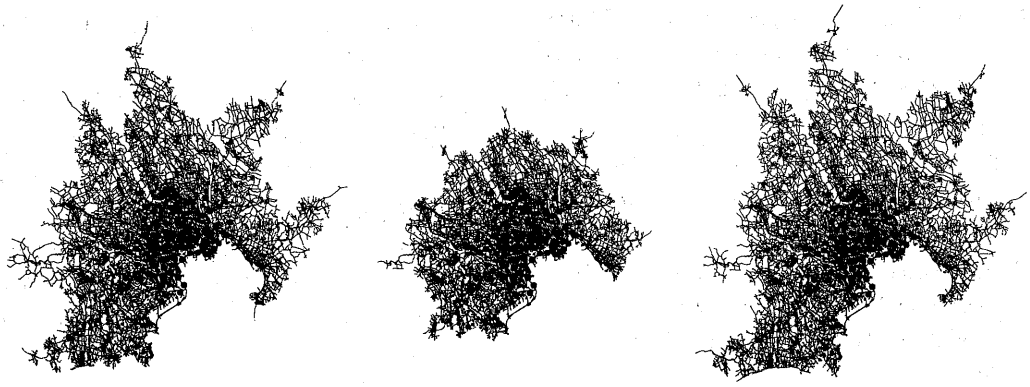
[15] Y. Shirai and J. Tsuji, "Artificial Intelligence," *Iwanami Course: Information Science, vol. 22*, Iwanami, Japan, 1982 (in Japanese).

表 1: Computational Results

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

(a) Dijkstra method (b) A* algorithm (c) Bidirectional method

図 2: Searched regions from one of the sources by various algorithms



(a) Dijkstra method (b) A* algorithm (c) Bidirectional method

図 3: Regions of vertices loaded by various algorithms