Design and Analysis of Computer Algorithms
CS 6363.005: Homework #4

Due on Monday November 28, 2016 at 11:59pm

Instructor: Benjamin Raichel

Hanlin He (hxh160630)
hanlin.he@utdallas.edu

Contents

Problem 1 Running the Algorithms

Part (a) Dijkstra's Algorithm

Active	Null	s	a	d	b	c	e (Final)
s	0						0
a	∞	3					3
b	∞		7				7
c	∞		12	10	9		9
d	∞	6	5				5
e	∞			12	11	10	10

Part (b) Bellman-Ford Algorithm

Start from the edge with the smallest weight up to the edge with the largest weight. The ordering is as follow:

1. $c \to e$;
2. $a \to d$ and $b \to c$;
3. $s \to a$;
4. $a \to b$.

Problem 2 SSSP

Given a directed graph $G = (V, E)$, with positive edge weights and a single source shortest tree from vertex s.

Based on definition, for every vertex $v \in V$,

- $dist(v) = pred(v) + w(pred(v) \to v)$;
- if $v \to w \in E$, then $dist(w) \leq dist(v) + w(v \to w)$.

Otherwise, the SSSP tree is wrong.

Thus, we can traverse the graph, at each vertex, check these values to verify the correctness. The algorithm is shown in algorithm 1 applying DFS.

Problem 2 continued on next page...

Algorithm 1 Algorithm to Check Correctness of SSSP

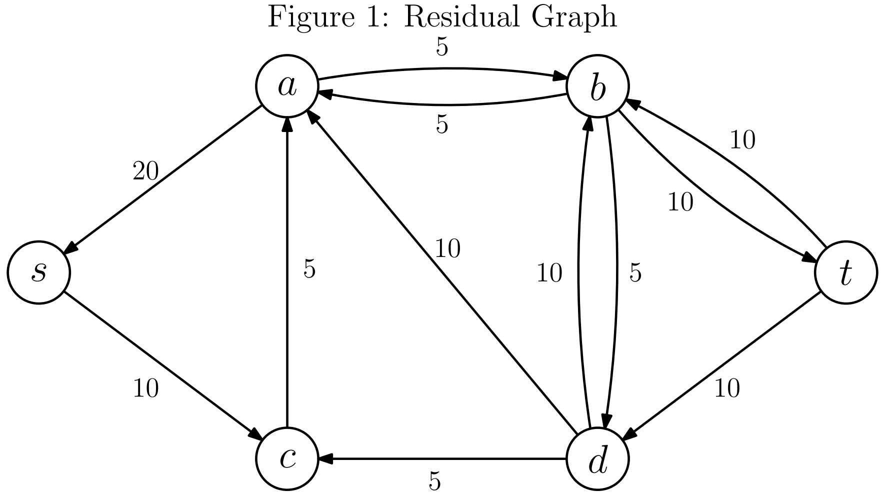
```
procedure CheckSSSP(v)
    Mark v.
    for each edge v → w. do
        if dist(v) ≠ pred(v) + w(pred(v) → v) then
            return False
        end if
        if dist(w) > dist(v) + w(v → w) then
            return False
        else if dist(w) = dist(v) + w(v → w) then
            if dist(w) ≠ v then
                return False
            end if
        end if
        if w is unmarked. then
            return CheckSSSP (w)
        end if
    end for
end procedure
```

Problem 3 Computing Flows and Cuts

Part (a) Residual Graph

The residual graph of the flow network is shown in fig. 1.

Figure 1: Residual Graph



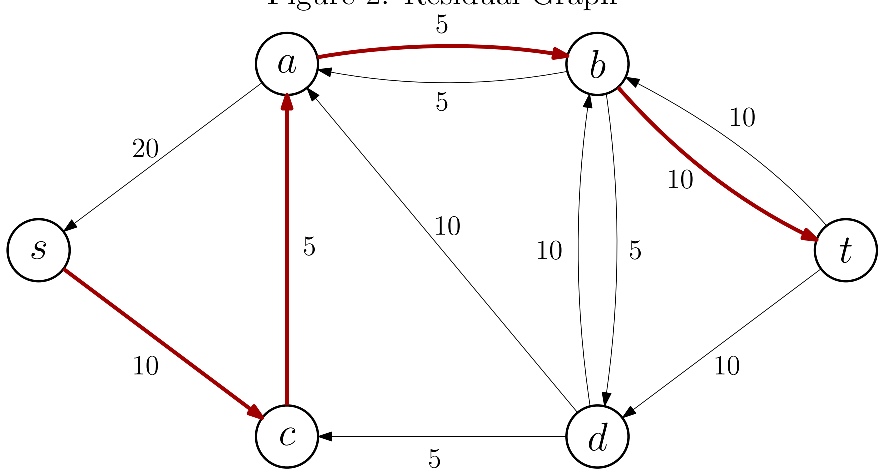
Part (b) Augmenting Path

From the residual graph we know that,

$$s \longrightarrow c \longrightarrow a \longrightarrow b \longrightarrow t$$

is an augmenting path, as shown in fig. 2.

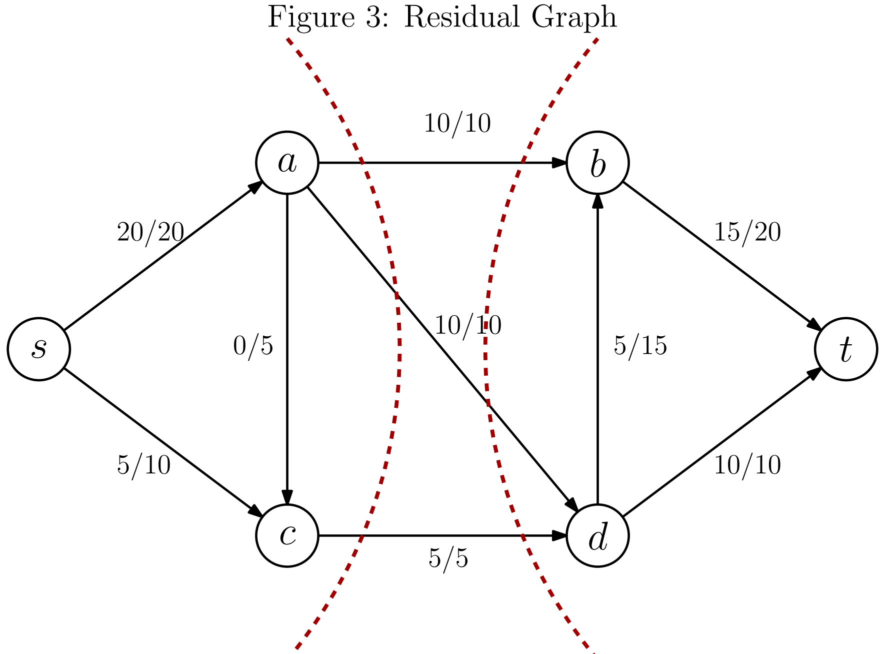
Figure 2: Residual Graph



Part (c) Minimum Cut

A minimum $s-t$ cut is $\{s, a, c\}$ and $\{b, d, t\}$, as shown in fig. 3.

Figure 3: Residual Graph



Problem 4 Flow and Cut Problems

Part (a) Class Scheduling

First, let S, the set of student, be the source of the graph. Turn each student's interest classes $C(x_i)$ into a node, and turn each class C_i into a node. Let T denote the target, the sum of the enrolled total in each class.

Then,

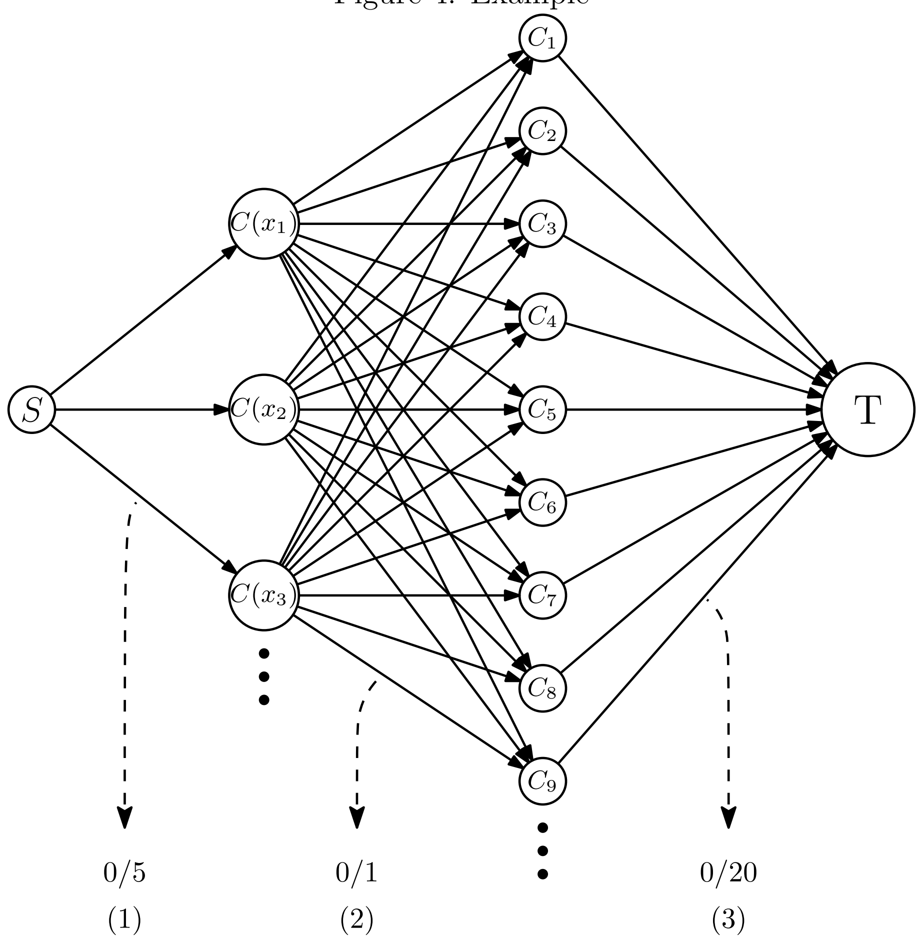
(1) Add a directed edge from S to each $C(x_i)$ with capacity of 5, representing a student can choose at most 5 classes.

(2) Add a directed edge from each $C(x_i)$ to each C_i contained in $C(x_I)$ with capacity of 1, representing a student can register for specific class in the interest list at most once.

(3) Add a directed edge from each C_i to T with capacity of 20, representing a class can have at most 20 enrolled students.

Compute the maximum flow from S to T would be the maximum class enrollment. The maximum flow graph is shown in fig. 4.

Figure 4: Example



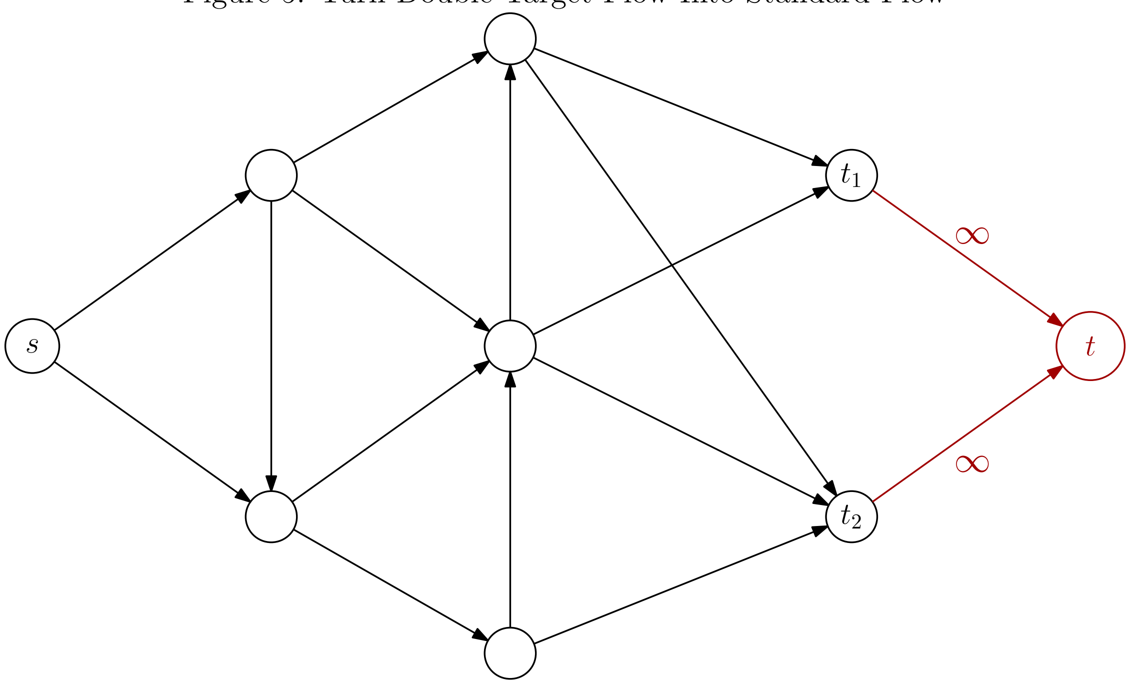
Part (b) Double Target Flow

Add a third target node in G, with two directed edges of infinite capacity from t_1 and t_2 to t, meaning $c(t_1 \to t) = \infty$, $c(t_2 \to t) = \infty$. Then, the double target flow network is turned into a single target flow network, where t is the target.

It is obvious that the valid flow that maximizing the net flow leaving s is the same in both graph. Hence, we can solve the problem using a standard flow algorithm.

The graph is shown in fig. 5.

Figure 5: Turn Double Target Flow Into Standard Flow



Let StandardMaxFlow (G, s, t) be the given algorithm to compute the max flow in any standard flow. The algorithm is shown in algorithm 2.

Algorithm 2 Algorithm to Solve Double Target Maximum Flow

procedure DoubleMaxFlow(G, s, t_1, t_2)
    Add vertex t to G, namely G'. Add edges from t_1, t_2 to t in G'
    $c(t_1 \to t) = \infty$
    $c(t_2 \to t) = \infty$
    return StandardMaxFlow (G', s, t)
end procedure

Part (c) Unique Cuts

The steps of the algorithm can be described as follow, let f^* be a maximum flow of $s - t$.

1. Compute the residual graph of f^*, denoted by $G_f = (V, E_f)$.
2. Traverse G_f from the source, we can find a set of vertices reachable from source. Let S be that set.
3. Let $T = V \setminus S$. Reverse all the edges in E_f which has endpoint in T, i.e. for all $v \to w \in E_f$ that $v, w \in T$, replace the edge with $w \to v$. Let $G' = (T, E'_f)$ be the new graph.
4. Traverse T, find the set of all the vertices which can be reached from t. Let T' be the set.
5. If $S \cap T' = V$, there is a unique minimum cut. Otherwise, there is no unique minimum cut in G.

Running Time Analysis

1. Computing residual graph takes $\mathcal{O}(|E|)$ time.
2. Traversing G_f to get S takes $\mathcal{O}(|V|)$ time.
3. Reversing the edges in G_f takes $\mathcal{O}(|V|)$ time.
4. Traversing G' to get T' takes $\mathcal{O}(|E|)$ time.

In total, the algorithm takes $\mathcal{O}(|V| + |E|)$ time.

Problem 5 NP-completeness

Part (a) Knapsack Problem

$\mathcal{O}(nb)$ time does not imply P=NP, since b is not polynomial in the length of the input to the problem.[1]

Part (b) Independent-Set

Use the idea of binary search.

Let Independent-Set (G, k) return True if exist an independent set of size greater than k, otherwise return False.

Search the max value in $[0 \dots k]$. First try $k/2$, if Independent-Set $(G, k/2) =$ True, call Independent-Set $(G, (k + k/2)/2)$, otherwise call Independent-Set $(G, (0 + k/2)/2)$, and recursively continue. Stop until True is returned, or has made $\lceil \log k \rceil$ recursive calls.

[1] Cited from Wikipedia: Knapsack Problem

Problem 5 continued on next page...

Part (c) 4SAT

Reduce 3SAT to 4SAT:

Change every 3CNF formula into a 4CNF formula.

$$a \vee b \vee c \longrightarrow (a \vee b \vee c \vee x) \wedge (a \vee b \vee c \vee \overline{x})$$

For each 3CNF formula of n size, there is at most $\binom{n}{3}$ types of clauses. So this operation takes $\mathcal{O}(n^3)$ time (polynomial time).

Thus, a 3SAT is reduced to a 4SAT problem, plus a polynomial time operation.

By definition, 3SAT is a NP-hard problem. Hence, we conclude 4SAT is NP-hard. And because 4SAT is a special case of 3SAT, so it can be verified in polynomial time, thus, it is also in NP. Therefore, 4SAT is NP-complete. □

Part (d) Hitting-Set

Reduce Vertex-Cover to Hitting-Set:

Given an undirected graph G on n nodes and m edges and a parameter k. We define:

- S to be the set of nodes in G;
- S_i to be the set of the two endpoints of edges e_i, i.e. $S_i = \{u, v\}, \forall e \in E$, where $e = (u, v)$;
- Let the parameter k be the same k we are given.

Then G has a vertex cover of size k if and only if set S has a hitting set of size k for $C = \{S_1, \ldots, S_m\}$, since a set of nodes in G is a vertex cover S' if and only if it has an element in common with each of the edges, i.e. $S' \cap S_i \neq \emptyset$.

This reduction is in polynomial time because we only need to list the edges of G, which takes $\mathcal{O}(n + m)$ time. Therefore, Hitting-Set is NP-hard.

On the other hand, given a "Hitting Set", it takes $\mathcal{O}(n^2)$ to find the intersection of two set with size n. For totally m set, it is checkable in polynomial time ($\mathcal{O}(n^2 m)$ in particular), i.e. Hitting-Set is in NP.

Hence, Hitting-Set is NP-complete. □