Algorithm Design
M. T. Goodrich and R. Tamassia
John Wiley & Sons
Solution of Exercise C-7.9

We will describe two solutions for the flight scheduling problem.

1. We will reduce the flight scheduling problem to the shortest paths problem. That means that we will construct an instance for the shortest paths problem, that when solved, gives us a solution for the flight scheduling problem. Given the set of airports $\mathcal{A}$ and the set of flights $\mathcal{F}$, we consider a weighted directed graph $\overrightarrow{G}$ that is constructed as follows.
   - For each airport $a_i \in \mathcal{A}$, draw a circle $circle_i$ to represent the time (24 hours).
   - For each flight $f_i \in \mathcal{F}$, find the origin airport a_o, the destination airport a_d, the departure time t_d, and the arrival time t_a. Draw a vertex v_1 on $circle_o$ marked the time t_d and also draw a vertex v_2 on $circle_d$ marked the time (t_a+c(a_d)). Draw an directed edge from v_1 to v_2 with weight t_a+c(a_d)-t_d (of course we must compute the correct weight (flight time) in case like the departure time is 10:00PM and the arrival time is 1:00AM next day). The direction of edges on a circle should be clockwise.

   Now we have a new graph like the Figure **??** below:
   Note that we have included the minimum connecting time in the total flight time, so we can only consider the modified new flights without worrying about the connecting times.
   Given graph $\overrightarrow{G}$, in order to solve the flight scheduling problem, we can just find a shortest path from the first vertex on $circle(a)$ (origin airport a) representing time t or after t to one vertex on $circle(b)$ (destination airport b) in the graph. The flights' sequence can be obtained from the shortest path from origin to destination.
   We can use Dijkstra's algorithm to find the desired shortest path (observe that $\overrightarrow{G}$ has only positive weights), but we need to slightly modify the algorithm (as it is presented in the textbook), so that it is applicable to directed graphs. The modifications should be straightforward: in the relaxation step,

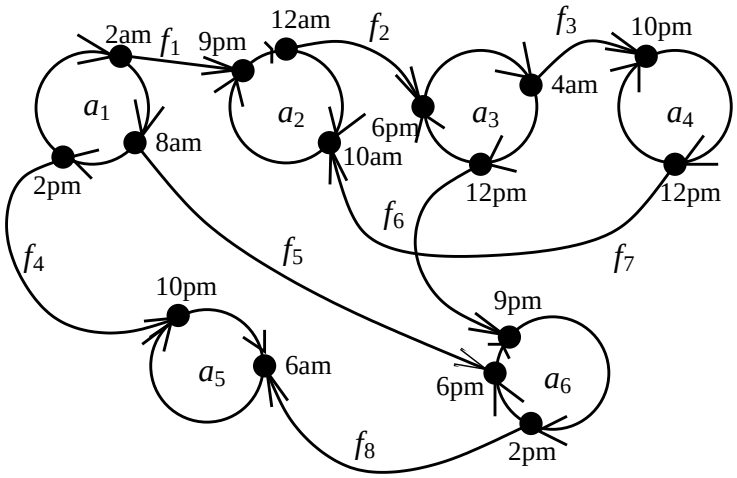

Figure 0.1: Graph used to reduce a flight scheduling problem to a shortest paths problem.

we only consider edges with orientation from the previously selected vertex to some other vertex. Finally, we need keep track (mark) the shortest path, so we include a parent-child relationship to the visited nodes (value p).

Algorithm FlightScheduling(a,b,t):
construct graph $\overrightarrow{G}$
$v \leftarrow$ first vertex on $circle(a)$ representing time t or after t
$D[v] \leftarrow 0$
for each vertex $u \neq v$ of $\overrightarrow{G}$ **do**
  $D[u] \leftarrow \infty$
let a priority queue Q contain all the vertices of $\overrightarrow{G}$ with D values as keys
while Q is not empty **do**
  $u \leftarrow Q$.removeMin()
  for each vertex z such that $\overrightarrow{(u,z)} \in E_{\overrightarrow{G}} \wedge z \in Q$ **do**
    if $D[u] + w(\overrightarrow{(u,z)}) < D[z]$ **then**
      $D[z] \leftarrow D[u] + w(\overrightarrow{(u,z)})$
      change to $D[z]$ the key of vertex z in Q
      $p[z] \leftarrow u$
$w \leftarrow$ vertex on $circle(b)$ with minimum value D
return reversed path from w to v (using values p)

The time complexity of the algorithm essentially the time complexity of Dijkstra's algorithm (since the construction of the graph can be done in linear ($O(n+m)$) time). Using a heap as the priority queue, we achieve a total $O((N+M)\log N)$ time complexity, where N the number of vertices of $\overrightarrow{G}$ and M the number of edges of $\overrightarrow{G}$. Note that, generally, $M = O(m)$ and $N = O(m)$, so the running time of the algorithm is $O(m \log m)$.

2. The following algorithm finds the minimum travel time path from airport $a \in \mathcal{A}$ to airport $b \in \mathcal{A}$. Recall, we must depart from a at or after time t. Note, "$\oplus$" and "$\preceq$" are operations which we must implement. We define these operations as follows: If $x \oplus y = z$, then z is the point in time (with date taken into consideration) which follows x by y time units. Also, if $a \preceq b$, then a is a point in time which preceeds or equals b. These operations can be implemented in constant time.

Algorithm FlightScheduling(a,b,t):
  initialize a set $\mathcal{P}$ to a
  initialize incoming_flight(x) to nil for each $x \in \mathcal{A}$
  earliest_arrival_time(a)$\leftarrow t$
  for all $x \in \mathcal{A}$ such that $x \neq a$ **do**
    earliest_arrival_time(x)$\leftarrow \infty$
    time_can_depart$\leftarrow \infty$
  repeat
    remove from $\mathcal{P}$ airport x such that earliest_arrival_time(x) is soonest
    if $x \neq b$ **then**
      if $x = a$ **then**
          time_can_depart $\leftarrow t$
      else
          time_can_depart $\leftarrow$ earliest_arrival_time(x) $\oplus$ c(x)
      for each flight $f \in \mathcal{F}$ such that $a_1(f) = x$ **do**
          if $time\_can\_depart \preceq t_1(f) \wedge$
          $t_2(f) \preceq earliest\_arrival\_time(a_2(f))$ **then**
              earliest_arrival_time($a_2(f)$) $\leftarrow t_2(f)$
              incoming_flight($a_2(f)$) $\leftarrow f$
              add $a_2(f)$ to $\mathcal{P}$ if it is not yet there
  until $x = b$

  initialize *city* to b

initialize $flight\_sequence$ to *nil*
while (city$\neq a$) **do**
  append $incoming\_flight(city)$ to $flight\_sequence$
  assign to *city* the departure city of $incoming\_flight(city)$
reverse $flight\_sequence$ and output

The algorithm essentially performs Dijkstra's Shortest Path Algorithm (the cities are the vertices and the flights are the edges). The only small alterations are that we restrict edge traversals (an edge can only be traversed at a certain time), we stop at a certain goal, and we output the path (a sequence of flights) to this goal. The only change of possible consequence to the time complexity is the flight sequence computation (the last portion of the algorithm). A minimum time path will certainly never contain more that m flights (since there are only m total flights). Thus, we may discover the path (tracing from b back to a) in $O(m)$ time. Reversal of this path will require $O(m)$ time as well. Thus, since the time complexity of Dijkstra's algorithm is $O(m\log n)$, the time complexity of our algorithm is $O(m\log n)$.

Note: Prior to execution of this algorithm, we may (if not given to us) divide $\mathcal{F}$ into subsets $F_1, F_2, \ldots, F_N$, where F_i contains the flights which depart from airport i. Then when we say "for each flight $f \in \mathcal{F}$ such that $a_1(f) = x$", we will not have to waste time looking through flights which do not depart from x (we will actually compute "for each flight $f \in F_x$"). This division of $\mathcal{F}$ will require $O(m)$ time (since there are m total flights), and thus will not slow down the computation of the minimum time path (which requires $O(m\log n)$ time).