CSE 421: Introduction to Algorithms

Complexity and Representative Problems

Paul Beame

Measuring efficiency: The RAM model

- RAM = Random Access Machine
- Time = # of instructions executed in an ideal assembly language
  - each simple operation (+, -, *, =, if, call) takes one time step
  - each memory access takes one time step

Complexity analysis

- Problem size $N$
  - Worst-case complexity: max # steps algorithm takes on any input of size $N$
  - Best-case complexity: min # steps algorithm takes on any input of size $N$
  - Average-case complexity: avg # steps algorithm takes on inputs of size $N$

Stable Matching

- Problem size
  - $N = 2^n$ words
    - $2n$ people each with a preference list of length $n$
    - $2n \log n$ bits
    - specifying an ordering for each preference list takes $\log n$ bits
  - Brute force algorithm
    - Try all $n!$ possible matchings
  - Gale-Shapley Algorithm
    - $n^2$ iterations, each costing constant time
      - For each man an array listing the women in preference order
      - For each woman an array listing the preferences indexed by the names of the men
      - An array listing the current partner (if any) for each woman
      - An array listing the preference index of the last woman each man proposed to (if any)

Complexity

- The complexity of an algorithm associates a number $T(N)$, the best/worst/average-case time the algorithm takes, with each problem size $N$.
- Mathematically,
  - $T$ is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

Efficient = Polynomial Time

- Polynomial time
  - Running time $T(N) \leq cN^k + d$ for some $c,d,k > 0$
- Why polynomial time?
  - If problem size grows by at most a constant factor then so does the running time
    - E.g. $T(2N) \leq c(2N)^k + d \leq 2^k(cN^k + d)$
    - Polynomial-time is exactly the set of running times that have this property
  - Typical running times are small degree polynomials, mostly less than $N^3$, at worst $N^6$, not $N^{100}$
Given two positive functions $f$ and $g$
- $f(N)$ is $O(g(N))$ iff there is a constant $c > 0$ so that $f(N)$ is eventually always $\leq c g(N)$
- $f(N)$ is $o(g(N))$ iff the ratio $f(N)/g(N)$ goes to 0 as $N$ gets large
- $f(N)$ is $\Omega(g(N))$ iff there is a constant $\varepsilon > 0$ so that $f(N)$ is $\geq \varepsilon g(N)$ for infinitely many values of $N$
- $f(N)$ is $\Theta(g(N))$ iff $f(N)$ is $O(g(N))$ and $f(N)$ is $\Omega(g(N))$

Note: The definition of $\Omega$ is the same as "$f(N)\text{ is not } o(g(N))\)"

5 Representative Problems

Interval Scheduling
- Single resource
- Reservation requests
  - Of form "Can I reserve it from start time $s$ to finish time $f$?"
  - $s < f$
  - **Find**: maximum number of requests that can be scheduled so that no two reservations have the resource at the same time

Formally
- Requests $1, 2, \ldots, n$
  - request $i$ has start time $s_i$ and finish time $f_i \geq s_i$
  - Requests $i$ and $j$ are **compatible** iff either
    - request $i$ is for a time entirely before request $j$
      - $f_i \leq s_j$
    - or, request $j$ is for a time entirely before request $i$
      - $f_j \leq s_i$
  - Set $A$ of requests is **compatible** iff every pair of requests $i, j$: $A$, $i, j$ is compatible
- **Goal**: Find maximum size subset $A$ of compatible requests
Interval Scheduling

- We shall see that an optimal solution can be found using a “greedy algorithm”
  - Myopic kind of algorithm that seems to have no look-ahead
  - These algorithms only work when the problem has a special kind of structure
  - When they do work they are typically very efficient

Weighted Interval Scheduling

- Same problem as interval scheduling except that each request $i$ also has an associated value or weight $w_i$
  - $w_i$ might be
    - amount of money we get from renting out the resource for that time period
    - amount of time the resource is being used
  - Goal: Find compatible subset $A$ of requests with maximum total weight

Weighted Interval Scheduling

- Input. Set of jobs with start times, finish times, and weights.
  - Goal. Find maximum weight subset of mutually compatible jobs.

Weighted Interval Scheduling

- Ordinary interval scheduling is a special case of this problem
  - Take all $w_i = 1$
  - Problem is quite different though
    - E.g. one weight might dwarf all others
    - “Greedy algorithms” don’t work
  - Solution: “Dynamic Programming”
    - builds up optimal solutions from smaller problems using a compact table to store them

Bipartite Matching

- A graph $G = (V, E)$ is bipartite iff
  - $V$ consists of two disjoint pieces $X$ and $Y$ such that every edge $e$ in $E$ is of the form $(x, y)$ where $x \in X$ and $y \in Y$
  - Similar to stable matching situation but in that case all possible edges were present
  - $M \subseteq E$ is a matching in $G$ iff no two edges in $M$ share a vertex
  - Goal: Find a matching $M$ in $G$ of maximum possible size

Bipartite Matching

- Input. Bipartite graph.
  - Goal. Find maximum cardinality matching.
Bipartite Matching

- Models assignment problems
  - $X$ represents jobs, $Y$ represents machines
  - $X$ represents professors, $Y$ represents courses
- If $|X| = |Y| = n$
  - $G$ has perfect matching if maximum matching has size $n$
- Solution: polynomial-time algorithm using “augmentation” technique
  - Also used for solving more general class of network flow problems

Independent Set

- Given a graph $G = (V, E)$
  - A set $I \subseteq V$ is independent iff no two nodes in $I$ are joined by an edge
- Goal: Find an independent subset $I$ in $G$ of maximum possible size
  - Models conflicts and mutual exclusion

Independent Set

- Input. Graph.
- Goal. Find maximum cardinality independent set.

Independent Set

- Generalizes
  - Interval Scheduling
    - Vertices in the graph are the requests
    - Vertices are joined by an edge if they are not compatible
  - Bipartite Matching
    - Given bipartite graph $G = (V, E)$ create new graph $G' = (V', E')$ where
      - $V' = E$
      - Two elements of $V'$ (which are edges in $G$) are joined if they share an endpoint in $G$

Independent Set

- No polynomial-time algorithm is known
  - But to convince someone that there was a large independent set all you’d need to do is show it to them
    - They can easily convince themselves that the set is large enough and independent
    - Convincing someone that there isn’t one seems much harder
- We will show that Independent Set is NP-complete
  - Class of all the hardest problems that have the property above
Competitive Facility Location

- Two players competing for market share in a geographic area
  - e.g. McDonald’s, Burger King

- Rules:
  - Region is divided into $n$ zones, $1, \ldots, n$
  - Each zone $i$ has a value $b_i$
  - Revenue derived from opening franchise in that zone
  - No adjacent zones may contain a franchise
  - i.e., zoning regulations limit density
  - Players alternate opening franchises

- Find: Given a target total value $B$ is there a strategy for the second player that always achieves $\geq B$?

Model geography by

- A graph $G=(V,E)$ where
  - $V$ is the set $\{1, \ldots, n\}$ of zones
  - $E$ is the set of pairs $(i,j)$ such that $i$ and $j$ are adjacent zones

Observe:

- The set of zones with franchises will form an independent set in $G$

Checking that a strategy is good seems hard

- You’d have to worry about all possible responses at each round!
- a giant search tree of possibilities

Problem is PSPACE-complete

- Likely strictly harder than NP-complete problems
- PSPACE-complete problems include
  - Game-playing problems such as $n \times n$ chess and checkers
  - Logic problems such as whether quantified boolean expressions are always true
  - Verification problems for finite automata

Target $B = 20$ achievable?

What about $B = 25$?

Five Representative Problems

- Variations on a theme: independent set.
  - Interval scheduling: $n \log n$ greedy algorithm.
  - Weighted interval scheduling: $n \log n$ dynamic programming algorithm.
  - Bipartite matching: $n^2$ max-flow based algorithm.
  - Independent set: NP-complete.
  - Competitive facility location: PSPACE-complete.