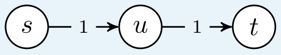
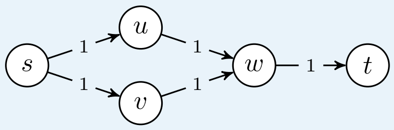
Task 1

a) There can be more than one s-t cut with minimal capacity. A simple example is the following flow network, where both s-t cuts have the same capacity.

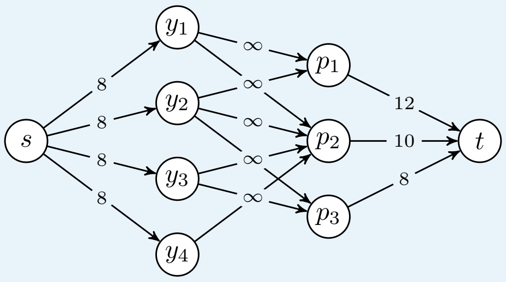


b) No, there can be many ways to maximize flow for a given flow problem. An example of this can be seen in the graph below, where the flow can go either through u or v, but not through both parts (given integer flow).



Task 2

Here we can model the problem as a flow problem where the flow represents man-years. We create a vertex for each year and a vertex for each project. We then allow an edge with capacity equal to the number of man-years available in each year from the source to each year. From a year there are edges with infinite capacity for each of the projects that can be worked on in that year (minimum start year, maximum deadline). From each of the projects, an edge with a capacity equal to the project size runs to the drain. A flow network based on this is shown below.

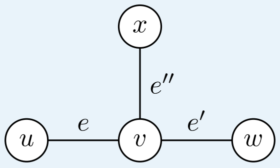


Note that in the flow network there cannot be more flow through any of the years than there are man-years available for that year. In the same way, no more flow can come into a project than what is not used by the other projects in the years the project is available. Nor can there be more flow through a project than is required to complete the project. This means that if and only if the flow is equal to what is required to complete all the projects, then it is possible to complete all the projects. Then the flow from year to project will also provide a plan for when the projects will be worked on.

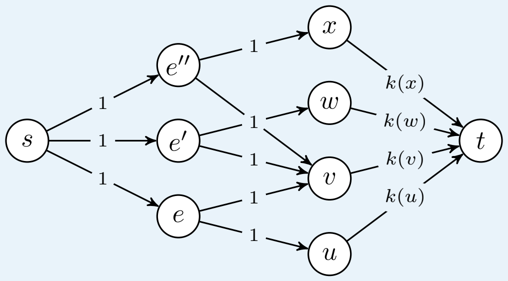
If we find the maximal flow for the flow network, we get that this is 30. This means that it is possible to carry out all the projects. This can e.g. give a plan to put 8 man-years in project 1 in year 1 and 4 in year 2. Project 3 is carried out with 4 man-years in year 2 and 4 in year 3. Then project 2 is carried out with 4 man-years in year 3 and 6 in year 4 .

Task 3

Create a flow network with a vertex for each edge and each vertex in the graph G. For each vertex representing an edge, $\{u, v\}$, let there be an edge of capacity 1 from the source to the vertex and an edge of capacity from the vertex to each of the vertices representing u and v . From each vertex representing a vertex in the original graph $v \in V$, let an edge of capacity $k(v)$ go to the drain.



If we let the graph be as above, we end up with the following flow network



Since all the edges have integer capacity, we know that there is an optimal integer flow. In an arbitrary integer flow, a unit of flow from the vertex of an edge, $e \in E$, to the vertex of a vertex, $v \in V$, indicates that the edge e goes to v in the directed graph. Note that no flow can enter a node v from more than $k(v)$ edges, which obeys the definition of a k-orientation. For there to be a k-orientation, we must have a directed edge in D for every edge in E. This means, it is only possible if the max flow is equal to $|E|$. Then the flow will also define a directed graph which is a k-orientation of G.

Task 4

a) $n = m = C$

We need to push flow in and out of the n first vertices, then we need to push C flow along the k vertices in the path in the middle, then in and out of the m last vertices before the sink. If we use the minimum distance version of the algorithm with the *Discharge* operation, we then need $2n + (k-1) + 2m$ operations, since we do not need to push flow backwards again.

b) $n = C$, $m \ll n$

The same method until the flow reaches the last m vertices. There is no room for C flow between them, so $n-m$ flow ends up being pushed back. This will take at worst $k-1+2(n-m)$ operations if we do not use any other techniques. In total:

$2n + k - 1 + 2m + k - 1 + 2(n - m) = 4n + 2(k - 1)$ operations.

c) $n \ll m$, $m = C$

Here there is more than enough room for the flow that we manage to push forward to the last vertices, so it corresponds to problem 1., only with slightly less total flow.

$2n + k - 1 + 2n = 4n + k - 1$ operations.

Task 5

This is a standard example of a situation where multiplicative weight update works well. If we let the advisors correspond to the choices in Algorithm 7.1 (page K229) and let $v_t(i)$ be 1 if advisor i gave a good advice in time step t and 0 if the advisor gave a bad advice in the time step. Then we end up with the result from theorem 7.2 (page K229) applying. In this situation, this means that we expect to do not much worse than the adviser who gives the most good advice.

Task 6

a) In this situation, we do not know anything about how the values, $v_t(i)$, are set. Thus, it is impossible to pick values in any way that, over all possible outcomes, is better than guessing at random. So, we cannot expect to be closer to an optimal solution for all possible ways of setting values than choosing randomly.

b) No. If we always make the choice that so far has the greatest weight, then for some ways the values can be set we will never be able to achieve what the evidence suggests. For example, if we draw two random choices, i and j ($i < j$), and for the remaining selections, the value at all time steps is set to 0. For i and j , set the values as as in the table below ($0 < \epsilon \leqslant 1/4$). Then in each time step we will end up making a choice that has a value of 0, although we could have ended up with a total value of approximately 3T/8 if we chose either i or j in all time steps. This is because i will always have the greatest weight in time steps where i has value 0. The same is true for j.

t	1	2	3	4	5	6	…
i	0	$\frac{3}{4} - \epsilon$	$\frac{3}{4} + \epsilon$	0	0	$\frac{3}{4} - \epsilon$	…
j	$\frac{3}{4}$	0	0	$\frac{3}{4} - \epsilon$	$\frac{3}{4} + \epsilon$	0	…

This means that if we choose deterministically, we can end up with given distributions of the values where we always do poorly. For the random choice made in multiplicative weight update, we do well on average anyway for all possible ways the values can be distributed. So, with non-deterministic choices we can guarantee that we do relatively well even if the values are set by a malicious adversary, which we cannot with the deterministic choices.

Task 7

Here one can basically follow the same procedure as in the proof of theorem 7.2 in the compendium (pages K229 and K230). A couple of changes are needed. For the upper bound, we must use that $1 - x \leqslant e^{-x}$ rather than $1 + x \leqslant e^x$. Thus we get that

$$\mathrm{W}_{\mathrm{T}+1} \leqslant \mathrm{N} \cdot \exp\left(-\epsilon \sum_{t=1}^{\mathrm{T}} \sum_{i=1}^{\mathrm{N}} c_t(i) p_t(i)\right)$$

For the lower bound, in order to be able to apply the same calculation rules when combining the upper and lower bounds, you must factor in two terms, one when $c_t(j) \geqslant 0$ and one when $c_t(j) < 0$. Thus, we get that the lower bound is

$$\mathrm{W}_{\mathrm{T}+1} \geqslant (1-\epsilon)^{\sum_{t:c_t(j)\geqslant 0} c_t(j)} (1+\epsilon)^{\sum_{t:c_t(j)<0} c_t(j)}$$

A more detailed review of the proof can be seen in the multiplicative weight update slides from 2019 (attached to the exercise on Blackboard). Note that slightly different notation is used there than in this year's compendium.