Randomized Algorithms

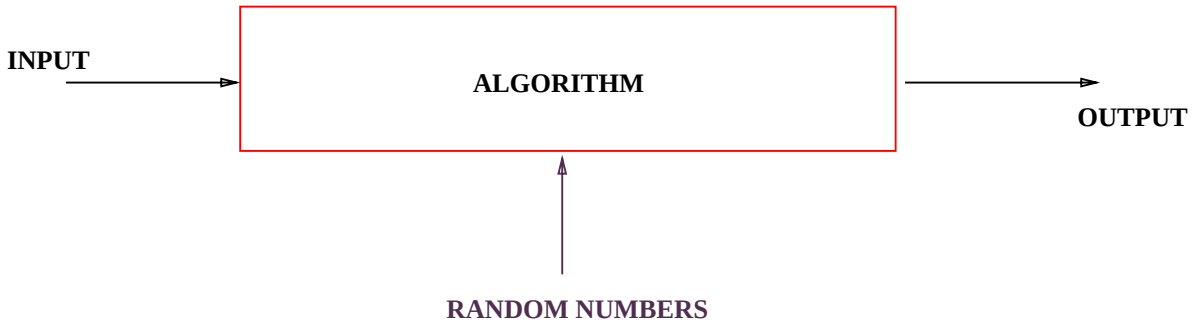
Prabhakar Raghavan
IBM Almaden Research Center
San Jose, CA.

– Typeset by FoilTEX –

Deterministic Algorithms

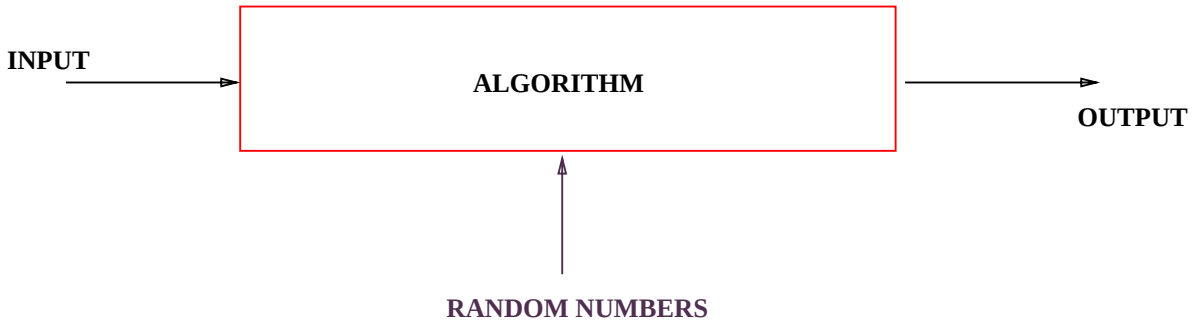
Goal: To prove that the algorithm solves the problem *correctly* (always) and *quickly* (typically, the number of steps should be polynomial in the size of the input).

Randomized Algorithms



- In addition to input, algorithm takes a source of random numbers and makes random choices during execution.
- Behavior can vary even on a fixed input.

Randomized Algorithms



- Design algorithm + analysis to show that this behavior is likely to be good, on *every input*.
  (The likelihood is over the random numbers only.)

Not to be confused with the Probabilistic Analysis of Algorithms

- Here the *input* is assumed to be from a probability distribution.
- Show that the algorithm works for most inputs.

Monte Carlo and Las Vegas

A Monte Carlo algorithm runs for a fixed number of steps, and produces an answer that is correct with probability $\geq 1/3$.

A Las Vegas algorithm always produces the correct answer; its running time is a random variable whose expectation is bounded (say by a polynomial).

Monte Carlo and Las Vegas

These probabilities/expectations are only over the random choices made by the algorithm – *independent of the input.*

Thus independent repetitions of Monte Carlo algorithms drive down the failure probability exponentially.

Advantages of randomized algorithms

- Simplicity
- Performance

For many problems, a randomized algorithm is the simplest, the fastest, or both.

Scope

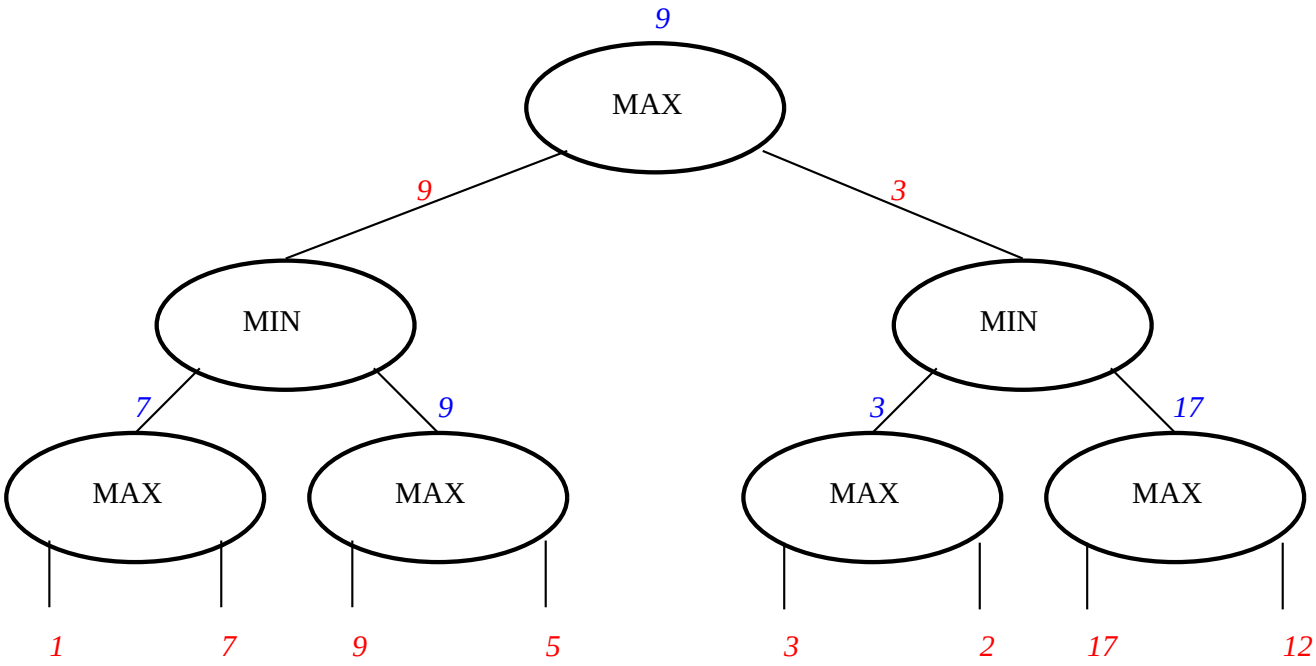
- Number-theoretic algorithms: Primality testing (Monte Carlo).
- Data structures: Sorting, order statistics, searching, computational geometry.
- Algebraic identities: Polynomial and matrix identity verification. Interactive proof systems.

- Mathematical programming: Faster algorithms for linear programming. Rounding linear program solutions to integer program solutions.
- Graph algorithms: Minimum spanning trees, shortest paths, minimum cuts.
- Counting and enumeration: Matrix permanent. Counting combinatorial structures.

- Parallel and distributed computing: Deadlock avoidance, distributed consensus.
- Probabilistic existence proofs: Show that a combinatorial object arises with non-zero probability among objects drawn from a suitable probability space.

- Derandomization: First devise a randomized algorithm, then argue that it can be "derandomized" to yield a deterministic algorithm.

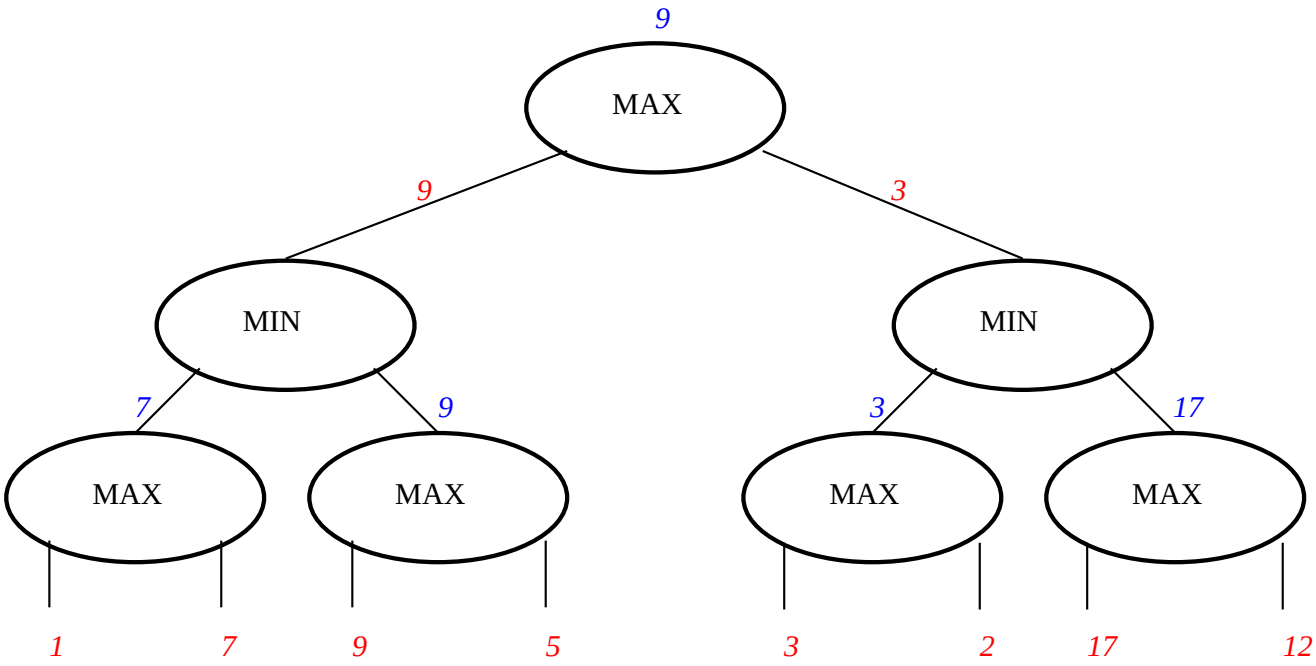
Game-tree evaluation



- Tree with alternating MAX/MIN nodes.
  Each leaf has a real value.

- Goal: To evaluate the root.

Game-tree evaluation



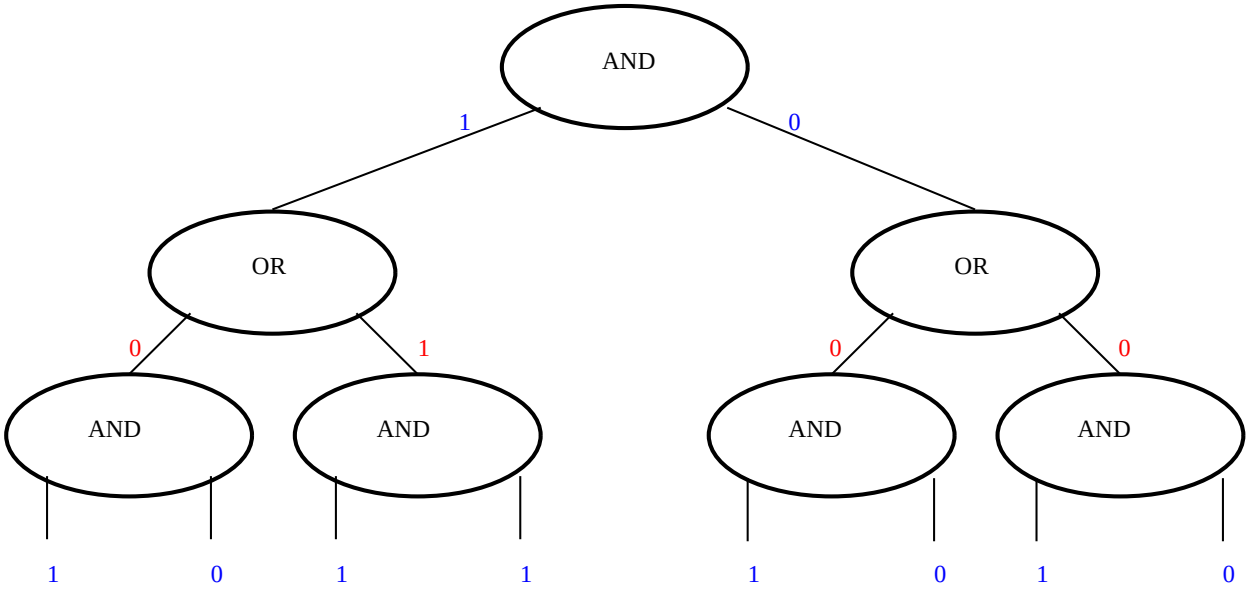
- Cost: The number of leaves inspected.
- Algorithm: Computes leaf-inspection sequence.

Simple special case

- Uniform binary tree, height h, with $n = 2^h$ leaves.
- All values are boolean, so MAX$\rightarrow$OR and MIN$\rightarrow$AND.

Exercise: Every deterministic algorithm can be forced to read n leaves.

Randomized algorithm



Evaluate (recursively) a random child of the current node.

If this does not determine the value of the current node, evaluate the other child.

Analysis of tree evaluation

AND-OR trees of depth $2k$ (k levels of AND and k levels of OR on any path); will prove by induction on k.

Expected cost for an OR subtree evaluating to 1:

$$\frac{1}{2} \cdot 3^{k-1} + \frac{1}{2} \cdot 2 \cdot 3^{k-1} = \frac{3}{2} \cdot 3^{k-1}.$$

Analysis of tree evaluation

For an OR subtree evaluating to 0, we must evaluate both of its subtrees, incurring cost $2(3^{k-1})$.

Now consider the root AND:
(a) If it is 1, both OR subtrees must evaluate to 1; expected cost paid is twice above, $= 3^k$.
(b) If it is 0, at least one OR subtree evaluates to 0.

Game tree analysis

Expected cost for root AND evaluating to 0:

$$2 \cdot 3^{k-1} + \frac{13}{22} 3^{k-1} \leq 3^k.$$

Number of leaves in tree $= 4^k$.

Thus expected cost of randomized algorithm $\leq n^{0.793}$

Lower bounds and the minimax principle

Basic Idea: For a lower bound on the performance of all randomized algorithms for a problem, derive instead a lower bound for any deterministic algorithm for the problem *when the inputs are drawn from a probability distribution* (of your choice).

Let $\mathcal{I}$ denote the set of instances of the problem, and $\mathcal{A}$ the set of deterministic algorithms for the problem. Any randomized algorithm can be viewed as a probability distribution on the algorithms in $\mathcal{A}$.

Minimax Principle

For all distributions p over $\mathcal{I}$ and q over $\mathcal{A}$,

$$\min_{A \in \mathcal{A}} \mathbf{E}[C(I_p, A)] \leq \max_{I \in \mathcal{I}} \mathbf{E}[C(I, A_q)].$$

Lower bound for game tree evaluation

We will specify a probability distribution on instances ($0/1$ values for the leaves) and lower bound the expected running time of any deterministic algorithm.

NOR trees instead

Exercise: Show that a balanced AND-OR tree of even depth is equivalent to a tree of the same depth, all of whose nodes are NOR nodes.

We will show the lower bound for a NOR tree with n leaves.

The input distribution

Let $p = (3 - \sqrt{5})/2$.

Claim: The value of any node is 1 with probability p, independent of all other nodes at the same level.

Claim: Any deterministic algorithm may as well determine the value of one sub-tree of a node, before inspecting any leaf of its sibling sub-tree.

The Analysis

Let $W(h)$ be the expected number of leaves it inspects in determining the value of a node at distance h from the leaves.

Clearly

$$W(h) = W(h-1)+(1-p)W(h-1).$$

Letting $h = \log_2 n$, this gives a lower bound of $n^{0.694}$.

Exercise: Why is this lower bound weak?

The 2-SAT Problem

Given a set of boolean clauses in CNF each containing two literals, find a satisfying assignment if one exists.

$$(x_1 + x_4)(\bar{x}_4 + x_3)(x_2 + \bar{x}_3)\cdots$$

- Start with any tentative assignment
- If there is an unsatisfied clause, pick one of its two literals at random, and flip it.

- If no solution found in $2n^2$ steps, declare "none exists".

Monte Carlo: If a solution exists, will find it with probability $> 1/2$.
If not, will always declare "none exists".

Random Walk Analysis

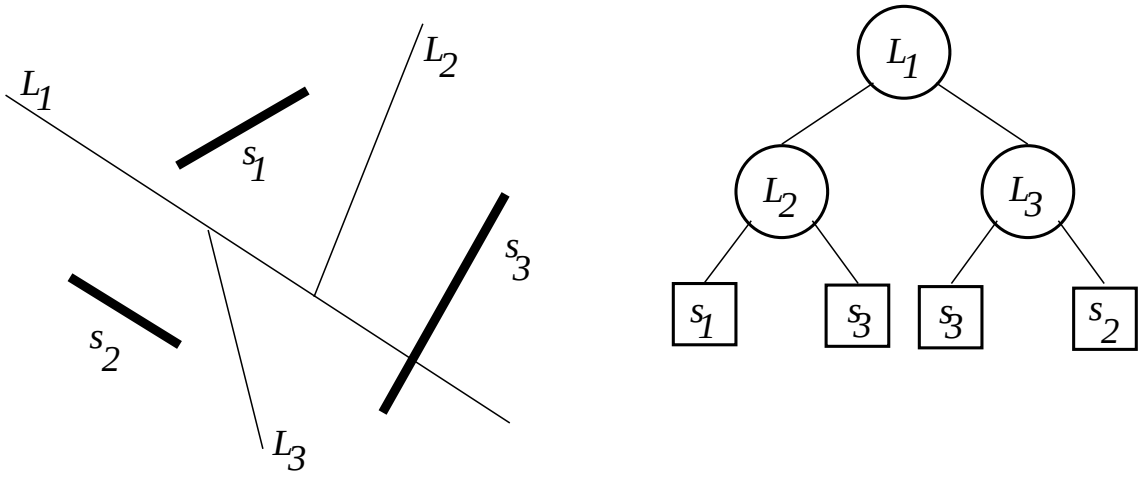
What is the probability of missing a satisfying assignment in $2n^2$ steps?

- Fix any *particular* satisfying assignment A.
- Consider the number of literals on which the algorithm's tentative assignment agrees with A.

- This is a random walk on the integers that increases with probability at least $1/2$ at each step.

Expected time to find an assignment $< n^2$.
Markov's inequality $\Rightarrow$ probability of missing an assignment in $2n^2$ steps is $< 1/2$.

Binary planar partitions



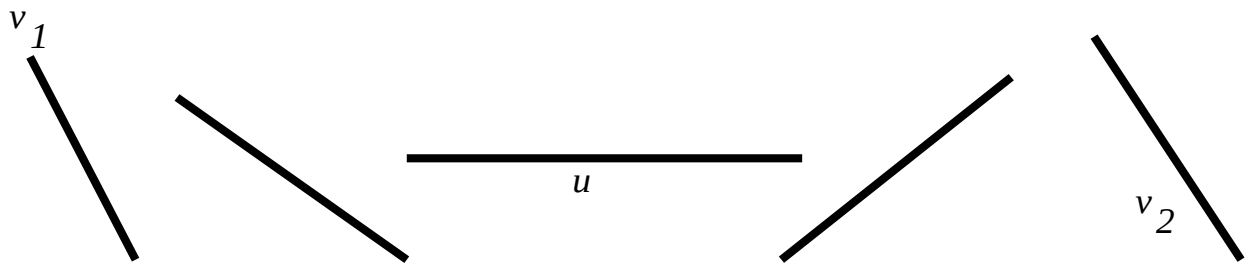
Given n non-intersecting line-segments in the plane, build a small linear decision tree that has (pieces of) at most one segment in each cell.

Autopartitions

Choose the input segments in random order.
Extend the chosen segment (if necessary) to cut any cells where it is not "alone".

The expected size of the resulting tree is $\leq n + 2nH_n$.

Analysis of autopartition size



Define $l(u)$ to be the line containing segment u.

$u \dashv v$ denotes $l(u)$ cuts v in the constructed tree.

$index(u, v)$ is i if $l(u)$ intersects $i - 1$ other segments before hitting v.

Autopartitions

$$\mathbf{Pr}[u \dashv v] \leq 1/(index(u,v)+1).$$

Thus the expected size of the tree constructed is

$$n + \sum_u \sum_{v \neq u} \mathbf{Pr}[u \dashv v] \leq$$

$$n + \sum_u \sum_{v \neq u} \frac{1}{index(u,v)+1}$$

For any u, i, $index(u, v) = i$ for at most two v; thus the above sum is $\leq n + 2nH_n$.

It follows that for any n segments in the plane, *there always exists* an autopartition of size $O(n \log n)$.

Matrix product verification

Suppose A, B and C are $n \times n$ matrices with entries from a finite field $\mathcal{F}$. We wish to verify whether $AB = C$.

Multiplying out takes about n^3 steps – this can be cut to $O(n^{2.38})$ steps using sophisticated algorithms.

Simple randomized algorithm

- Pick an n-vector x with entries randomly drawn from $\mathcal{F}$.
- Compute $z = A(Bx)$.
  If $z = Cx$ output $AB = C$,
  else output $AB \neq C$.

Takes $O(n^2)$ steps.

Simple randomized algorithm

If $AB = C$, will always output $AB = C$.

If $AB \neq C$, will output $AB = C$ with probability at most $1/|\mathcal{F}|$.

Sources

R.M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34:165–201, 1991.

R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

D. J. A. Welsh. Randomised algorithms. *Discrete Applied Mathematics*, 5:133–145, 1983.