For the computation of the δ's and the update of the dual variables, priority queues are very useful. The best asymptotic bounds are obtained if we use for the implementation of priority queues socalled Fibonacci heaps.

Michael L. Fredman, Robert E. Tarjan, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, JACM 34 (1987), 596–615.

A priority queue or a heap is an abstract data structure consisting of a set of items, each with a real-valued key, subject to the following operations

make heap:	Return a new, empty heap.
insert (i, h):	Insert a new item i with predefined key into heap h.
find min (h):	Return an item of minimum key in heap h.
delete min (h):	Delete an item of minimum key from heap h and return it.

In addition, the following operations on heaps are often useful:

meld (h_1, h_2):	Return the heap formed by taking the union of the item-disjoint heaps h_1 and h_2. This operation

destroys h_1 and h_2.

decrease key (Δ, i, h): Decrease the key of item i in heap h by subtracting the nonnegative real number Δ. This operation assumes that the position of i in h is known.

delete (i, h): Delete arbitrary item i from heap h (position i in h is known).

Fibonacci heaps support

delete min and delete — in $O(\log n)$ amortized time

each other operation — in $O(1)$ amortized time

Computation of δ:

- All M-free nodes have the same dual weight.
  $\Rightarrow$
  δ_0 can be computed by the consideration of any M-free node.

- For the computation of δ_1, we maintain a priority queue P_1 which contains for all $[j, A] \notin A_T$ with $[j, B] \notin B_T$ all edges (i, j) with $[i, B] \in B_T$ and the property that (i, j) has minimum reduced cost within all such edges, if such an edge exists. Furthermore, using an array of size n, we have direct access to the

elements in $\mathcal{P}_1$ corresponding to the node $[j, A]$ (for each j).
The weight of such an element will be the reduced cost of the current edge (i,j) such that $[i,B] \in \mathcal{B}_T$ and $r(i,j)$ is minimum within all such edges.
We use a Fibonacci heap for the realization of the priority queue.

Note that each extension step decreases all weights of the elements in the priority queue by the current δ.

Idea

Maintain the property that always the weight of all elements in $\mathcal{P}_1$ has to be decreased by the same amount.

↝

We maintain the sum Δ_1 of all dual changes done so far. If we add an edge (i,j) to the priority queue, we define the weight of the edge to be $r(i,j) + \Delta_1$.

↝

We update $\mathcal{P}_1$ with respect to $[i,B] \in \mathcal{B}_T$ at the moment when $[i,B]$ is added to $\mathcal{B}_T$ in the following way:

- For all edges (i,j) with $[j,B] \notin \mathcal{B}_T$ and $[j,A] \notin \mathcal{A}_T$ perform the following update operations:

(1) If no element with respect to j is contained in the priority queue then insert the element (i,j) with weight $r(i,j) + \Delta_1$.

(2) If P_1 contains an element with respect to j with larger weight than $r(i,j) + \Delta_1$ then replace the corresponding edges by (i,j) and decrease its weight such that its value becomes $r(i,j) + \Delta_1$

(3) If P_1 contains an element with respect to j with weight $r(i,j) + \Delta_1$ then add the edge (i,j) to the corresponding list of edges.

(4) In other cases do nothing.

- If P_1 contains some edges with respect to the node $[i, A]$ delete this entry and the corresponding list of edges.

If $\delta = \delta_1$ then we have to delete at least one minimal element from P_1.

It is easy to see that the total time for the updates of P_1 is bounded by $O(m + n \log n)$.

Exercise:

Prove that the total time used for the manipulation of P_1 is bounded by $O(m + n \log n)$.

For the computation of δ_2, we maintain a priority queue P_2 which contains all edges

(i,j) such that $[i,B], [j,B] \in B_T$ and $r(i,j) > 0$.

We can use a heap for the realization of P_2 or Fibonacci heaps.

Note that each extension step decreases all weights of the elements (i,j) in P_2 with

- $\#\, D'_{[q+1]}$ ~~alive~~ current such that $i,j \in D'_{[q,A]}$

by 2δ where δ is the current value chosen for the current extension step.

Analogously to the manipulation of P_1, we maintain with respect to P_2 the sum Δ_2 of all dual changes done so far with respect to edges in P_2 and modify the weights in the appropriate manner.

- We update P_2 with respect to $[i,B] \in B_T$ at the moment when $[i,B]$ is added to B_T in the following way:
  - For all $[j,B] \in B_T$ with $r(i,j) > 0$ insert the edge (i,j) with weight

$$r(i,j) + \Delta_2.$$

Since at most n edges are inserted, the used time is

- heaps	$O(n \log n)$
- Fibonacci heaps	$O(n)$

Computation of δ_2:

(1) findmin

(* let (i,j) be the output findmin *)

(2) If $\exists D_{[q,A]}$ ~~alive~~ current with

$$(i,j) \in D'_{[q,A]}$$

then

deletemin ;

goto (1)

else

$$\delta_2 := \text{weight}(i,j) - \Delta_2 .$$

If $\delta = \delta_2$ then we have to delete at least one minimal element from P_2

total time:

- heaps $O(m \log m)$
- Fibonacci heaps $O(m \log m)$

02.07.

For the computation of δ_3, we maintain a priority queue P_3 which contains for all alive $D_{[q,A]}$ with $[q,B] \notin B_T$ and $[q,A] \in A_T$ the value $\mu(E_q)$.

We can use a heap or Fibonacci heaps for the realization of P_3.

Each extension step decreases all weights of the elements in P_3 by two times the current δ. Hence, we can use the value Δ_2 defined above and modify the weights in the appropriate memos.

Update of P_3 before the computation of δ:

We have to insert

$\forall\, [q,A] \in V_A$ such that

- $D_{[q,A]}$ is alive,
- $[q,A]$ has been inserted to T_{exp} after the last dual change and the last augmentation, respectively but
- $[q,B] \notin T_{exp}$

the value

$$\mu(E_q) + \Delta_2.$$

We have to delete

$\forall\, [q,B]$ which are inserted to T_{exp} after the last dual change and the last augmentation, respectively

the corresponding value if in P_3 such a value exists.

total time $O(n \log n)$

If $\delta = \delta_3$ we have to delete at least one minimal element from P_3. As observed above, the number of such deletion is bounded by

total time $O(n \log n)$.

We have proved the following theorem

Theorem 1.10

The primal-dual method can be implemented such that its time complexity is $O(nm \log n)$.