Theory of Algorithms Class Notes
Calvin Walker

Lecture 1: Gale-Shapley Algorithm for Stable Matching

Stable Matching Problem: There are two groups of people: $A = \{a_1 \dots a_n\}$ and $B = \{b_1 \dots b_n\}$, and we want to find a stable matching between A and B such that there is no pair of people a_i, b_j who would rather be with each other than their current partners. i.e We want to find a permutation π such that there are no $i, j \in [n]$ for whom

1. $\pi(i) \neq j$
2. a_i prefers b_j to $b_{\pi(i)}$
3. b_j prefers a_i to $b_{\pi^{-1}(j)}$

Gale-Shapley algorithm: The people in group A make offers in the order of their preference lists. When b_j receives an offer from a_i, if b_j is unmatched or preferes a_i to their current partner, they accept and become partners with a_i. Otherwise, b_j rejects and a_i moves onto the next person.

Properties:

- The Gale-Shapley algorithm terminates in $O(n^2)$ steps and results in a stable matching.

  Proof: There are n^2 possible offers and each offer is made at most once.

  Terminates: Oberve that if some a_i makes an offer to their last choice b_j, then all people except b_j must be unavailable for a_i and thus already matched. There are only $n - 1$ other people in A, so b_j must be unmatched, and b_j accepts a_i's offer, and the algorithm terminates.

  Stable: Assume there are $i, j \in [n]$ such that a_i and b_j prefer each other to their current partners. Let b_j be a'_i's current partner and b'_j be matched with a_i. Then b_j is available to a_i. However, since a_i is matched with b'_j, b_j must have been unavailable to a_i at some point, a contradiction.

- As Gale-Shapley progresses, (1) people in B only become happier, and (2) if b_j becomes unavailable to a_i they never become available again.

  Proof: (1) b_j only breaks off from its match if it prefers the new a_i. (2) Assume b_j becomes unavailable to a_i and then becomes available again. Let a'_i be b_j's partner when b_j became unavailable to a_i. Let a''_i be b_j's partner when b_j became available to a_i again. Then, b_j prefers a'_i to a_i and a_i to a''_i. However, b_j must prefer a''_i to a'_i, a contradiction.

Lecture 2: Big O Notation

Big O Notation: Given functions $f, g : \mathbb{N} \mapsto \mathbb{R}^+$

1. We say that $f(n)$ is $O(g(n))$ if there exist $n_0, C > 0$ such that for all $n \geq n_0$, $f(n) \leq Cg(n)$
2. We say that $f(n)$ is $\Omega(g(n))$ if there exist $n_0, c > 0$ such that for all $n \geq n_0$, $f(n) \geq cg(n)$
3. We say that $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $\Omega(g(n))$ such that there exist n_0, c, C' such that for all $n \geq n_0$, $cg(n) \leq f(n) \leq C'g(n)$

Examples:

- If $f(n) \leq 8n \log_2(n) + 20n + 100$ then $f(n)$ is $O(n \log n)$
- If $f(n) \geq n^2 - 3n - 2$ then $f(n)$ is $\Omega(n^2)$
- If $\frac{1}{2} \log_2 n - 2 \leq f(n) \leq 4 \log_2 n + 1$ then $f(n)$ is $\Theta(\log n)$

Properties:

- Given $f, g, h : \mathbb{N} \mapsto \mathbb{R}^+$, if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$
- Given $f_1, f_2, g_1, g_2 : \mathbb{N} \mapsto \mathbb{R}^+$, if $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n))$ then:
  - $f_1(n) + f_2(n)$ is $O(g_1(n) + g_2(n))$ and $O(\max\{g_1(n), g_2(n)\})$
  - $f_1(n)f_2(n)$ is $O(g_1(n)g_2(n))$

    Proof: There exist n_1, C_1 such that $\forall n \geq n_1(f_1(n) \leq C_1 g_1(n))$ and there exist n_2, C_2 such that $\forall n \geq n_2(f_2(n) \leq C_2 g_2(n))$. Let $n' = \max\{n_1, n_2\}$ and $C' = C_1 C_2$, So for all $n \geq n'$, $f_1(n)f_2(n) \leq C' g_1(n) g_2(n)$

  Comparing logarithms, polynomials, and exponential functions:

1. $\log n^{C'}$ is $O(n^C)$
2. $n^{O(1)}$ means at most n^C for some $C > 0$. This is polynomial time
3. $2^{O(n)}$ means at most 2^{Cn} for some $C > 0$. This is exponential time.

Lecture 3: Greedy Algorithms

Interval Scheduling Problem 1 (Maximizing number of jobs): Given n jobs with a set time interval $[a_i, b_i]$ and one processor, find a schedule S which accepts the maximum number of jobs without having two jobs running at the same time. Formally, we say that a sequence $S = (i_1, \ldots, i_m)$ is a valid schedule if for all $j \in [m-1](b_{i_j} \leq a_{i_{j+1}})$

Greedy Algorithm for Interval Scheduling 1: When choosing the next job, always choose the job which can be finished first.

- *Stored Data*: $S_k = (i_1, \ldots, i_k)$ of the jobs which have been accepted thus far, and the time $t_k = b_{i_k}$
- *Initialization*: $S_0 = \emptyset$ and $t_0 = 0$
- *Iterative Step*: Choose the next job i_{k+1} from the set $\{i \in [n] \mid a_i \geq t_k = b_{i_k}\}$ such that b_{i_k} is minimized. Update S_{k+1} and $t_{k+1} = b_{i_{k+1}}$, if there are no available jobs left then the algorithim terminates and we take $S = S_k$

Properties:

- Given a valid schedule $S = (i_1, \ldots, i_m)$, for all $j \in [m]$, we denote the time at which the jth hob in S finishes as $t_j(S) = b_{i_j}$. For all $j > m$, we say $t_j(S) = \infty$
- If the greedy algorithim gives a valid schedule $S = (i_1, \ldots, i_m)$, then for any other valid schedule $S' = (i_1, \ldots, i'_m)$, $m' \leq m$

  Proof (Using the lemma proved below): Consider $j = m+1, t_{m+1}(S) = \infty$ so we must have that $t_{m+1}(S') = \infty$. Thus, $m' \leq m$
- If $S = (i_1, \ldots, i_m)$ is the schedule given by the greedy algorithm then for any other valid schedule $S' = (i_1, \ldots, i'_m)$, for all $j \in \mathbb{N}$, either $t_j(S) \leq t_j(S')$ or $t_j(S') = \infty$

  Proof: By induction on j. For the inductive step, assume that $t_k(S) \leq t_k(S')$. Note that $a_{i'_{k+1}} \geq t_k(S') \geq t_k(S)$ so job i'_{k+1} is available to S. Since S always chooses the job with the earliest completion time out of the available jobs, $t_{k+1}(S) = b_{i_{k+1}} \leq b_{i'_{k+1}} = t_{k+1}(S')$.
- This algorithm can be implemented in $O(n \log n)$ time by first sorting the jobs in increacing order of b_i

Interval Scheduling Problem 2 (Mimimizing maximum lateness): Given n jobs, each of which has length $l_i > 0$ and a deadline d_i, complete the jobs one by one such that the maximum lateness of any single job is minimized. Formally, define $t_S(i_k) = \sum_{j=1}^{k} l_{i_j}$ to be the time at which job i_k is finished, and find a schedule $S = (i_1, \ldots, i_n)$ such that $\max\limits_{j \in [n]}\{t_S(i_j) - d_{i_j}\}$ is minimized.

Greedy Algorithm for Interval Scheduling 2 (Earliest Deadline First): Take the schedule S_{greedy} such that $d_{i_1}, \ldots, d_{n_1}$ are in increacing order.

Properties:

- The schedule S such that $d_{i_1}, \ldots, d_{n_1}$ are in increacing order minimizes $\max\limits_{j \in [n]}\{t_S(i_j) - d_{i_j}\}$

  Proof:
- If S is an optimal schedule, then ...

  Proof:

  1. Swapping i_j and i_{j+1} does not affect the lateness of any other job as for all $j' \in [n] \setminus \{j, j+1\}$, $t_{S'}(i_{j'}) = t_S(i_{j'})$
  2. $t_S(i_{j+1}) = t_{S'}(i_j) = t_{S'}(i_{j+1}) + l_{i_j}$ as schedule S' puts job i_{j+1} before job i_j

  This implies that $t_S(i_{j+1}) - d_{i_{j+1}} > t_{S'}(i_{j+1}) - d_{i_{j+1}}$ and $t_S(i_{j+1}) - d_{i_{j+1}} > t_{S'}(i_j) - d_{i_j}$ as $d_{i_j} > d_{i_{j+1}}$. Thus, the lateness of job i_{j+1} for schedule S is greater than the lateness of both job i_j and i_{j+1} for schedule S'