CS 345A Assignment 5

Yasharth Bajpai (170822), Nidhi Hegde (180472)

October 30, 2020

Difficult

Overview:

Since the shortest path may not always be feasible, we adopt a constrained approach in this problem towards finding the shortest feasible path from s to d. In our solution, we make use of the following sub-routine:
Augmented Dijkstra(v):
This function is a modification of the original Dijkstra, which starts at node v and follows the normal Dijkstra algorithm with a slight change that the shortest path length from v to any vertex is assigned a value ∞ if its length is greater than C. The set returned by this algorithm is the set of vertices reachable from v along the shortest path, with fuel usage less than C.

In our first step, we compute the above sub-routine for all vertices in R_0.
Next, we take the union of all sets returned by the sub-routine when applied on elements of R_0 (only s in this case), and call this set R_1. We then add R_1 to the set R. In the next step, we again apply our Augmented Dijkstra sub-routine only on those vertices in the set R_1 which represent fuel-stations, and similarly compute R_2 by taking a union over all the sets returned by the sub-routine.
At the end of 2 steps, the set R contains all the vertices reachable from s (or R_0) with a re-fueling of less than 2 times.
Similarly at end of i steps, the set R contains all the vertices reachable from s with a re-fueling of less than i times. Using this idea, we iterate till n such steps or till we finally get the vertex d, depending on whichever happens first.
If $d \notin R$ at the end of n steps, it means that vertex d is not reachable from s under the constraints given in problem.

Pseudo Code:

Algorithm 1 Algorithm to find shortest feasible route from s to d

```
function FindShortestFeasiblePath($G, s, d$)
    $R \longleftarrow \phi$
    $R_0 \longleftarrow s$
    $R \longleftarrow R \cup R_0$
    $Path(s, *) \longleftarrow NULL$
    $Path(s, s) \longleftarrow s$
    for $i$ from 1 to n do
        for each $v$ in $R_{i-1}$ do
            if $v$ not a Fuel-Station Node or VisitedAugDijkstra(v) $==$ true then
                continue
            end if
            $P, r \leftarrow AugmentedDijkstra(G, v)$
                    ▷ r[j] represents node j and P[j] represents path from node $v$ to node $j$
            for $j$ in (0, |r|-1) do
                if $Path(s, r[j])$ not NULL then
                    $Path(s, r[j]) \leftarrow Append(Path(s, v), P[j])$  ▷ Appending the path from node $v$ to node $j$
                end if
            end for
            $VisitedAugDijkstra(v) = true$
            $R_i \longleftarrow R_i \cup r$
        end for
        $R \longleftarrow R \cup R_i$
        if $d \in R$ then
            break
        end if
    end for
    if $i == n + 1$ then
        return $No\ path\ exists$
    end if
return $Path(s, d)$
end function
```

Proof of Correctness:

The δ used in the proof correspond to the length of the shortest feasible paths between two vertices. Consecutive fuel pumps, implies that there is no third fuel pump that exists on the path between these two fuel pumps.

Lemma 1: In an optimal path, the distance between two consecutive fuel pumps on the path $\leq C$.

It is easy to see that the vehicle can only travel a maximum of C distance starting from a fuel pump. Let's assume that the vehicle starts with a full tank from a pump on the optimal path and doesn't reach another fuel pump before the last drop of fuel is used. In this case, the vehicle cannot go any further and thus never will reach d. Thus, contradicting our original assumption of travelling on the optimal path.

Lemma 2: In the optimal path, the optimal subpath between two consecutive fuel pumps is the shortest path between them.

Using Lemma 1, we know that the maximum distance between two consecutive pumps on the optimal path is C. Therefore, for a travel between consecutive fuel pumps, the constraint of refueling is relaxed, as one can reach the other without refueling enroute. Let u & v be consecutive fuel pumps on the optimal path P. Then, $\delta(u, v) \leq C$ i.e we can reach from u to v without refueling. If there exists another path from u to v without refueling, such that $\delta^*(u, v) < \delta(u, v) \implies \delta^*(s, d) < \delta(s, d)$ where the path P^* uses the new shorter path between the fuel pumps. Thus, the path P we originally considered is no more the optimal path. Thus, contradicting our original assumption of u & v being consecutive fuel pumps on the optimal path P.

Lemma 3: A fuel pump is visited at most once in an optimal path

Leaving the fuel pump, we are certain for travelling further for a distance C. Let's assume that the optimal path P has two visits to some fuel pump vertex a such that $P \equiv s \rightsquigarrow a \rightsquigarrow b \rightsquigarrow a \rightsquigarrow d$ where b is any other vertex. Then $\delta(s, d) = \delta(s, a) + \delta(a, b) + \delta(b, a) + \delta(a, d)$. We can safely say that as a is a fuel pump, thus the vehicle was at maximum capacity leaving it. Therefore, the vertex d was as reachable from a in its first visit as in the second visit. Thus, skipping the journey to b, we travel directly towards d following the optimal path post the second visit. Thus the path $P^* \equiv s \rightsquigarrow a \rightsquigarrow d$ has a distance $\delta^*(s, d) = \delta(s, a) + \delta(a, d)$. As edge weights are positive for the graph $\implies \delta^*(s, d) < \delta(s, d)$. Thus, P is not an optimal path contradicting our original assumption

Lemma 4: If we do not reach the destination vertex in n iterations of the procedure, then the destination vertex is not reachable with the current constraints and no optimal path exists.

The Augmented Dijkstra's is applied for all unvisited fuel pumps in the Reachable set at any iteration level. If no new reachable petrol pump is returned at some point in the iteration, it implies that there exists no more fuel pumps in the graph reachable with the given fuel constraint. This can occur in two cases: Firstly, either we have covered all the pumps in the graph or Secondly, the other pumps are not reachable. In both these cases, if the destination vertex has not already been found, no feasible path exists further for the destination as the algorithm makes no progress beyond this point.
So for the algorithm to proceed to the next iteration with some progress, we must at least receive one new

fuel pump every iteration. In the worst case, when all the vertices are fuel pumps, then we would need n iterations to complete a visit and Augmented Dijkstra's to all the vertices. At this point we will definitely have covered all reachable fuel pumps in the graph, and there will be no more pumps to be visited. So we are now back to the first case discussed above. If d has not been encountered yet, we don't have any progress further in the algorithm and thus it is fine to safely assume that d won't be discovered in any of the later iterations.

Using the Lemmas above, we'll prove the correctness of the algorithm by Induction.

Induction Hypothesis/Loop Invariant: At end of each iteration i of the loop, R stores the set of vertices reachable from the s (start vertex) in $< i$ refuelings.

Base Case: At $i = 1$ i.e. the first iteration, we are at s (Starting vertex is the only vertex in R). Augmented Dijkstra's executes on it, s to these vertices is calculated and we get finite values for shortest distances only for the vertices with a distance less than C from s, the vertices returned are added to R_1. The set R_1 is then added to the set R i.e. $R = R \bigcup R_1$. Trivially, R contains the vertices which are reachable without refueling (< 1) from s as the maximum distance we can travel without a refueling is C (Including s itself). Therefore, the hypothesis holds true for $i = 1$.

Induction Step: Assuming that the claim holds true for iteration i, we will try and prove that it holds true for iteration $i + 1$.
R stores the vertices reachable from s in $< i$ refuelings along the path and the union of all vertices returned by Augmented Dijkstra's at level i is stored as R_i. It is trivial to see that reaching any of these vertices in the i^{th} iteration, the tank of the vehicle will never be fully filled (Positive edge weights).
Now, moving to the $i + 1^{th}$ iteration. For each fuel pump vertex $v \in R_i$, we refill the tank to the maximum capacity C. Now, we have to execute the Augmented Dijkstra's at each of these vertices with fuel pumps. Now, as a result, we get back all the vertices that are reachable from the fuel pumps in R_i without refueling further. These, vertices are stored in R_{i+1}.
As only one additional refueling was done while starting from the fuel pump vertices in R_i, we can say that these vertices R_{i+1} that were reached in this iteration can be reached in $<= i$ or $< i + 1$ refuelings. These newly found vertices are added to R i.e. $R = R \bigcup R_{i+1}$. As the newly found vertices R_{i+1} are reachable from some vertex in R, which are in turn reachable from s implies that these newly added vertices are also reachable from s. Earlier, R comprised of vertices reachable from s in $< i$ refuelings. Now, that we have appended the set R_{i+1} into R, we also have vertices with one additional fueling than before .i.e. $< i + 1$ refuelings. Therefore, after $i + 1^{th}$ iteration, the set R contains vertices reachable from s with $< i$ refuelings. Hence, the induction hypothesis/loop invariant holds for $i + 1$ as well.
Therefore, we prove that the algorithm works for iteration $i + 1$ if it works for all the preceding iterations in the procedure. So, our Induction Hypothesis is proved.

Failing Case: The maximum number of times we can run the iteration is bound by n i.e.the number of vertices in the graph. Employing Lemma 4, If we do not encounter the destination vertex d before the n iterations, implies that the vertex is not feasibly reachable from s. This is when we return that no feasible path is possible from s to d. Thus, the case when no shortest feasible path exists is also taken care of by the algorithm.

Time Complexity:

To find the shortest feasible path between s & d, we employ an Augmented Dijkstra's Algorithm with an additional check for the distance being less than C (Tank Capacity) and return the weights accordingly. This is a constant time operation added to the regular Dijkstra's Algorithm. Therefore our Augmented Dijkstra's Algorithm also runs in $O(E \log V) \equiv O(m \log n)$. (Assuming that we have a graph implemented as an Adjacency list)
Now, in every iteration we pick the refueling vertices from the set of reachable vertices (performable in $O(1)$) and apply the Augmented Dijkstra's Algorithm to each of these vertices. Thus, the total number of times the Augmented Dijkstra's is executed is bounded by the number of fuel pumps, which is in turn bounded by the number of vertices in the graph. Thus, the Augmented Dijkstra's algorithm is used $O(n)$ times through our entire implementation. At each instance of return from Dijkstra's algorithm, we receive $O(n)$ new vertices to be processed for path, as complex paths are possible and non-fuel vertices may be visited more than once.

Therefore, the complexity of the entire procedure is $O(n) \times (O(m \log n) + O(n)) \implies O(mn \log n + n^2)$. For a graph to be connected, we know that $n - 1 \leq m \leq n^2$. Sandwiching the values for m, $O(n^2 \log n) < O(mn \log n + n^2) < O(n^3 \log n)$. In both cases, asymptotically, the first term dominates the order, while the second term in the complexity grows fairly slower as compared to the first term. Therefore, we can conclude that the algorithm has a run time of $O(mn \log n)$.

Note: If the graph is not completely connected we only refer to the connected component to which s belongs,hence essentially we have a connected graph with reduced size, $G'(n', m')$. if d is belongs to the same connected component, it might be found in time $O(m'n' \log n')$ where $n' - 1 \leq m' \leq n'^2$. Else, d is never reached and no optimal paths exist.