2WO08: Graphs and Algorithms	**Lecture 4**	**Date: 26/2/2012**
Instructor: Nikhil Bansal	**The Circulation Problem**	**Scribe: Tom Slenders**

1 The basic circulation problem

We will consider the max-flow problem again, but this time there will be no source and sink. Each vertex v has a demand $d(v)$, which is more or less the flow that should enter that vertex. If the demand is negative that amount should leave the vertex. A bit more concrete we can write it like this:

Definition 1 $d(v) := \sum_{e \in N^-(v)} f(e) - \sum_{e \in N^+(v)} f(e)$

We can say that a vertex v is a semi-sink if $d(v) < 0$ and a semi-source if $d(v) > 0$.

Note that we can rewrite the original max-flow problem as a circulation problem as follows:

$$\begin{aligned} d(s) = &\quad D \quad \text{(this is our original source)} \\ d(t) = &\quad -D \quad \text{(this is our original sink)} \\ d(v) = &\quad 0 \quad \forall v \neq s, t \end{aligned}$$

Theorem 2 *Solving if a circulation problem of a graph G is feasible and is the same as solving a max-flow problem for a graph G'.*

Proof:

We note that a necessary condition for a flow to exist is:

$$\sum_{v:d(v)<0} d(v) = -\left(\sum_{v:d(v)>0} d(v) \right) =: D \qquad (1)$$

Now we add two extra vertices to the graph G: s and t. s is a source that has an outgoing edge to every vertex v that has $d(v) < 0$, also the edge has a capacity of $-d(v)$. t is a sink that has an incoming edge from every vertex v that has $d(v) > 0$, also the edge has a capacity of $d(v)$. For an example see Figure 1.

Claim 3 *A circulation problem is feasible if and only if the max-flow value is D*

We see that there is a cut separating s from all other vertices, this cut has capacity D. If all these edges are using full capacity, then also all edges going to t must have full capacity. Therefor all other vertices have their demand met.

This also concludes the proof of Theorem 2. □

We also want to solve the minimal cost for a circulation problem if edges have cost, also known as the minimum cost circulation problem.

Theorem 4 *If (1) is met and there is no cycle with infinite capacity edges s.t. wt(cycle)¡0 then there exists a polynomial algorithm that finds a solution for the minimum cost circulation problem.*

The proof was omitted during the lecture. For a proof see [2]

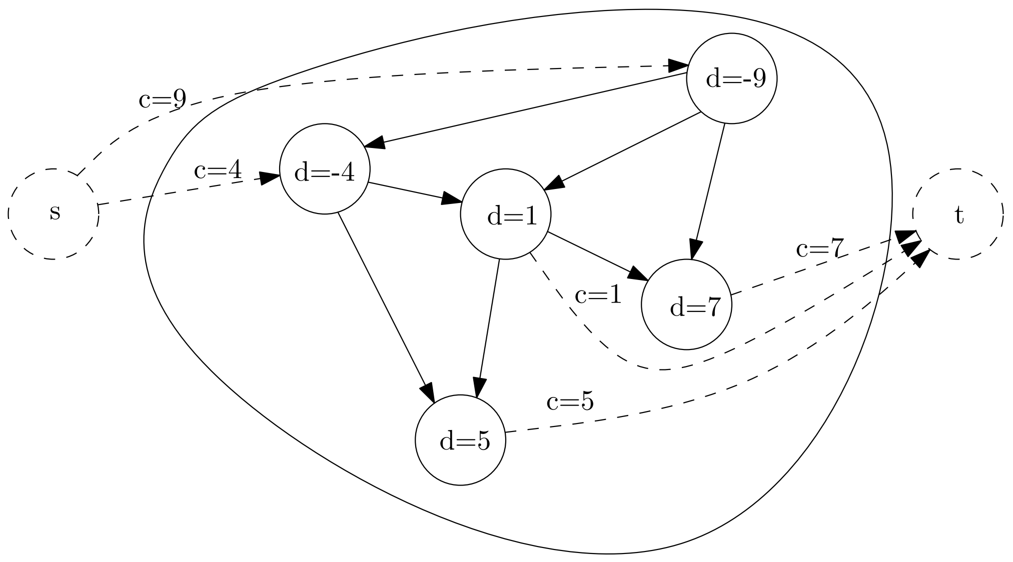


Figure 1: By adding the vertices s and t we can rewrite a circulation problem into a max-flow problem.

2 lower bounds

We will now add some extra restrictions to the circulation problem. For every edge we now introduce a lower bound on the circulation flow. For every edge e we have a lower bound $l(e)$ and the following must hold for every edge: $l(e) \leq f(e) \leq c(e)$. Note that we included the restriction for capacity as well.

Theorem 5 *Solving the circulation problem with lower bounds is feasible and is the same as solving the basic circulation problem.*

Proof: For every edge e with a lower bound going from vertex u to vertex v we do the following: We add a vertex u' and an edge going from u to u', also $d(u') = l(e)$. We add a vertex v' and an edge going from v' to v, also $d(v') = -l(e)$. We change the capacity of edge e to $c(e) - l(e)$. This is illustrated in figure 2.

Now we have reduced every edge to an edge without lower bound and if only if the demands in u' and v' can be met, the lower bound circulation problem can be solved. $\square$

2.1 Example: min-cost perfect matching

We want to create a perfect matching with minimal costs. We can reduce this to a circulation problem. We add a source s and sink t as in figure 3. The newly created edges all have capacity and lower bound 1. The edges in the original graph have capacity 1 and no lower bound.

2.2 Example: Rounding in a matrix

Consider a matrix A with real values. We denote the sum of each row to be $R_i := \sum_j (a_{ij})$ and the sum of each column to be $C_j := \sum_i (a_{ij})$. We want to round all values in the matrix, for each

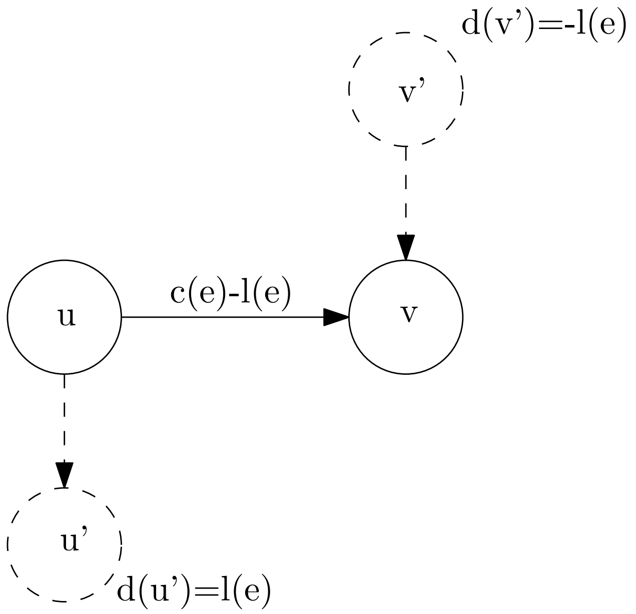


Figure 2: The change to every edge with a lower bound as in proof 5.

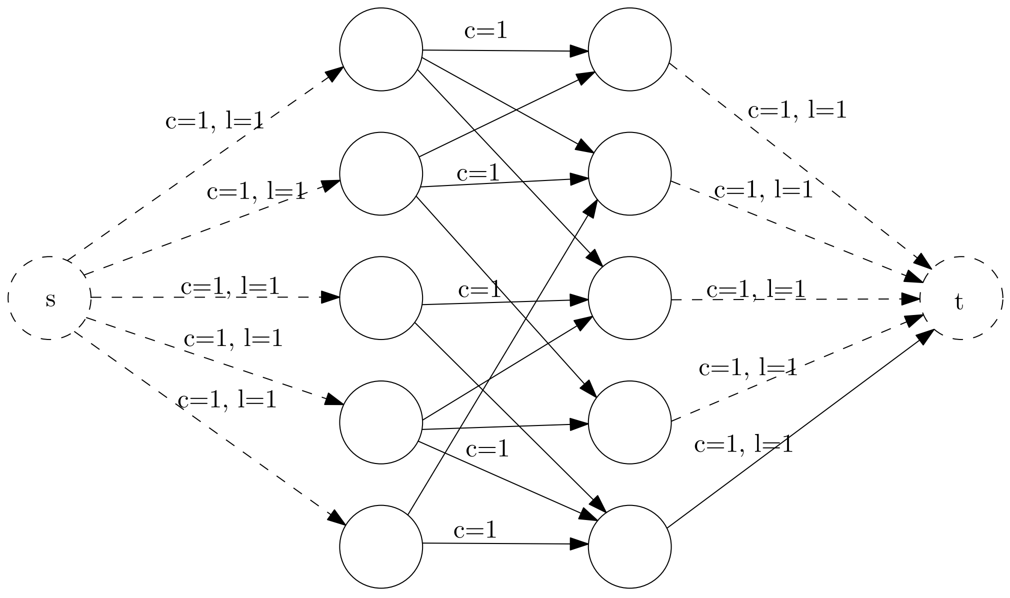
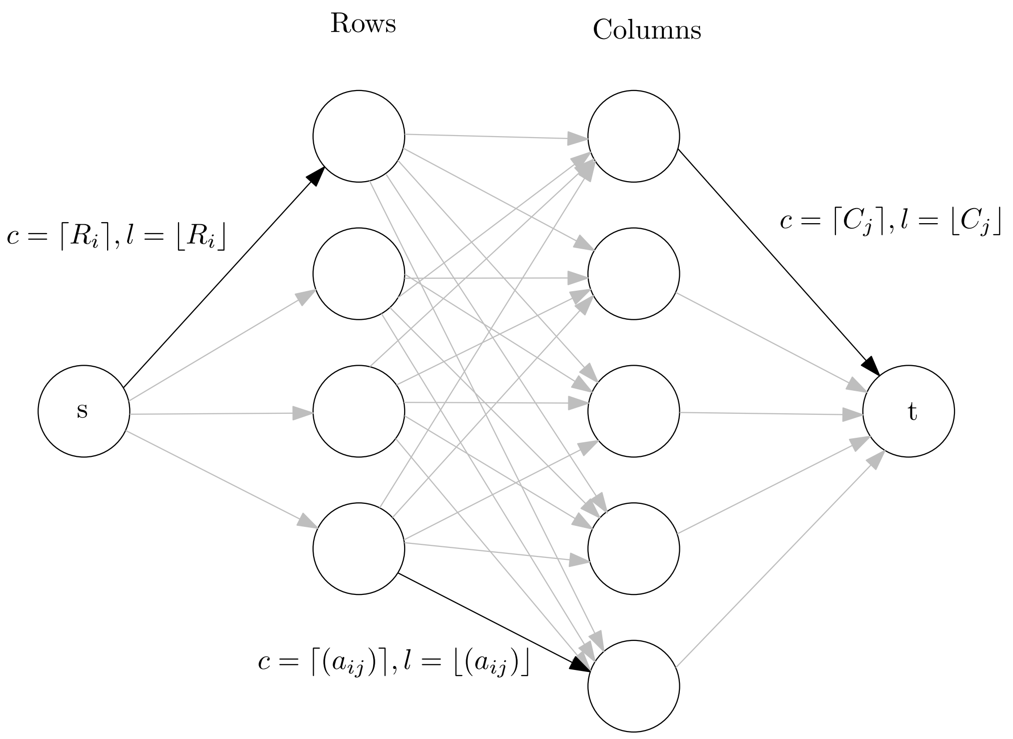


Figure 3: A representation of how to solve a min-cost perfect matching by solving a max-flow.



Rows
Columns
$c = \lceil R_i \rceil, l = \lfloor R_i \rfloor$
$c = \lceil C_j \rceil, l = \lfloor C_j \rfloor$
$c = \lceil (a_{ij}) \rceil, l = \lfloor (a_{ij}) \rfloor$
s
t

Figure 4: The graph created from the rounding in a matrix problem.

value we can choose to round it up or down. We denote these new values in the matrix B, so $(b_{ij}) = \lceil (a_{ij}) \rceil$ or $(b_{ij}) = \lfloor (a_{ij}) \rfloor$. Also we want to round all C_j and R_i up or down, such that for every j: $\sum_i (b_{ij}) = \lceil C_j \rceil$ or $\sum_i (b_{ij}) = \lfloor C_j \rfloor$ and also for every i: $\sum_j (b_{ij}) = \lceil R_i \rceil$ or $\sum_j (b_{ij}) = \lfloor R_i \rfloor$. We want to know if such a matrix B exists and also how to find one.

We create a graph as in figure 4. For every row we create a vertex, for every column we create a vertex and we create a source vertex and a sink vertex. Now we draw edges from the source to the row vertices, these edges have a capacity of $\lceil R_i \rceil$ and a lower bound of $\lfloor R_i \rfloor$. We also draw edges from the column vertices to the sink, these edges have a capacity of $\lceil C_j \rceil$ and a lower bound of $\lfloor C_j \rfloor$. Lastly we draw a vertex from every row vertex i to every column vertex j, these edges have a capacity of $\lceil (a_{ij}) \rceil$ and a lower bound of $\lfloor (a_{ij}) \rfloor$.

If we can solve this circulation problem, we have found a solution to our problem, since if we solve a circulation problem with only integer values, our circulation algorithm will return an integer only solution. We know that a non-integer solution exists, because we can simply fill in the original values of the (a_{ij}). This means we will always find an integer solution to the stated circulation problem.

3 Chinese Postman Tour

For the Chinese Postman Tour (CPT) we want to visit all edges of a partially directed graph Γ at least once by doing a tour. Every edge has a cost d and we want to minimize the total cost of all edges visited.

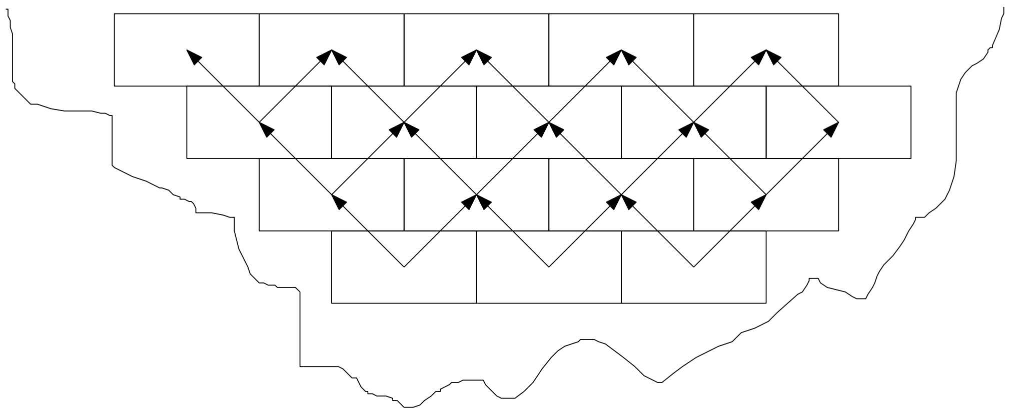


Figure 5: A representation of the edges for a dig site. The top layer must be dug first before the lower parts can be reached.

We first note that any solution is at least $\sum_{(i,j)\in E} d_{i,j}$, since we have to visit every edge at least once.

If the graph is Eulerian (this means $\forall v : d^+(v) = d^-(v)$ or in words: The in-degree is the same as the out-degree), we can simply find an Euler route and visit every edge exactly once. Finding this Euler route can be done by starting with some cycle and adding cycles of unvisited edges.

Solving this problem with both undirected and directed edges is known to be NP-complete, though with only undirected eges it can be solved in polynomial time, see [3][1].

4 Project management

In project management it is often necessary to distribute the way money or time is spent. To gain money or time one often has to invest to learn certain skills before one gets (hopefully more) money in return by doing a job that requires those skills.

We can model this problem with graphs as follows. Every vertex v represents a project that can be done and has an associated revenue $p(v)$, possibly negative. Every (directed) edge represents the prerequisites of a vertex, an incoming arrow means that vertex can only be done if the vertex from which the edge comes is also done.

For example we can look at a mining site where one can only dig for the lowest parts of the mine if the parts on top are dug first. The corresponding edges are shown in Figure 5. Some of the lower parts might be very valuable, but also need a lot of digging before they can be reached.

We wonder how we can maximize the profit for the Project management problem. Note that we may assume that there are no cycles in our graph G, since a cycle means that all vertices in the cycle must all be done or none. We could simply represent this as one vertex.

We create the following graph G': We use the same vertices as in G and every prerequisites edge e is added with $c(e) = \text{inf}$. We add two vertices: a source s with outgoing edges to all vertices v for which $p(v) \geq 0$, the capacity for each edge is $p(v)$, a sink t with incoming edges from all vertices v for which $p(v) < 0$, the capacity for each edge is $-p(v)$. Now we search for a min-cut, the vertices that are on the same side of the cut as s are the projects that should be done to maximize profit, we shall call this set S, the other vertices will be called T. See Figure 6.

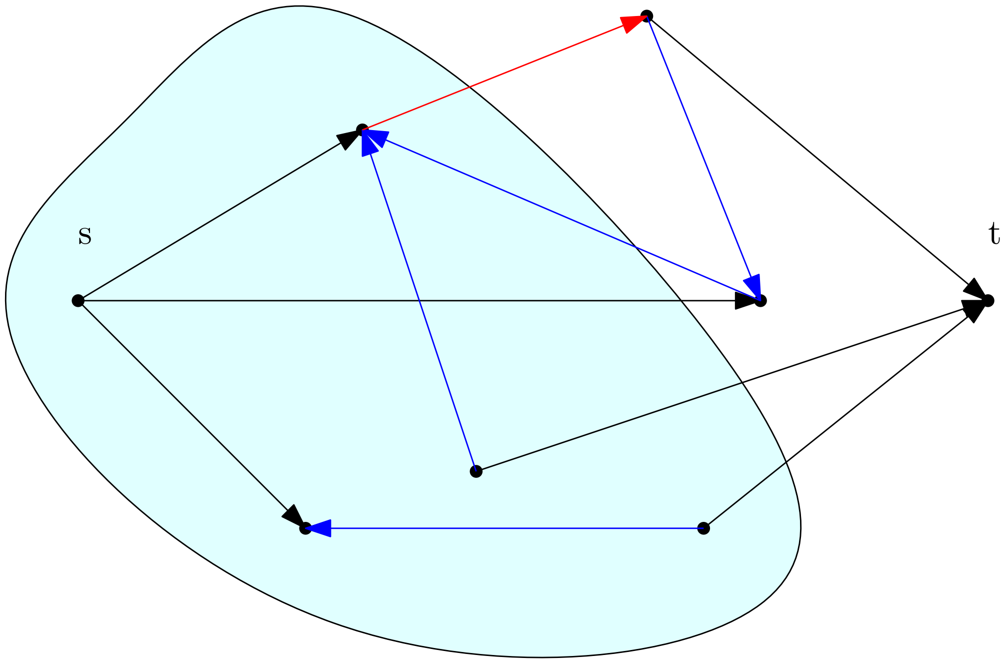


Figure 6: An example of a min-cut to find the lowest project costs. Note that the blue and red arrows are arrows with infinite capacity, but that only the blue arrows are allowed. The vertices in the light blue are are in set S, the others are in T.

We now denote $P(S)$ the positive profit that lies in S, $P(T)$ the positive profit that lies in T, $N(S)$ the negative profit in S and $N(T)$ the negative profit that lies in T. Now we see we want to solve $\max(P(S) - N(S))$. This also means we want to solve $\max(P(S) - N + N(T)$ with $N = N(T)+N(S)$ a constant. This means we want to solve $\max P(S)+N(T) = P-P(T)+N-N(S)$ with $P = P(S) + P(T)$ a constant. Now we want to solve $\min(P(T) + N(S))$ and we can see that this is actually the same as finding a min-cut for our graph G'. Hence finding a min-cut in G' results in an optimal choice S to maximize the total profit.

References

[1] Jack Edmonds & Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[2] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255, 1985.

[3] NIST. Chinese postman problem. `http://xlinux.nist.gov/dads//HTML/chinesePostman.html`, May 2013.