Proceedings of
the XXVII Summer School, Sozopol'01

Applications of Mathematics in Engineering and Economics

edited by
D. Ivanchev and M.D. Todorov

FACULTY OF
APPLIED MATHEMATICS & INFORMATICS
TECHNICAL UNIVERSITY OF SOFIA

HERON PRESS · Sofia · 2002

Applications of Mathematics in Engineering and Economics'27
Eds. D. Ivanchev and M.D. Todorov, Heron Press, Sofia, 2002

Monotone priority queues in the quickest path problem

L. A. Pilipchuk[1], Y. V. Malakhouskaya[1], Y. H. Pesheva[2]

[1] Faculty of Applied Mathematics and Computer Science,
Belorussian State University, F.Skarina Avenue 220050, Minsk, Belarus
[2]Technical University of Sofia, P.O.Box 384, 1000 Sofia, Bulgaria

Let $S = \{I, U\}$ be a finite oriented net with nodes I and arcs U. The net should not have multiple arcs and loops. Each arc has two characteristics: c_{ij} is the time needed to pass the arc and d_{ij} is the carrying capacity. The net has two special nodes — a source s and a destination t. We need to find a path from s to t with a minimum time to transfer a flow of data G. A mathematical model of this problem is the following:

$$\sum_{(i,j)\in U} \left(c_{ij} + \delta_{ij} \frac{G}{d_{ij}} \right) x_{ij} \to \min,$$

$$\sum_{j\in I_i^+(U)} x_{ij} - \sum_{j\in I_i^-(U)} x_{ji} = \begin{cases} 1, & i = s, \\ -1, & i = t, \\ 0, & i \in I \setminus \{s,t\}. \end{cases}$$

$$\delta_{ij} = \begin{cases} 1, & (i,j) = (i_*, j_*), \quad \text{if} \quad d_{i_*j_*} = \min_{x_{ij}\neq 0}\{d_{ij}, (i,j) \in U\}, \\ 0, & (i,j) \in U \setminus (i_*, j_*). \end{cases}$$

$$x_{ij} \geq 0, \quad (i,j) \in U,$$

$$I_i^+(U) = \{j : (i,j) \in U\}, \qquad I_i^-(U) = \{j : (j,i) \in U\}.$$

This problem for any amount of data to transfer can be reduced to a prevalent path search [1]. For the search it is convenient to use an algorithm with decreasing carrying capacities [1].

First consider a net having only nodes but no arcs. We sort the arcs so that their carrying capacities do not increase. We add arcs to the net in that sorted order. We assign a label $D[i]$ equal to the minimum time needed to go from s to i for each node i.

Initially $D[i] = \infty$ for all nodes except the source, $D[s] = 0$.

When adding an arc (i, j) we check: if $D[i] \neq \infty$ and $D[i] + c_{ij} \leq D[j]$, that is if we have found a path to j with a lower time than known before, then we should recalculate graph's labels starting with node j. If the label $D[t]$ has changed, then we have found a path from s to t with the smallest time among all paths with carrying capacity not less than d_{ij}. It means [1] that this path is prevalent. We add it to the list of prevalent paths. After all arcs are added we will have a full list of prevalent paths sorted by carrying capacity.

Let us show the work of the algorithm on an example. Let $S = \{I, U\}$ be the given net shown on Figure 1.

Node 0 is a source and node 5 is a destination.

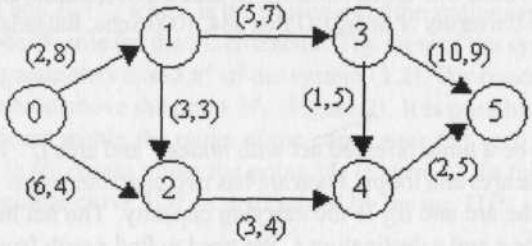

Figure 1.

1) Let us order the arcs with their capacities not increasing: $(3,5)$, $(0,1)$, $(1,3)$, $(3,4)$, $(4,5)$, $(0,2)$, $(2,4)$, $(1,2)$.

2) Then we take all the nodes of the net and assign each of them an initial label (see Figure 2).

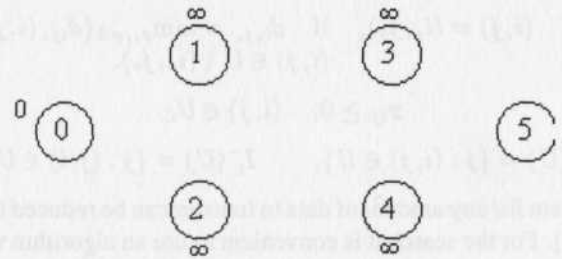

Figure 2.

3) When we add arc (3,5) nothing changes, because $D_3 = \infty$.

4) We add arc (0,1) and recount node labels starting with node 1. The only node we should process is node 1. Now we can reach it in 2 time units (see Figure 3.)

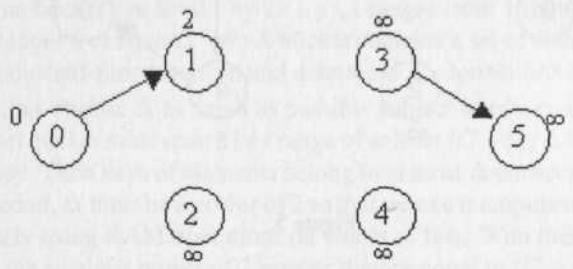

Figure 3.

5) We add arc (1,3). While recalculating node labels, D_5 changes (see Fig.4). This means that we have found a prevalent path 0-1-3-5 with the carrying capacity equal to 5 and the passing time equal to 17 (see Figure 4).

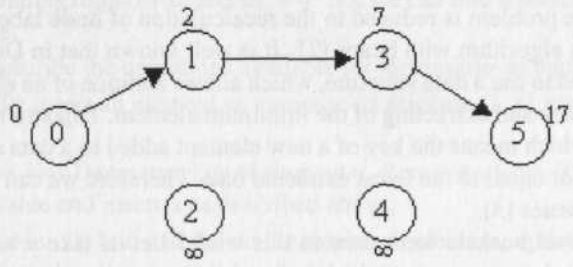

Figure 4.

6) When we add arc (3,4), the label of node 4 changes. Now $D_4 = 8$.

7) We add arc (4,5). After the recount operation the label of node 4 changes from 17 to 10. So the second prevalent path is 0-1-3-4-5. Its carrying capacity is 5 and passing time equals to 10 (see Figure 5).

8) We add arc (0,2). After its insertion the label of node 4 changes.

9) After addition of arc (1,2) we obtain one path more from the source to the destination. The path is 0-2-4-5. But the label of node 5 has not changed, therefore this path is not prevalent (see Figure 6).

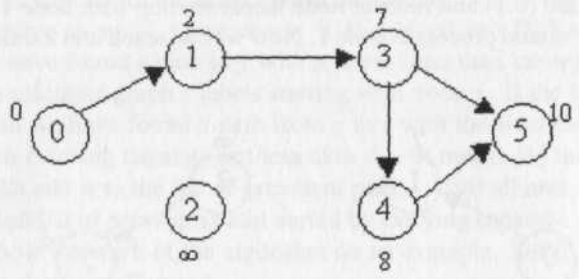

Figure 5.

10) After addition of the last arc we obtain one path more: 0-1-2-4-5, but it also will not be prevalent.

Result: the net has 2 prevalent paths: 0-1-3-5, 0-1-3-4-5. Each of these paths can be prevalent at various values of the flow.

So, path 0-1-3-5 is the quickest for $G > 122,5$ and path 0-1-3-4-5 is the quickest for $G < 122,5$. The others paths can not be the quickest no matter what the value of the flow is.

So, now the problem is reduced to the recalculation of node labels. We use for it Dijkstra's algorithm with heaps [2]. It is well known that in Dijkstra's algorithm we need to use a data structure, which allows addition of an element, decreasing of its key and extracting of the minimum element. Dijkstra's algorithm is monotone, which means the key of a new element added to a data structure is always greater or equal to the latest extracted one. Therefore we can use monotone priority queues [3].

The multilevel buckets were used in this work. Let us take a look at their structure.

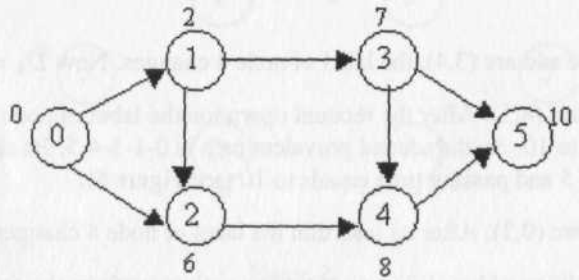

Figure 6.

We treat element keys as base-Δ numbers. Let us consider a bucket structure that contains k levels, let k be a positive integer. Except for the top level, a level contains an array of Δ buckets. The top level contains infinitely many buckets. Each top-level bucket corresponds to an interval of keys $[i\Delta^k, (i+1)\Delta^k - 1]$.

We denote bucket j at level i by $B(i,j)$, i ranges from 1 (bottom level) to k (top) and j ranges from 0 to $\Delta - 1$. A bucket contains a set of elements in a way that allows constant-time insertion and deletion, *e.g.*, double linked list.

Given k, we choose Δ as small as possible subject to two constraints. First, each top-level bucket must span a key range of at least $(C+1)/\Delta$ where C is the maximum key. Then keys of elements belong to at most Δ consecutive top-level buckets. Second, Δ must be a power of 2 so that we can manipulate base-Δ numbers efficiently using RAM operations on words of bits. With these constraints, we set Δ to the smallest power of 2 greater than or equal to $(C+1)^{1/k}$.

Let μ be the key of the latest element extracted from the queue. Denote μ_i as the i-th least significant digit of μ in a base-Δ representation. Similarly for element u with a key $\rho(u)$, denote u_i — the i-th least significant digit of $\rho(u)$. Let i be the index of the most significant digit in which $\rho(u)$ and μ differ, or 1, if $\rho(u) = \mu$. Given μ and u with $\rho(u) \geq \mu$, we denote the position of u with respect to μ by (i, u_i). If u is inserted into B, it is inserted into $B(i, u_i)$. If an element u is in $B(i,j)$, then only the i least significant digits of $\rho(u)$ differ from the corresponding digits of μ, and $u_i = j$. So, we can find a position of element in $O(1)$ time.

Let us describe the operations available for the multilevel bucket [3].

Insert. To insert an element u, compute its position (i,j) and insert u into $B(i,j)$.

Decrease_key. Decrease a key of element u. Remove u from $B(i,j)$, set $\rho(u)$ to the new value and insert u as described above.

Extract_min. To find and delete the minimum element, update μ and move elements affected by the change of μ, find the lowest nonempty level i, find j the first nonempty bucket at level i. If $i = 1$, delete an element from $B(i,j)$, set $\mu = \rho(u)$ and return u. (In this case all elements' positions remain the same.) If $i \geq 1$, examine elements $B(i,j)$ and delete the minimum element u from $B(i,j)$. Set $\mu = \rho(u)$ and expand bucket $B(i,j)$. Return u. The bucket expansion procedure moves elements to the new positions with respect to a changed μ.

Dijkstra's algorithm begins with an empty bucket, does a sequence of operations and stops when the bucket is empty again. Such a sequence of operations is called balanced.

Theorem 1 *For a balanced sequence, amortised bounds for the multilevel bucket implementation of priority queue operations are as follows: $O(1)$ for insert, $O(1)$ for decrease_key and $O(k + C^{1/k})$ for extract_min [3].*

Proof. In the worst case we can add an element at $O(1)$ time. It takes the same time to decrease the element key, provided we know its address.

Let us consider searching and extracting of the minimal element. We are able to find the bucket with the minimal element at $O(1 + \Delta) = O(\Delta)$ time. We should add the cost of bucket expansions. This cost is proportional to the number of elements in the bucket. Each element can move down by expansion operation at most $k - 1$ times. In a balance sequence each inserted element should be extracted therefore we charge moves of elements to a lower level to the extract_min operation. Notice that $\Delta = O(C^{1/k})$ and therefore $O(k + \Delta) = O(k + C^{1/k})$.

The best bound of $O(\log C / \log\log C)$ for extract_min is obtained for $k = [\log C / 2 \log\log C]$.

Let us see the use of multilevel bucket on an example. Let us recalculate node labels for the following net.

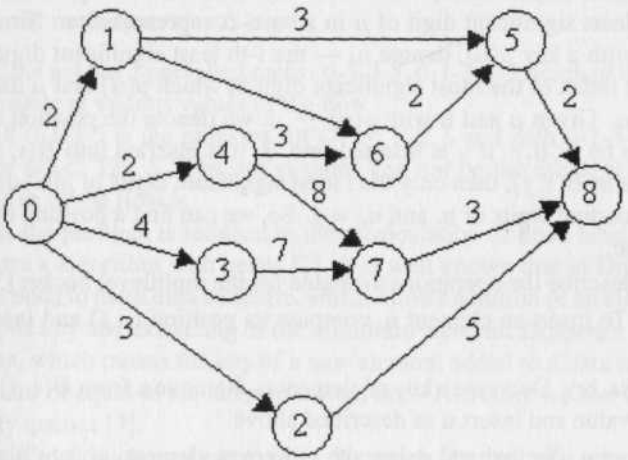

Figure 7.

We will use multilevel bucket with $k = 2$, $\Delta = 2^2 = 4$. Initial $\mu = 0$.

1) Examine all nodes, which are reachable from node 0 and insert their labels into the bucket:
   - node 1: Insert(1,2). $2_{10} = 2_4 \Rightarrow$ the element goes into $B(1, 2)$.
   - node 2: Insert(2,3). The element goes into $B(1, 3)$.
   - node 3: Insert(3,4). $4_{10} = 10_4 \Rightarrow$ element goes into $B(2, 1)$.
   - node 4: Insert(4,2). This one goes into $B(1, 2)$.

   After the first step of Dijkstra's algorithm the elements are organized in the bucket in the following way:

	j=1	j=2	j=3
i=1		(1,2);(4,2)	(2,3)
i=2	(3,4)		

2) Extracting minimum. The element we are looking for should be in the first nonempty bucket on the first nonempty level. That bucket is $B(1,2)$ and the element to extract is (1,2). After extracting μ changes to 2 and bucket $B(1,2)$ should be expanded. The expansion operation moves element (4,2) into $B(1,1)$.

   Further, let us examine nodes that can be reached from 1.

   - node 5: Insert(5,5). $5_{10} = 11_4 \Rightarrow$ the most significant digit, in which 011 differs from 002 is 2, so the element goes into $B(2,1)$.
   - node 6: Insert(6,12). $12_{10} = 30_4 \Rightarrow$ the element goes into $B(2,3)$.

	j=1	j=2	j=3
i=1	(4,2)		(2,3)
i=2	(3,4);(5,5)		(6,12)

3) Extract_min returns element (4,2). The quantity μ has not changed. While passing the next nodes the labels of node 6 and node 7 are being changed:

   - Insert (7,10). $10_{10} = 22_4 \Rightarrow B(2,2)$
   - Decrease_key(6,5). $5_{10} = 11_4 \Rightarrow B(2,1)$

	j=1	j=2	j=3
i=1			(2,3)
i=2	(3,4);(5,5);(6,5)	(7,10)	

4) The next extracted element is (2,3). The quantity $\mu = 3$.

   - Insert (8,8). $8_{10} = 20_4 \Rightarrow B(2,2)$

	j=1	j=2	j=3
i=1			
i=2	(3,4);(5,5);(6,5)	(7,10);(8,8)	

5) Extract_min returns (3,4). The quantity $\mu = 4_{10} = 10_4$ and $(1,2)$ has to be expanded. $5_{10} = 11_4 \Rightarrow$differs from $\mu = 10_4$ in first digit. As a result we have:

	j=1	j=2	j=3
i=1	(5,5);(6,5)		
i=2		(7,10);(8,8)	

6) Extract operation returns (5,5).

   – Decrease_key (8,7). $7_{10} = 13_4 \Rightarrow B(1,3)$.

7) Extracting (6,5).

8) Extracting (8,7).

9) Extracting (7,10).

Now the bucket is empty. We found the following node labels:

I	0	1	2	3	4	5	6	7	8
D[i]	0	2	3	4	2	5	5	10	7

References

[1] Pilipchuk, L.A., Laput, I.V. (2000) The quickest path problem, *International science conference "Computer Network Technologies", October, 25-29, Minsk*

[2] Kormen, T., Leiserson, C., Rivest, R. (1999) *Algorithms: Construction and Analysis*, MCNMO, M.

[3] Cherkassky, B. V., Goldberg, A. V., Silverstein, C. (1999) Buckets, heaps, lists and monotone priority queues, *SIAM J. Comput.* **28**(4) 1326-1346.