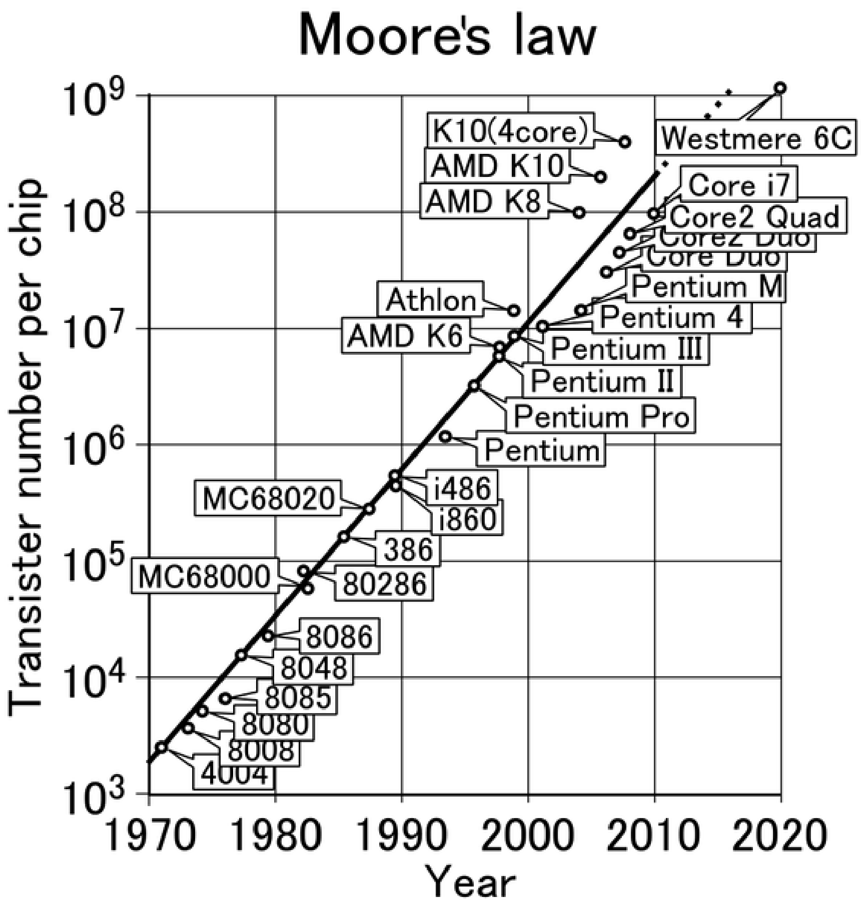
地図情報処理特論 第３回、第4回

授業計画

回	授業内容	予習・復習内容	備考
1	地図情報の歴史と概要	データの種類や解析目的による各手法の位置づけを理解する。	
2	電子地図情報のメッシュ構造:ＪＩＳ標準の地図のメッシュ構造について述べる		
3	電子地図情報で扱う属性:電子地図情報で扱われる情報の表現方法について		
4	電子地図情報の可視化技術(1):コンピュータで地図情報を可視化する方法		
5	電子地図情報の可視化技術(2):コンピュータで地図情報を可視化する方法		レポート#1　DRMの可視化
6	GPSの原理と位置情報表現:GPS:全地球測位衛星の原理について		
7	GPSデータの電子地図への対応（マップマッチング）:マップマッチングアルゴリズムの概要		
8	電子地図情報の検索:地図情報の階層管理、ハッシングによる高速検索		
9	経路探索(1):最適経路探索アルゴリズムダイクストラ法		
10	経路探索(2):最適経路探索アルゴリズムA-star法など		
11	経路探索演習	経路探索問題を指定した実ネットワークで行います。	レポート#2　最速経路
12	プローブ情報処理（１）:プローブ情報の分析：交通情報生成、災害時の利用など		
13	プローブ情報処理（２）:プローブ情報の多様な利用方法：地図作成、クレンジング、圧縮方法など		
14	プローブ情報処理演習I	例題のプローブ情報を分析して渋滞情報を生成します。	
15	プローブ情報処理演習2	例題のプローブ情報を分析して渋滞情報を生成します。	レポート#3：渋滞情報生成

How to Reduce Computation Time

Progress of computer is fast. It doubles in every 18months.



A super computer in late 1980's is comparable to about 1/100 of a today's PC.

HITAC S-810/20K(1987)

Memory 64MB
HDD ～2GB
0.63 GFLOPS

x 100

Memory ～16GB
HDD ～TB
51 GFLOPS

How to Reduce Computation Time

However, computation time often explodes if you don't take extra care in developing your algorithms even though you have a high speed computer. Computers amplify your lack of consideration millions of times more.

Map-matching algorithm is a good example.

Examples of probes on 2nd meshes 2次メッシュ状のプローブ情報数

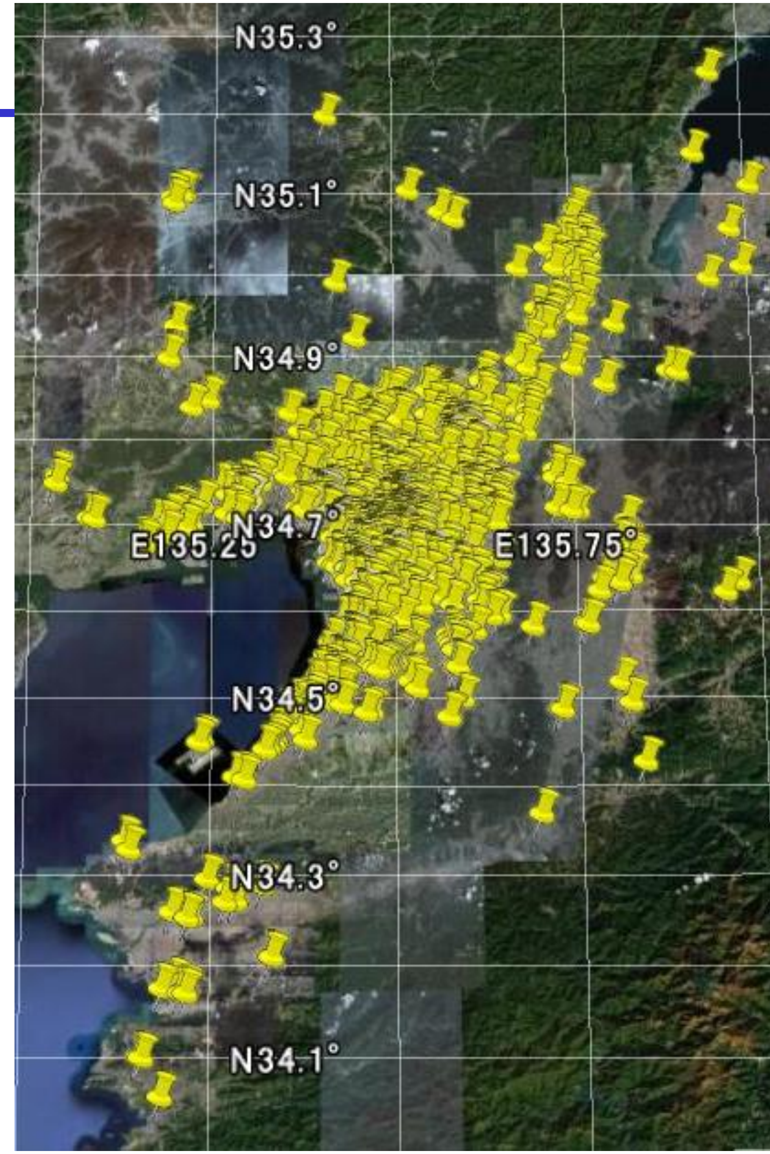
北緯35度20分

北緯34度40分

北緯34度

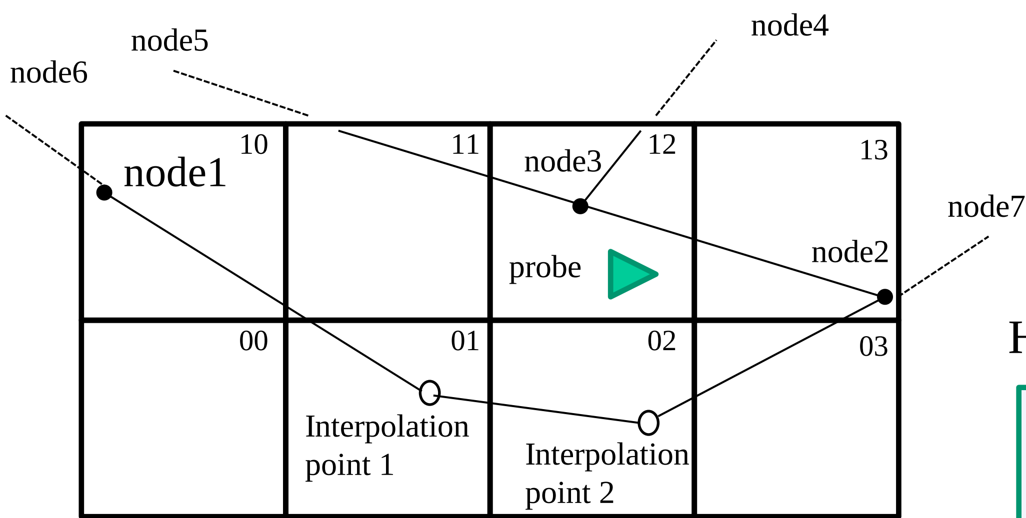
52	7	3,430	5,225	1,283	94	0	0	0	2,936
	6	2,805	3,185	1,736	1,632	0	0	6	10,843
	5	5,041	4,269	1,484	3,203	460	0	310	17,471
	4	278	12,282	1,727	724	14,458	28,191	52,295	25,573
	3	3,496	11,313	759	8,303	8,578	277,971	113,075	80,538
	2	19,312	76,007	17,747	39,348	218,696	492,143	111,611	6,808
	1	17,498	28,630	117,947	757,524	1,754,517	238,029	30,542	1,279
	0	83,837	346,871	1,069,772	3,202,566	1,836,787	259,601	61,005	0
51	7	128,694	99,387	0	2,840,180	1,063,173	67,431	176,772	82,555
	6	4,291	0	1	1,181,389	532,371	130,624	21,523	1,551
	5	0	1,243	152,867	261,392	59,277	30,914	7,000	182
	4	0	5,333	150,438	25,607	1,923	12,154	416	0
	3	147	50,675	67,740	2,468	3,972	193	0	0
	2	129	82,448	6,988	0	0	0	0	0
	1	868	15,711	590	1	0	0	0	0
	0	176	5,707	87	0	0	0	0	0
		0	1	2	3	4	5	6	7
		34							

東経135度　　東経135度



Hash Table Preparation

Hash table helps to know which links are in a sub mesh of current focus.



If the probe is known to
be within a sub-mesh 12,
You can find the link quickly
by looking up the table of 12

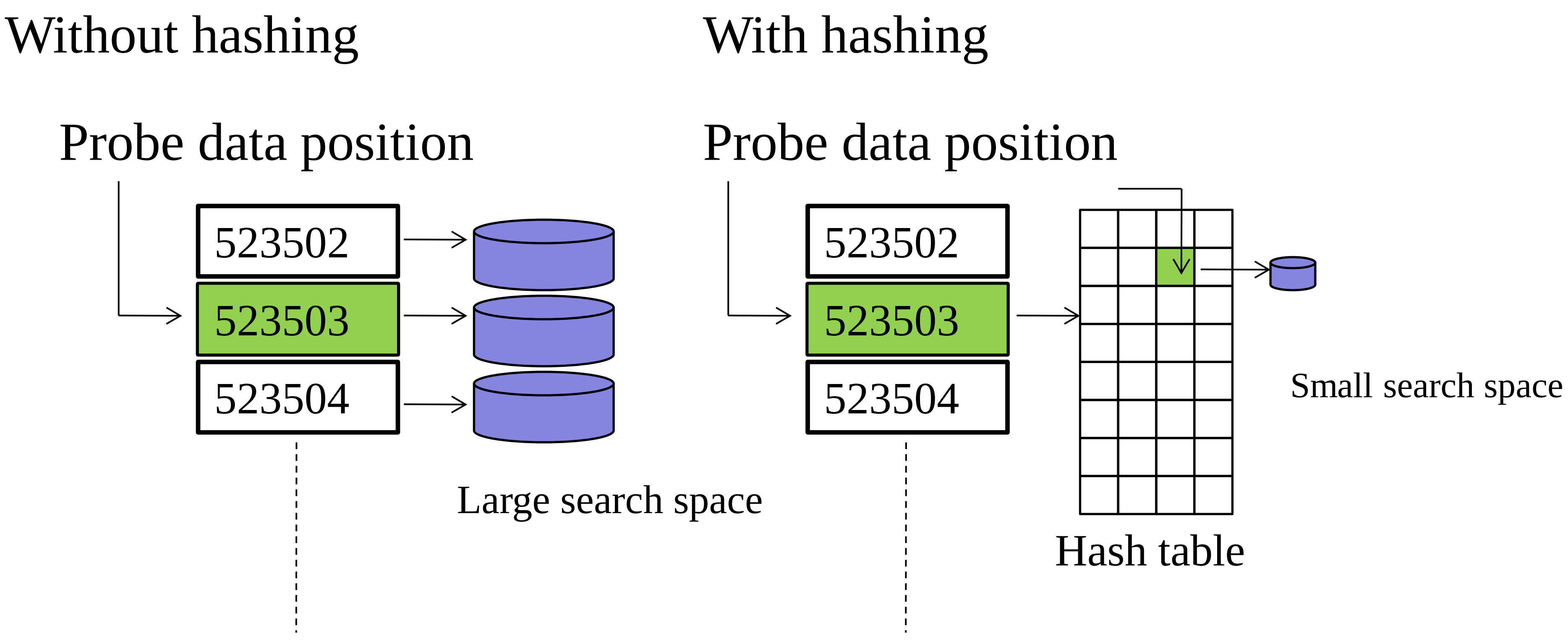
Current position of the probe

2ndmesh 523534
Sub-mesh 12

Hash table for searching neighbor links

00 NULL
01 link(node1,node2)
02 link(node1,node2)
03 link(node1,node2)
10 link(node1,node2),link(node1,node6)
11 link(node3,node5)
12 link(node2,node3),link(node3,node4),link(node3,node5)
13 link(node1,node2),link(node2,node7),link(node2,node5)

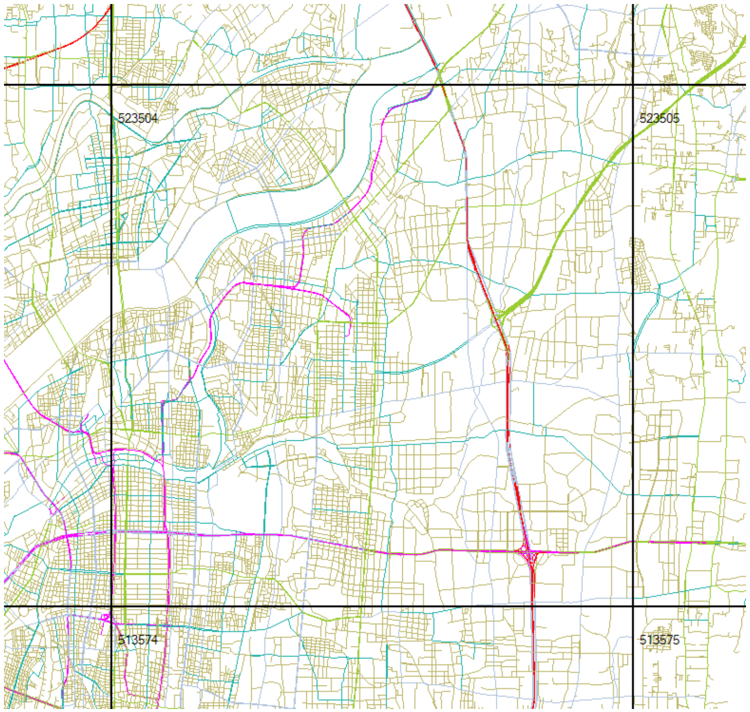
Hash Table Preparation

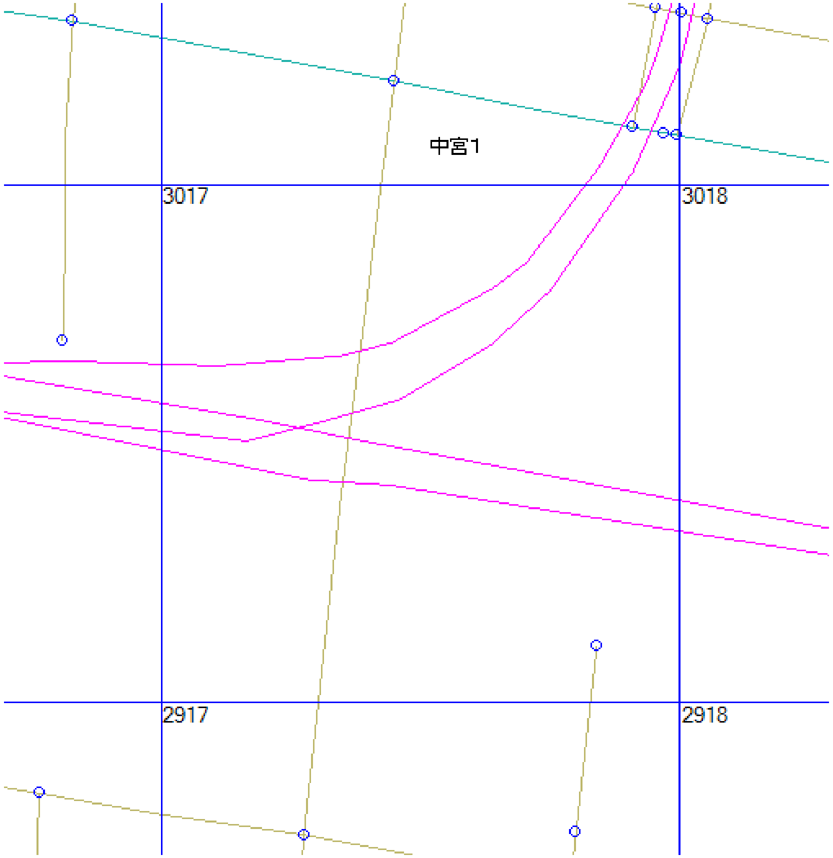


How to Reduce Computation Time

Number of links in a 2nd mesh:
12,924 links(maximum)
Exhaustive search is not efficient.

e.g.
50x50 divided sub mesh
57 links(maximum)
6 links(following example sub-mesh 30-17)

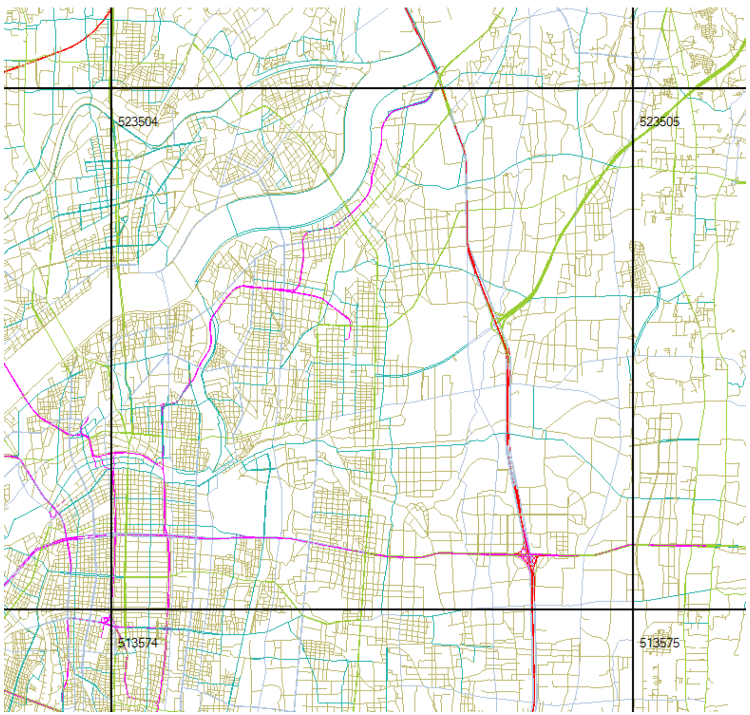


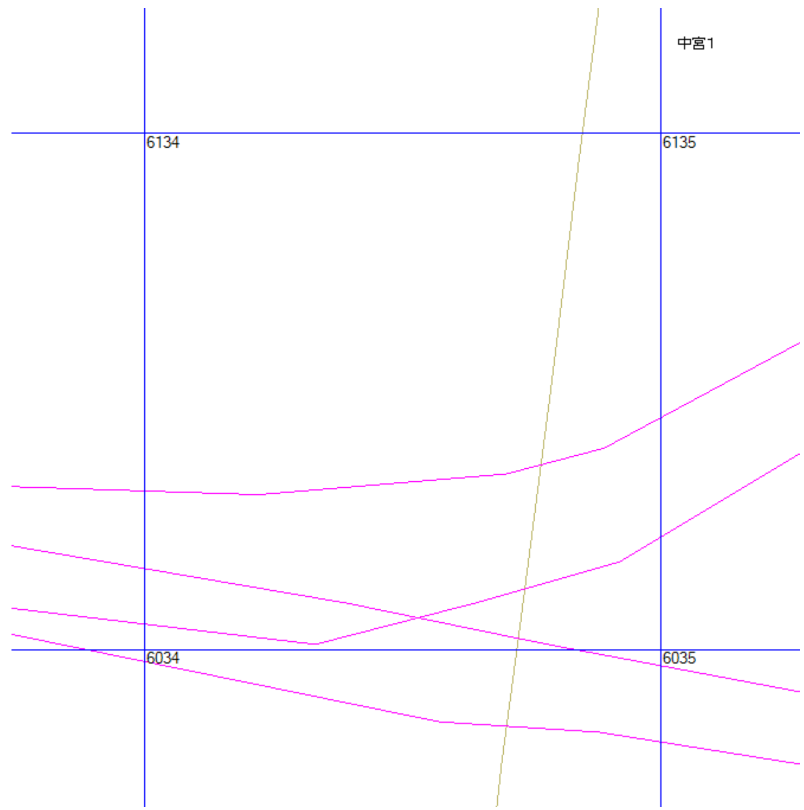


How to Reduce Computation Time

Number of links in a 2nd mesh:
12,924 links(maximum)
Exhaustive search is not efficient.

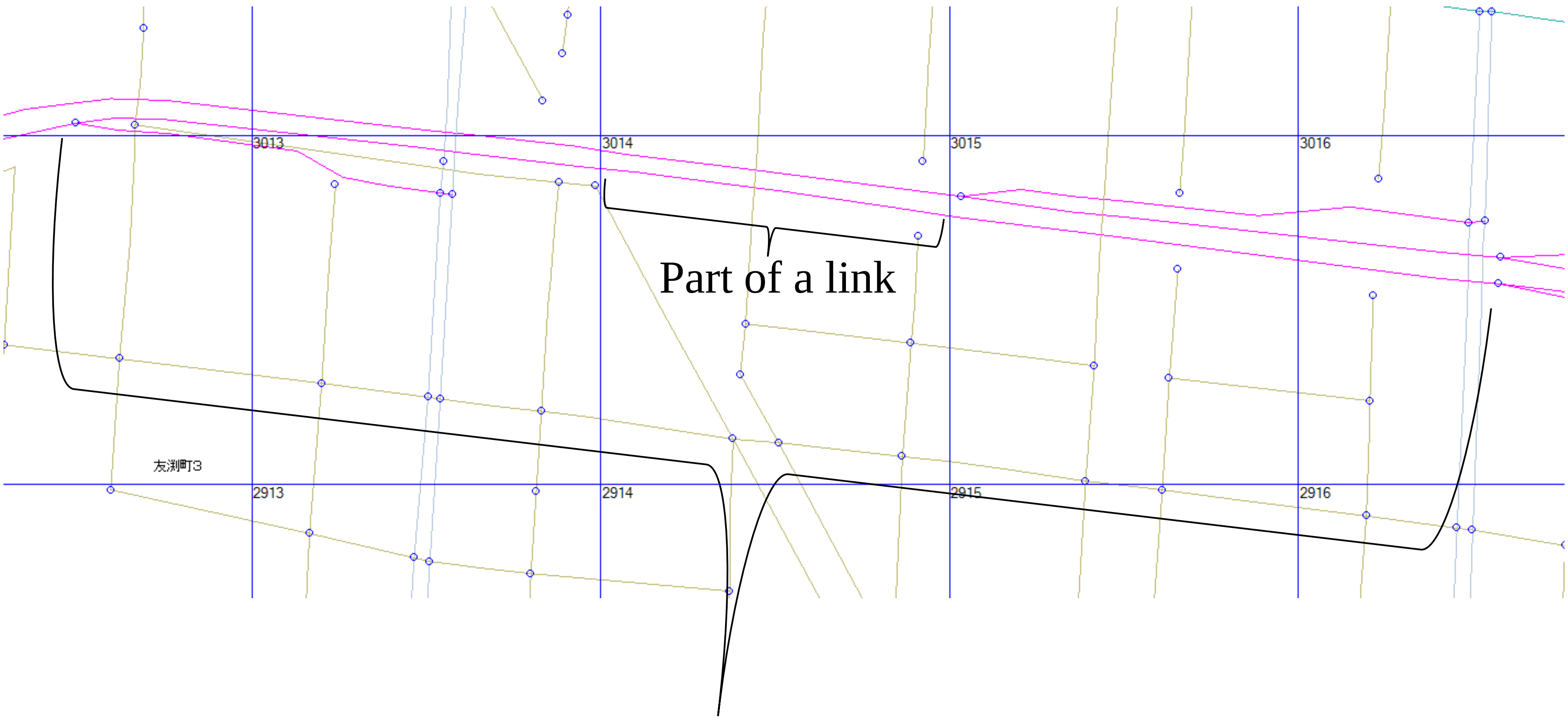
e.g.
100x100 divided sub mesh
33 links(maximum)
4 links (following example sub-mesh 61-34)





Only parts of links are in the sub meshes

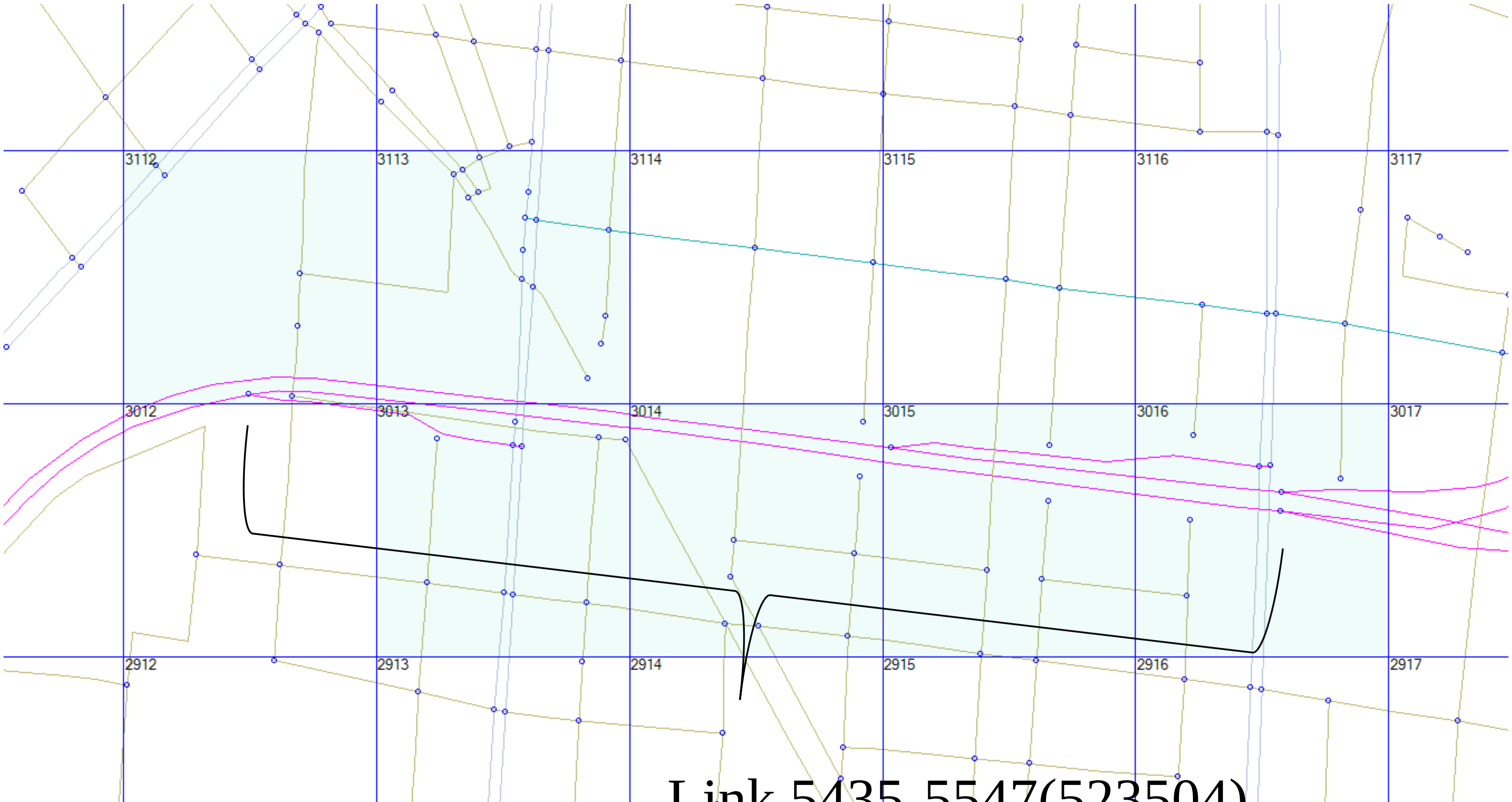
e.g.
50x50 divided sub mesh (approximately 200m by 200m)



A whole link

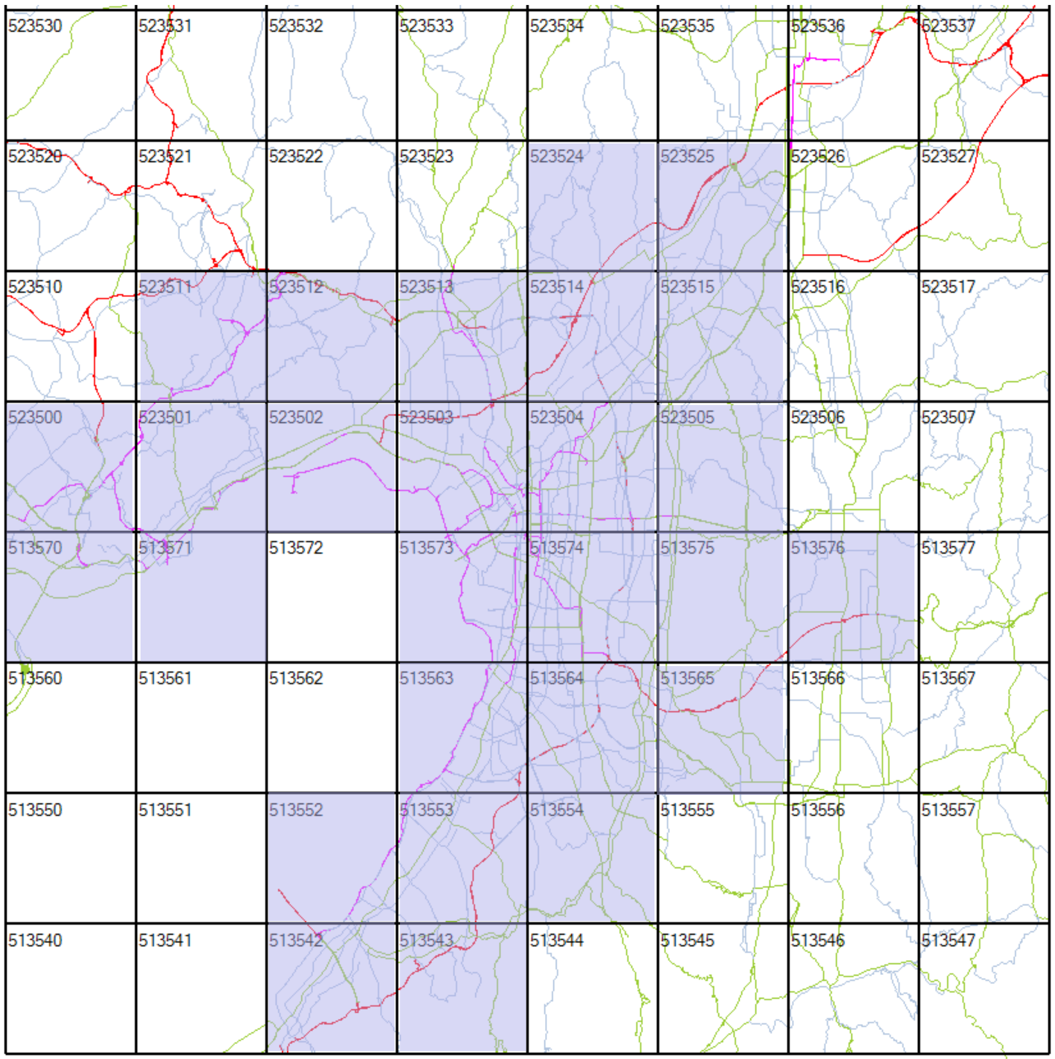
Hash Table Preparation

All links must be investigated to which sub meshes they belong.
In the example below, a link bridges over 6 sub meshes.



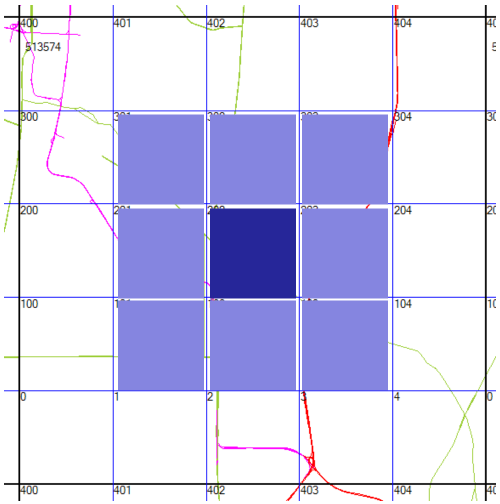
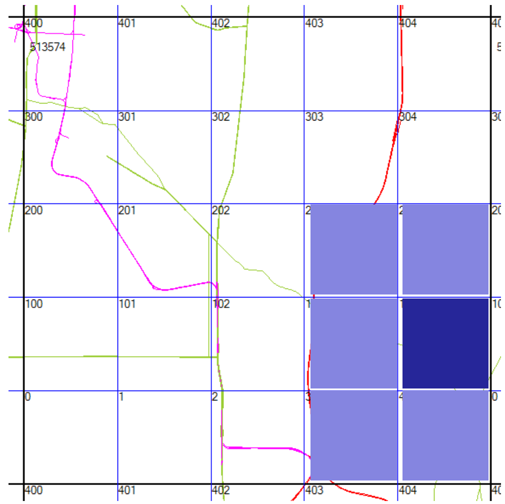
Link 5435-5547(523504)

Hash Table Preparation



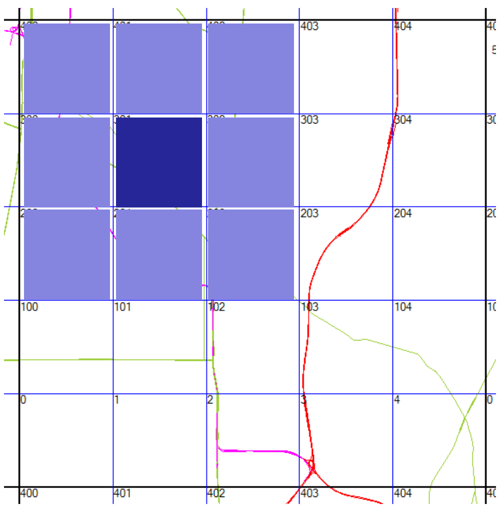
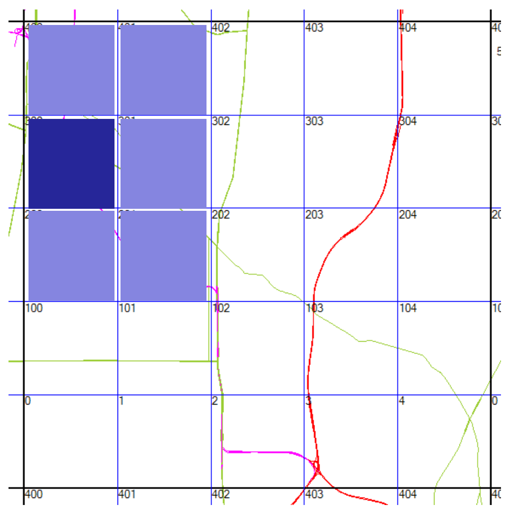
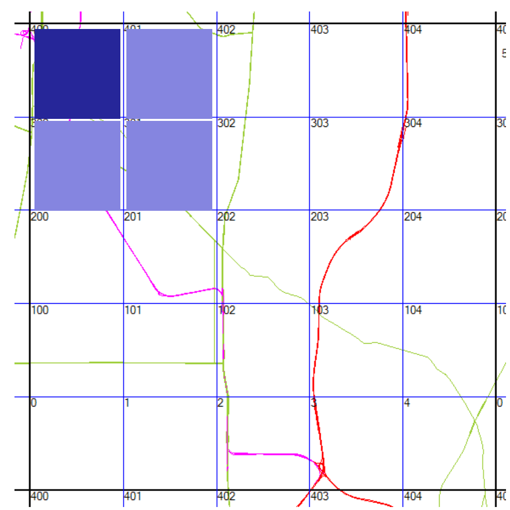
27 2nd meshes are processed.

Hash Table Preparation

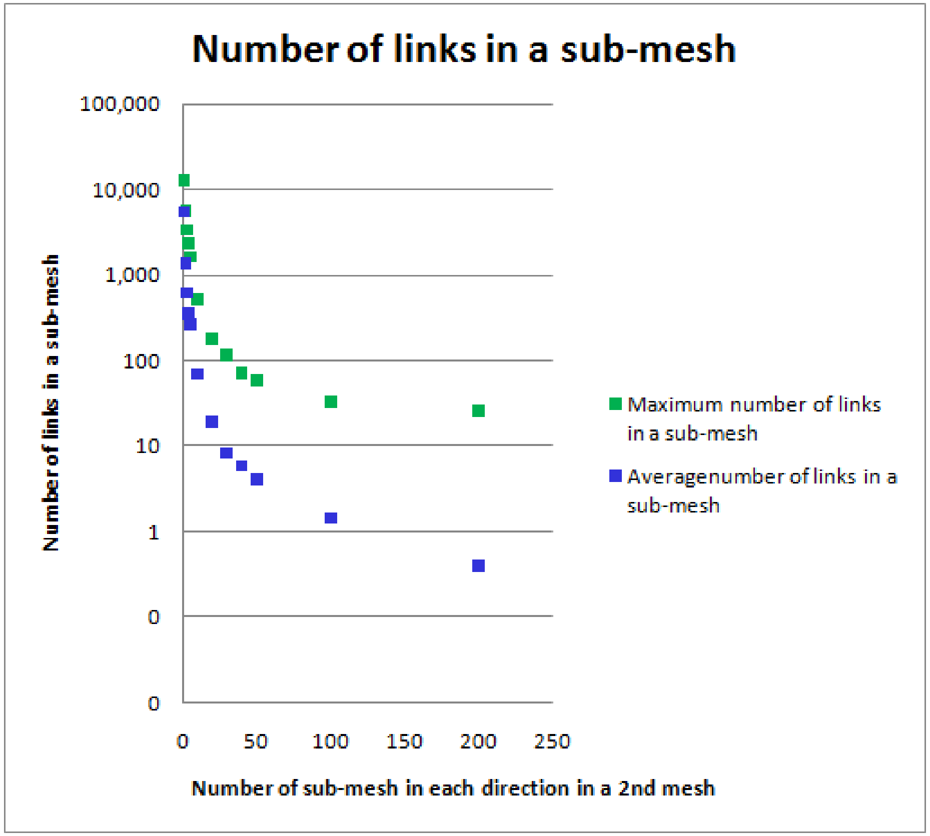
It searches for fringed sub-meshes to avoid missing border links like these.

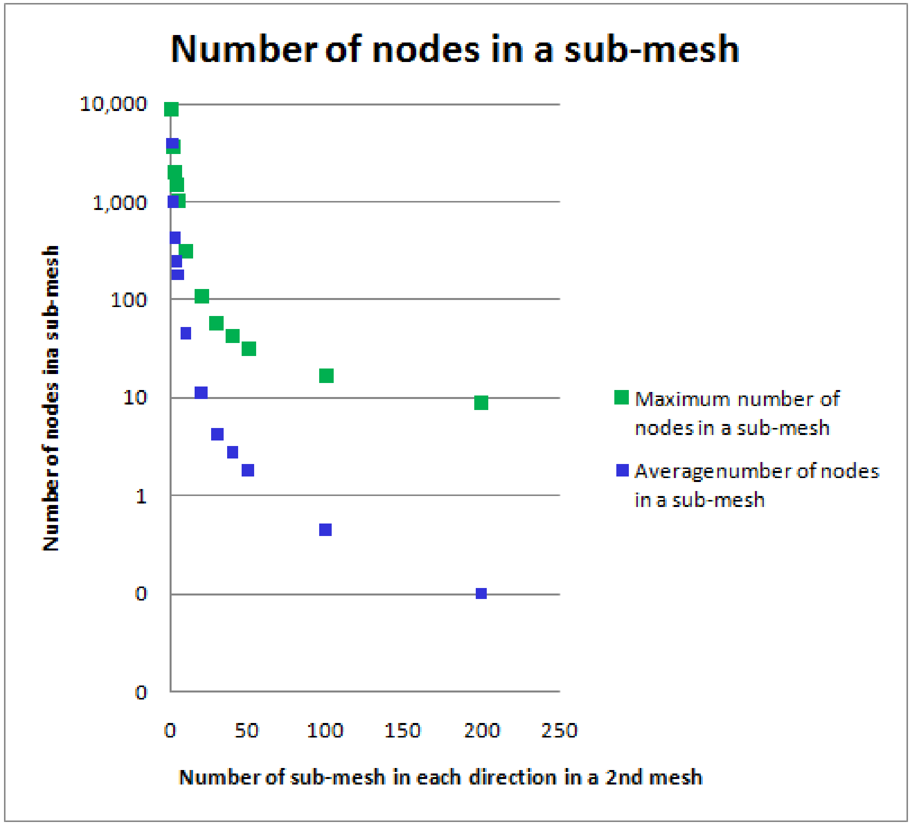
N ≧4 has the meaning

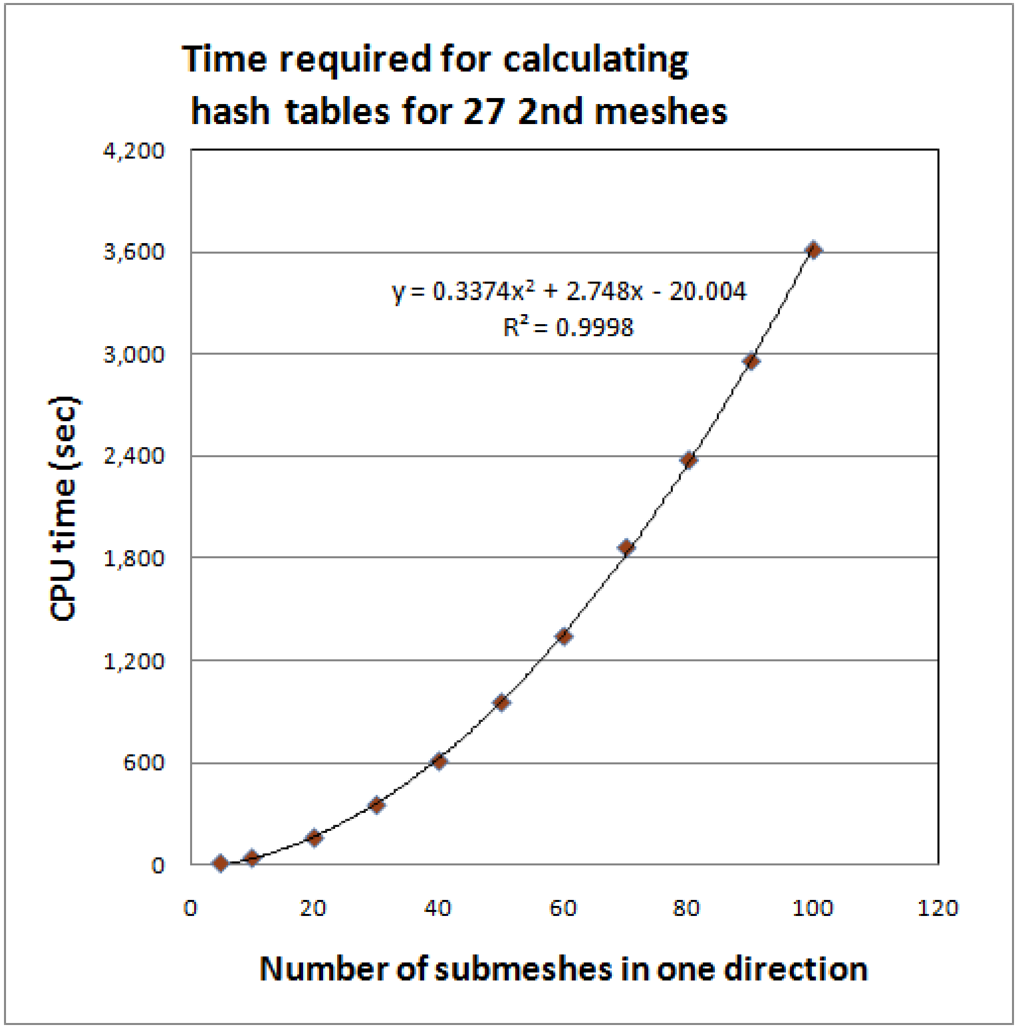
Hash Table Preparation

Number of links and nodes decrease in a square manner with respect to the number of sub-meshes in each axis.





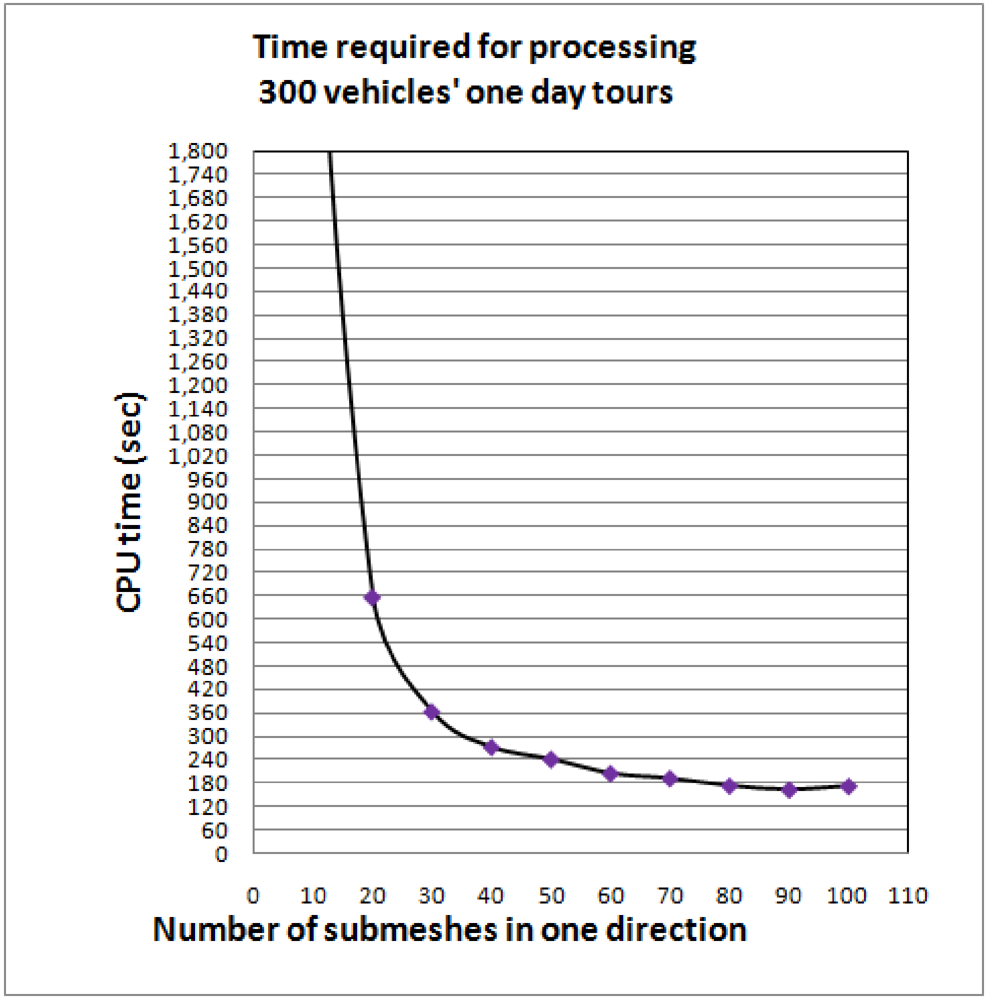
Hash Table Preparation



Calculating the hash tables is time consuming, but you need to do it once.

It is almost linear with respect to the total number sub-meshes. (square to sub-meshes in one axis)

Hash Table Preparation



Effectiveness of hashing is significant.

Too small sub-mesh increases the possibility of spoiling the candidate links for the matching.

N=100 (sub-mesh size about 100m by 100m) is the upper limit.

You can process 1 month 300 vehicle data within 90 minutes.

Without hashing(N=1), it takes about 3 months. It is painfully slow.

Conclusion

To reduce the computation time in map-matching, a hashing algorithm is developed.

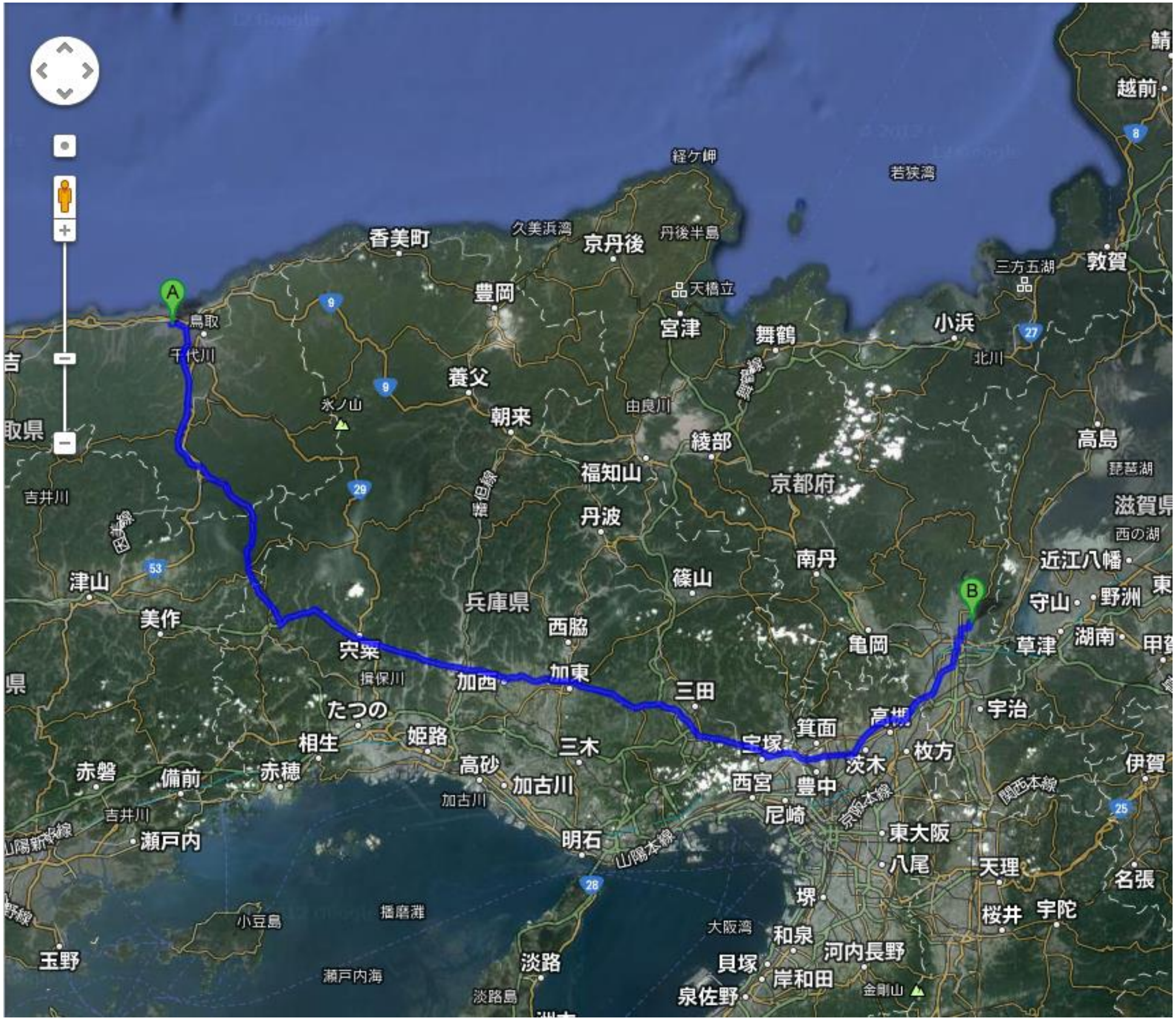
More than 30 million probe data which is obtained from 300 freight vehicles tours in one month survey can be handled within 90 minutes.

Without the hashing algorithm, it would have taken 3 months to process the one month probe data.
(Development of the hashing algorithm took 2 weeks)

ディジタル道路地図を用いた最も
代表的なアプリケーション

最適経路計算

例　Google Map



鳥取大学

↓

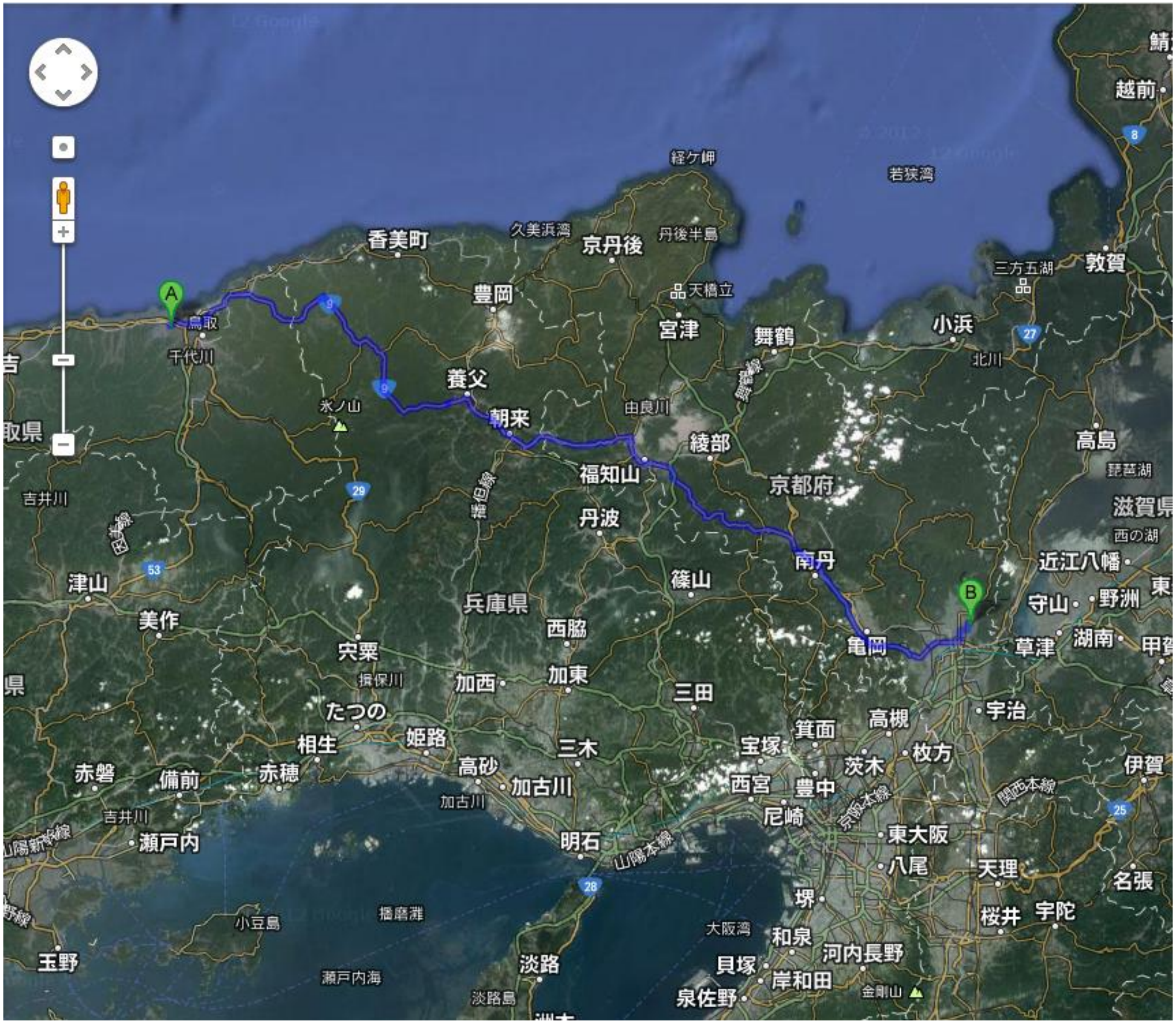
京都大学

中国自動車道経由

225km

3時間13分

例　Google Map



鳥取大学

京都大学

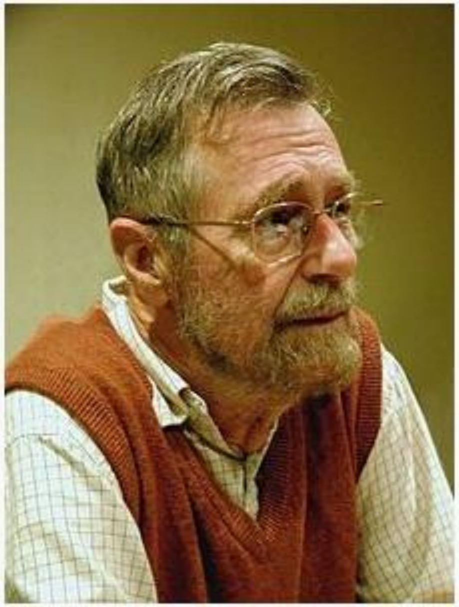
国道9号線
経由

225km

3時間56分

経路探索の基本アルゴリズム

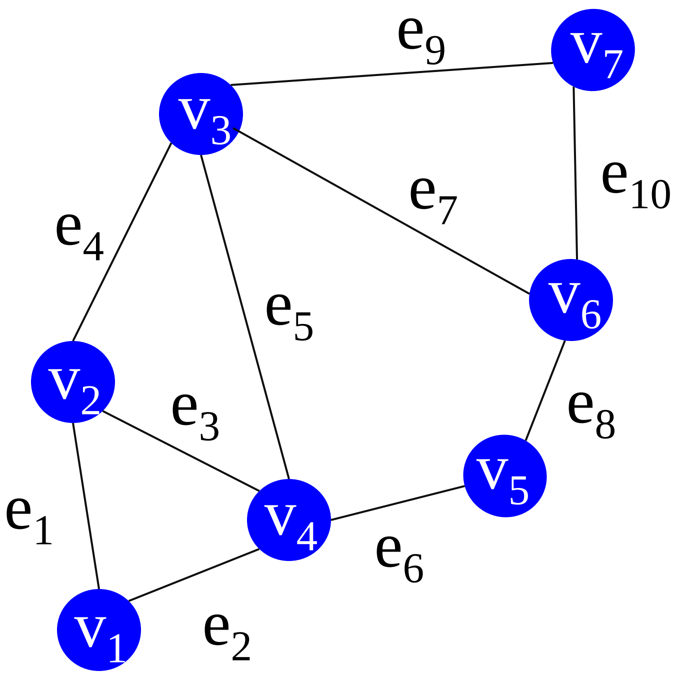
ダイクストラ法　Dijkstra Method



Edsger Wybe Dijkstra,
1930年5月11日 - 2002年8月6日
オランダ、ロッテルダム出身

グラフの基礎：無向グラフ

頂点（ノード）と枝（リンク）



グラフの表記法

$G=(V,E)$

ノード集合　V

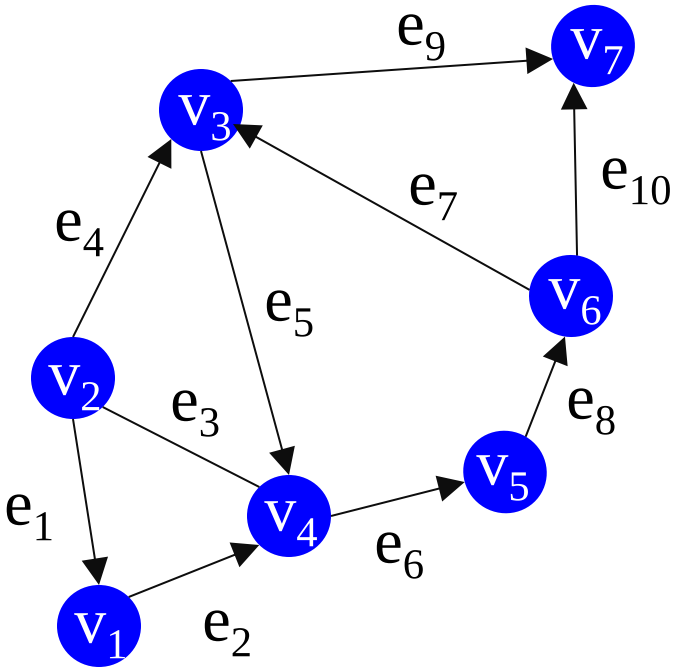
ノードの対を表す
リンクの集合　E

$e=(u,v)$

ノードu,vはリンクeの端点

グラフの基礎：有向グラフ

頂点（ノード）と枝（リンク）



グラフの表記法

$G=(V,E)$

ノード集合　V

ノードの対を表す
リンクの集合　E

$e=(u,v)$

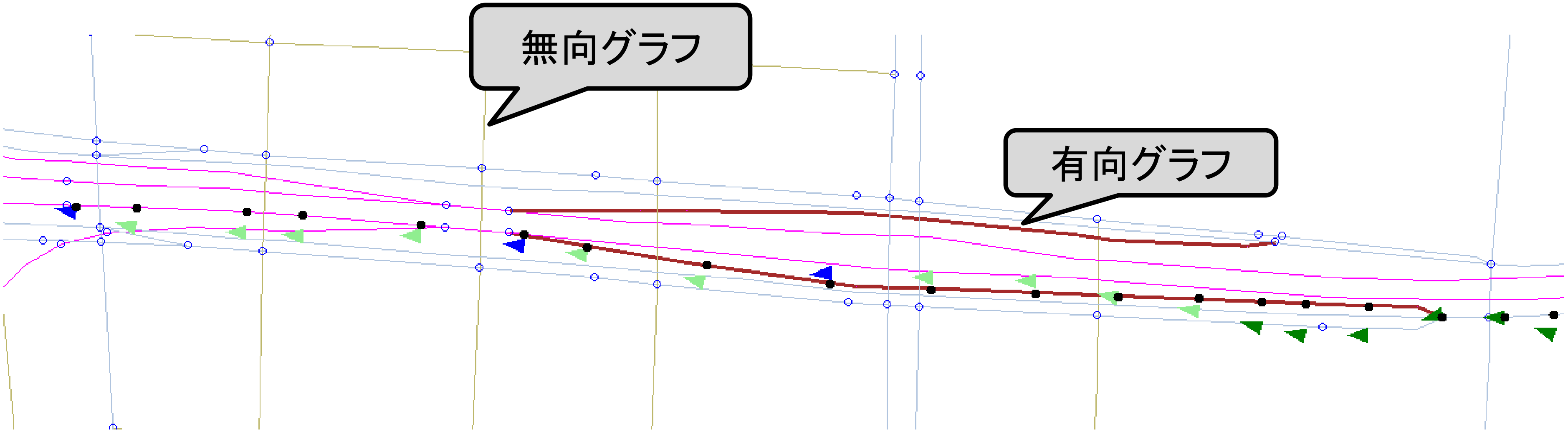
ノードuはリンクeの始点
ノードvはリンクeの終点

無向グラフと有向グラフ

ほとんどの一般道路.....................無向グラフで表現

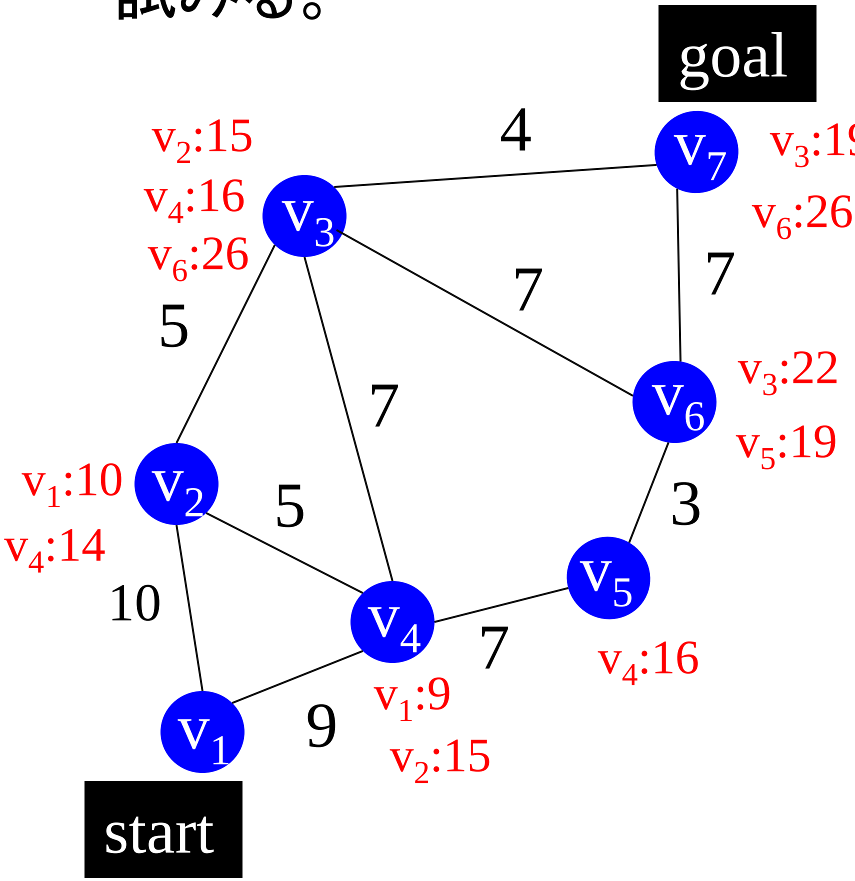
高速道路など上下線間の距離が大きい路線では
上下線を分けて表現　..................有向グラフで表現

リンクの属性情報で識別する。



Dijkstra法の概要

厳密な説明はかえってわかりにくいので直感的な説明を試みる。



始点　v_1,終点 v_7

1. まず、始点v1から隣接で未訪問のノードを調べる。
2. 調べたノードにはどのノードから訪れたかとスコアをメモする
3. 既に誰かが訪れていてスコアが負けていたらあきらめそこまでの経路は不採用とする。

注目ノードV1:　V2,V4を訪れる

注目ノードV2:　V3,V4を訪れる

v1→v2→v3　15

注目ノードV4:　V2,V3,V5を訪れる

v1→v2→v4　15

v1→v4→v2　14>10 NG

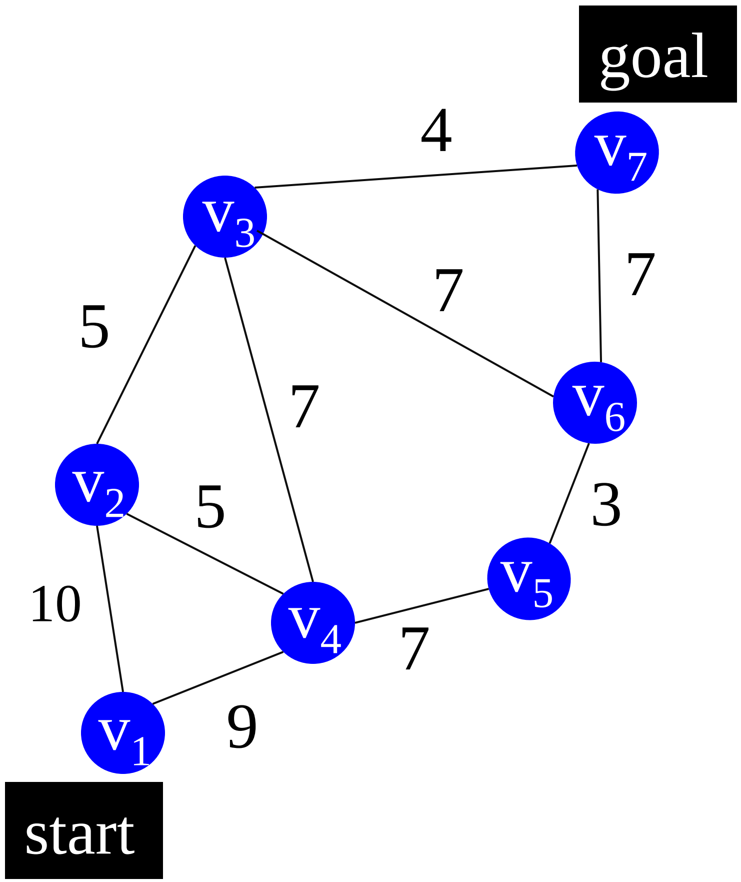
v1→v4→v3　16>15 NG

v1→v4→v5　22>16 NG

注目ノードV3:　V4,V6,V7を訪れる

v7←v3←v2←v1

Dijkstra法の簡単な実装



変数

start： 始点ノード番号

visited[N]:
各ノードへの最小コストが
確定しているかどうかのフラグ
=0 YES
=1 NO

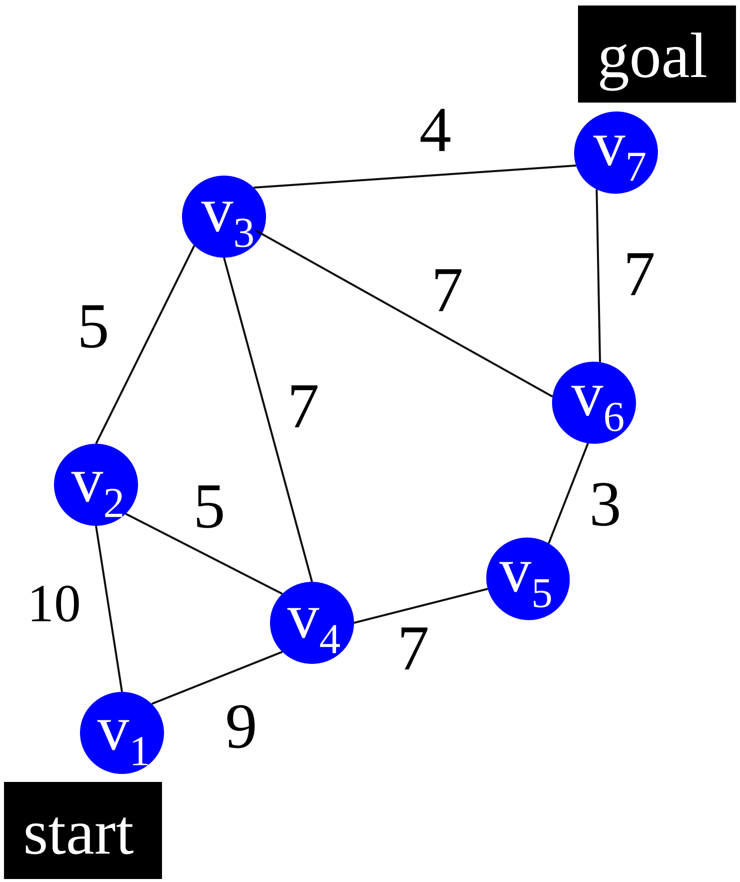
cost[N]:
各ノードへの最小累積コストを格納

previous[N]:
各ノードへの最小コストを与える
手前のノード番号

link_cost[N][N]: リンクのコスト

N: ノード数

Dijkstra法の簡単な実装　　配列 **link_cost[N][N]**



下流ノード

上流ノード

	1	2	3	4	5	6	7
1		10		9			
2	10		5	5			
3		5		7		7	4
4	9	5	7		7		
5				7		3	
6			7		3		7
7			4			7	

数値はリンクコストを表す。

は隣接しない（無限）

必ずしも効率的な表現方法ではない。

Dijkstra法の簡単な実装

初期化

全ノードkについて、
cost[k]=∞,
fixed[k]=NO
previous[k]=0

cost[start]=0

root_node=start

fixed[root_node]=YES　とする。

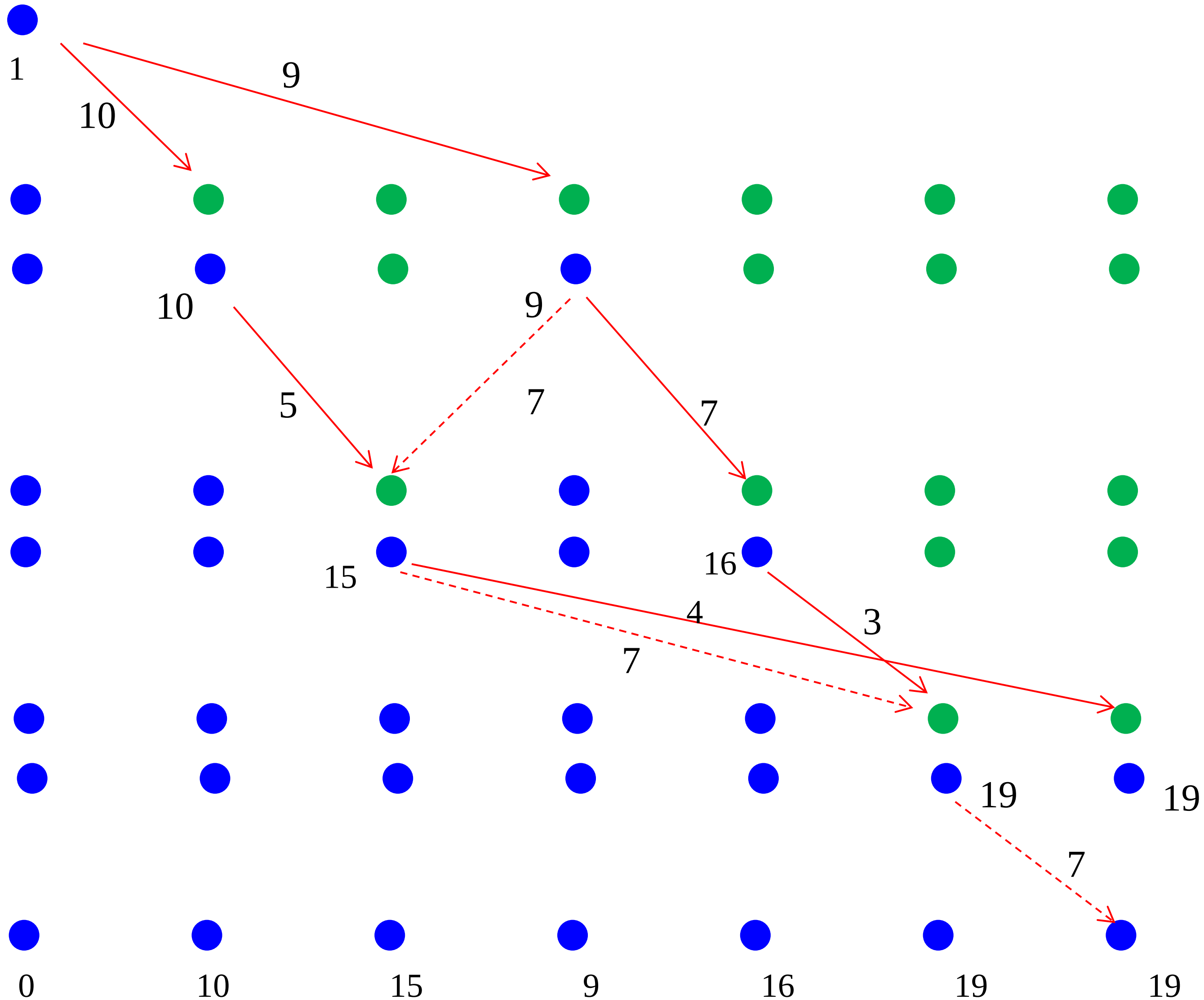
fixed[k]==NO　のノードの中で、
root_nodeの全ての隣接ノードkについて
リンクコストを計算し、各隣接ノードの累積コスト　cost[k]に
格納する。その際の条件は

root_nodeの累積コスト＋リンクコスト＜隣接ノードkの現状の累積コスト
すなわち、
cost[root_node]+link_cost[root_node][k] < cost[k]

また、隣接ノードの上流ノード番号
previous[k]にroot_nodeを格納する。

fixed[k]==NOのノードの中で累積コストが
小さいものを選びそれを新たな
root_nodeとする。

Dijkstra法の簡単な実装



Dijkstra法の簡単な実装

LOOP1

	1	2	3	4	5	6	7
fixed	YES	NO	NO	NO	NO	NO	NO
cost	0	∞	∞	∞	∞	∞	∞
previous	0	0	0	0	0	0	0

LOOP2

	1	2	3	4	5	6	7
fixed	YES	YES	NO	YES	NO	NO	NO
cost	0	10	∞	9	∞	∞	∞
previous	0	1	0	1	0	0	0

LOOP3

	1	2	3	4	5	6	7
fixed	YES	YES	YES	YES	YES	NO	NO
cost	0	10	10+5	9	9+7	∞	∞
previous	0	1	2	1	4	0	0

LOOP4

	1	2	3	4	5	6	7
fixed	YES	YES	YES	YES	YES	YES	YES
cost	0	10	15	9	16	16+3	15+4
previous	0	1	2	1	4	5	3

演習課題

実際のリンクデータを用いて最短経路を算出するプログラムを作りなさい。

リンク番号　始点ノード　終点ノード　始点ノード座標　リンク長さ(m)

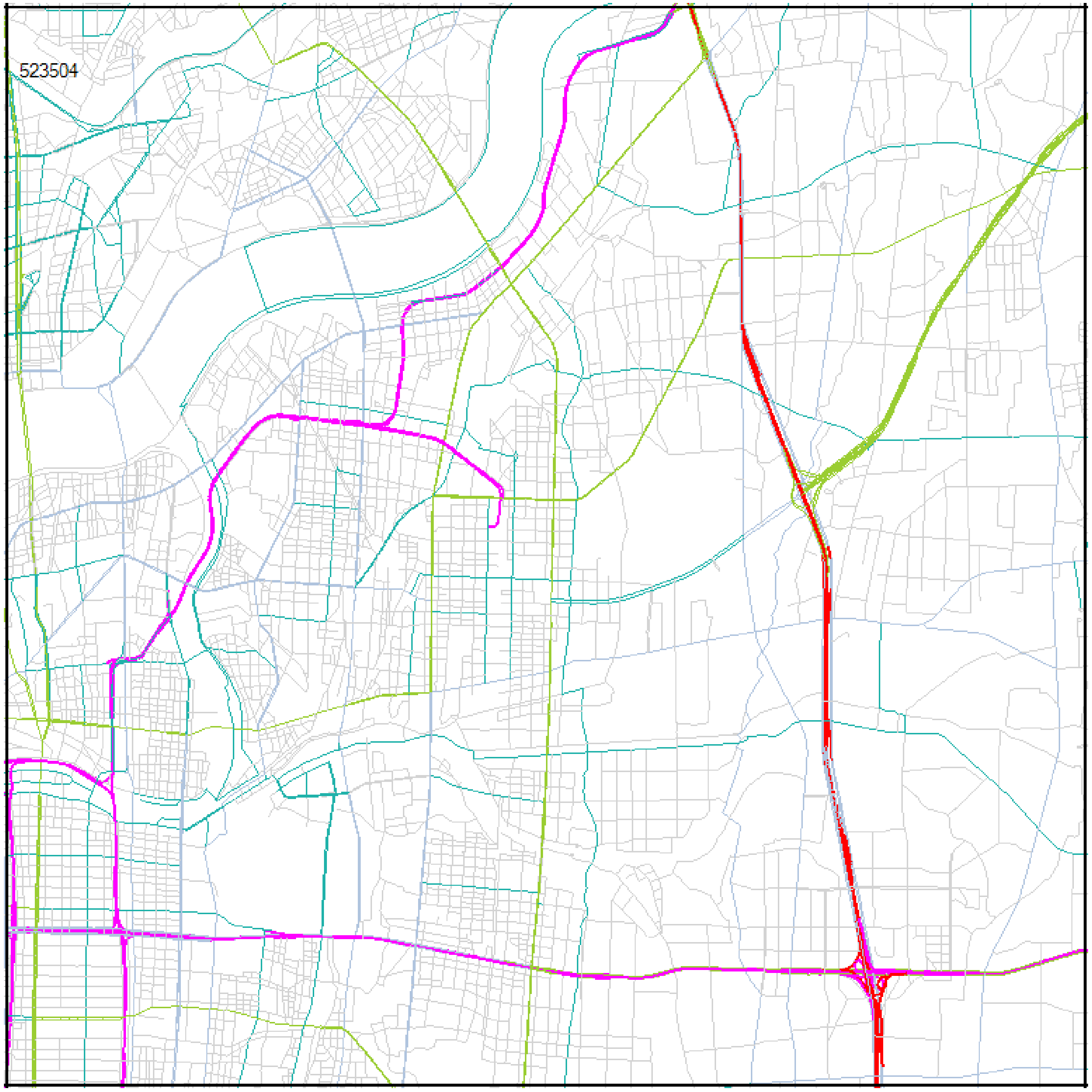
```
link, 0,node1,1,node2,1006,x,27,y,0,type,9,link_type 0,link_length, 46
link, 1,node1,2,node2,7683,x,177,y,0,type,9,link_type 0,link_length, 48
link, 2,node1,3,node2,7685,x,253,y,0,type,3,link_type 0,link_length, 48
link, 3,node1,4,node2,7687,x,718,y,0,type,6,link_type 0,link_length, 37
link, 4,node1,5,node2,7688,x,1043,y,0,type,9,link_type 0,link_length, 26
link, 5,node1,6,node2,1082,x,1069,y,0,type,2,link_type 0,link_length, 296
link, 6,node1,7,node2,1082,x,1083,y,0,type,2,link_type 5,link_length, 296
link, 7,node1,8,node2,1085,x,1110,y,0,type,9,link_type 0,link_length, 215
```

サンプルデータ

kihonlink_523504.csv

12,545リンク

演習課題



サンプルデータ
kihonlink_523504.csv

PROGRAM例

```
#include "stdafx.h"
/*dijkstra.c最短経路問題			shortest path problem */
#include <stdio.h>
#include <stdlib.h>

#define			N			7
#define			START			0
#define			FALSE			0
#define			TRUE			1
//#define			INT_MAX			999

void readweight(int **weight);
void readweight(int **weight)
{
		int i, j;

		for(i = 0; i < N; i++)	for(j = 0; j < N; j++)	weight[i][j] = INT_MAX;
		weight[0][1] = 10;
		weight[0][3] = 9;
		:
		weight[5][6] = 7;
		weight[6][2] = 4;
		weight[6][5] = 7;
}

int _tmain(int argc, _TCHAR* argv[])
{
		int i, j, next, min;
		int *distance, *prev, *visited;
		int **weight, **p;
		distance = (int *)calloc(N, sizeof(int));
		prev = (int *)calloc(N, sizeof(int));
		visited = (int *)calloc(N, sizeof(int));
		weight = (int **)malloc(N * sizeof(int *));
		*weight = (int *)malloc(N * N * sizeof(int));
		for(p = weight, i = 1; i < N; i++, p++)		*(p + 1) = *p + N;
		readweight(weight);	/* 距離 weight[0..n-1][0..n-1] を読む */
		for(i = 0; i < N; i++)
		{
				for(j = 0; j < N; j++)
				{if(weight[i][j]>100) printf("*");
				else			printf("%d", weight[i][j]);
				// printf("¥n");
				};printf("¥n");
		}
for(i = 0; i < N; i++)
		{
		visited[i] = FALSE;
		distance[i] = INT_MAX;
		}
		distance[START] = 0;
		next = START;
		do
		{
		i = next;
		visited[i] = TRUE;
		min = INT_MAX;
		for(j = 0; j < N; j++)
		{
		if(visited[j])		continue;
		if(weight[i][j] < INT_MAX
		&& distance[i] + weight[i][j] < distance[j])
				{
				distance[j] = distance[i] + weight[i][j];
				prev[j] = i;
						}
				if(distance[j] < min)
				{
				min = distance[j];
				next = j;
						}
				}
		}while(min < INT_MAX);
		printf("点  直前の点  最短距離¥n");
		for(i = 0; i < N; i++)
		if(i != START && visited[i])
		{printf("%2d%10d%10d¥n", i, prev[i], distance[i]);
		   getchar();
		};
		return 1;
}
```

経路探索の課題

ドライバー向けを想定した場合

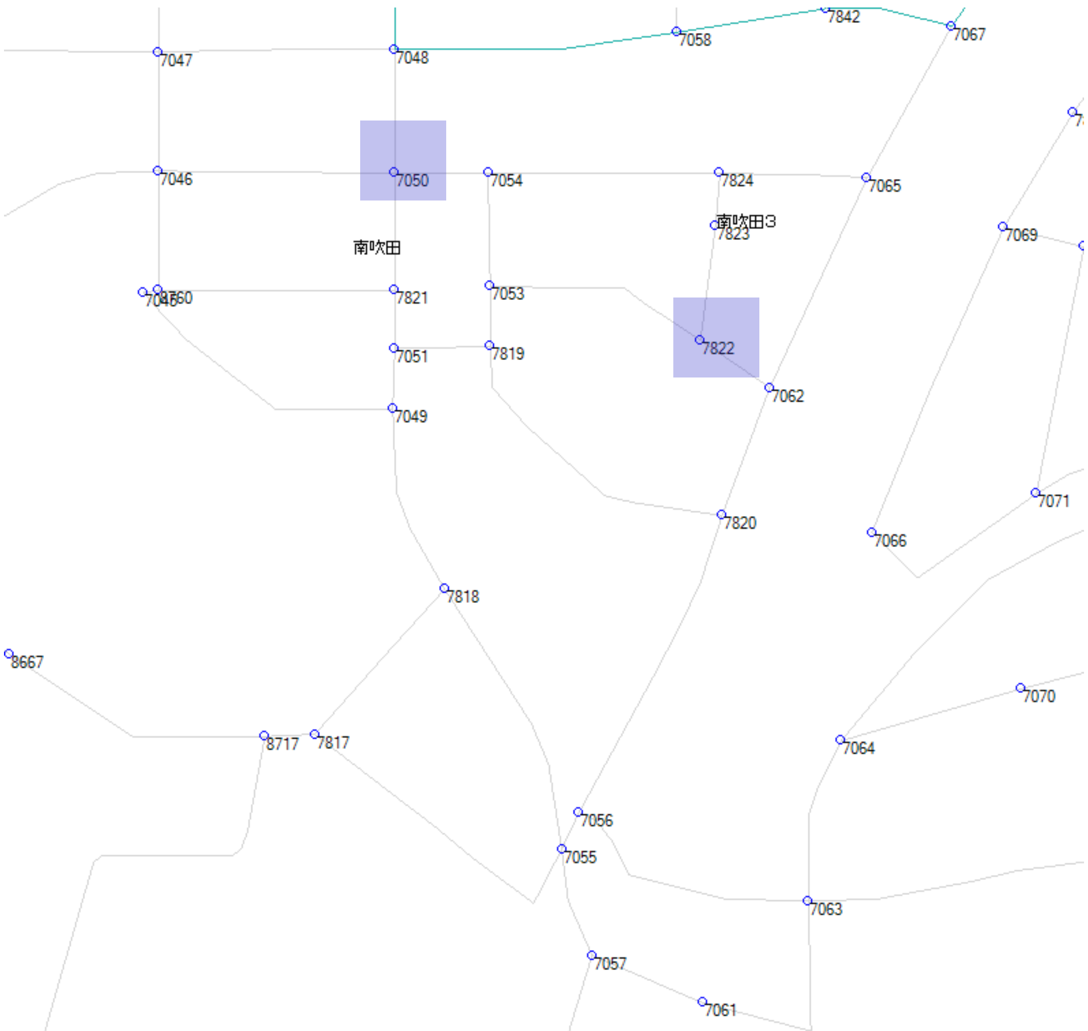
リンクコストとして何が妥当かは長年の議論の的

距離、旅行時間、右折時間を考慮するしない

道路種別、有料道路の料金、景観、走りやすさ等々

演習課題

リンクデータが多すぎて処理速度が大きくなる場合、
リンクデータを削って実行してかまいません。



ノード番号7000番台の
リンクのみにして

node 7050
⬇
node 7822

までの最短経路を
求めなさい

Dijkstra法の改善

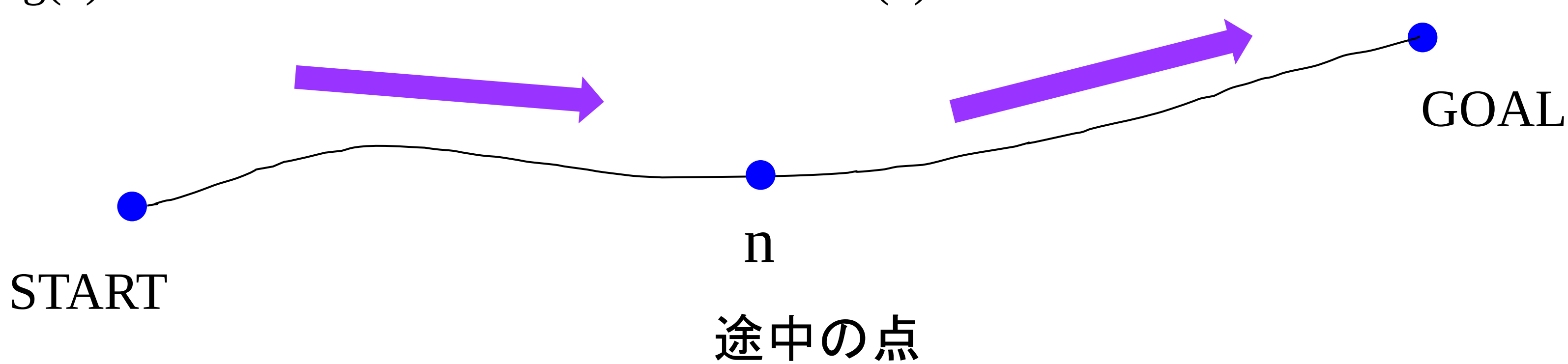
Dijkstra法をベースに高速化を図るアイデアがいくつか生まれている。そのひとつがA*法（エイスター法）

http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf　で原論文入手可能

STARTからGOALまでの最小コスト: $f(n)=g(n)+h(n)$

STARTから地点nまでのコスト:
$g(n)$で最小コスト

地点n からGOALまでのコスト:
$h(n)$で最小コスト



最短ルート

1968年に掲載された論文

IEEE Trans.on
Systems Science & Cybernetics
pp.100-107,
Vol.SSC-4,No.2,July,1968

[5] J. E. Falk, "Lagrange multipliers and nonlinear programming," *J. Math. Anal. Appl.*, vol. 19, July 1967.

[6] O. L. Mangasarian and J. Ponstein, "Minimax and duality in nonlinear programming," *J. Math. Anal. Appl.*, vol. 11, pp. 504–518, 1965.

[7] J. Stoer, "Duality in nonlinear programming and the minimax theorem," *Numerische Mathematik*, vol. 5, pp. 371–379, 1963.

[8] R. T. Rockafellar, "Duality and stability in extremum problems involving convex functions," *Pacific J. Math.*, vol. 21, pp. 167–187, 1967.

[9] P. Wolfe, "A duality theorem for nonlinear programming," *Q. Appl. Math.*, vol. 19, pp. 239–244, 1961.

[10] R. T. Rockafellar, "Nonlinear programming," presented at the American Mathematical Society Summer Seminar on the Mathematics of the Decision Sciences, Stanford University, Stanford, Calif., July–August 1967.

[11] D. G. Luenberger, "Convex programming and duality in normal space," *Proc. IEEE Systems Science and Cybernetics Conf.* (Boston, Mass., October 11–13, 1967).

[12] J. M. Danskin, "The theory of max-min with applications," *J. SIAM*, vol. 14, pp. 641–665, July 1966.

[13] W. Fenchel, "Convex cones, sets, and functions," mimeographed notes, Princeton University, Princeton, N. J., September 1963.

[14] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Computer J.*, vol. 6, p. 163, July 1963.

[15] L. S. Lasdon and A. D. Waren, "Mathematical programming for optimal design," *Electro-Technol.*, pp. 53–71, November 1967.

[16] J. B. Rosen, "The gradient projection method for nonlinear programming, pt. I, linear constraints," *J. SIAM*, vol. 8, pp. 181–217, 1960.

[17] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *Computer J.*, vol. 7, July 1964.

[18] D. Goldfarb, "A conjugate gradient method for nonlinear programming," Ph.D. dissertation, Dept. of Chem. Engrg., Princeton University, Princeton, N. J., 1966.

[19] L. S. Lasdon, "A multi-level technique for optimization," Ph.D. dissertation, Systems Research Center, Case Institute of Technology, Cleveland, Ohio, Rept. SRC 50-C-64-19, 1964.

[20] L. S. Lasdon and J. D. Schoeffler. "A multi-level technique for optimization," Preprints, Joint Automatic Control Conf., Troy, N. Y., June 22–25, 1965, pp. 85–92.

[21] ——, "Decentralized plant control," *ISA Trans.*, vol. 5, pp. 175–183, April 1966.

[22] C. B. Brosilow and L. S. Lasdon, "A two level optimization technique for recycle processes," *1965 Proc. AICHE—Symp. on Application of Mathematical Models in Chemical Engineering Research, Design, and Production* (London, England).

[23] L. S. Lasdon, "Duality and decomposition in mathematical programming," Systems Research Center, Case Institute of Technology, Cleveland, Ohio, Rept. SRC 119-C-67-52, 1967.

[24] A. V. Fiacco and G. P. McCormick, *Sequential Unconstrained Minimization Techniques for Nonlinear Programming*. New York: Wiley, 1968.

[25] R. Fox and L. Schmit, "Advances in the integrated approach to structural synthesis," *J. Spacecraft and Rockets*, vol. 3, p. 858, June 1966.

[26] B. P. Dzielinski and R. E. Gomory, "Optimal programming of lot sizes, inventory, and labor allocations," *Management Sci.*, vol. 11, pp. 874–890, July 1965.

[27] J. E. Falk, "A relaxed interior approach to nonlinear programming," Research Analysis Corp., McLean, Va. RAC-TP-279, 1967.

A Formal Basis for the Heuristic Determination of Minimum Cost Paths

PETER E. HART, MEMBER, IEEE, NILS J. NILSSON, MEMBER, IEEE, AND BERTRAM RAPHAEL

***Abstract*—Although the problem of determining the minimum cost path through a graph arises naturally in a number of interesting applications, there has been no underlying theory to guide the development of efficient search procedures. Moreover, there is no adequate conceptual framework within which the various ad hoc search strategies proposed to date can be compared. This paper describes how heuristic information from the problem domain can be incorporated into a formal mathematical theory of graph searching and demonstrates an optimality property of a class of search strategies.**

I. Introduction

A. The Problem of Finding Paths Through Graphs

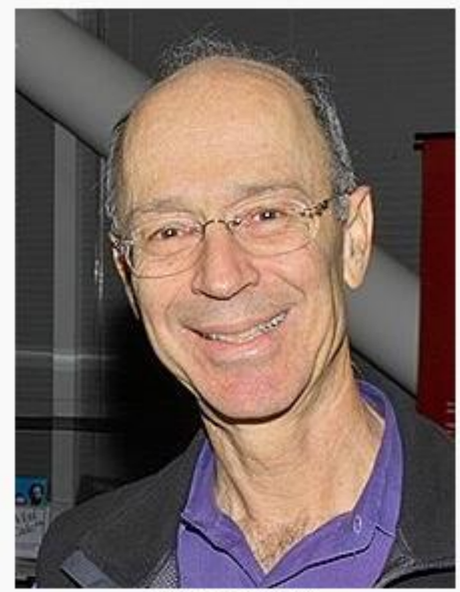
MANY PROBLEMS of engineering and scientific importance can be related to the general problem of finding a path through a graph. Examples of such problems include routing of telephone traffic, navigation through a maze, layout of printed circuit boards, and mechanical theorem-proving and problem-solving. These problems have usually been approached in one of two ways, which we shall call the *mathematical approach* and the *heuristic approach*.

1) The mathematical approach typically deals with the properties of abstract graphs and with algorithms that prescribe an orderly examination of nodes of a graph to establish a minimum cost path. For example, Pollock and Wiebenson[1] review several algorithms which are guaranteed to find such a path for any graph. Busacker and Saaty[2] also discuss several algorithms, one of which uses the concept of dynamic programming.[3] The mathematical approach is generally more concerned with the ultimate achievement of solutions than it is with the computational feasibility of the algorithms developed.

2) The heuristic approach typically uses special knowledge about the domain of the problem being represented by a graph to improve the computational efficiency of solutions to particular graph-searching problems. For example, Gelernter's[4] program used Euclidean diagrams to direct the search for geometric proofs. Samuel[5] and others have used ad hoc characteristics of particular games to reduce

Manuscript received November 24, 1967.
The authors are with the Artificial Intelligence Group of the Applied Physics Laboratory, Stanford Research Institute, Menlo Park, Calif.

Peter E. Hart



Peter Hart in 2005

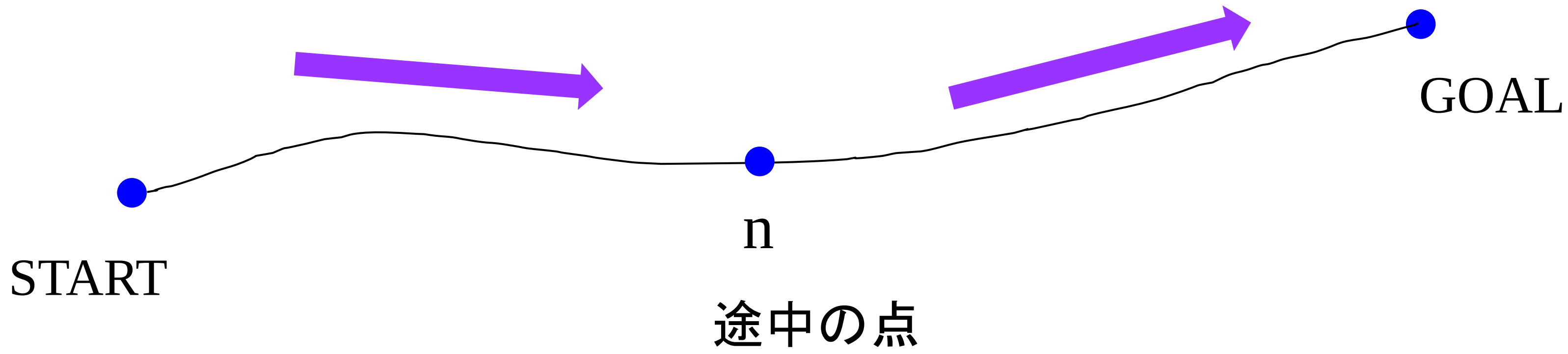
1940年代生まれ
現在　(株)リコー　副社長？？

Dijkstra法の改善: A*法

START からGOALまでの最小コストの推定値: f(n)=g(n)+h*(n)

STARTから地点nまでのコスト:
g(n)で最小コストで既知

地点n からGOALまでの最小コストの
推定値: h*(n)



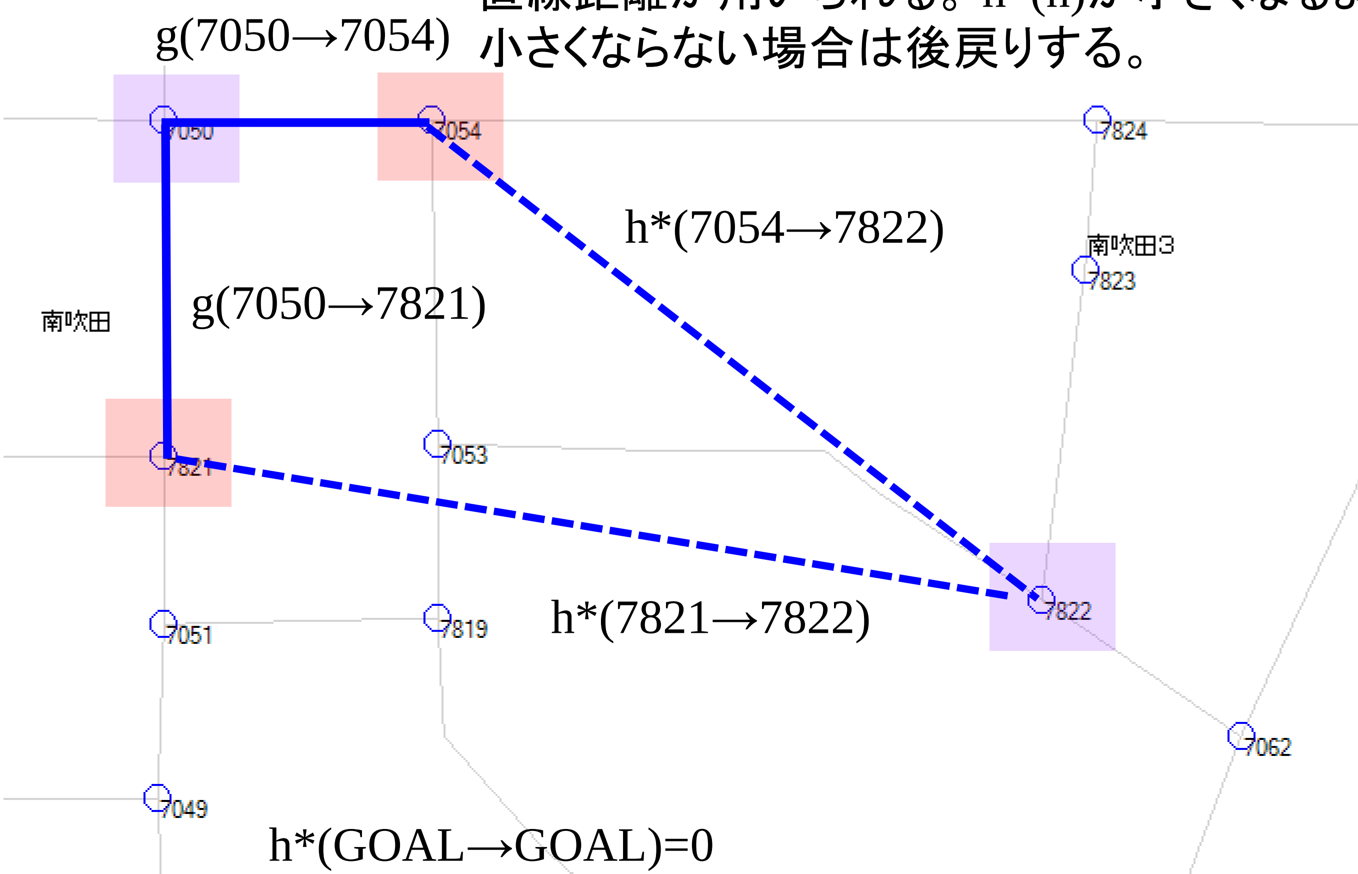
h*(n)の条件
「0以上」かつ「地点nからGOALまでの実際の最短経路コスト以下」の範囲の値

すなわち、 $0 \leq h^*(n) \leq h(n)$

であれば、最小コストの解に到達することが保証されている。

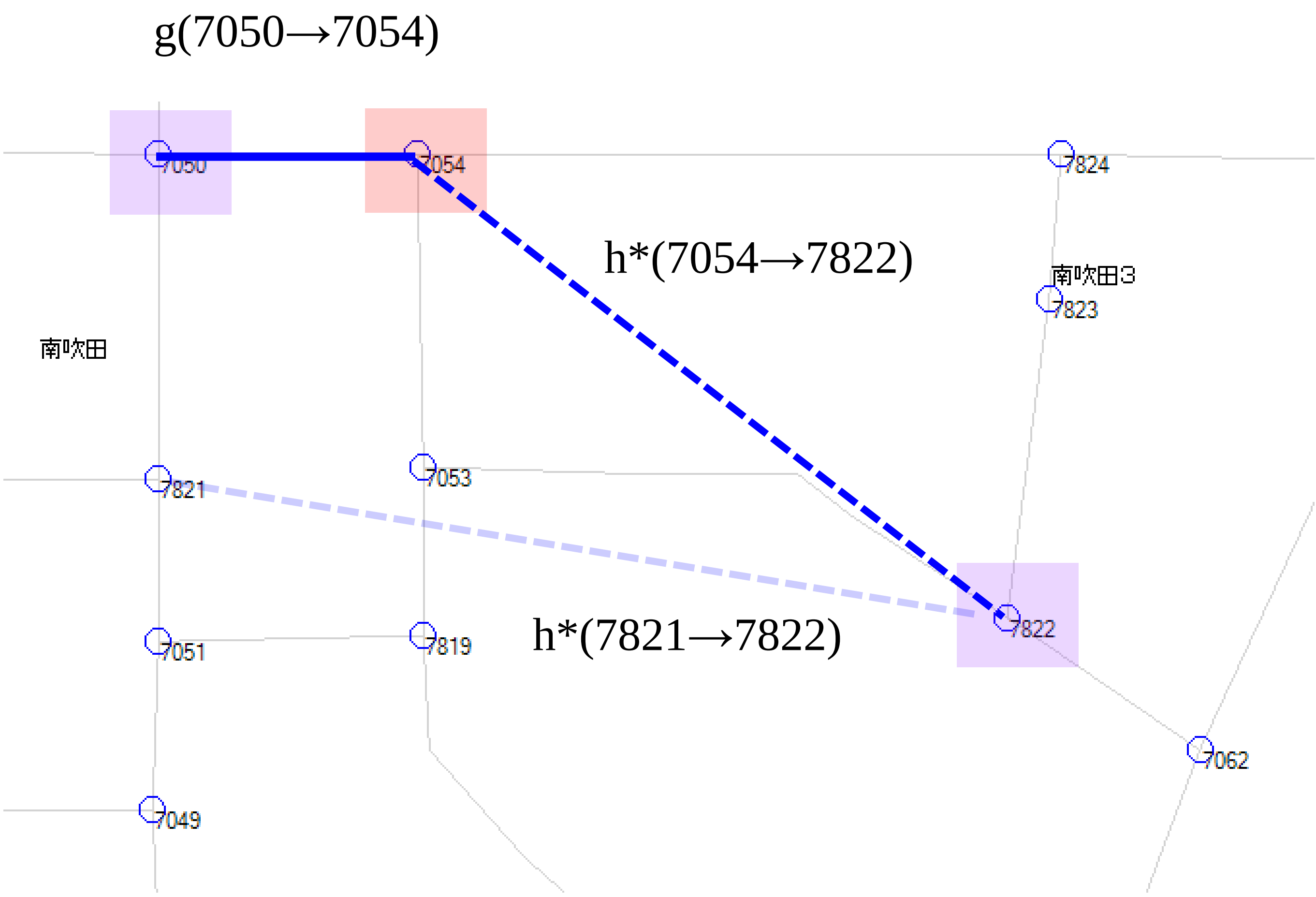
Dijkstra法の改善: A*法

$0 \le h^*(n) \le h(n)$ を満たす h*(n)としては、通常、点nからGOALまでの直線距離が用いられる。h*(n)が小さくなるようにnを選択する。小さくならない場合は後戻りする。

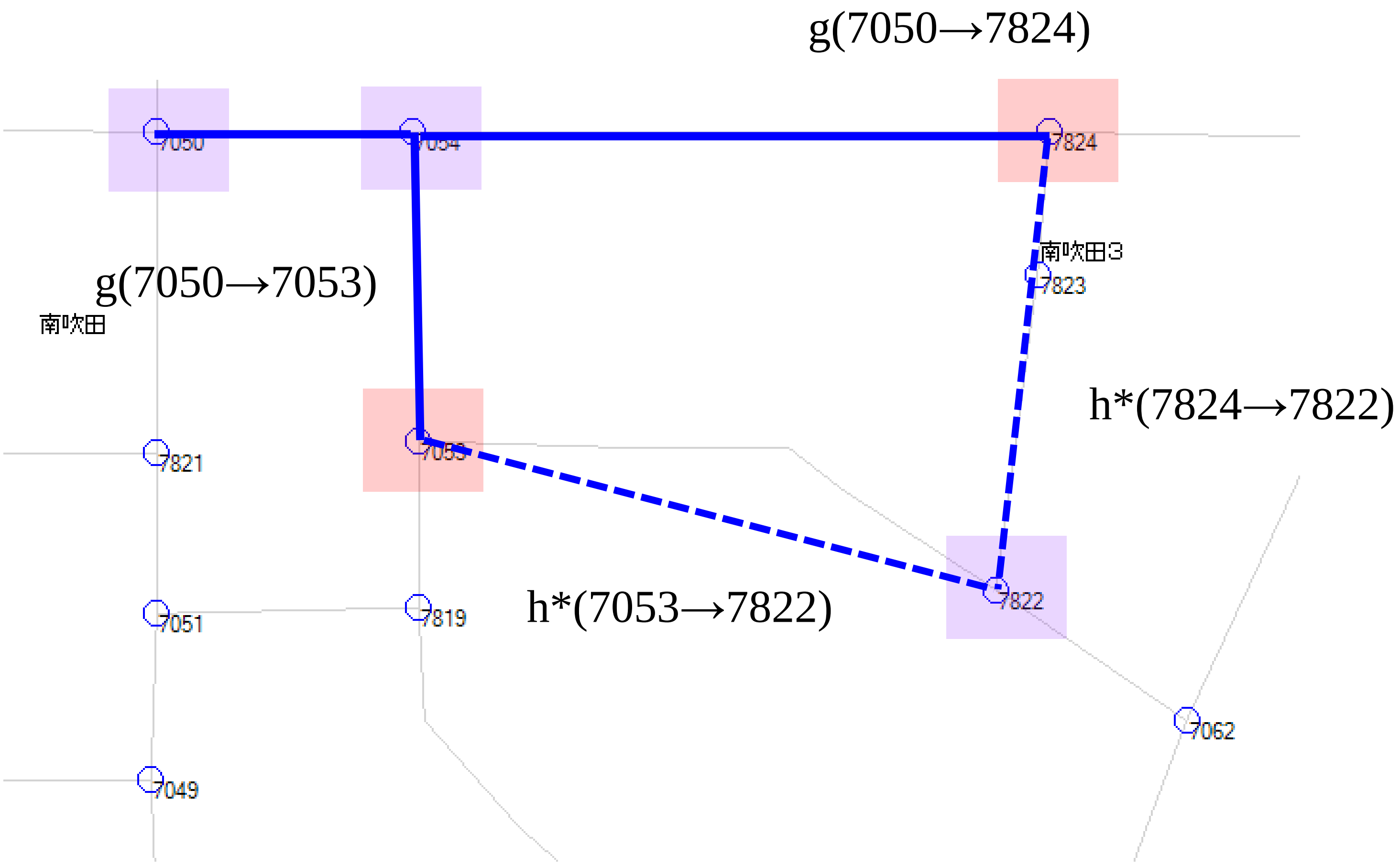


$$f(\mathrm{START} \to \mathrm{GOAL}) = g(\mathrm{START} \to \mathrm{GOAL}) + h^*(\mathrm{GOAL} \to \mathrm{GOAL})$$
$$= g(\mathrm{START} \to \mathrm{GOAL})$$

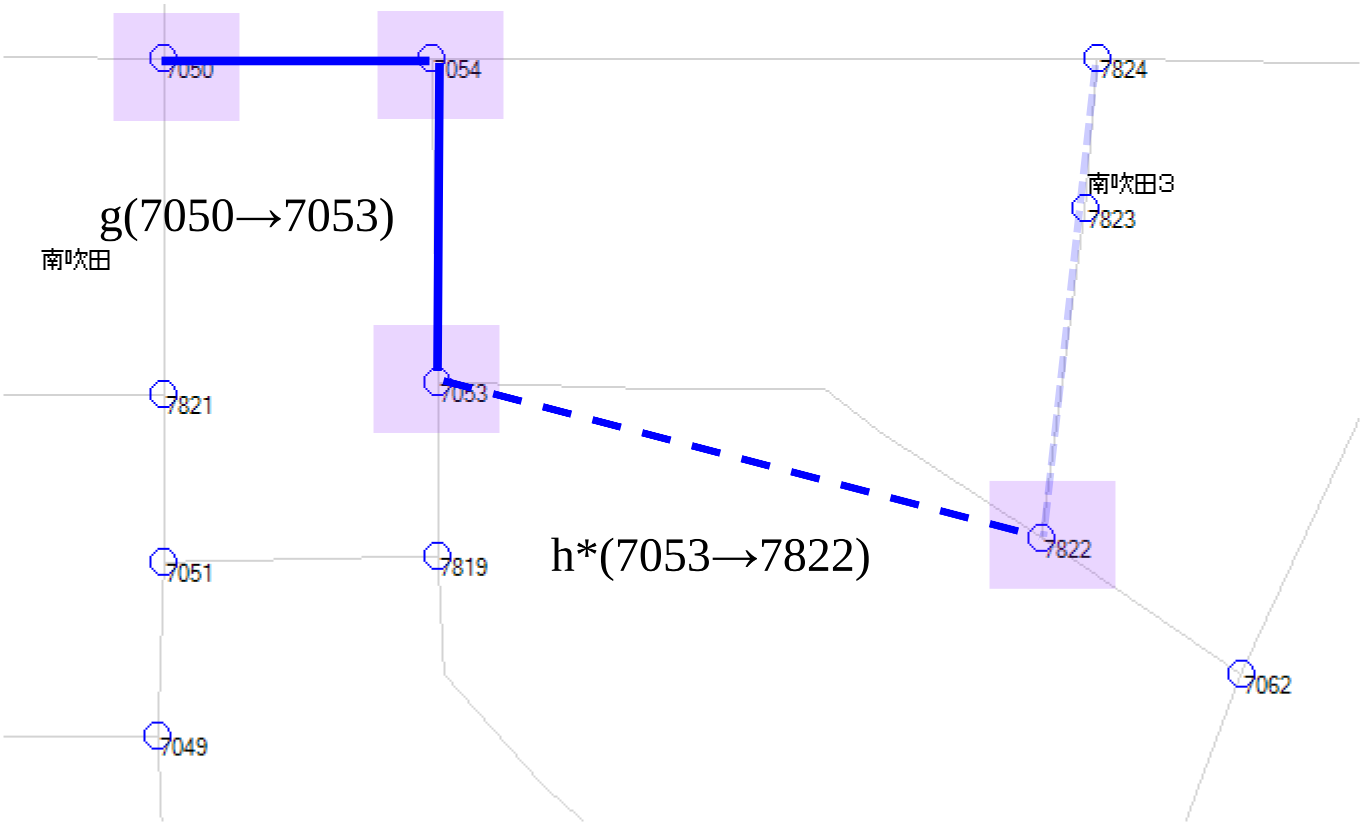
Dijkstra法の改善: A*法



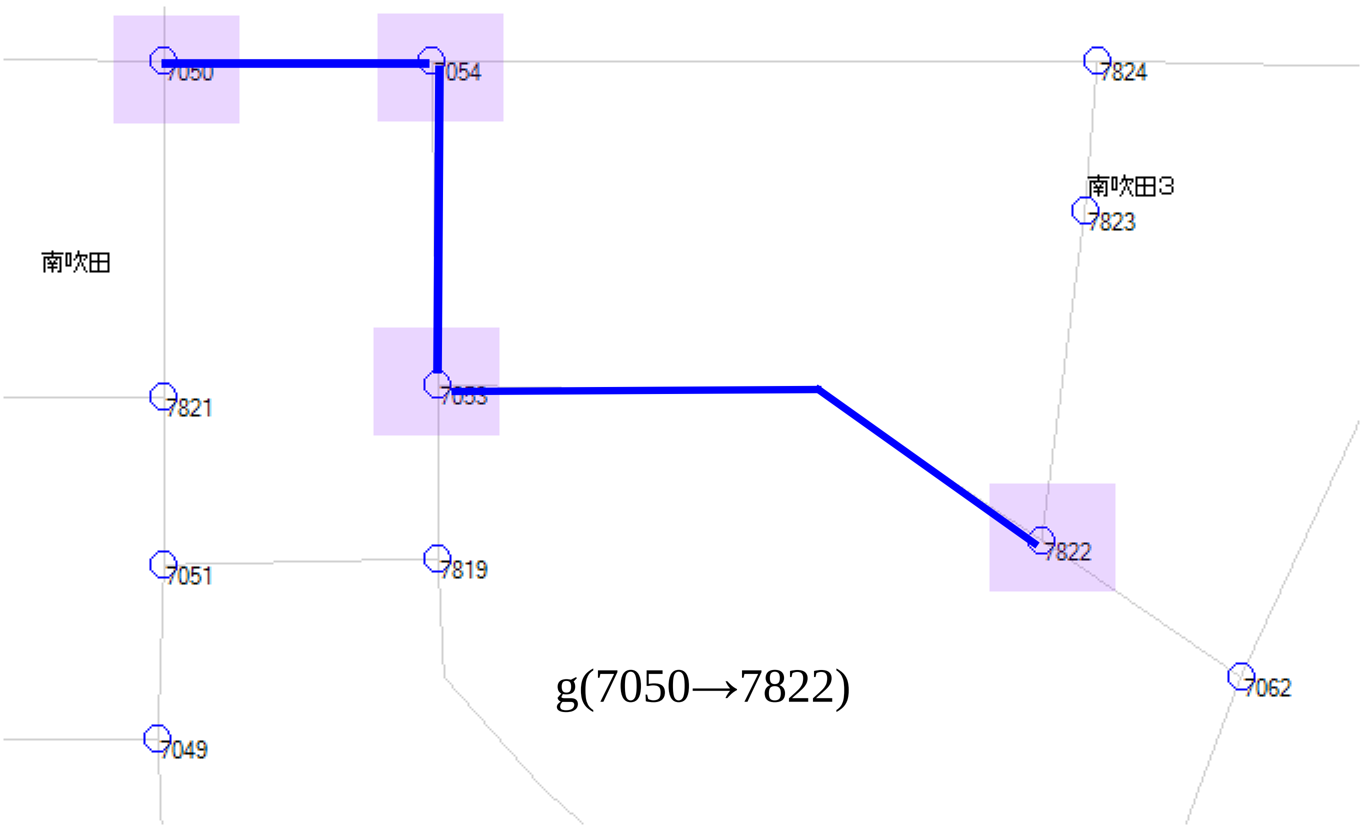
Dijkstra法の改善: A*法



Dijkstra法の改善: A*法

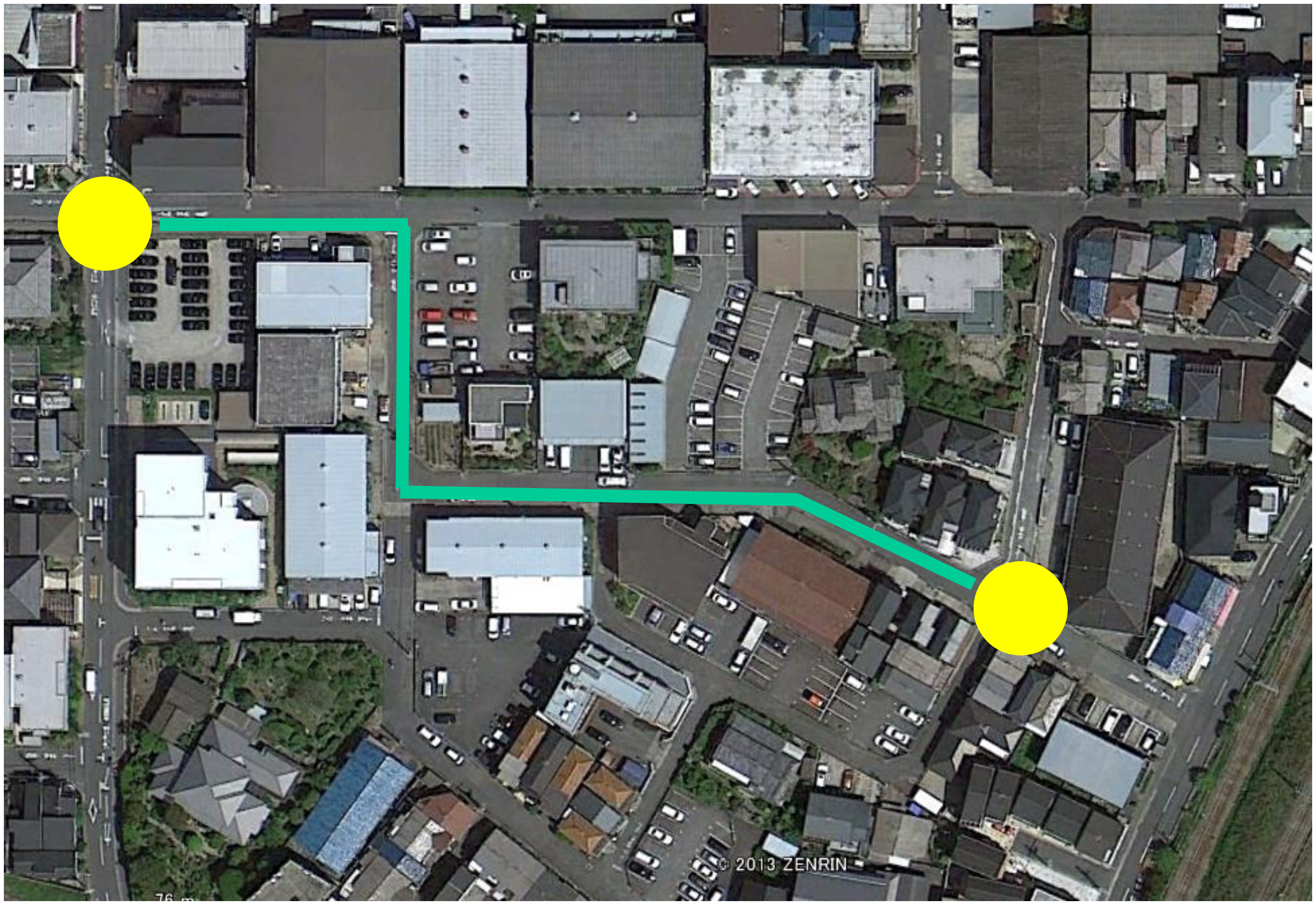


Dijkstra法の改善: A*法



g(7050→7822)

Dijkstra法の改善: A*法



Dijkstra法の改善: A*法

アルゴリズム自体は直感的に理解しやすい。

メリット：探索戦略があるため少ない探索回数で
　　　　　最適解が得られやすい。
課題：　コストを推定する関数（直線距離など）の
　　　　　計算負荷が増える。

演習課題

鳥取市の２次メッシュデータを提供するので道路の形状を描画しなさい。
地図の仕様書は手渡しします。

データは523421.csv
523411.csv

mapdata.zip