

The min cost flow problem

Course notes for Optimization

Spring 2007

Peter Bro Miltersen

February 7, 2007

Version 3.0

1 Definition of the min cost flow problem

We shall consider a generalization of the max flow problem known as the min cost flow problem.

A **flow network with costs** is a directed graph $G = (V, E, c, b, k)$ where V is a set of vertices, E is a set of directed edges (arcs), each arc $(u, v) \in E$ has a **capacity** $c(u, v) \in \mathbf{R}$ and each pair of vertices (u, v) has a **cost** $k(u, v) \in \mathbf{R}$ satisfying $k(u, v) = -k(v, u)$. By convention, $c(u, v) = 0$ for $(u, v) \notin E$. We allow positive as well as negative capacities. Each vertex $v \in V$ has a **balance** $b(v) \in \mathbf{R}$.

A *feasible flow* in G is a real valued function $f : V \times V \rightarrow \mathbf{R}$ satisfying

1. **Capacity constraints:** For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.
2. **Skew symmetry:** For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.
3. **Balance constraints:** For all $u \in V$, $\sum_{v \in V} f(u, v) = b(u)$.

The **cost** of a feasible flow f is defined to be

$$\text{cost}(f) = \frac{1}{2} \sum_{(u,v) \in V \times V} k(u, v) f(u, v)$$

Intuitively, we are charged $k(u, v)$ monetary units for each unit of flow carried by the arc (u, v) . Since $k(u, v) = -k(v, u)$ and $f(u, v) = -f(v, u)$ the terms $k(u, v)f(u, v)$ and $k(v, u)f(v, u)$ are identical. We multiply the sum by $\frac{1}{2}$ in order to get only one of them.

The min cost flow problem is the following: Given a flow network with costs, find the feasible flow f in G that minimizes $\text{cost}(f)$ among all feasible flows f in G . If no feasible flow in G exists, then report this.

2 The max flow problem vs. the min cost flow problem

Comparing the min cost flow problem to the max flow problem we see the following differences:

- We now allow a negative capacity $c(u, v) < 0$ on an arc (u, v) . Note that if we have such a negative capacity and a feasible flow f in G , skew symmetry and the capacity constraint on (u, v) dictates that $f(v, u) = -f(u, v) \geq -c(u, v) > 0$. Thus, we may interpret a negative capacity as putting a non-trivial **lower bound** $l(v, u) = -c(u, v)$ on the flow carried by the opposite arc (v, u) , i.e., a requirement that (v, u) carries at least this amount of flow.

This interpretation is in fact the motivation for allowing negative capacities.

- In the max flow problem, $\sum_{v \in V} f(u, v)$ was required to be zero for all $u \in V$, except the two special vertices s and t . Now there are no special vertices and rather than demanding $\sum_{v \in V} f(u, v)$ to be zero, we are allowed to demand it to be any particular value $b(u)$. We may think of a vertex u with $b(u) > 0$ as a *producer* of $b(u)$ units of flow and a vertex u with $b(u) < 0$ as a *consumer* of $-b(u)$ units of flow. A vertex u with $b(u) = 0$ neither produces nor consumes flow, but, as in the max flow case, has to send out exactly the flow it receives.
- Because of the above, it is no longer clear that a given flow network with costs has *any* feasible flow: For instance, the zero flow could fail to satisfy the capacity constraints as well as the balance constraints. It is fairly clear that a *necessary* condition for a feasible flow to exist is that $\sum_u b(u) = 0$ (Exercise 1).
- Rather than trying to maximize the *value* of the flow we are looking for, we seek to minimize its *cost*.

Given these differences, we seem to have a totally *different* problem. But let us first convince ourselves that we in fact have a *more general* problem, i.e., that the max flow problem may be directly phrased as a min cost flow problem. This also serves well as our first “programming exercise” using the min cost flow formalism.

Given an instance of the max flow problem $G = (V, E, c, s, t)$, consider the min cost flow instance $G' = (V', E', c', b, k)$ where

1. $V' = V \cup \{z\}$,
2. $E' = E \cup \{(t, z), (z, s)\}$,
3. $c'(u, v) = c(u, v)$ for all $u, v \in V$. We let $c'(t, z) = c'(z, s) = c(s, V)$. All other capacities are 0.

4. $b(u) = 0$ for all $u \in V'$.
5. $k(u, v) = 0$ for $\{u, v\} \neq \{t, z\}$ and $k(t, z) = -1, k(z, t) = 1$.

Claim 1 1. All feasible flows in G' have cost less than or equal to zero.

2. If there is a flow f in G with value r , then there is a feasible flow f' in G' with cost $-r$.

3. Conversely, if there is a feasible flow f' in G' with cost $-r$, then there is a flow f in G with value r and we may easily compute f from f' .

Proof Exercise 2.

Thus, given an algorithm for solving the min cost flow problem, we may easily use it to derive an algorithm for finding max flows and we can thus regard the min cost flow problem as a more general formalism in which to phrase optimization problems and instances.

The flow network G' above has $b(v) = 0$ for all v . Such networks form an important special case and we shall refer to them as *circulation networks*. We shall also refer to a feasible flow in such a network as a *feasible circulation*.

3 A local search algorithm for finding min cost flows

We shall describe Klein's algorithm for finding a min cost feasible flow in a flow network with costs. As the Ford-Fulkerson method for the max flow problem, Klein's algorithm is an example of an algorithm following the *local search pattern*, as sketched in figure 1. To make the pattern concrete for the case at hand, we have to specify how to find the first feasible solution and design a neighborhood relation, i.e., specify how we are allowed to improve one feasible solution to obtain a new one.

3.1 Finding the first feasible solution

We shall show that the problem of determining whether a feasible flow exists in a given flow network with negative capacities allowed can be reduced to the max flow problem and hence has an efficient solution.

```

LocalSearch(ProblemInstance  $x$ )
   $y :=$  feasible solution to  $x$ ;
  while( $\exists z \in N(y) : z$  is better than  $y$ ) {
     $y := z$ 
  }
  return  $y$ 

```

Figure 1: Local Search Pattern, $N(y)$ is the neighborhood of y .

Let $G = (V, E, c, b, k)$ be a flow network. We want to find some feasible flow in G . As we are not yet trying to minimize the cost of the feasible flow we find, we can ignore the costs $k(u, v)$ completely in this subsection. Also, as Exercise 1 tells us that a feasible flow cannot exist unless $\sum_u b(u) = 0$, we shall assume without loss of generality that this is the case.

We shall define a max flow instance $G' = (V', E', c', s, t)$ from G and argue that a maximum flow in G' of a certain value can be used to derive a feasible flow in G while a maximum flow below this value indicates that no such feasible flow exists.

We first include in V' all the vertices in V and two new vertices s and t . Further vertices will be included below.

For each $u \in V$ with $b(u) > 0$, we include in E' an arc (s, u) with capacity $b(u)$. For each $u \in V$ with $b(u) < 0$, we include in E' an arc (u, t) with capacity $-b(u)$.

We now examine each unordered pair of vertices $\{u, v\}$ from V . For each such pair, there are the following cases:

1. $c(u, v) \geq 0$ and $c(v, u) \geq 0$. In this case, we include in E' an arc from u to v with capacity $c(u, v)$ (unless $c(u, v) = 0$ in which case we make no arc) and an arc from v to u with capacity $c(v, u)$ (unless $c(v, u) = 0$ in which case we make no arc), i.e., we copy the arcs between u and v “verbatim” from G to G' .
2. One of (u, v) and (v, u) , and without loss of generality (v, u) , has a negative capacity $c(v, u) < 0$. Define $l(u, v) = -c(v, u)$. There are two subcases:
 - (a) $l(u, v) > c(u, v)$. In this case we immediately conclude that no flow on the arc $c(u, v)$ can satisfy the capacity constraints on (u, v) and

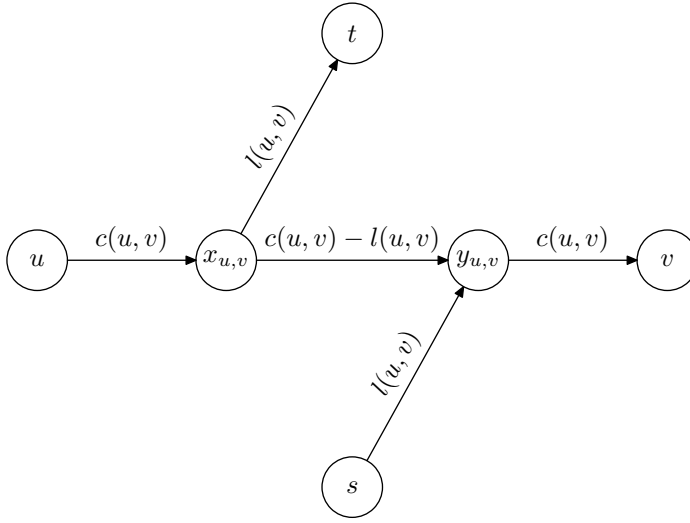


Figure 2: The part of G' corresponding to an arc (u, v) with $l(u, v) \leq c(u, v)$

(v, u) as well as skew symmetry. We thus abort the construction of G' and conclude that G does not have a feasible flow.

(b) $l(u, v) \leq c(u, v)$. In this case we add two new vertices $x_{u,v}$ and $y_{u,v}$ to V' and include five new arcs in E' (see Figure¹ 2)

- An arc between u and $x_{u,v}$ with capacity $c(u, v)$,
- An arc between $x_{u,v}$ and $y_{u,v}$ with capacity $c(u, v) - l(u, v)$
- An arc between $y_{u,v}$ and v with capacity $c(u, v)$,
- An arc between s and $y_{u,v}$ with capacity $l(u, v)$,
- An arc between $x_{u,v}$ and t with capacity $l(u, v)$

Having done the above for all unordered pairs u, v from V , our construction of G' is complete. Now, the following claim proves the usefulness of our construction.

Claim 2 G has a feasible flow if and only if the maximum flow f^* in G' saturates all arcs out of s , i.e., if and only if $|f^*| = c'(s, V)$.

Furthermore, if f^* is a flow in G' that saturates the arcs out of s , the flow f defined as follows is a feasible flow in G :

- $f(u, v) = f^*(u, v)$ for vertices $\{u, v\}$ as in case 1 above,

¹Thanks to Troels Bjerre Sørensen for preparing the figure

- $f(u, v) = f^*(u, x_{u,v})$ for vertices $\{u, v\}$ as in case 2(b) above.

The proof of Claim 2 is left as Exercise 3.

By Claim 2, we can use a max flow algorithm to find the first feasible flow in our local search algorithm for the min cost flow problem - it will either find a feasible flow or establish there is no feasible flow.

3.2 Neighborhood design

Given a feasible flow in our network, we have to find a way of improving its cost by changing it slightly, i.e., we have to design a neighborhood relation for our local search algorithm. We seek inspiration in the Ford-Fulkerson algorithm for max flow, where the notion of *the residual network* proved very useful. In the min cost case, the definition of a residual network is completely analogous.

Let $G = (V, E, c, b, k)$ be a flow network with costs. The **residual network** $G_f = (V, E_f, c_f, b_f, k)$ is defined by

- $E_f = \{(u, v) \in V \times V \mid c(u, v) - f(u, v) > 0\}$
- $c_f(u, v) = c(u, v) - f(u, v)$ for all u, v . Hence all capacities G_f are non-negative.
- $b_f(u) = 0$ for all u . Hence G_f is a circulation network.

The following lemmas are very useful and easy to prove - we already saw analogous lemmas in the max flow case.

Lemma 3 *If f is a feasible flow in G and f' is a feasible circulation in G_f then $f + f'$ is a feasible flow in G with $\text{cost}(f + f') = \text{cost}(f) + \text{cost}(f')$.*

Lemma 4 *If f, f' are feasible flows in G then $f' - f$ is a feasible circulation in G_f .*

In the Ford-Fulkerson method for the max flow problem, one step of the local search was done by finding an *augmenting path* in the residual network and adding the corresponding path flow to the flow at hand. For the min cost flow problem, the notion analogous to augmenting paths and path flows are *augmenting cycles* and *cycle flows*.

```

MinCostFlow( $G = (V, E, c, b, k)$ )
  Using a max flow algorithm, find a feasible flow  $f$  in  $G$ 
  (if no such flow exists, abort)
  while( $\exists$  an augmenting cycle  $C = (u_1, \dots, u_r)$  in  $G_f$ ) {
     $\delta := \min\{c_f(u_i, u_{i+1}) \mid i = 1, \dots, r - 1\}$ 
     $f := f + \gamma_C^\delta$ 
  }
  return  $f$ .

```

Figure 3: Klein's algorithm for finding a min cost flow.

Let $C = (u_1, u_2, \dots, u_r)$ in V^r be a sequence of vertices with $u_r = u_1$ but all other u_i 's being different. Note that if we for a digraph $G = (V, E)$ have that $(u_i, u_{i+1}) \in E$ for all i , then C is a **simple cycle** in G .

The **cycle flow** γ_C^δ is the circulation defined by

1. $\gamma_C^\delta(u_i, u_{i+1}) = \delta, i = 1, \dots, r - 1,$
2. $\gamma_C^\delta(u_{i+1}, u_i) = -\delta, i = 1, \dots, r - 1,$
3. $\gamma_C^\delta(v, w) = 0$ for all other (v, w) .

If G is a circulation network (i.e., $b \equiv 0$), a cycle flow f_C^δ may or may not be a feasible circulation in G , depending on whether or not it satisfies the capacity constraints. The skew symmetry and balance constraints are clearly satisfied.

Let $G = (V, E, c, b, k)$ be a flow network, let f be a feasible flow in G and let G_f be the residual network. An **augmenting cycle** in G_f is a simple cycle $C = (u_1, u_2, \dots, u_r)$ in G_f ($u_1 = u_r$ and $(u_i, u_{i+1}) \in E_f$) with the property that $\sum_{i=1}^{r-1} k(u_i, u_{i+1}) < 0$.

We are now ready to state Klein's algorithm for finding a min cost flow. As seen in Figure 3, the algorithm works by repeatedly improving the flow at hand by adding an augmenting cycle flow. Lemma 3 and the definition of an augmenting cycle ensures that f is always a feasible flow in G during the execution of Klein's algorithm. Also, the cost of f will decrease in each iteration of the while loop, as the cost of the cycle flow added to f is

$$\delta \sum_{i=1}^{r-1} k(u_i, u_{i+1}) < 0.$$

We now have to argue that

1. the algorithm is partially correct, i.e., that it produces a minimum cost flow in case it terminates and that
2. it indeed terminates.

3.3 Partial correctness

We shall use the following lemma which is also of general interest.

Lemma 5 (circulation decomposition lemma) *Let $G = (V, E, c, b, k)$ be a circulation network (i.e., $b \equiv 0$) with no negative capacities (for instance, G may be a residual network). Let f be a feasible circulation in G . Then, f may be written as a sum of cycle flows*

$$f = \gamma_{C_1}^{\delta_1} + \gamma_{C_2}^{\delta_2} + \cdots + \gamma_{C_m}^{\delta_m}$$

where each cycle flow $\gamma_{C_i}^{\delta_i}$ is also a feasible circulation in G , i.e., the capacity constraints are not violated.

Proof The proof is an induction in the number of pairs (u, v) for which $f(u, v) > 0$.

Induction Base If the number of pairs (u, v) for which $f(u, v) > 0$ is zero, i.e., $f(u, v) \leq 0$ for all (u, v) , by skew symmetry we have that in fact $f(u, v) = 0$ for all (u, v) so f is the zero flow. The statement is vacuously true for that flow.

Induction Step. Assume f is not the zero flow but that the statement to be proven holds for all feasible circulations f' having strictly fewer arcs carrying a positive flow than the circulation f . We shall show that it also holds for f .

We shall use f to locate a cycle in G by taking a walk in G .

Since f is not the zero flow, there are vertices u_0 and v , so that $f(u_0, v) \neq 0$. Pick such a u_0 . The start of our walk is u_0 . Since $\sum_w f(u_0, w) = 0$, there must be at least one vertex u_1 so that $f(u_0, u_1)$ is in fact strictly bigger than 0. The next vertex in our walk is u_1 . As $f(u_1, u_0) = -f(u_0, u_1) < 0$ and $\sum_w f(u_1, w) = 0$, there must be a vertex u_2 so that $f(u_1, u_2) > 0$. The next vertex in our walk is u_2 . We thus continue our walk, obtaining u_3, u_4, \dots . We stop when we encounter a previously seen vertex, i.e., at the minimum value j so that $u_j = u_{j-i}$ for some value $i > 0$.

Let δ be the smallest value among the values $f(u_{j-h}, u_{j-h+1})$, $h = 1, 2, \dots, i$. Then, γ_C^δ is a cycle flow which is also feasible circulation in G .

Let $f' = f - \gamma_C^\delta$. As G has no negative capacities, f' is a feasible circulation in G . Also, it has at least one more arc than f *not* carrying a positive flow, namely the arc (u_{k+i}, u_{k+i+1}) for which $f(u, v) = \delta$. Hence the induction hypothesis may be applied to f' and we conclude that f' is a sum of feasible cycle flows that are also feasible circulations in G . As $f = f' + \gamma_C^\delta$, we have established that the same holds for f .

We can now establish the partial correctness of Klein's algorithm:

Lemma 6 *Let G be a flow network with costs and let f be a feasible flow in G . If f is not a minimum cost flow, then there is an augmenting cycle in G_f .*

Proof Let G be fixed and let f^* be the optimal (i.e., minimum cost) feasible flow in G .

By Lemma 4, $g = f^* - f$ is a circulation in the residual network G_f . By the same lemma, $\text{cost}(g) = \text{cost}(f^*) - \text{cost}(f)$. Since f is not a minimum cost flow, this value is strictly negative.

By Lemma 5, g can be written as a sum of cycle flows, each being a feasible cycle flow in G_f . Since the cost of g is the sum of the costs of the cycle flows, and the cost of g is strictly negative, one of the cycle flows must also have a strictly negative cost. But the cycle corresponding to this cycle flow is then, by definition, an augmenting cycle.

3.4 Termination

As in the case of Ford-Fulkerson, we shall only prove termination for the case of integer inputs. Indeed, for general reals, Klein's algorithm may not terminate unless care is taken on how to choose the augmenting cycles (Exercise 6).

If all capacities are integers and an implementation of Ford-Fulkerson is used to find the first feasible flow (which is therefore integer valued), we note that all flows along all arcs in f and all residual capacities in G_f will remain integer valued throughout the algorithm. This means that for each pair of vertices (u, v) , there is only a finite number of possibilities for the value of $f(u, v)$ as it will be an integer between $-c(v, u)$ and $c(u, v)$. Thus, there are also only finitely many possibilities for the entire feasible flow f . As the cost

of the feasible flow f decreases in each iteration of the while loop, we will not see at particular f more than once during the executing of the algorithm. Thus, the algorithm terminates. At the same time we immediately have the following important integrality theorem.

Theorem 7 (Integrality theorem for min cost flow) *If a flow network has capacities and balances which are all integer valued and there exists some feasible flow in the network, then there is a minimum cost feasible flow with an integer valued flow on every arc. Furthermore, Klein's algorithm finds such a feasible flow.*

3.5 Complexity

To estimate the complexity of Klein's algorithm, we should first determine the complexity of performing a single iteration of the local search, i.e., a single iteration of the while-loop. The dominating task is to determine if the residual graph contains an augmenting cycle and to find one if it does. That this can be done efficiently is left as Exercise 7.

Then, we should determine how many steps the local search may take before terminating. As stated above, in the case of non-integer inputs, we are not certain of termination and even in the case of integer input, the algorithm may take exponential time (Exercise 6). In this way, the algorithm is very similar to the Ford-Fulkerson method for the max flow problem.

As in that case, it *is* possible to obtain a polynomial time algorithm for min cost flow by specifying more precisely *which* augmenting cycle to pick in case there is more one. The resulting analysis is somewhat more complicated than the analysis of the analogous Edmonds-Karp algorithm for max flow and is therefore not included in these notes. In practice, Klein's algorithm tend to terminate in fairly few iterations, even if no special care is taken about how to choose the augmenting cycle.

However, we note for future use that a polynomial time version of Klein's algorithm for the min cost flow problem indeed does exist.

4 Applications of min cost flow

Ahuja *et al* contains many examples of expressing natural optimization problems and instances in the min cost flow formalism. When reading their examples one should note that their formalism is slightly different from ours: While

we have been working with *net flows* (i.e., always assuming flows satisfying the skew symmetry constraint), this is not the case for Ahuja *et al.* Thus, when Ahuja *et al.* describe a flow x , we should interpret it, in our formalism, as the corresponding net flow f defined by $f(u, v) := x(u, v) - x(v, u)$, and thus obeying skew symmetry. In most of the examples of Ahuja *et al.*, the translation will be trivial as either $x(v, u)$ or $x(u, v)$ can be seen in advance to be zero. Also, rather than putting a negative capacity on the arc (v, u) , the formalism of Ahuja *et al.* allows putting a positive lower bound on the arc (u, v) , but as we already discussed, these two alternatives are equivalent. Thus, in most cases, the reader should have no difficulty translating the examples of Ahuja *et al.* into the formulation of the min cost flow problem of these notes. The only difficulty arises when it is not clear in advance in which direction the flow will be along a given arc (u, v) . Solving this difficulty is the subject of Exercise 9.

Here we shall mention one additional example that will be of importance to us later in the course, namely the problem of finding a *minimum weight perfect matching* in a complete bipartite graph.

This is the following problem: Let a **weight matrix** $W = (w(u, v))_{u, v \in \{1, 2, \dots, n\}}$ be given, representing the weights of the arcs of a complete bipartite graph with two times n vertices. We wish to find a perfect matching in this graph of minimum total weight, i.e., a permutation π on $\{1, 2, \dots, n\}$ minimizing $\sum_{i=1}^n w(i, \pi(i))$.

We formalize this as a flow network G with costs as follows: We let $V = \{l_1, l_2, \dots, l_n, r_1, r_2, r_3, \dots, r_n\}$. For each (i, j) , we put an arc between l_i and r_j with capacity 1 and cost $w(i, j)$. The balance of each vertex l_i we put to 1 and the balance of each vertex r_j we put to -1 .

Claim 8 *If W has a perfect matching of total weight v , then G has a feasible flow of cost v . Conversely, if G has a feasible flow of cost v where the flow through every arc is integer valued, then W has a perfect matching of total weight v . Furthermore, given this flow, we can easily compute the matching.*

The proof of the claim is left as Exercise 8. Combining Claim 8 with Theorem 7 gives us that we may use any implementation of Klein's algorithm to compute minimum weight perfect matchings, and we conclude (using the efficient version of Klein's algorithm alluded to in Section 3.5) that this problem has an efficient algorithm.

4.1 Exercises

Exercise 1 Let $G = (V, E, c, b, k)$ be a flow network with costs. Show that a feasible flow in G cannot possibly exist, unless $\sum_u b(u) = 0$.

Exercise 2 Prove Claim 1.

Exercise 3 Prove Claim 2.

Exercise 4 Prove Lemma 3.

Exercise 5 Prove Lemma 4.

Exercise 6 Show that Klein's algorithm may fail to terminate on certain inputs with non-integer capacities and costs. Also show that it may run in exponential time with integer capacities and costs. **Hint:** Use the analogous examples for the Ford-Fulkerson algorithm and the reduction from the max flow problem to the min cost flow problem of Section 2.

Exercise 7 We consider the algorithm in Figure 4 for determining whether a negative cost cycle exists in a given network. The algorithm is supposed to work only if the network is strongly connected (i.e., when there are paths using arcs in E , between any pair of vertices). If this is not the case, we can split the network into strongly connected components and deal with each component separately.

Prove that the method correctly determines if the strongly connected network G has a negative cost cycle. Show how to augment the method so that a negative cycle is returned if one exists. What is the complexity of the method?

Exercise 8 Prove Claim 8.

Exercise 9 Show how to translate an instance of the min cost flow problem as defined by Ahuja et al, pages 296-297, into an instance of the min cost flow problem as defined by these notes. Note that the main difference is that these notes operate with net flows and thus demand costs and flows to be skew symmetric while this is not the case in Ahuja et al. **Hint:** Replace each node v in the network with two nodes v_{in} and v_{out} connected by an arc with infinite capacity and zero cost.

```

ExistsNegativeCycle( $G = (V, E, k)$ )
  pick arbitrary vertex  $s$  in  $G$ ;
  let  $d(0, s) = 0$ ;
  for  $v \in V - \{s\}$ , let  $d(0, v) = \infty$ ;
  for  $i := 1$  to  $n$  do
    Invariant:  $d(i - 1, v)$  is the cost of the cheapest path from  $s$  to  $v$  using  $< i$  arcs.
    for  $v \in V$ , let  $d(i, v) = \min\{d(i - 1, v), \min_{(u,v) \in E} (d(i - 1, u) + k(u, v))\}$ 
  od;
  if  $(\exists v : d(n, v) < d(n - 1, v))$  return true;
  return false;

```

Figure 4: Negative cycle detection

Exercise 10 Let $G = (V, E)$ be a directed graph and $d : E \rightarrow \mathbf{R}^+$ a measure of the length of the arcs of G . Let s and t be two distinguished vertices in G . Formulate the problem of finding the shortest path from s to t (with respect to the distance measure d) as a min cost flow problem, using exercise 9.