Homework 3

Thomas Kim tsk389 51835

David Munoz dam2989 51840

CS331 Algorithms and Complexity

Problem Q1. Prove each of the following heuristics to be ideal or not ideal for scheduling jobs on a machine while minimizing:

$$\sum_{i=1}^{n} w_i C_i$$

1. Smallest time first.
2. Most important first.
3. Maximum $\frac{w(i)}{t(i)}$.

Suppose jobs are expressed as tuples (w_k, t_k) where w_k is the weight of the job and t_k is the time that job takes.
Claim: Smallest time first is **not** ideal.

Proof. Suppose the following set of jobs is scheduled using smallest time first.
$\{j_1 = (1, 2), j_2 = (3, 3)\}$
The smallest time first heuristic would schedule j_1 before j_2.
Thus, j_1 would finish at time 2 and j_2 would finish at time 5.
This results in a total cost of 17.
Suppose j_2 is scheduled before j_1.
j_2 would finish at time 3 and j_1 would finish at time 5.
This results in a total cost of 14.
$14 < 17 \implies$ the smallest time first heuristic is not ideal.

□

Claim: Most important first is **not** ideal.

Proof. Suppose the following set of jobs is scheduled using most important first.
$\{j_1 = (2, 3), j_2 = (1, 1)\}$ The most important first heuristic would schedule j_2 before j_1.
Thus, j_1 finishes at time 3 and j_2 finishes at time 4.
The total cost is therefore 10
Suppose j_2 was scheduled before j_1.
j_1 would finish at time 4 and j_2 would finish at time 1.
The total cost is therefore 9.
$9 < 10 \implies$ the most important first heuristic is not ideal.

□

Claim: Maximum $\frac{w(i)}{t(i)}$ first **is** ideal.

Proof. Let the queue of jobs be represented by a set of jobs $\{j_1, j_2, \ldots, j_n\}$ where j_k is scheduled before j_c iff $k < c$.
Suppose there exists an inversion between adjacent jobs in the scheduling queue.
This means that for some k, $\frac{w_k}{t_k} < \frac{w_{k+1}}{t_{k+1}}$.
This means that $w_k t_{k+1} < w_{k+1} t_k$. **(1)**
Base case: The inversion exists between jobs j_1 and j_2.
$w_1 t_1 + w_2(t_1 + t_2) > w_2 t_2 + w_1(t_1 + t_2)$

$w_1t_1 + w_2t_1 + w_2t_2 > w_2t_2 + w_1t_1 + w_1t_2$
$w_2t_1 > w_1t_2$ is true by equation (1)
Induction hypothesis:
$w_{n+1}\left(\sum_{i=1}^{n+1} t_i\right) + w_{n+2}\left(\sum_{i=1}^{n+2} t_i\right) > w_{n+2}\left(\sum_{i=1}^{n} t_i + t_{n+2}\right) + w_{n+1}\left(\sum_{i=1}^{n+2} t_i\right)$
$w_{n+2}\left(\sum_{i=1}^{n+2} t_i\right) - w_{n+2}\left(\sum_{i=1}^{n} t_i + t_{n+2}\right) > w_{n+1}\left(\sum_{i=1}^{n+2} t_i\right) - w_{n+1}\left(\sum_{i=1}^{n+1} t_i\right)$
$w_{n+2}\left(\sum_{i=1}^{n+2} t_i - \sum_{i=1}^{n} t_i - t_{n+2}\right) > w_{n+1}\left(\sum_{i=1}^{n+2} t_i - \sum_{i=1}^{n+1} t_i\right)$
$w_{n+2}\left(t_{n+1} + t_{n+2} - t_{n+2}\right) > w_{n+1}t_{n+2}$
$w_{n+2}t_{n+1} > w_{n+1}t_{n+2}$
This is true by equation (1).

□

Problem Q2(a). Briefly explain which steps of Dijkstra's algorithm fail when a graph can have negative weight edges.

Dijkstra's algorithm relies on a breadth-first search using a priority queue and stops execution when the goal is reached.
The stopping of execution when the goal has been reached relies on the invariant that the current path is the shortest path found so far, and that any other paths will either increase or stay the same in cost if explored.
However, negative-weight edges can result in the cost of a path being explored to decrease, violating this invariant.
This problem can be trivially solved by simply running BFS until all vertices are explored.

Problem Q2(b). Given a set of 4-tuples (s_i, d_i, p_i, q_i), where each entry corresponds to source, destination, departure time, arrival time respectively, calculate the plan that arrives at d_i as early as possible, while allowing for at least 1 hour for each connecting flight.

1: Let $G(V, E)$ be an empty graph
2: **for** $\forall(s_i, d_i, p_i, q_i)$ **do**
3: Create some vertex v which corresponds to s_i
4: Create some vertex v' which corresponds to d_i
5: Create a directed edge $e(v, v')$ with weight $q_i - p_i$
6: Create a mapping $f : e \to (q_i + 1)$
7: Create a mapping $g : e \to p_i$
8: **end for**
9: Execute Dijkstra's algorithm starting at node S and using $g(e)$ as the initial weight of any path starting with edge e
10: Dovetail the following steps in between iterations of Dijkstra's algorithm.
11: **for** $\forall e \in$current cutset **do**
12: **if** $f(e) > g(e)$ **then**
13: Remove e from the graph
14: **end if**
15: **end for**

Runtime
The graph creation will explore every flight once, resulting in a complexity of $O(m)$.
The additional steps in this algorithm will explore every node precisely once, so the complexity is $O(m)$.

The runtime of this algorithm is $O(mlog(n))$ from Dijkstra's and $O(2m)$ from the additional steps.
Therefore, the total runtime is $O(mlog(n))$

Correctness

Proof. An invalid edge is defined to be an edge where q_i for an incoming flight and p_i for an outgoing connection fulfill $p_i < q_i + 1$
By lines 6, 7, 12, the algorithm will delete such edges before they are explored by Dijkstra's algorithm.
Each edge represents the total amount of time spent on the flight plus 1h overhead for connections.
The initial weight (time) of any path is given to be the departure time of the edge explored.
This means the lowest weight path will have the minimal time difference from $t = 0$.
Since there are no negative weight edges (time cannot be negative), Dijkstra's algorithm will successfully find the min weight path.
This min weight path has the minimal time difference from $t = 0$.
This means the arrival time with respect to $t = 0$ will be minimized.

□

Problem Q3(a). Give an algorithm to find the minimum-product spanning tree.

1: Let $G(V, E)$ be the graph in question
2: **for** $\forall e \in E$ **do**
3: $\quad weight(e) = ln(weight(e))$
4: **end for**
5: Run Prim's algorithm on the new graph G'

Proof of correctness

Proof. Note that by the problem, $w > 1$
Restricting the domain to $(1, \infty), ln(x)$ is one-to-one and monotonically increasing
$ln(\prod_{i=1}^{n} w_i) = \sum_{i=1}^{n} ln(w_i)$
So the sum of the modified weights is equal to the natural log of the product-weight.
Since $ln(x)$ is one-to-one and monotonically increasing for the domain of this problem, minimizing $\sum_{i=1}^{n} ln(w_i)$ will minimize $\prod_{i=1}^{n} w_i$

□

The time taken to change the weights of each edge is $O(m)$.
The time taken to run Prim's algorithm is $O(mlog(n))$.
Therefore, the total runtime is $O(mlog(n) + m) = O(mlog(n))$.

Problem Q3(b). Prove that if the weights of an undirected graph are unique, there exists a unique minimum spanning tree.

Proof. Suppose two MSTs T_1, T_2 differ by some edge e.
Suppose $e \in T_1 \wedge e \notin T_2$.
Take $T' = T_2 \cup \{e\}$
Note that T' must contain a cycle, since T' is a spanning tree and e is a non-tree edge.
T_2 must contain some edge e', such that e' and e are in the same cycle in T' and $w(e) < w(e')$.
Replace e' with e in T_2. This creates a new tree with a smaller cost than T_2.
This means T_2 is not an MST.

□

Problem Q3(c). Let $\mathcal{T}_G$ be the set of minimum spanning trees for some graph G. Let T be a spanning tree for G. Prove that:

$$\forall e \in T, e \in T' \in \mathcal{T}_G \implies T \in \mathcal{T}_G$$

Lemma 1: Consider two MSTs which differ by 2 edges. These edges must be in the same cycle and must have the same weight.

Proof. Suppose these edges are not in the same cycle. Removing one of these edges would result in a disconnected graph, meaning the two MSTs are not trees.
Suppose edge e has a greater weight than e'. Replace e with e', and the resulting tree has a smaller weight than the original tree, implying the original tree was not an MST.

□

Main proof:

Proof. Take $T' \in \mathcal{T}_G$.
For each edge $e \in T' \wedge e \notin T$, find some $e' \in T$ s.t. e' forms a cycle with e and replace e with e'.
Note that every edge $e' \notin T'$ will form a cycle in T'
By lemma 1, these replacements result in new MSTs.
Note that both T and T' must have $n-1$ edges because they are both trees.
The resulting transformation will transform T' into T, as all edges $e \in T' \wedge e \notin T$ have been replaced by some edge $e' \in T$, and since both trees have the same number of edges to begin with they will end up with the same number of edges.

□