

Network Optimization (Fall 2008) – Notes on the Project

1. The improvement makes sense when $c = \min_{(i,j) \in A} c_{ij} > 1$. Consider the case when we pick the node(s) in bucket(k). The distance labels of some other nodes in buckets of higher value might get updated (because they are the heads of arcs going out of the current node, and the optimality conditions are violated). The updated value of these distance labels will at least be $k + c$. Hence, no nodes will be added to the buckets numbered $k + 1, k + 2, \dots, k + c - 1$, and we can lump the buckets $k, k + 1, \dots, k + c - 1$ in to one bucket. To implement this idea, we can use $\lceil (nC + 1)/c \rceil$ buckets, and place node i in bucket k if $ck \leq d(i) \leq ck + c - 1$. Subsequently, the running time is $O(m + nC/c)$.
2. (b) One way to generate random networks is to randomly assign the ones in the node-node adjacency matrix. Assuming the number of arcs m is known, we can generate an $n \times n$ matrix of random values (between 0 and 1). We can pick the m largest values from among the *non-diagonal* entries in this matrix, convert all of them to 1, and set the remaining $n^2 - m$ entries to zero. We avoid self-loops in the network by avoiding the diagonal entries. The commands in MATLAB could be given as shown below (assuming n and m are known).

```
>> N = rand(n);
>> N = N - diag(diag(N));
>> rvals = -sort(-reshape(N,1,n^2)); % sort in descending order
>> N = (N>=rvals(m));
```

When we pick the source node s randomly, the above method may generate networks where not all nodes are reachable from s . This is not a big concern for testing the algorithms, but we may prefer to work with networks where all nodes are reachable from s . One way to ensure reachability is to build a spanning tree rooted at node s . Starting with node s , we can add the remaining nodes sequentially, by including an arc from one of the nodes already added, which is selected randomly, to the node currently being added. One way to achieve this task in MATLAB *without using loops* is given below (assuming n, m , and the source node s are known).

```
>> addord = randperm(n);
>> [remnd,ind] = setdiff(addord,s);
>> addord = addord(sort(ind));
>> nds = [s addord];
>> arcind = min((1:n-1),randint(1,n-1,n)+1);
>> arcs = (addord-1)*n + nds(arcind);
>> NTr = zeros(n);
>> NTr(arcs) = 1;
>> N = rand(n);
>> N = N-diag(diag(N));
>> N(arcs) = 0;
>> rvals = -sort(-reshape(N,1,n^2));
>> N = (N >= rvals(m-n+1)) + NTr
```

- (c) Overall, Dijkstra's algorithm (original implementation) is expected to perform very well in practice for dense networks, i.e., when the number of arcs m is reasonably large compared to the number of nodes n . Dial's implementation works well when C/c is small, i.e., when most costs lie in a small range of values. If the costs occur in a huge range (i.e., $C \gg c$), Dial's implementation will be slower. FIFO label correcting algorithm is expected to be efficient in practice as well, but will be slower on average than the other two methods in question. Hence, one may choose Dijkstra's algorithm if all costs are known to be non-negative; else go with the FIFO label correcting algorithm.