

Network Optimization (Fall 2008) – Brief Solutions to Homework 4

1. First-fit heuristic

```

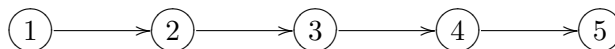
LIST := {a1, ..., an};
num_Bins := 0; (number of bins)
Cap := ∅; (vector of available bin capacities – size is variable)
Bin_Id := [0 0 ... 0]; (bin to which each item is assigned, initialized to the vector of n zeros)

begin
while LIST ≠ ∅ do
  select item ai from LIST
  for k := 1 to num_Bin if Bin_Id(i) = 0 then do
    if Cap(k) > ai then do
      Bin_Id(i) := k;
      Cap(k) := Cap(k) - ai;
      LIST := LIST - {ai};
    end_if
  end_for
  if Bin_Id(i) = 0 then do
    num_Bins := num_Bins + 1;
    Cap(num_Bins) := 1 - ai;
    Bin_Id(i) := num_Bins;
    LIST := LIST - {ai};
  end_if
end_while
end
    
```

The **while** loop is executed n times – once for each item. In the worst case, $num_Bins = n$, and hence the **for** loop gets executed $O(n)$ times. Thus, the algorithm runs in $O(n^2)$ time.

For any n , consider the instance where $a_i > 0.5$ for $i = 1, \dots, n$. For this instance, the first-fit heuristic will examine *all* current bins before putting the next item in a new bin (as no two items will fit in the same bin). Hence, the number of times the **for** loop is executed becomes $0 + 1 + \dots + n - 1$, proving that the heuristic runs in $\Omega(n^2)$ time, and hence in $\Theta(n^2)$ time.

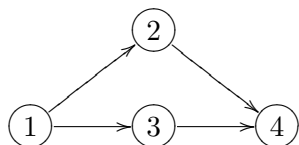
2.



3.



4.



$[1, 2, 3, 4]$ and $[1, 3, 2, 4]$ are both topological orderings.

Assume without loss of generality that the path is $1-2-\dots-n$. Then, the unique topological ordering is $[1, 2, \dots, n]$. Note that the network cannot contain arcs of the form (j, i) where $j > i$, as such an arc will form a cycle along with the section of the directed $1-n$ path from i to j . For all other arcs of the form (i, j) with $i < j$, the topological ordering is of course valid. Any other permutation of $\{1, 2, \dots, n\}$ cannot be a topological ordering, as *at least* two arcs from the $1-n$ directed path will violate the order.

To prove the converse, we can use the contrapositive argument. Assume w.l.o.g. that the network does not contain the path $1-2-\dots-n$. We will then prove that the topological ordering is not unique. Since there is no $1-n$ path, there must exist a node i with at least two subgraphs $G_i^1 = (N_i^1, A_i^1)$ and $G_i^2 = (N_i^2, A_i^2)$ rooted at i , such that there are no arcs in the original network connecting the nodes in N_i^1 and N_i^2 . When performing DFS from node 1, the first node which the DFS back-tracks to *and* finds an admissible arc from can be a candidate for the node i . Again, assume w.l.o.g. that $|N_i^1| \leq |N_i^2|$. Given a topological ordering for the original network, another topological ordering can then be obtained by swapping the orders assigned to the nodes in N_i^1 with those assigned to an appropriately chosen subset of N_i^2 of size $|N_i^1|$, with possible adjustments of the new orders among the nodes in N_i^2 . Hence, the topological ordering is not unique.

Alternative Proof: One could come up with a shorter proof for the result by analyzing the topological ordering algorithm that was discussed in class (given in Figure 3.8, page 79 in the book). The topological ordering determined by this algorithm is unique if and only if there are no ties for node selection from the LIST. W.l.o.g., the topological ordering $[1, 2, \dots, n]$ is unique if the node i is not in the LIST when node $i-1$ is in it. Hence, the network must contain the arc $(i-1, i)$ for all $2 \leq i \leq n$, thus proving the existence of the path $1-2-\dots-n$. Conversely, if the network contains the path $1-2-\dots-n$, then two distinct nodes i and j cannot be present in the LIST at the same time – since the nodes which are in the LIST have zero indegree, and because of the $1-n$ path, only one node can have zero indegree at any time. Hence the algorithm gives a unique topological ordering, as there will be no ties for the node selection from the LIST.

5. Strongly Connected Components Algorithm ($O(n^2)$ algorithm)

```

LIST := {1, ..., n}; LABEL := {0, ..., 0};
num_Comp := 1; (number of strongly connected components)
while LIST ≠ ∅ do
  remove node i from LIST
  LABEL(i) = num_Comp;
  for k in LIST do
    if τik = 1 and τki = 1 then do
      LABEL(k) := num_Comp;
      LIST := LIST - {k};
    end_if
  end_for
  num_Comp := num_Comp + 1;
end_while

```

6. Check the course web page for MATLAB code `BFS.m`.