Ex. 2 Advanced algorithms

MCMF

26.2.10

26.2-10

In short: assume the flow is known, trace back how many augmenting paths we have from the source to the root.

At each time we can take out one edge that has the minimum capacity in the augmenting path.

26.2-10

long version:

Work backwards take flow away until it is 0 in the following way:

Pick an edge (u,v) such that f(u,v) is minimum, find a path using that edge, and reduce the flow on that path by f(u,v). By now you know that (u,v) will never be used for its flow is 0.

Since in each iteration an edge is discarded, the operation will take at most |E| times

Each path on which we reduced flow could have been an augmenting path if we started with 0, so we could get to our max flow with at most |E| augmentations.

26.3-2

26.3-2

We prove by induction on iterations of Ford-Fulkerson.

- Base case: No iterations performed yet. The flow is integral because it's 0.
- Inductive case: at the i'th iteration and all previous iterations resulted in integral flow. This implies all residual capacities are also integral (addition and subtraction with integers result in more integers).

26.3-2(cont.)

As a result, the augmenting path found in this iteration has an integral bottleneck, and the algorithm pushes an integral amount of flow on each edge in the augmenting path, thus keeping all flows integral.

Session 2

26.2-6 and 26.2-7

Preparing questions:

Demonstrate the case in which Edmonds Karp is preferable over General FF implementation

Is a general FF preferable in any circumstance?

26.2-6:

Let S be the set of all the sources and set T be the set of all the sinks.
Convert the problem of multiple sources and multiple sinks into a problem with single source, s and single sink t similar to how it is shown in figure 26.3. But make capacities of all the edges from super source s to the sources s_i, $C(s, s_i) = p_i$. Similarly make capacity of all edges from sinks t_i to the super sink t, $C(t_j, t) = q_j$. Find max flow, f' in the resulting flow network. If $|f'| = \sum_{s_i \in S} p_i = \sum_{t_j \in T} q_i$ then flow f with the additional constraint is f' otherwise no such flow is possible in the given network.

26.2-7

It is easy to see that f_p obeys the capacity constraints.For all edges not *on the* path, we have that $f_p(u,v) = 0 \leq cf(u,v)$, since we do not allow negative capacities. For all edges *on* the path, we have that

$$f_p(u,v) = c_f(p) \leq c_f(u,v) \tag{1}$$

For the flow conservation we see look at each vertex along the path. As the path is simple, no vertex is visited twice. Then each vertex only have a single input and a single output, and as they are all assigned the same value ($c_f(p)$). For vertices not on the path, all adjacent edges will have zero flow.

Finally we can note that $|f_p| = c_f(p)$ from the fact that the path is single and then all cuts must have the same value.

A

Let J = {J1, J2, ..., Jn} be the set of jobs, and T = {T1, T2, .., Tk} be slots available on a machine where these jobs can be performed. Each job J has a set of valid slots Sj ⊆ T when it can be scheduled. The constraint is that no two jobs can be scheduled at the same time. The problem is to find the largest set of jobs which can be scheduled.

Answer

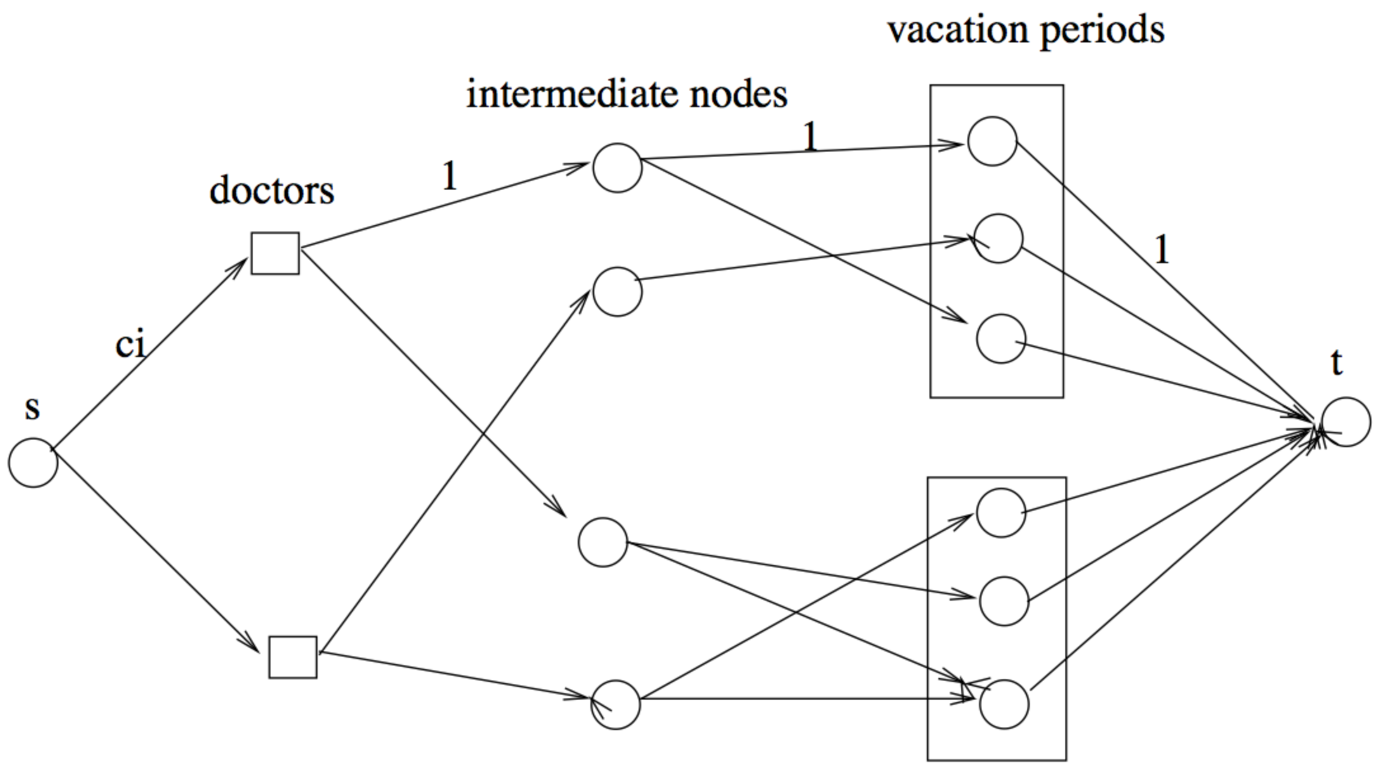
This problem can be solved by reducing it to a bipartite matching problem. For every job, create a node in X, and for every timeslot create a node in Y . For every time slot T in Sj, create an edge between J and T. The maximum matching of this bipartite graph is the maximum set of jobs that can be scheduled.

B

There are k vacation periods each spanning multiple contiguous days. Let D_j be the set of days included in the jth vacation period. There are n doctors in the hospital, each with a set of vacation days when he or she is available. We need to maximize the assignment of doctors to days under the following constraints:

1. Each doctor has a capacity c_i which is the maximum total number of days he or she can be scheduled.
2. For every vacation period, any given doctor is scheduled at most once.

Answer



26-1

Escape problem