Randomized Algorithms and other selected topics

CMSC 451, Summer 2009

Closest Pair of Points Randomized Algorithm

**Closest pair.** Given n points in the plane, find a pair with smallest Euclidean distance between them.

**Consider points in the unit square.**
- No loss of generality - scale the points in linear time.

**Basic algorithm outline:**
- Consider the points in some random order p₁, p₂, ..., pₙ
- Maintain the current value of δ = distance between closest pair
- When considering a new point, look in the vicinity to see if any previously considered points are < δ away. Update δ if necessary.

**How do we look for points in the vicinity?**
Closest Pair of Points Randomized Algorithm

Basic algorithm outline:
- Consider the points in some random order \( p_1, p_2, \ldots, p_n \)
- Maintain the current value of \( \delta = \) distance between closest pair
- When considering a new point, look in the vicinity to see if any previously considered points are \( < \delta \) away. Update \( \delta \) if necessary.

Consider the algorithm in stages:
- The closest pair of points remains constant
- Each stage either verifies that \( \delta \) is the minimum distance or finds a pair of points with separating distance less than \( \delta \). The newly found distance becomes the \( \delta \) value for the next stage.

How many stages are there? It depends on the random ordering.
- Minimum: 1. \( p_1 \) and \( p_2 \) are the closest pair.
- Maximum: \( n-1 \). Adding a new point always decreases \( \delta \).
- Will show: expected running time is within a constant factor of the minimum case!

How do we look for points in the vicinity?

Main points:
If \( p \) is involved in the closest pair, then the other point lies in a close subsquare.

This defines a 5x5 grid around the point.

If \( p \) and \( q \) are in the same subsquare, \( d(p,q) < \delta \)
Closest Pair of Points Randomized Algorithm

How do we look for points in the vicinity?

In a single stage (i.e. for a single value of $\delta$):

- for each point considered, keep track of the subsquare containing it
- when a new point is considered, determine its subsquare and check the neighboring subsquares (each of these subsquares contains at most one point)

Note: this is similar in flavor to the original divide and conquer algorithm's combine step.

Closest Pair of Points Randomized Algorithm

How do we look for points in the vicinity?

In a single stage (i.e. for a single value of $\delta$):

- for each point considered, keep track of the subsquare containing it
- when a new point is considered, determine its subsquare and check the neighboring subsquares (each of these subsquares contains at most one point)
- look up subsquares in a dictionary that stores points indexed by the subsquare containing them (e.g. a hash table).
- if a neighboring subsquare contains a point at distance $< \delta$, update $\delta$ and the dictionary

Note: this is similar in flavor to the original divide and conquer algorithm's combine step.
Closest Pair of Points Randomized Algorithm [Khuller, Matias '95]

RandomizedClosestPair(S) {
    order the points randomly p₁, p₂, ..., pₙ
    let δ denote the minimum distance so far
    initialize δ = d(p₁, p₂)
    make dictionary for subsquares of length δ/2
    for i=1 to n
        find the subsquare sᵢ containing pᵢ
        lookup the 25 subsquares S’ close to pᵢ
        for each p’ in S’
            if d(pᵢ, p’) < δ
                delete the current dictionary
                δ’ = d(pᵢ, p’)
                make dictionary for length δ’/2
                for each pⱼ for j=1 to i
                    find the subsquare sⱼ containing pⱼ
                    insert sⱼ into the new dictionary
            else
                insert pᵢ into the current dictionary
    return the points associated with δ
}

Closest Pair of Points Randomized Algorithm: Time Analysis

RandomizedClosestPair(S) {
    order the points randomly p₁, p₂, ..., pₙ
    let δ denote the minimum distance so far
    initialize δ = d(p₁, p₂)
    make dictionary for subsquares of length δ/2
    for i=1 to n
        find the subsquare sᵢ containing pᵢ
        lookup the 25 subsquares S’ close to pᵢ
        for each p’ in S’
            if d(pᵢ, p’) < δ
                delete the current dictionary
                δ’ = d(pᵢ, p’)
                make dictionary for length δ’/2
                for each pⱼ for j=1 to i
                    find the subsquare sⱼ containing pⱼ
                    insert sⱼ into the new dictionary
            else
                insert pᵢ into the current dictionary
        return the points associated with δ
}
Closest Pair of Points Randomized Algorithm: Time Analysis

How many insert operations can we expect?

Cost of closest pair update operations:
- in iteration $i$, causes $i$ inserts
- large $i$ causes a large cost

Will show: the expected number of insert operations is $O(n)$

Intuition: as the cost of updates increases, an update becomes less likely

define $X_i = 1$ if the $i^{th}$ point causes $\delta$ to change and 0 otherwise

Total number of insert operations = $n + \sum_i iX_i$

Bounding $Pr[X_i = 1]$:
Consider the first $i$ points (as randomly ordered)
Assume $p$ and $q$ are the points that define $\delta$ for this set
The minimum distance only decreases when considering $p$ or $q$
The probability that $p_i$ is $p$ or $q$ is $2/i$

$Pr[X_i = 1] \leq 2/i$

$E[number of insert ops] = n + \sum_i iX_i \leq n + \sum_i i (2/i) = n + 2n = 3n = O(n)$

Total Time: $O(n)$ time $+$ $O(n)$ dictionary operations
Closest Pair of Points Randomized Algorithm: Time Analysis

**Total Time:** $O(n)$ time + $O(n)$ dictionary operations

**Dictionary operations - Hash Table**
- Make dictionary $O(1)$
- Lookup $O(1)$
- Insert $O(1)$

$O(n)$ total time using hashing

9 Are there harder problems?
**Geography Game**

*Geography.* Alice names capital city $c$ of country she is in. Bob names a capital city $c'$ that starts with the letter on which $c$ ends. Alice and Bob repeat this game until one player is unable to continue. Does Alice have a forced win?

*Ex.* Budapest → Tokyo → Ottawa → Ankara → Amsterdam → Moscow → Washington → Nairobi → …

*Geography on graphs.* Given a directed graph $G = (V, E)$ and a start node $s$, two players alternate turns by following, if possible, an edge out of the current node to an unvisited node. Can first player guarantee to make the last legal move?

*Remark.* Some problems (especially involving 2-player games and AI) defy classification according to P, EXPTIME, NP, and NP-complete.

---

9.1 PSPACE
PSPACE

**P.** Decision problems solvable in polynomial time.

**PSPACE.** Decision problems solvable in polynomial space.

**Observation.** \( P \subseteq \text{PSPACE}. \)
- Poly-time algorithm can consume only polynomial space.

---

**PSPACE**

- **Binary counter.** Count from 0 to \( 2^n - 1 \) in binary.
- **Algorithm.** Use \( n \) bit odometer.

**Claim.** 3-SAT is in PSPACE.
**Pf.**
- Enumerate all \( 2^n \) possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses.

**Theorem.** NP \( \subseteq \) PSPACE.
**Pf.** Consider arbitrary problem \( Y \) in NP.
- Since \( Y \leq_p \text{3-SAT} \), there exists algorithm that solves \( Y \) in poly-time plus polynomial number of calls to 3-SAT black box.
- Can implement black box in poly-space.
Some PSPACE Problems
Quantified Satisfiability

**QSAT.** Let \( \Phi(x_1, \ldots, x_n) \) be a Boolean CNF formula. Is the following propositional formula true?

\[
\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \forall x_{n-1} \exists x_n \Phi(x_1, \ldots, x_n)
\]

Assume \( n \) is odd.

**Intuition.** Amy picks truth value for \( x_1 \), then Bob for \( x_2 \), then Amy for \( x_3 \), and so on. Can Amy satisfy \( \Phi \) no matter what Bob does?

**Ex.** \((x_1 \lor x_2) \land (x_2 \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor x_3)\)

Yes. Amy sets \( x_1 \) true; Bob sets \( x_2 \); Amy sets \( x_3 \) to be same as \( x_2 \).

**Ex.** \((x_1 \lor x_2) \land (x_2 \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor x_3)\)

No. If Amy sets \( x_1 \) false; Bob sets \( x_2 \) false; Amy loses; if Amy sets \( x_1 \) true; Bob sets \( x_2 \) true; Amy loses.

**QSAT is in PSPACE**

**Theorem.** QSAT \( \in \) PSPACE.

**Pf.** Recursively try all possibilities.

- Only need one bit of information from each subproblem.
- Amount of space is proportional to depth of function call stack.

\[
\begin{align*}
\exists x_1 & \lor \exists x_2 \\
\exists x_3 & \lor \exists x_4 \\
\exists x_5 & \lor \exists x_6 \\
\exists x_7 & \lor \exists x_8 \\
\end{align*}
\]
15-Puzzle

**8-puzzle, 15-puzzle.** [Sam Loyd 1870s]

- **Board:** 3-by-3 grid of tiles labeled 1-8.
- **Legal move:** slide neighboring tile into blank (white) square.
- **Find sequence of legal moves to transform initial configuration into goal configuration.**

**Initial configuration**

```
1 2 3
4 5 6
8 7 _
```

**Goal configuration**

```
1 2 3
4 5 6
8 7 8
```

**Planning Problem**

**Conditions.** Set $C = \{ C_1, ..., C_n \}$.

**Initial configuration.** Subset $c_0 \subseteq C$ of conditions initially satisfied.

**Goal configuration.** Subset $c^* \subseteq C$ of conditions we seek to satisfy.

**Operators.** Set $O = \{ O_1, ..., O_k \}$.

- To invoke operator $O_i$, must satisfy certain prereq conditions.
- After invoking $O_i$, certain conditions become true, and certain conditions become false.

**Planning.** Is it possible to apply sequence of operators to get from initial configuration to goal configuration?

**Examples.**

- 15-puzzle.
- Rubik’s cube.
- Logistical operations to move people, equipment, and materials.
Competitive Facility Location

**Input.** Graph with positive edge weights, and target $B$.

**Game.** Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors has been selected.

**Competitive facility location.** Can second player guarantee at least $B$ units of profit?

Yes if $B = 20$; no if $B = 25$.

12 Local Search
Coping With NP-Hardness

**Q.** Suppose I need to solve an NP-hard problem. What should I do?

**A.** Theory says you’re unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

12.1 Landscape of an Optimization Problem
Gradient Descent: Vertex Cover

**VERTEX-COVER.** Given a graph $G = (V, E)$, find a subset of nodes $S$ of minimal cardinality such that for each $u\cdot v$ in $E$, either $u$ or $v$ (or both) are in $S$.

**Neighbor relation.** $S \sim S'$ if $S'$ can be obtained from $S$ by adding or deleting a single node. Each vertex cover $S$ has at most $n$ neighbors.

**Gradient descent.** Start with $S = V$. If there is a neighbor $S'$ that is a vertex cover and has lower cardinality, replace $S$ with $S'$.

**Remark.** Algorithm terminates after at most $n$ steps since each update decreases the size of the cover by one.

---

Gradient Descent: Vertex Cover

**Local optimum.** No neighbor is strictly better.

- Optimum: center node only
  - Local optimum: all other nodes
- Optimum: all nodes on left side
  - Local optimum: all nodes on right side
- Optimum: even nodes
  - Local optimum: omit every third node
Local Search

Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbor relation. Let $S \sim S'$ be a neighbor relation for the problem.

Gradient descent. Let $S$ denote current solution. If there is a neighbor $S'$ of $S$ with strictly lower cost, replace $S$ with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.

A funnel

A jagged funnel

Metropolis Algorithm

Metropolis algorithm idea. [Metropolis, Rosenbluth, Rosenbluth, Teller, Teller 1953]

- Simulate behavior of a physical system according to principles of statistical mechanics.
- Globally biased toward "downhill" steps, but occasionally makes "uphill" steps to break out of local minima.

Metropolis algorithm.
- Given a fixed temperature $T$, maintain current state $S$.
- Randomly perturb current state $S$ to new state $S' \in \mathbb{N}(S)$.
- If $E(S') \leq E(S)$, update current state to $S'$.
  Otherwise, update current state to $S'$ with probability $e^{\Delta E / (kT)}$, where $\Delta E = E(S') - E(S) > 0$.

Intuition. Simulation spends roughly the right amount of time in each state.
Simulated Annealing

Simulated annealing.
- T large $\Rightarrow$ probability of accepting an uphill move is large.
- T small $\Rightarrow$ uphill moves are almost never accepted.
- Idea: turn knob to control $T$.
- Cooling schedule: $T = T(i)$ at iteration $i$.

Physical analog.
- Take solid and raise it to high temperature, we do not expect it to maintain a nice crystal structure.
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal either.
- Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures.

Epilogue: Algorithms that Run Forever
What is success?

Rubik’s cube

Tetris

Online / Streaming Algorithms

Goal: keep up with the state of the system.

Main Applications:
- Network routing
- Internet traffic
- Sensor networks

Problem: Packet Routing
- A switch has n input links and n output links
- Packets arrive to an input link of a switch
- Each packet has a header saying which output link it needs to depart on
- Time moves in discrete steps – at each time step one packet can arrive at a single input link and one can leave through a single output link