Chapter 4. BASIC TECHNIQUES

In this chapter we present a new way how to see randomized algorithms and several basic techniques how to design and analyse randomized algorithms: Especially we deal with:

- **Linearity of expectations + indicator variables.**
- **Game theory methods for lower bounds of randomized algorithms.**

PROLOGUE

How to see basics of

deterministic, randomized and quantum computations

MATHEMATICAL VIEWS of COMPUTATION

Let us consider an n bit systems.

To describe a deterministic computation we need to specify

an initial state - by an n-bit string

and an evolution, which can be described, by a vector of the length 2^n the elements of which are n-bit strings.

A computation step is then an application of the evolution mapping to the current state being represented by an n-bit string.

However, for any a bit significant task, the number of bits needed, $n2^n$, is much too much big. The task of programming is then to replace an application of such an enormously huge mapping by a short circuit/program.

To describe a randomized computation we need;

to specify an initial probability distribution on all n-bit strings, what can be done by an vector of length 2^n, indexed by n-bit strings, the elements of which are non-negative numbers that sum up to 1. and to specify a randomized evolution, which has to be done, in case of a homogeneous evolution, by an $2^n \times 2^n$ matrix A of conditional probabilities for obtaining a new state/string from an old state/string. The matrix A has to be stochastic - all columns have to sum up to one and $A[i, j]$ is a probability of going from a string representing j to a string representing i.

To perform a computation step one then needs to multiply by A the 2^n-elemnts vector specifying the current probability distribution on 2^n states.

However, for any nontrivial problem the number 2^n is larger than the number of particles in the universe and therefore the task of programming is to design a small circuit that can actually implement such a multiplication by a matrix of an enormous size.

In case of quantum computation on n quantum bits,

the initial state is given by an 2^n vector of complex numbers (probability amplitudes) the sum of the squares of which is one,

and the homogeneous quantum evolution is described by an $2^n \times 2^n$ unitary matrix of complex numbers - at which inner products of any two different columns and any two different vectors are 0.[1]

Concerning a computation step, this is again a multiplication of a vector of the probability amplitudes, representing the current state, by a very huge $2^n \times 2^n$ unitary matrix what has to be realized by a "small" quantum circuit.

[1] A matrix A is usually called unitary if its inverse matrix can be obtained from A by transposition around the main diagonal and replacement of each element by its complex conjugate.

LINEARITY OF EXPECTATIONS

A simple, but very often useful, fact is that for any random variables $X_1, X_2, \dots$ it holds

$$\mathbf{E}[\sum_i X_i] = \sum_i \mathbf{E}[X_i].$$

even if X_i are dependent and dependencies among X_i's are very complex.

Example A ship arrives at a port, and all 40 sailors on board go ashore to have fun. At night, all sailors return to the ship and, being drunk, each chooses randomly a cabin to sleep in. *What is the expected number of sailors sleeping in their own cabins?*

Solution Let X_i be a random variable, so called *(indicator variable)*, which has value 1 if the i-th sailor chooses his own cabin, and 0 otherwise.

Expected number of sailors who get to their own cabin is

$$\mathbf{E}[\sum_{i=1}^{40} X_i] = \sum_{i=1}^{40} \mathbf{E}[X_i]$$

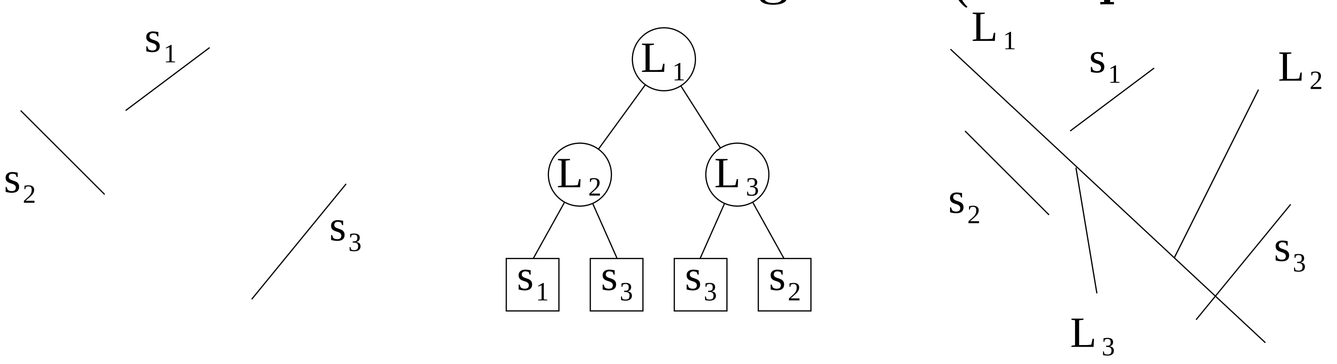
Since cabins are chosen randomly $\mathbf{E}[X_i] = \frac{1}{40}$ Therefore,

$$\mathbf{E}[\sum_{i=1}^{40} X_i] = 40.\frac{1}{40} = 1.$$

EXAMPLE

Binary partition of a set of n disjoint line segments

Problem Given a set $S = \{s_1, \ldots, s_n\}$ of non-intersecting line segments, find a binary partition of the plane such that every region contains at most one line segment (or a part of a line segment).



A binary partition is given by a binary tree + additional information. With each node v a region r_v is associated (the whole plane stands for the root) and also a line L_v intersecting r_v.

Each line L_v partitions the region r_v into two regions $r_{l(v)}$ and $r_{r(v)}$ which correspond to two children of v, namely $l(v)$ and $r(v)$.

Notation: $l(s_i)$ **denotes an infinite line-extension of the line segm.** s_i
Autopartitions use only lines from $\{l(s_1), \ldots, l(s_n)\}$**.**

Algorithm RandAuto

Input: **A set** $S = \{s_1, \ldots, s_n\}$ **of non-intersecting line segments.**

Output: **A binary autopartition** P_Π **of** S**.**

1: **Pick a permutation** Π **of** $\{1, \ldots, n\}$ **uniformly and randomly.**

2: **While there are regions** R **that contain more than one segment, choose one of them randomly and cut it with** $l(s_i)$ **where** i **is the first element in the ordering induced by** Π **such that** $l(s_i)$ **cuts the region** R**.**

Theorem **The expected size of the autopartition** P_Π **of** S**, produced by the above RandAuto algorithm is** $\theta(n \ln n)$**.**

Proof Notation (for line segments u**,** v**).**

$$index(u, v) = \begin{cases} i & \textbf{if } l(u) \textbf{ intersects } i-1 \textbf{ segments before hitting } v \\ \infty & \textbf{if } l(u) \textbf{ does not hit } v \end{cases}$$

$u \dashv v$ **is an event that** $l(u)$ **cuts** v **in the same region in the constructed autopartition.**

Probability: **Let u and v be segments, $index(u,v) = i$ and let $u_1, \ldots, u_{i-1}$ be segments line $l(u)$ intersects before hitting v.**
The event $u \dashv v$ happens, during an execution of RandPart, only if u occurs before any of $\{u_1, \ldots, u_{i-1}, v\}$ in the permutation Π. Therefore the probability that event $u \dashv v$ happens is $\frac{1}{i+1}$.

Notation: **Let $C_{u,v}$ be the indicator variable that has value 1 if $u \dashv v$ and 0 otherwise.**

$$\mathbf{E}[C_{u,v}] = \boldsymbol{Pr}[u \dashv v] = \frac{1}{index(u,v)+1}.$$

Clearly, the size of the created partition P_Π equals n plus the number of intersections due to the cuts. Its expectation value is therefore

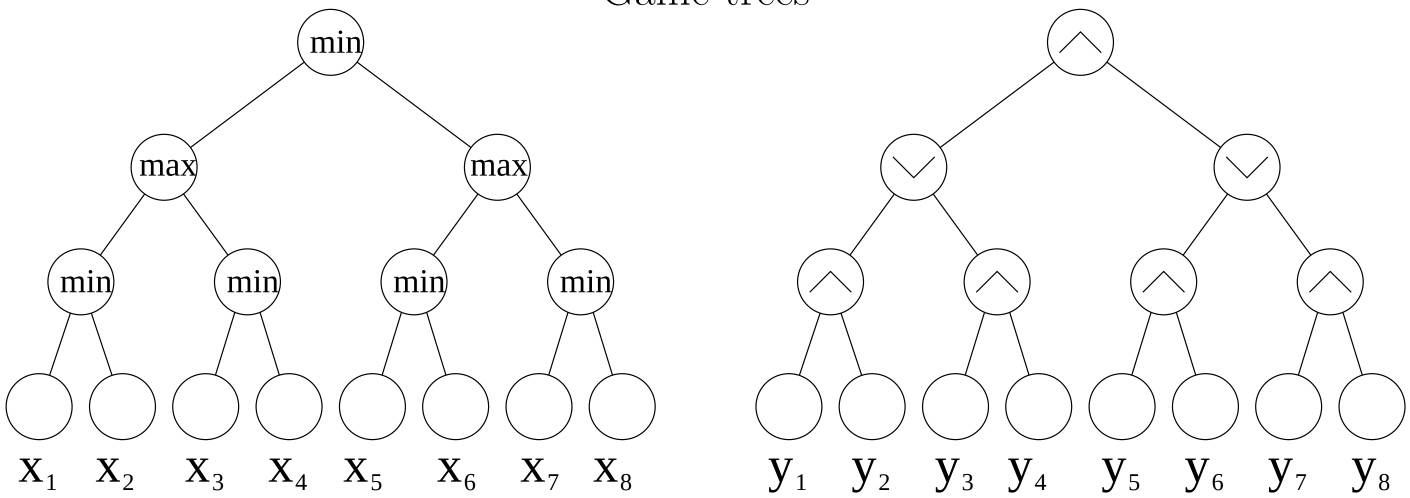
$$n + E[\sum_u \sum_{v \neq u} C_{u,v}] = n + \sum_u \sum_{v \neq u} Pr[u \dashv v] = n + \sum_u \sum_{v \neq u} \frac{1}{index(u,v)+1}.$$

For any line segment u and integer i there are at most two v, w such that $index(u,v) = index(u,w) = i$. Hence $\Sigma_{v \neq u} \frac{1}{index(u,v)+1} \leq \Sigma_{i=1}^{n-1} \frac{2}{i+1}$ and therefore

$$n + \mathbf{E}[\sum_u \sum_{v \neq u} C_{u,v}] \leq n + \sum_u \sum_{i=1}^{n-1} \frac{2}{i+1} \leq n + 2nH_n.$$

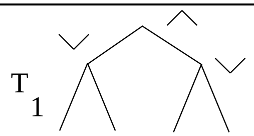
GAME TREE EVALUATION

Game trees



Evaluation of game trees plays a crucial role in AI, in various game playing programs.

> Assumption: An evaluation algorithm chooses at each step (somehow) a leaf, reads its value and performs all evaluations of internal nodes it can perform. Determine the total number of such steps.



T_k – binary game tree of depth $2k$.

The worst-case complexity of a deterministic algorithm: $n = 4^k = 2^{2k}$.

Randomized algorithm:

To evaluate an AND-node v, the algorithm chooses randomly one of its children and evaluates it.

If 1 is returned, algorithm proceeds to evaluate other subtree and returns as the value of v the value of that subtree.. If 0 is returned, algorithm returns imediately 0 for v (without evaluating other subtree).

To evaluate an OR-node v, algorithm chooses randomly one of its children and evaluates it.

If 0 is returned, algorithm proceeds to evaluate other subtree and returns as the value of v the value of the subtree. If 1 is returned, the algorithm imediately returns 1 for v.

Theorem Given any instance of T_k, the expected number of steps for the above randomized algorithm is at most 3^k.

Proof by induction:
Base step: Case $k = 1$ easy - verify by computations for all cases.
Inductive step: Assume that the expected cost of the evaluation of. any instance of T_{k-1} is at most 3^{k-1}

Consider an OR-node tree T with both children being T_{k-1}-trees. If the root of T were to return 1, at least one of its T_{k-1}-subtrees has to return 1. With probability $\frac{1}{2}$ this child is chosen first, given in average at most 3^{k-1} leaf-evaluations. With probability $\frac{1}{2}$ both subtrees are to be evaluated.
The expected cost of determining the value of T is therefore:

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

If the root of T were to return 0 both subtrees have to be evaluated, giving the cost $2 \times 3^{k-1}$.

Consider the root of T_k.

If the root evaluates to 1, both of its OR-subtrees have to evaluate to 1. The expected cost is therefore $2 \times \frac{3}{2} \times 3^{k-1} = 3^k$.

If the root evaluates to 0, at least one of the subtrees evaluates to 0. The expected cost is therefore

$$\frac{1}{2} \times 2 \times 2 \times 3^{k-1} + \frac{1}{2} \times \frac{3}{2} \times 3^{k-1} \leq 3^k = n^{\lg_4 3} = n^{0.793}.$$

Our algorithm is therefore a *Las Vegas algorithm.* Its running time (number of leaves evaluations) is: $n^{0.793}$.

BASIC CONCEPTS of CLASSICAL GAME THEORY

We consider **games with two players**, Alice and Bob. Let X and Y be nonempty sets of their (so-called pure) strategies. Mappings $p_X : X \times Y \rightarrow \mathbf{R}$ and $p_Y : X \times Y \rightarrow \mathbf{R}$ are called **payoff functions** and the quadruple (X, Y, p_X, p_Y) is called a (mathematical) **game**.

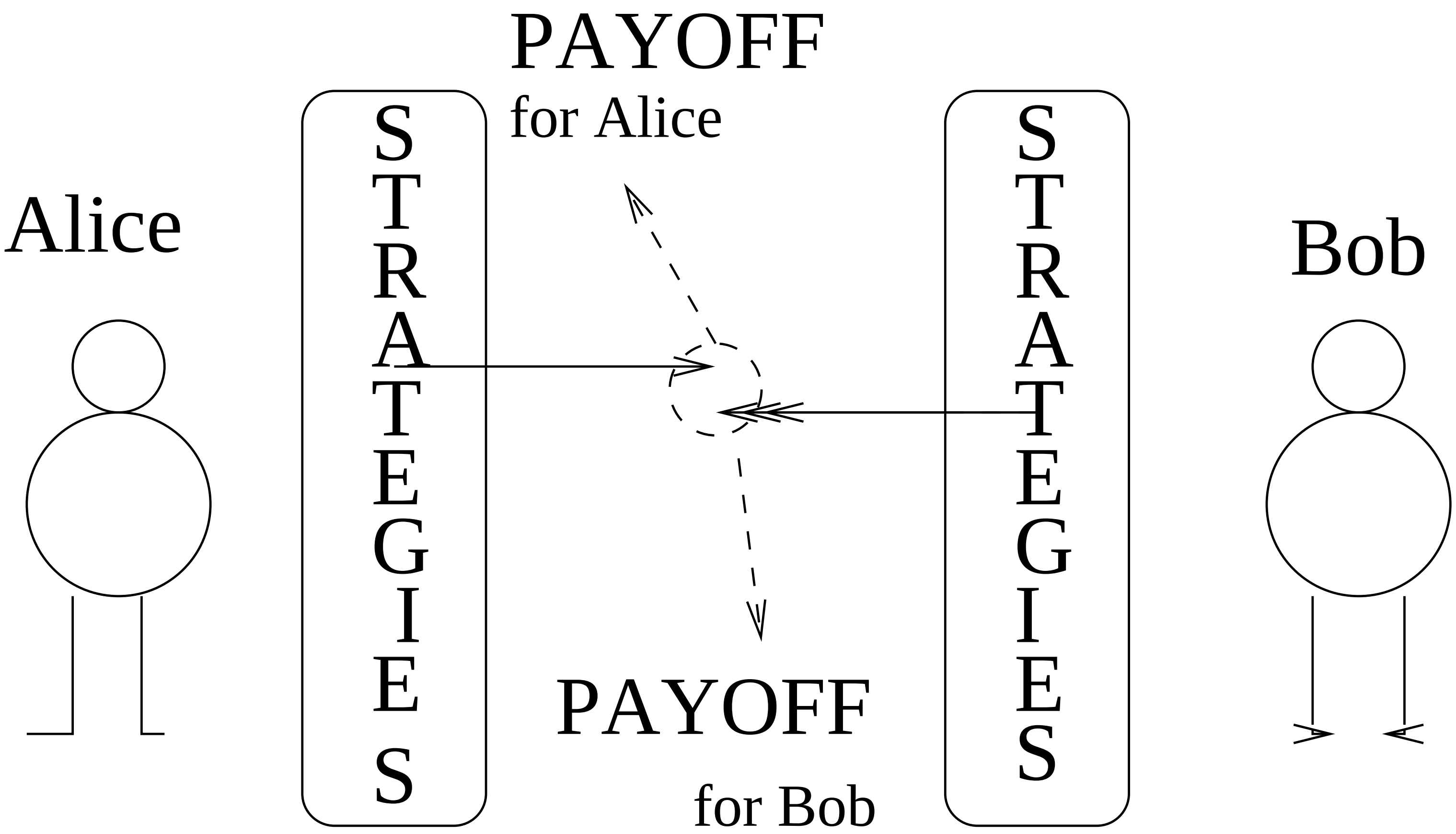
A **mixed strategy** is a probability distribution on pure strategies.

An element $(x, y) \in X \times Y$ is said to be a **Nash equilibrium** of the game (X, Y, p_X, p_Y) iff $p_X(x', y) \leq p_X(x, y)$ for $\forall x' \in X$, and $p_Y(x, y') \leq p_Y(x, y)$ **for** $\forall y' \in Y$.

Informally, Nash equilibrium is such a pair of strategies that none of the players gains by changing his/her strategy.

A game is called **zero-sum game** if $p_X(x, y) + p_Y(x, y) = 0$ for all $x \in X$ and $y \in Y$.

One of the basic result of the classical game theory is that not every two-players zero-sum game has a Nash equilibrium in the set of pure strategies, but there is always a Nash equilibrium if players follow mixed strategies.


PAYOFF
for Alice
Alice
STRATEGIES
STRATEGIES
Bob
PAYOFF
for Bob

EXAMPLE - PENNY FLIP GAME

There are two players, A (Alice) and B (Bob). They "play" as follows:

- **Alice places a penny head up in a box.**
- **Bob flips or does not flip the coin**
- **Alice flips or does not flip the coin**
- **Bob flips or does not flip the coin**

After the "game" is over, they open the box and Bob wins if the penny is head up.

It is easy to check that using pure strategies chances to win are $\frac{1}{2}$ for each player and there is no (Nash) equilibrium in the case of pure classical strategies.

However, there is equilibrium if Alice chooses its strategy with probability $\frac{1}{2}$ and Bob chooses each of the four possible strategies with probability $\frac{1}{4}$.

PRISONERS' DILEMMA

Another game that has been much studied is so-called Prisoners' dilemma.

Two prisoners, Alice and Bob, can use, independently, any of the following two strategies: **cooperate** or **defect** (not to cooperate).

The problem is that the payoff function (p_a, p_b), in millions, is a very special one:

$\frac{\text{Alice}}{\text{Bob}}$	C_A	D_A
C_B	$(3,3)$	$(5,0)$
D_B	$(0,5)$	$(1,1)$

What is the best way for Alice and Bob to proceed in order to maximize their payoffs?

A strategy s_A is called **dominant** for Alice if for any other strategy s'_A of Alice and s_B of Bob, it holds

$$P_A(s_A, s_B) \geq P_A(s'_A, s_B).$$

Clearly, defection is the dominant strategy of Alice (and also of Bob) in the case of Prisoners Dilemma game.

Prisoners Dilemma game has therefore **dominant-strategy equilibrium**

$$\begin{array}{ccc} \frac{\text{Alice}}{\text{Bob}} & C_A & D_A \\ C_B & (3,3) & (5,0) \\ D_B & (0,5) & (1,1) \end{array}$$

BATTLE of SEX GAME

Alice and Bob have to decide, independently of each other, where to spent the evening.

Alice prefers to go to opera (O), Bob wants to watch TV (T).

However, at the same time both of them prefer to be together than to be apart.

Pay-off function is given by the matrix

$$\begin{array}{ccc} & O & T \\ O & (\alpha, \beta) & (\gamma, \gamma) \\ T & (\gamma, \gamma) & (\beta, \alpha) \end{array}$$

where $\alpha > \beta > \gamma$.

what kind of strategy they should choose?

The two Nash equilibria are (O, O) and (T, T), but players are faced with tactics dilemma, because these equilibria bring them different payoffs.

COIN GAME

There are three coins: one fair, with both sides different, and two unfair, one with two heads and one with two tails.

The game proceeds as follows.

- **Alice puts coins into a black box and shakes the box.**
- **Bob picks up one coin.**
- **Alice wins if coin is unfair, otherwise Bob wins**

Clearly, in the classical case, the probability that Alice wins is $\frac{2}{3}$.

FROM GAMES to LOWER BOUNDS

for RANDOMIZED ALGORITHMS

Next goal is to present, using zero-sum games theory, a method how to prove a lower bound for the average running time of randomized algorithms for a given problem.

This techniques can be applied to algorithms that terminate for all inputs and all random choices.

TWO–PERSON ZERO–SUM GAMES – EXAMPLE

PAYOFF–MATRICES

Example

Bob

Alice		Scissors	Paper	Stone
	Scissors	0	1	-1
	Paper	-1	0	1
	Stone	1	-1	0

→ Table shows how much Bob has to pay to Alice

Any two players zero–sum game is represented by an $n \times m$ payoff–matrix M with all rows and columns summing up to 0.
Possible strategies of Alice are given by rows of M.
Possible strategies of Bob are given by columns of M.

$M_{i,j}$ is the amount paid by Bob to Alice if Alice chooses strategy i and Bob's choices strategy j.

The goal of Alice (Bob) is to maximize (minimize) her payoff.

STRATEGIES for ZERO-INFORMATION and ZERO-SUM GAMES

(Games with players having no information about their opponents strategies.)

Observe that if Alice chooses a strategy i, then she is guaranteed a payoff of $\min_j M_{ij}$ regardless of Bob's strategy.

An optimal strategy for Alice is such an i that maximises $\min_j M_{ij}$.

$$V_A = \max_i \min_j M_{ij}$$

denotes therefore the lower bound on the value of the payoff Alice gains (from Bob) when she uses an optimal strategy.

An optimal strategy for Bob is such a j that minimizes $\max_i M_{ij}$. Bob's optimal strategy ensures therefore that his payoff is at least

$$V_B = \min_j \max_i M_{ij}$$

Theorem

$$V_A = \max_i \min_j M_{ij} \leq \min_j \max_i M_{ij} = V_B$$

Often $V_A < V_B$. In our last (scissors-...) example, $-1 = V_A < V_B = +1$.

If $V_B = V_A$ we say that the game has a solution – a specific choice of strategies that leads to this solution.

ϱ and γ are so called optional strategies for Alice and Bob if

$$V_A = V_B = M_{\varrho\gamma}$$

Example

of the game which has a solution ($V_A = V_B = 0$)

0	1	2
-1	0	1
-2	-1	0

What happens if a game has no solution ?

There is no clear–cut strategy for any player.

Way out: to use randomized strategies.

Alice chooses strategies according to a probability vector $p = (p_1, \ldots, p_n)$; p_i is probability that Alice chooses strategy $s_{A,i}$

Bob chooses strategies according to a probability vector $q = (q_1, \ldots, q_n)$; q_j is a probability that Bob chooses strategy $s_{B,j}$.

Payoff is now a random variable – **if p, q are taken as column vectors** then

$$E[\text{payoff}] = p^T M q = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i M_{ij} q_j$$

Let V_A (V_B) denote the best possible lower (upper) bound on the expected payoff of Alice (Bob)

$$V_A = \max_p \min_q p^T M q \qquad V_B = \min_q \max_p p^T M q$$

Theorem (von Neumann Minimax theorem) For any two–person zero–sum game specified by a payoff matrix M it holds

$$\max_p \min_q p^T M q = \min_q \max_p p^T M q$$

Observe that once p is fixed, $\max_p \min_q p^T M q = \min_q \max_p p^T M q$ is a linear function and is minimized by setting to 1 the q_j with the smallest coefficient in this linear function.

This has interesting/important implications:

If Bob knows the distribution p used by Alice, then his optimal strategy is a pure strategy.

A similar comment applies in the opposite direction. This leads to a simplified version of the minimax theorem, where e_k denotes a unit vector with 1 at the k-th position and 0 elsewhere.

Theorem (Loomis' Theorem) For any two–persons zero–sum game

$$\max_p \min_j p^T M e_j = \min_q \max_i e_i^T M q$$

YAO'S TECHNIQUE

provides an application of the game-theoretic results to the establishment of lower bounds for randomized algorithms.

For a given algorithmic problem $\mathcal{P}$ let us consider the following payoff matrix.

$$\begin{array}{cc|ccc|}
 & & \multicolumn{3}{c}{\textit{deterministic algorithms}} \\
 & & \mathcal{A}_1 & \mathcal{A}_2 & \mathcal{A}_3 \\
\text{I} & c_1 & & & \\
\text{N} & c_2 & & \text{entries} & \\
\text{P} & c_3 & & = & \\
\text{U} & c_4 & & \text{resources} & \\
\text{T} & & & \text{(i.e. used computation time)} & \\
\text{S} & & & &
\end{array}$$

Bob – a designer choosing good algorithms

Alice – an adversary choosing bad inputs

Pure strategy for Bob corresponds to the choice of a deterministic algorithm

Optimal pure strategy for Bob corresponds to a choice of an optimal deterministic algorithm.

V_B is the worst-case running time of any deterministic algorithm

Problem: How to interpret mixed strategies ?

A *mixed strategy* for Bob is a probability distribution over (always correct) deterministic algorithms—so it is a Las Vegas randomized algorithm.

An optimal mixed strategy for Bob is an optimal Las Vegas algorithm. **Distributional complexity** of a problem is an expected running time of the best deterministic algorithm for the worst distribution on the inputs.

Loomis theorem implies that distributional complexity equals to the least possible time achievable by any randomized algorithm

Reformulation of von Neumann+Loomis' theorem in the language of algorithms

Corollary Let Π be a problem with a finite set I of input instances and $\mathcal{A}$ be a finite set od deterministic algorithms for Π. For any input $i \in I$ and any algorithm $A \in \mathcal{A}$, let $T(i, A)$ denote computation time of A on input i. For probability distributions p over I and q over $\mathcal{A}$, let i_p denote random input chosen according to p and A_q a random algorithm chosen according to q. Then

$$\max_p \min_q E\left[T(i_p, A_q)\right] = \min_q \max_p E\left[T(i_p, A_q)\right]$$

$$\max_p \min_{A \in \mathcal{A}} E\left[T(i_p, A)\right] = \min_q \max_{i \in I} E\left[T(i, A_q)\right]$$

Consequence:

Theorem(Yao's Minimax Principle) For all distributions p over I and q over $\mathcal{A}$.

$$\min_{A \in \mathcal{A}} \mathbf{E}[T(i_p, A)] \leq \max_{i \in I} \mathbf{E}[T(i, A_q)]$$

Interpretation Expected running time of the optimal deterministic algorithm for an arbitrarily chosen input distribution p for a problem Π is a lower bound on the expected running time of the optimal (Las Vegas) randomized algorithm for Π.

IMPLICATIONS OF YAO'S MINIMAX PRINCIPLE

Interpretation again Expected running time of the optimal deterministic algorithm for an arbitrarily chosen input distribution p for a problem Π is a lower bound on the expected running time of the optimal (Las Vegas) randomized algorithm for Π.

Consequence:
In order to prove a lower bound on the randomized complexity of an algorithmic problem, it suffices to choose *any* probability distribution p on the input and prove a lower bound on the expected running time of deterministic algorithms for that distribution.

The power of this technique lies in

1. the flexibility at the choice of p
2. the reduction of a lower bound for randomized algorithms to a lower bound for deterministic algorithms.

(It is important to remember that we can expect that the deterministic algorithm "knows" the chosen distribution p.)

The above discussion holds for Las Vegas algorithms only.

THE CASE OF MONTE CARLO ALGORITHMS

Let us consider Monte Carlo algorithms with error probability $0 < \varepsilon < \frac{1}{2}$.

Let us define the **distributional complexity with error** ε, notation

$$\min_{A \in \mathcal{A}} E\,[T_\varepsilon(I_p, A)],$$

to be the minimum expected time of any deterministic algorithm that errs with probability at most ε under the input I_p with distribution p.

Let us denote by

$$\max_{i \in \mathcal{I}} E\,[(T_\varepsilon(i, A_q)]$$

the expected time (under the worst input) of any randomized algorithm A_q that errs with probability at most ε.

Theorem For all distributions p over inputs and q over Algorithms, and any $\varepsilon \in [0, 1/2]$, it holds

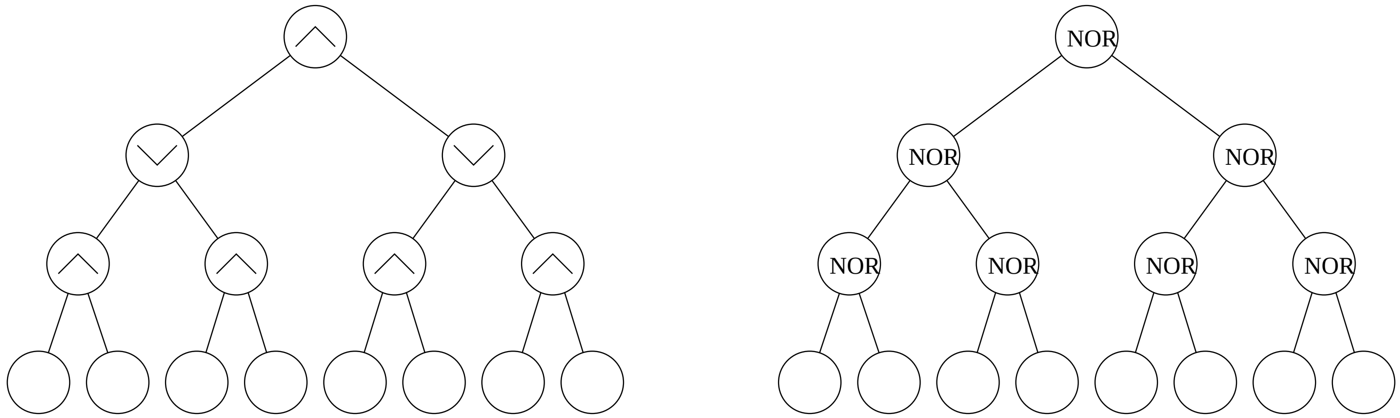
$$\frac{1}{2}(\min_{A \in \mathcal{A}} \mathbf{E}\,[T_{2\varepsilon}(i_p, A)]) \leq \max_{i \in \mathcal{I}} \mathbf{E}\,[T_\varepsilon(i, A_q)]$$

GAME TREE REVISITED

A randomized algorithm for game-tree evaluations can be viewed as a probability distribution over deterministic algorithms, because the length of computation and the number of choices at each step are finite. (We may imagine that all coins are tossed before the beginning of execution.)

Instead of AND–OR trees we can consider NOR–trees

$$(a \vee b) \wedge (c \vee d) \equiv (a \text{ NOR } b)\text{NOR}(c \text{ NOR } d)$$



Note: It's important to distinguish between

- the expected running time of the randomized algorithm with a fixed input
  (where probability is considered over all random choices made by the algorithm)
- the expected running time of the deterministic algorithm when proving the lower bound (the average time is taken over all random input instances).

LOWER BOUND FOR GAME TREE EVALUATION - I

Assume now that each leaf is set up to have value 1 with probability $p = \frac{3-\sqrt{5}}{2}$ (observe that $(1-p)^2 = p$ for such a p).

Observe that if inputs of a NOR-gate have value 1 with probability p then its output value is 1 with probability $(1-p)(1-p) = p$.

Consider now only *depth–first pruning algorithms* for tree evaluation.
(They are such depth–first algorithms that make use of the knowledge that subtrees that provide no additional useful information can be ”pruned away”.)

Of importance for the overall analysis is the following technical lemma:

Lemma Let T be a NOR–tree each leaf of which is set to 1 with a fixed probability. Let $W(T)$ denote the minimum, over all deterministic algorithms, of the expected number of steps to evaluate T. Then there is a depth–first pruning algorithm whose expected number of steps to evaluate T is $W(T)$.

The last lemma tells us that for the purposes of our lower bound, we may restrict our attention to the depth–first pruning algorithms.

LOWER BOUND FOR GAME TREE EVALUATION - II

For a depth–first pruning algorithm evaluating a NOR–tree, let $W(h)$ be the expected number of leaves the algorithm inspects in determining the value of a node at distance h from the leaves.

It holds

$$W(h) = pW(h-1) + (1-p)2W(h-1) = (2-p)W(h-1)$$

because with the probability $1-p$ the first subtree produces 0 and therefore also the second tree has to be evaluated. If $h = \lg_2 n$, then the above recursion has a solution

$$W(h) \geq n^{0.694}.$$

This implies:

Theorem The expected running time of any randomized algorithm that always evaluates an instance of T_k correctly is at least $n^{0.694}$, where $n = 2^{2k}$ is the number of leaves.

The upper bound already shown was $n^{0.79}$, what is more than n^{694}. It is therefore natural to ask what does the previous theorem really says?

For example, is our lower bound technique weak? ?

The above result just says that in order to get a better lower bound another probability distribution on inputs could be better.

Appendix

The concept of the number of wisdom introduced in the following and related results helped to show that randomness is deeply rooted even in arithmetic.

Ω - number of wisdom

For a universal computer C, the number of wisdom Ω_C is the halting probability of self-delimiting programs for C. More formally

$$\Omega_C = \sum_{p \text{ halts}} 2^{-|p|}$$

where p are self-delimiting halting programs for C.

Ω_C is therefore the probability that a self-delimiting computer program for C generated at random, by choosing each of its bits using an independent toss of a fair coin, will eventually halt.

Properties of numbers of wisdom

:

- $0 \leq \Omega_C \leq 1$
- Ω_C is a uncomputable and random real
- At least n-bit theory is needed to determine n bits of Ω_C.
- At least n bits long program is needed to determine n bits of Ω_C
- n-bit theory is needed in order to prove that a specific n-bit program is elegant.
- Bits of Ω can be seen as mathematical facts that are true for no reason (simpler than by themselves).

- G. Chaitin, who introduced numbers of wisdom, designed a specific universal computer C and a two hundred pages long Diophantine equation E, with 17000 of variables and with one parameter k, such that for a given k the equation E has a finite (infinite) number of solutions if and only if the k-th bit of Ω_C is 0 (is 1). { As a consequence, we have that randomness, unpredictability and uncertainty occur even in the theory of Diophantine equations of elementary arithmetic.}
- Knowing the value of Ω_C with n bits of precision allows to decide which programs for C with at most n bits halt.
- Incompleteness of axioms may also be a reason why some open problems remain open for hundred years.