Comparison of two algorithms for the search of shortest routes in urban networks

Sartenaer, Annick

Published in:
Belgian Journal of Operations Research, Statistics and Computer Science

Publication date:
1998

Document Version
Early version, also known as pre-print

Comparison of two algorithms for the search of shortest routes in urban networks

A. Sartenaer

Department of Mathematics
Facultés Universitaires ND de la Paix
61 rue de Bruxelles
5000 Namur, Belgium

e-mail : **as@math.fundp.ac.be**

(This work was supported by the Belgian
National Fund for Scientific Research.)

Abstract

This paper presents an application of the so-called "auction algorithm" by Bertsekas (for finding shortest paths in a directed graph) to the search of optimal routes in an urban network. A comparison of this algorithm with Dijkstra's algorithm is performed which shows that Dijkstra's algorithm is particularly well adapted to urban networks.

Keywords : Shortest path problem, sparse networks, auction algorithm.

1 Introduction

Shortest path problems are by far the most fundamental and also the most commonly encountered problems in the study of transportation and communication networks (see [4] and [5], for instance). This type of networks[1], as noticed by Van Vliet in [11], are generally large and sparse (that is, with a number of arcs to number of nodes ratio small – typically around 3).

Dijkstra's algorithm (see [6]) is known as one of the most efficient methods for the search of optimal routes in urban networks, especially when the sparsity of the network is exploited through the use of a *heap*, as suggested by Johnson (see [10]). On the other hand, the so-called "auction algorithm" of Bertsekas for finding shortest paths in a directed graph (see [1] and [2]) is reputed to perform very well on random graphs and problems with few destinations. As mentioned in [3], the auction algorithm even runs faster than Johnson's algorithm on that kind of problems. This paper presents a comparison of the auction algorithm with Dijkstra's method (using a heap) *when applied to the search of optimal routes in an urban network.* The choice to compare Bertsekas' algorithm with the algorithm of Johnson has been motivated by a study performed by Burton in his Ph.D. thesis [3]. In this work, Burton compares several methods for shortest path problems, drawing the conclusion that Dijkstra's algorithm implemented with a binary heap is quite attractive for its practical performance, as well as for the fact that it requires few storage and takes advantage of the sparsity in a very efficient way.

We consider a *directed graph* (N, A) consisting of a set of *nodes* $N = \{1, 2, \ldots, n\}$ and a set of *arcs* $A \subseteq \{(i,j) | i, j \in N \text{ and } i \neq j\}$. We define a *path* P as a sequence of nodes $(i_1, i_2, \ldots, i_k)$ such that (i_l, i_{l+1}) is an arc for all $l = 1, \ldots, k-1$. The path P is called *a simple path* if the nodes $i_1, i_2, \ldots, i_k$ are distinct, and a *cycle* if $i_1 = i_k$. If we now associate a length, a_{ij} say, with each arc (i,j), we can define the *length* of path P as the sum of its arc lengths. The *shortest path problem* then consists of finding the *shortest path* from one particular node to another node, that is, the path of minimum length among all paths between these two nodes.

Section 2 of the paper describes Bertsekas' algorithm for the single origin and single destination case. Section 3 gives the properties and an interpretation of the algorithm of Section 2 when applied to urban networks. We shortly present a variant of Dijkstra's method proposed by Johnson [10] for solving the same problem in Section 4. We discuss computational complexities and some numerical issues for both algorithms in Section 5 and Section 6, respectively. Finally conclusions are given in Section 7.

2 The auction algorithm of Bertsekas

Traditionally, shortest path methods are divided into two categories, label setting (Dijkstra-like) and label correcting (Bellman-Ford-like) (see the surveys given in [8] and [9]). The auction algorithm of Bertsekas (see [1] and [2]) shares charateristics of both categories. As a label setting algorithm, it finds the shortest distance of a node at the first time the node is labelled and, as a

[1] Throughout the paper, we will use the term *urban* network (which is the representative of a class of networks modelling many practical problems that involve graph theory).

label correcting algorithm it may continue to update the label of a node after its shortest distance is found.

We briefly describe here the auction algorithm for the single origin and single destination case (see [2] for more details). The underlying idea of the algorithm is the following. Let us consider the graph as a maze. In order to reach the destination, the auction algorithm will travel through the maze, sometimes advancing, sometimes backtracking along its current path. Each time it backtracks from a node, the algorithm records a measure of the desirability of revisiting and advancing from that node in the future. The algorithm will then revisit and proceed forward from a node when its measure of desirability is higher than the measure of desirability of other nodes. The algorithm stops the first time the destination node is visited.

In his paper, Bertsekas uses the following assumptions on the graph.

AS.1 All cycles have positive lengths.

AS.2 Each node except for the destination has at least one outgoing arc.

AS.3 There is at most one arc between two nodes in each direction.

Let node 1 be the origin node and let t be the destination node. The algorithm maintains at all times a simple path $P = (1, i_1, i_2, \ldots, i_k)$, starting at the origin, and a vector p of *prices* p_i for each node $i \in N$ that represents the measure of desirability mentioned above. It proceeds in iterations, transforming a given pair (P, p) satisfying the *complementarity slackness* condition (or CS for short)

$$p_i \leq a_{ij} + p_j, \quad \text{for all } (i,j) \in A, \tag{2.1}$$

$$p_i = a_{ij} + p_j, \quad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } P, \tag{2.2}$$

into another pair satisfying the same condition. In order to do so at each iteration, the path P is either *extended* by adding a new node i_{k+1}, or *contracted* by deleting its *terminal* node i_k. In the latter case, the price of the terminal node is increased strictly. For the degenerate case that occurs when the path consists of just the origin node 1, this path is either extended or is left unchanged with the price p_1 being strictly increased. When the destination becomes the terminal node of the path, the algorithm terminates.

Assuming that an initial pair (P, p) that satisfies CS is available, it gives the following framework.

Algorithm 1

Step 0 : Let i be the terminal node of P. If

$$p_i < \min_{(i,j) \in A} \{a_{ij} + p_j\}, \tag{2.3}$$

go to Step 1; else go to Step 2.

Step 1 : (Contract path) Set

$$p_i = \min_{(i,j)\in A}\{a_{ij} + p_j\}, \tag{2.4}$$

and if $i \neq 1$, contract P. Go to Step 0.

Step 2 : (Extend path) Extend P by node j_i where

$$j_i = \arg\min_{(i,j)\in A}\{a_{ij} + p_j\}. \tag{2.5}$$

If j_i is the destination t, stop; P is the desired shortest path. Otherwise, go to Step 0.

Figure 2 in Appendix A provides an example for the operations of Algorithm 1 when starting with $P = (1)$ and $p_i = 0$ for all $i \in N$.

3 Interpretation of Algorithm 1 when applied to an urban network

Beside Assumptions AS.1 to AS.3, an urban network often satisfies the following assumption.

AS.4 All arc lengths are nonnegative.

Arc lengths in urban networks indeed usually reflect "travel costs" which are generally nonnegative.

This assumption in turn allows to greatly simplify the initialization phase of Algorithm 1 to find a pair (P, p) satisfying CS. Indeed, if AS.4 holds, the default pair

$$P = (1), \quad p_i = 0 \text{ for all } i, \tag{3.1}$$

satisfies condition CS and can be used initially. In this case, as can be seen on the example of Figure 2 in Appendix A, the terminal node draws the tree of shortest paths from the origin to the nodes that are closer to the origin than the given destination. This results from the properties that the nodes become terminal for the first time *in the order* of their proximity to the origin, and that the shortest distance of a node is found *at the first time* the node becomes the terminal node of the path.

In order to interpret Algorithm 1, let D_i be the shortest distance from the origin 1 to node i. Then, as shown in [2], we have that

$$p_1 - p_j \leq D_j \text{ for all } j \in N$$

and

$$p_1 - p_i = D_i \text{ for all } i \in P$$

throughout the course of the algorithm. We thus deduce that

$$D_i + p_i \leq D_j + p_j \text{ for all } i \in P \text{ and } j \in N. \tag{3.2}$$

Since p_j is an underestimate of the shortest distance from j to t (see [2]), we may view the quantity $D_j + p_j$ as an underestimate of the shortest distance from 1 to t *using only paths passing*

through j. Thus, intuitively, it makes sense to consider a node j as "eligible" for inclusion in the algorithm's path only if $D_j + p_j$ is minimal (see (3.2)).

In consequence to this, we can say that the auction algorithm maintains a path consisting of eligible candidates for participating in a shortest path from 1 to t. It extends this path by a node only if this node is an eligible candidate. Otherwise, if the terminal node i of the path has no eligible neighbour, then the algorithm contracts the path by deleting node i and improves the estimate p_i of the shortest distance from i to t. Note that node i will be revisited only if $D_i + p_i$ becomes again minimal, after sufficiently large increases of the prices of the currently eligible nodes.

4 Dijkstra's method

For comparison purposes and completeness, we present, in Algorithm 2 below, the variant of Dijkstra's method proposed by Johnson [10]. This method is a label setting method.

Algorithm 2

Step 1 : (Initialization) Let $l(i)$ be the label on node i. Set $l(1) = 0$ and mark this label as permanent. Set $l(i) = \infty$ for all $i \neq 1$ and mark these labels temporary. Set $q = 1$.

Step 2 : (Updating of labels) For all arcs $(q, i) \in A$ which have temporary labels, set

$$l(i) = \min\{l(i), l(q) + a_{qi}\}.$$

Step 3 : (Fixing a label as permanent) Among all temporarily labelled nodes i, find j for which

$$j = \arg\min\{l(i)\}.$$

Mark the label of j as permanent and set $q = j$. If q is the destination t, stop; $l(q)$ is the desired shortest path length. Otherwise, go to Step 2.

In each iteration of Algorithm 2, exactly one label that corresponds to the closest node of the origin whose label was not yet permanent, is marked as permanent. As a consequence, the nodes are becoming permanent in the order of their proximity to the origin.

Johnson's method uses a heap (i. e. a binary tree ordered by label values) to store the nodes that are not yet permanently labelled (see [10]). This type of storage exploits the possible sparsity of the network and is therefore well-suited to an urban network.

5 Computational complexities

In this section, we present and discuss the computational complexities of Algorithms 1 and 2.

The computational complexity of the auction algorithm is $O(mn)$, where m is the number of arcs and n the number of nodes in the network (see [2] for details). This bound can be

reduced by considering some more characteristics of the network. If μ bounds the number of arcs in the subgraph of nodes that are closer to the origin than the destination, a more accurate estimate is $O(\mu n)$. Network with a small diameter still improves this computational bound: if ν is the minimum number of arcs in a shortest path from the origin to the destination and $L = \max_{(i,j)\in A} a_{ij}$, then the computational complexity is $O(m\nu L)$ or $O(\mu\nu L)$.

The computational complexity of Johnson's algorithm is $O(m\log n)$, due to the use of a binary heap, when the arc lengths are nonnegative. For sparse networks, this complexity becomes $O(n\log n)$ because $m = O(n)$.

While the complexity of Johnson's method emphasizes the number of nodes, that of Bertsekas' method strongly depends on the number of arcs, their length and the network geometry. The auction algorithm should thus perform better on urban networks having a particular geometry: with arc lengths smaller near the destination than near the origin (for μ to be small), with close origin-destination pairs (for ν to be small), and moreover with small maximal lengths (for L to be small). On the other hand these characteristics should not affect the performance of Johnson's method.

6 Computational results

The main computational bottleneck we encountered in the implementation of Algorithm 1 is the calculation of $\min_{(i,j)\in A}\{a_{ij} + p_j\}$ for a given i, which is done every time node i becomes the terminal node of the path. However, improvements to the algorithm can be applied in order to reduce the number of these calculations. These improvements are presented in [2] and are used in our implementation, except those introduced by Bertsekas to solve the case of multiple origins and which exploit parallel computation.

We have tested both codes on the urban network of Namur, Belgium (see Appendix B). This network has 283 nodes and 724 arcs, and satisfies all the assumptions AS.1 to AS.4. Moreover, like all urban networks, the network of Namur has also the important property of being sparse (i. e. m/n is small). Table 1 presents the results for the single origin/single destination case, for 10 different origins and with destination being always node 283. The origin nodes, the optimal routes computed, the associated optimal costs and the cpu-times in milliseconds for Algorithms 1 and 2 are reported in the table. Table 2 shows the cpu-times in milliseconds for the "one-to-all" problem. All the computations have been performed on a DEC VAX 3500, under VMS, using the standard Fortran Compiler.

Comparing the results for Algorithm 1 and Algorithm 2 in these tables, we observe that Algorithm 1 is not well adapted to the search of optimal routes in an urban network (especially for the "one-to-all" problem). Nevertheless, based on his computational experience, Bertsekas concludes in [2] that his code is by far the fastest code *for random problems* of the type generated by NETGEN (a program that randomly generates shortest path problems [7]) and for few destinations (more than one, but much less than the maximum possible). The difference of performance observed here can be explained by several reasons.

Firstly, the sparsity of the urban network is very well exploited by the use of a binary heap

Origin	Optimal route	Opt. cost	Time	
			Algo 1	Algo 2
1	174, 4, 12, 216, 176, 177, 20, 21, 27, 182, 125, 253, 252, 186, 187, 235, 246, 283	704.929	160	20
5	4, 12, 216, 176, 177, 20, 21, 27, 182, 125, 253, 252, 186, 187, 235, 246, 283	650.929	130	10
10	9, 196, 175, 218, 16, 22, 28, 193, 250, 126, 251, 253, 252, 186, 187, 235, 246, 283	676.500	190	20
20	21, 27, 182, 125, 253, 252, 186, 187, 235, 246, 283	385.386	40	10
50	51, 233, 52, 190, 104, 106, 254, 187, 235, 246, 283	394.086	30	10
100	83, 205, 204, 108, 280, 278, 244, 117, 241, 274, 107, 282, 277, 246, 283	305.600	40	10
150	23, 178, 24, 220, 179, 25, 180, 26, 27, 182, 125, 253, 252, 186, 187, 235, 246, 283	806.672	100	10
200	189, 50, 51, 233, 52, 190, 104, 106, 254, 187, 235, 246, 283	473.886	40	10
250	126, 251, 253, 252, 186, 187, 235, 246, 283	223.486	40	10
280	278, 244, 117, 241, 274, 107, 282, 277, 246, 283	169.400	10	10

Table 1: Solutions for the single origin/single destination case, with destination being node 283 (times in milliseconds).

in Johnson's method while no such tool is used in the implementation of Algorithm 1. It would indeed be inefficient to exploit the sparsity through a binary heap in Bertsekas' method, since at each iteration a minimum value has to be computed on a *different set* of arcs *from one iteration to the other*, as this set is determined by the terminal node of the current path.

The second reason is closely related to the computational complexity discussed in Section 5 and can be illustrated by the following example. Consider the network given in Figure 1 that involves a cycle with relatively small length. As stated in the computational complexity of the auction algorithm and as shown in Table 3, the running time of Algorithm 1 on this example closely depends on the value of L^*. Indeed, by tracing the steps of Algorithm 1, we see that the price of node three will be first increased by 1 and then by increments of 3 (the length of the cycle) as many times as necessary for p_3 to reach or exceed L^*. If this last situation is unlikely to arise for randomly generated problems, it is not the case for urban networks. One only has to imagine a roundabout followed by a long road, situation which actually occurs in the network used for our computational tests. On the other hand, whatever value takes L^*, the behaviour of Dijkstra's method will be unchanged, since exactly one node becomes permanent per iteration and thus only four steps are required to reach the terminal node five. Moreover, the maximal

Origin	Algo 1	Algo 2
1	1190	30
5	1130	20
10	1140	20
20	770	20
50	1020	20
100	810	20
150	1020	20
200	1120	20
250	540	20
280	770	20

Table 2: Solution times in milliseconds for the "one-to-all" problem.

length, L, of the urban network of Namur is equal to 420 (see Table 5) and is not particularly small. This also may have an impact on the relative performance of Algorithm 1.

A third reason is the network geometry. The urban network of Namur is such that ν is not small in general, what also explains the higher cpu-times observed in Table 1 for some origin-destination pairs (see origins 1, 5, 10 and 150). For these pairs, the number of arcs in the shortest path is either 17 or 18.

Finally we observe a much larger diversity in the cpu-times reported in Table 1 for the auction algorithm than for Johnson's algorithm. This may be explained by the sensivity of the auction algorithm to the network geometry, as already stated, in opposition to Dijkstra's algorithm whose performance appears to be independent of the location of the origin in the network.

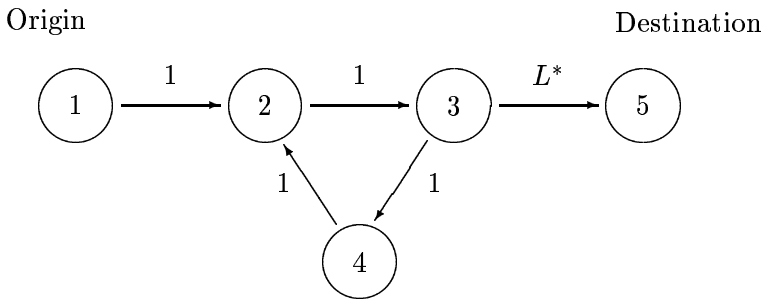


Figure 1: Example graph involving a cycle with relatively small length.

L^*	Algo 1
1	0
100	0
500	10
1000	40
5000	170
10000	350
50000	1730
100000	3470
500000	17340
1000000	173000
5000000	345600

Table 3: Solution times in milliseconds for the problem of Figure 1 with different values of L^*.

7 Conclusions

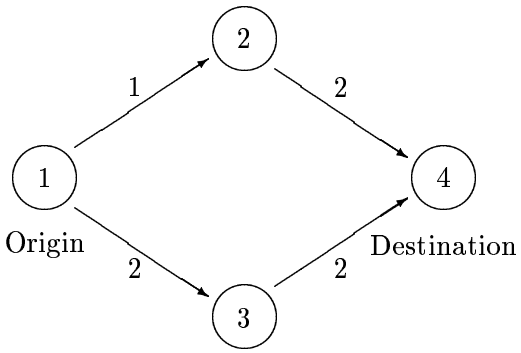
In this paper, we propose a comparison of Bertsekas' algorithm and Dijkstra's algorithm for the search of shortest routes in urban networks (such as Namur). While Bertsekas' algorithm is reputed to perform very well on general problems, this comparison shows that Dijkstra's algorithm using a binary heap is much more adapted to urban networks. Dijkstra's algorithm is indeed especially well suited to exploit the sparsity of this type of network, without being affected by its particular geometry and the presence of cycles with relatively small length.

We may thus further conclude that it is worthwhile taking into account the particular structure of a problem, if any, in the choice of the algorithm to be used for its solution.

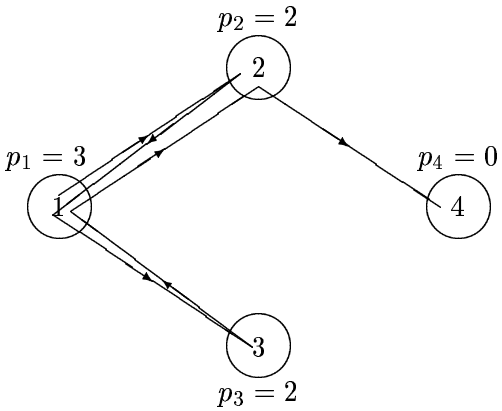
Acknowledgement

The author wish to thank Didier Burton and Michel Bierlaire for the interesting discussions on the above matter. Didier Burton also kindly gave access to his code.

Appendix A



Shortest path problem with arc lengths as shown.



Trajectory of terminal node and final prices generated by the algorithm.

Iteration	Path P prior to the iteration	Price vector p prior to the iteration	Type of action during the iteration
1	(1)	(0,0,0,0)	contraction at 1
2	(1)	(1,0,0,0)	extension to 2
3	(1,2)	(1,0,0,0)	contraction at 2
4	(1)	(1,2,0,0)	contraction at 1
5	(1)	(2,2,0,0)	extension to 3
6	(1,3)	(2,2,0,0)	contraction at 3
7	(1)	(2,2,2,0)	contraction at 1
8	(1)	(3,2,2,0)	extension to 2
9	(1,2)	(3,2,2,0)	extension to 4
10	(1,2,4)	(3,2,2,0)	stop

Figure 2: An example illustrating Algorithm 1.

Appendix B

Key: org = origin node, dst = destination node, cost = arc length.

org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost
1	167	32.400	1	174	25.200	2	148	102.000	2	238	42.000	3	11	78.000
3	174	65.400	4	5	54.000	4	11	166.800	4	12	10.800	4	174	82.800
5	4	54.000	5	8	234.000	6	185	167.000	6	190	52.000	7	199	27.000
8	5	234.000	8	9	237.600	8	10	151.200	9	8	237.600	9	10	100.000
9	196	24.000	10	8	151.200	10	9	100.000	10	149	4.000	11	3	78.000
11	4	166.800	11	17	162.000	12	4	10.800	12	216	18.000	13	176	36.600
13	217	72.000	14	15	28.800	14	176	58.200	14	202	57.000	15	14	28.800
15	177	87.000	16	21	150.000	16	22	39.000	16	218	39.000	17	11	162.000
17	178	81.000	18	179	50.400	19	180	51.000	20	21	54.000	20	177	90.000
21	16	150.000	21	20	54.000	21	27	60.000	21	180	153.000	22	16	39.000
22	28	102.000	22	177	102.429	23	150	168.000	23	178	12.000	24	178	12.000
24	220	82.286	25	179	42.000	25	180	51.000	26	27	87.000	26	180	63.000
27	21	60.000	27	26	87.000	27	28	84.000	27	182	60.000	28	22	102.000
28	27	84.000	28	193	32.100	29	30	123.428	29	150	72.000	29	151	5.143
30	29	123.428	30	31	60.000	31	30	60.000	31	32	60.000	32	31	60.000
32	201	21.000	33	181	87.000	33	182	152.400	34	116	33.720	34	194	14.500
34	272	16.000	35	152	30.000	35	183	36.000	36	37	42.000	36	183	30.000
37	36	42.000	37	45	60.000	37	230	30.857	38	181	87.000	38	184	144.000
38	231	60.000	39	184	51.429	39	185	60.000	40	185	36.000	40	186	221.000
41	42	54.000	41	153	43.200	42	41	54.000	42	188	24.000	43	44	108.600
43	188	54.000	44	43	108.600	44	46	54.000	44	48	36.000	45	37	60.000
46	44	54.000	46	47	60.000	47	46	60.000	47	229	30.000	48	44	36.000
48	49	61.714	48	154	118.286	48	228	51.429	49	48	61.714	49	189	18.000
50	51	43.200	50	189	28.800	51	50	43.200	51	233	43.800	52	90	60.000
52	190	75.000	52	233	30.000	53	54	54.000	53	156	78.000	54	53	54.000
54	55	30.000	54	57	66.000	55	54	30.000	55	191	60.000	55	219	54.000
56	191	72.000	56	232	7.200	57	54	66.000	57	58	28.800	58	57	28.800
58	67	7.200	58	191	36.000	58	237	31.500	59	60	99.000	59	237	9.000
60	59	99.000	60	61	45.000	60	155	51.429	61	60	45.000	61	62	144.000
62	61	144.000	62	63	111.000	62	166	113.143	62	210	66.429	63	62	111.000
63	127	36.600	63	157	25.714	63	236	36.000	64	209	45.857	64	210	46.500
65	66	57.600	65	113	108.000	65	209	60.000	66	65	57.600	66	67	50.400
66	173	109.800	67	58	7.200	67	66	50.400	68	69	64.800	68	209	33.000
69	68	64.800	69	73	36.000	69	208	39.000	70	210	63.000	70	211	43.200
71	211	28.800	71	212	7.200	72	172	64.800	72	221	28.800	73	74	72.000
73	100	104.000	73	172	36.000	73	239	24.000	74	73	72.000	74	75	57.600

Table 4: The urban network of Namur.

org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost
75	74	57.600	75	222	43.200	76	77	36.000	76	158	36.000	76	212	50.400
77	76	36.000	77	159	156.000	77	213	78.000	78	213	48.000	78	223	18.000
79	80	14.400	79	84	78.000	79	100	132.000	79	214	154.286	79	223	78.000
80	79	14.400	80	81	36.000	81	80	36.000	81	224	21.600	82	213	57.600
82	224	24.000	83	100	57.000	83	205	39.000	83	239	36.000	84	79	78.000
84	206	43.800	84	207	37.200	85	86	45.000	85	207	39.000	86	85	45.000
86	87	63.000	86	93	72.000	87	86	63.000	87	88	42.000	88	87	42.000
88	214	48.000	89	160	45.000	89	214	54.000	90	52	60.000	90	91	54.000
90	104	78.000	90	105	69.000	90	234	29.400	91	90	54.000	91	233	27.000
92	121	22.000	92	247	23.229	92	248	34.686	93	86	72.000	93	95	27.000
93	215	60.000	94	161	79.800	94	225	21.600	95	93	27.000	95	103	96.000
95	203	42.000	96	198	14.400	96	203	21.600	97	199	57.600	97	200	36.000
98	101	72.000	98	200	63.000	98	227	57.600	99	162	84.000	99	200	24.000
100	73	104.000	100	79	132.000	100	83	57.000	101	98	72.000	101	163	86.400
102	103	70.000	102	204	135.000	103	95	96.000	103	102	70.000	103	198	90.000
104	90	78.000	104	105	28.000	104	106	36.000	104	190	42.000	104	197	43.800
105	90	69.000	105	197	51.000	106	104	36.000	106	197	73.000	106	254	27.000
107	274	13.200	107	282	7.200	107	137	38.800	107	141	35.920	108	204	8.100
108	280	23.100	108	281	79.000	109	259	14.400	110	266	20.400	110	267	20.400
110	268	27.000	111	112	61.714	111	201	108.600	111	230	60.000	112	111	61.714
112	165	72.000	113	65	108.000	113	73	48.000	113	173	86.400	114	243	27.400
114	262	40.200	115	120	19.800	115	257	37.500	116	34	33.720	116	271	18.550
116	140	35.200	117	241	19.000	117	244	9.000	117	275	15.600	118	248	24.400
118	140	51.000	119	259	30.160	119	273	14.400	120	122	33.400	120	263	24.000
121	92	22.000	121	192	33.900	121	268	12.000	121	269	18.720	122	120	33.400
122	256	41.600	122	265	7.200	123	244	36.400	123	255	40.200	124	243	23.100
125	182	22.200	125	253	33.000	126	194	29.400	126	250	23.400	126	251	17.200
127	63	36.600	127	211	48.000	128	261	27.200	129	130	25.429	129	146	10.286
130	240	24.000	130	129	25.429	131	132	24.400	132	145	24.900	133	267	28.000
133	268	24.600	133	269	17.200	134	257	22.200	135	274	24.400	135	275	22.600
136	245	29.800	136	255	17.200	136	276	23.200	137	107	38.800	137	245	22.600
137	276	27.280	137	277	40.600	138	255	26.200	138	278	28.500	139	118	21.520
139	266	18.600	139	140	43.800	140	110	45.240	140	116	35.200	140	250	45.200
140	139	43.800	141	107	35.920	142	256	11.314	142	271	26.314	143	282	144.000
143	283	136.800	144	195	28.000	144	258	23.229	144	263	11.700	145	247	18.720
145	264	23.868	146	124	24.600	146	258	24.400	147	243	55.200	148	2	102.000
149	10	4.000	150	23	168.000	150	29	72.000	151	29	5.143	152	35	30.000
153	41	43.200	154	48	118.286	154	161	420.000	155	60	51.429	156	53	78.000
157	63	25.714	158	76	36.000	159	77	156.000	159	213	102.857	160	89	45.000
160	161	75.000	161	94	79.800	161	154	420.000	161	160	75.000	162	99	84.000
163	101	86.400	164	226	45.000	165	112	72.000	166	62	113.143	167	1	32.400
168	169	143.600	168	171	194.000	168	196	106.000	169	168	143.600	169	170	125.600
169	237	62.000	170	169	125.600	170	236	62.000	171	168	194.000	171	238	62.000
172	69	12.000	172	72	64.800	173	66	109.800	173	113	86.400	173	191	161.800
173	232	66.000	174	1	25.200	174	3	65.400	174	4	82.800	174	238	18.000

Table 5: The urban network of Namur (continued).

org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost
175	196	30.857	175	217	57.600	175	218	45.857	176	13	36.600	176	14	58.200
176	177	56.143	176	216	36.600	177	15	87.000	177	20	90.000	177	22	102.429
177	176	56.143	178	17	81.000	178	23	12.000	178	24	12.000	179	18	50.400
179	25	42.000	179	220	18.000	180	19	51.000	180	21	153.000	180	25	51.000
180	26	63.000	181	33	87.000	181	38	87.000	181	201	63.000	182	27	60.000
182	33	152.400	182	125	22.200	183	35	36.000	183	36	30.000	183	188	12.000
184	38	144.000	184	39	51.429	184	229	24.000	184	230	30.857	185	6	167.000
185	39	60.000	185	40	36.000	186	40	221.000	186	187	22.200	186	252	36.000
187	186	22.200	187	235	20.400	187	254	24.000	188	42	24.000	188	43	54.000
188	183	12.000	189	49	18.000	189	50	28.800	189	200	51.000	189	228	39.000
190	6	52.000	190	52	75.000	190	104	42.000	191	55	60.000	191	56	72.000
191	58	36.000	191	173	161.800	191	219	44.400	192	121	33.900	192	232	32.200
192	247	23.868	192	249	23.220	193	28	32.100	193	250	40.200	193	267	21.300
193	270	24.360	194	116	22.000	194	126	29.400	194	140	33.040	195	122	29.200
195	208	32.000	195	261	15.800	195	263	26.200	195	264	18.900	195	128	7.200
195	131	21.600	196	9	24.000	196	168	106.000	196	175	30.857	197	104	43.800
197	105	51.000	197	106	73.000	197	281	22.200	198	96	14.400	198	103	90.000
198	199	24.000	199	7	27.000	199	97	57.600	199	198	24.000	200	97	36.000
200	98	63.000	200	99	24.000	200	189	51.000	201	32	21.000	201	111	108.600
201	181	63.000	201	231	96.000	202	14	57.000	203	95	42.000	203	96	21.600
203	227	90.000	204	83	43.200	204	102	135.000	204	108	8.100	204	205	9.000
205	83	39.000	205	204	9.000	205	206	36.600	206	83	78.000	206	84	43.800
206	204	28.800	206	205	36.600	207	84	37.200	207	85	39.000	208	69	39.000
208	73	12.000	208	195	32.000	209	64	45.857	209	65	60.000	209	68	33.000
210	62	66.429	210	64	46.500	210	70	63.000	210	236	51.000	211	70	43.200
211	71	28.800	211	127	48.000	211	221	115.200	212	71	7.200	212	76	50.400
212	222	86.400	213	77	78.000	213	78	48.000	213	82	57.600	213	159	102.857
214	79	154.286	214	88	48.000	214	89	54.000	215	93	60.000	215	225	47.000
215	226	66.000	216	12	18.000	216	176	36.600	217	13	72.000	217	175	57.600
218	16	39.000	218	175	45.857	218	219	49.000	219	55	54.000	219	191	44.400
219	218	49.000	220	24	82.286	220	179	18.000	221	72	28.800	221	211	115.200
222	75	43.200	222	212	86.400	223	78	18.000	223	79	78.000	224	81	21.600
224	82	24.000	225	94	21.600	225	215	47.000	226	164	45.000	226	215	66.000
227	98	57.600	227	203	90.000	228	48	51.429	228	189	39.000	229	47	30.000
229	184	24.000	230	37	30.857	230	111	60.000	230	184	30.857	231	38	60.000
231	201	96.000	232	56	7.200	232	173	66.000	232	192	32.200	233	51	43.800
233	52	30.000	233	91	27.000	234	90	29.400	235	187	20.400	235	245	17.200
235	246	23.886	236	63	36.000	236	170	62.000	236	210	51.000	237	58	31.500
237	59	9.000	237	169	62.000	238	2	42.000	238	171	62.000	238	174	18.000
239	83	36.000	239	113	39.000	239	195	74.000	239	208	42.000	240	124	19.800
240	241	24.000	241	117	19.000	241	240	24.000	241	274	18.400	242	248	28.500
242	256	26.457	242	132	11.314	243	260	7.200	244	117	9.000	244	278	22.200
245	235	17.200	245	136	29.800	245	137	22.600	246	235	23.886	246	277	7.200
246	283	28.800	247	92	23.229	247	192	23.868	247	145	18.720	248	92	34.686
248	118	24.400	248	242	28.500	248	266	18.100	249	192	23.220	249	270	6.480

Table 6: The urban network of Namur (continued).

org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost	org	dst	cost
250	126	23.400	250	193	40.200	250	249	43.840	250	140	45.200	251	126	17.200
251	252	58.200	251	253	26.700	252	186	36.000	252	253	24.900	252	272	35.800
253	125	33.000	253	251	26.700	253	252	24.900	254	106	27.000	254	187	24.000
255	123	40.200	255	281	37.629	255	136	17.200	255	138	26.200	256	118	24.480
256	122	41.600	256	142	11.314	257	109	9.900	257	130	19.450	258	115	1.800
259	34	7.200	259	119	30.160	260	114	9.600	260	144	26.700	261	260	20.700
262	108	26.200	262	279	20.700	262	280	23.100	263	120	24.000	263	122	9.900
263	195	26.200	264	195	18.900	264	261	27.000	264	131	10.800	264	145	23.868
265	242	8.100	265	131	12.000	266	110	20.400	266	248	18.100	266	139	18.600
267	110	20.400	267	193	21.300	267	133	28.000	268	110	27.000	268	121	12.000
268	133	24.600	269	121	18.720	269	270	22.200	269	133	17.200	270	193	24.360
270	249	6.480	270	269	22.200	271	134	5.400	271	142	26.314	272	34	16.000
272	252	35.800	273	259	20.440	273	272	21.400	274	107	13.200	274	241	18.400
274	135	24.400	275	276	24.400	275	135	22.600	276	123	10.800	276	275	24.400
276	136	23.200	276	137	27.280	277	246	7.200	277	282	7.200	277	137	40.600
278	244	22.200	278	280	37.200	278	138	28.500	278	147	17.200	279	280	30.600
279	281	24.000	279	147	20.700	280	108	23.100	280	278	37.200	280	279	30.600
281	108	79.000	281	197	22.200	281	255	37.629	281	279	24.000	282	107	7.200
282	277	7.200	282	143	144.000	283	119	24.400	283	143	136.800			

Table 7: The urban network of Namur (continued).

References

[1] D. P. Bertsekas. The auction algorithm for assignment and other network flow problems: a tutorial. *INTERFACES*, 20:4:133–149, 1990.

[2] D. P. Bertsekas. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, 1(4):425–447, 1991.

[3] D. Burton. *On the inverse shortest path problem.* PhD thesis, Department of Mathematics, FUNDP, Namur, Belgium, 1993.

[4] B. V. Cherkassy, A. V. Goldberg and T. Radzik. Shortest path algorithms: theory and experimental evaluation. *Mathematical Programming*, 73(2):129–174, 1996.

[5] R. Dial et al. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9:215–248, 1979.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] A. Napier, D. Klingman and J. Stutz. Netgen: A program for generating large-scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.

[8] G. Gallo and S. Pallotino. Shortest path methods: a unifying approach. *Mathematical Programming Studies*, 26:38–64, 1986.

[9] G. Gallo and S. Pallotino. Shortest path algorithms. *Annals of Operations Research*, 13:3–79, 1988.

[10] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. Assoc. Comput. Mach.*, 24:1–13, 1977.

[11] D. Van Vliet. Improved shortest path algorithms for transport networks. *Transportation Research*, 12:7–20, 1978.