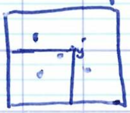
MPRi 2-24-2 2021/22 Solving Optimization Problems with Search Heuristics.

- Part I: Black-Box Optimization by Carola.Doerr@lip6.fr
- Part II: Local search strategies. by Evripidis.Bampis@lip6.fr

Final Grade:
1) Project
2) joint exam, 2 parts, 50-50

1) What are search heuristics?

- they are NOT problem-tailored
- general-purpose strategies for solving complex optimization heuristics.
- Example: TSP - many sophisticated algorithms
  - but: heuristics are not too bad either.
- star discrepancy computation.



- given a set P of points in $[0,1]^d$
- local star discrepancy $d^*_\infty(y,P) = \left| V_y - \frac{A(y,P)}{|P|} \right|$
- discrepancy $\sup_{y \in [0,1]^d} d^*_\infty(y,P)$

How do you solve this problem?

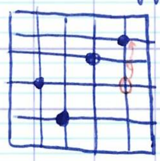
-> exact algo, complexity $O(n^{d/2+1})$ [complicated algo, using dynamic programming]

-> in practice, $d=4$ $n=1500$

$d=10$ $n=100$

but in many applications we have $d \gg 10$ or $n \gg 1500$

-> heuristic approaches!



- e.g. local search on the grid.

This is not unusual in practice. Theory of algorithms sharpens our minds and makes good contributions, but VERY often combined with heuristic approaches

Q What is the problem with local search? -> gets stuck in local optima :(

-> Simulated Annealing

-> Evolutionary Approaches // non-local sampling

-> random sampling

-> genetic algorithms, swarm intelligence algos, Bayesian Optimization, Nelder-Mead, ...

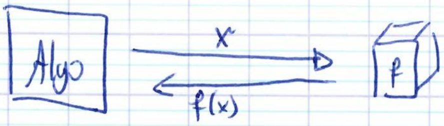
Q: Have you seen applications of such techniques?

Q What is different here from dynamic programming approach of DEM?

-> We fully evaluate a solution before sampling the next.

-> Black-Box Optimization Framework

2) Black-Box Optimization



Terminology:
- x: sample, search point
- $f(x)$: "fitness" of x
  - objective value of x
  - function value of x

Algo: Select distribution D
Sample $x \sim D$
Evaluate $f(x)$
Update D

When parallel evals allowed
Select distribution L
Sample $\lambda \sim L$
Sample $x^1, \dots, x^\lambda \sim D$
Evaluate $f(x^1), \dots, f(x^\lambda)$
Update L, and update D

fairly general framework & millions of algorithms that classify as black-box algorithms.

Our goal: Analyze their behavior by classical mathematical means w/ the goal to understand which algos / which ideas work well under which circumstances.

Performance Measure: # of calls/queries/evaluations made before ~~reaching~~ querying an optimal solution for the first time.

very convenient, because this is a very simple way of measuring complexity!

⚠ The query complexity of NP-hard problems can be polynomial!

Example: Max Clique problem
$f(S) = \begin{cases} 0 & \text{otherwise} \\ |S| & \text{if } S \text{ is a clique} \end{cases}$
Query all pairs, then you know the graph and can compute the maximal clique using a brute-force algorithm (which does not require further sampling)

3) Example: Mastermind

Formally, $f_z : \{0,1\}^n \to [0..n]$
$x \mapsto |\{i \in [n] \mid x_i = z_i\}|$
$\mathcal{F} = \{ f_z \mid z \in \{0,1\}^n \}$
set of problem instances

3a) First version: 2 colors, black and white
n positions.

Q: How many queries do you need to find my secret code?

You can easily obtain it in $n+1$ guesses, by querying
1) ○○○○○ … ○
2) ●○○○○ … ○
●●○○ … ○ ●○○○○ … ○○

→ do you need to be adaptive to learn my secret code? (No!, only for "optimizing", i.e. for querying a point x with $f(x) = \sup F$)

➡ Learning ≠ optimizing (but often close, as we only need 1 more query to "optimize")

→ ~~n+1 optimal?~~

→ Is this strategy generally done after $n+1$ querie? (meaning: on all functions $f : \{0,1\}^n \to \mathbb{R}$)
No! (Of course not ☺) → construct a counter-example.

→ This is why in practice we may want to change again a bit that we have changed before.

Notation: $[n] = \{1, 2, \dots, n\}$, $[a..b] = \{x \in \mathbb{Z} \mid a \le x \le b\}$

Randomized Local Search (RLS)

- Sample $x \in \{0,1\}^n$ uniformly at random (u.a.r) // Initialization
- Select a position $i \in [n]$ u.a.r.
- Create y by setting $y_j = \begin{cases} x_j & \text{if } j \neq i \\ 1 - x_j & \text{otw} \end{cases}$ } "mutation step"
- Evaluate $f(y)$
- If $f(y) \geq f(x)$ then $x \leftarrow y$ // Selection
- Repeat (if budget allows)

You may have seen variants of this algo, e.g. Schöning's 3-SAT algorithm

Q: How long does RLS need to optimize 2-color Mastermind (in expectation)?

1) At most $O(n \log n)$ steps:

we always have $f(y) \in \{f(x)-1, f(x)+1\}$

For a given x, $\Pr[f(y) > f(x) \mid f(x) = i] = \frac{n-i}{n}$

Ⓐ Hence $\frac{n}{n-i}$ steps in expectation to go from $f(x) = i$ to $f(x) = i+1$

total expected runtime $E[T] \leq \sum_{i=0}^{n-1} \frac{n}{n-i} = n \sum_{i=1}^{n} \frac{1}{i} = (1+o(1))\, n \ln(n)$

This is simply the coupon collector process

Ⓐ For a proof, see Lemma 1.6.3 in Doerr: Probabilistic Tools for the Analysis of Randomized Optimization Heuristics

Also $\Pr[T > C \cdot n H(n)] \leq \frac{1}{C}$ for all $C > 1$ [Markov]

$\Pr[T > (1+\varepsilon)\, n H(n)] \leq n^{-\varepsilon}$ for all $\varepsilon > 0$ [Chernoff]

$\Pr[T \geq n \ln(n) + \varepsilon n] \leq \exp(-\varepsilon)$ — " —

alternatively: $\Pr[T \geq (1+\varepsilon) E[T]] \leq \exp\left(-\frac{\varepsilon^2 E[T]}{3}\right)$ $\Pr[T \leq (1-\varepsilon) E[T]] \leq \exp\left(-\frac{\varepsilon^2 E[T]}{2}\right)$

2) At least $\Omega(n \log n)$ steps:

- Probability to start with $f(x) \leq n/2$ is at least $1/2$
- Starting from $n/2$, RLS still needs $\sum_{i=n/2}^{n-1} \frac{n}{n-i}$ iterations in expectation
- Hence, overall expected running time at least $\frac{1}{2} \sum \dots = \Omega(n \ln(n))$

Homework: Bound the running time of RLS on k-color Mastermind, i.e. the "game" where the secret code is a $z \in [k]^n$ and each query x gets a score $f_z(x) = |\{i \in [n] \mid x_i = z_i\}|$

Note that you have define a generalization of RLS from $\{0,1\}^n$ to $[k]^n$

∘ change to random color: $\leq n \cdot (k-1) H_{n-1} = \Theta(nk \ln(n))$

∘ change to ±1 color → random walk $O(n r_\ast + n \log n)$

Mastermind continued:

- We have seen an $n+1$ strategy
- RLS needs $\Theta(n \log n)$ queries.

little (?) overhead for randomized selection of position to flip — subjective ☺

A Formally

How much better can we get?

- intuitively, we need to learn n bits of information
- with every query, we receive a number between 0 and n (the "score", $f(x)$-value) $= \log_2(n+1)$ bits

$\Rightarrow$ we need at least $\frac{n}{\log(n+1)}$ queries

We call this the query complexity of the problem, (A) aka its black-box complexity

- Can you formalize the argument given above?

Key ingredients: 1) Yao's principle / Minimax principle

2) Query complexity of deterministic strategies on randomly chosen instances

Black-box complexity (definition)

Let A be an algorithm and $f: \{0,1\}^n \to \mathbb{R}$ a fct that we wish to optimize using A.

$T(A,f) =$ # of queries needed until A evaluates for the first time $x^* \in \arg\max f$

$\Rightarrow$ randomized A $\to$ $T(A,f)$ random variable.

$\Rightarrow$ We are mainly interested in $\mathbb{E}[T(A,f)]$

- Black-Box Complexity of a collection $\mathcal{F} = \{f: \{0,1\}^n \to \mathbb{R}\}$ of problems ~~and~~ with respect to a collection $\mathcal{A}$ of algorithms:

$$\mathcal{A}\text{-BBC}(\mathcal{F}) = \inf_{A \in \mathcal{A}} \sup_{f \in \mathcal{F}} \mathbb{E}[T(A,f)]$$

Step 1: Every deterministic strategy needs $\Omega(\frac{n}{\log n})$ queries to optimize 2-color Mastermind.

Proof idea: We can view deterministic algos as decision trees

We need to place 2^n possible optima into this tree.

On each level i we have at most $(n+1)^{i-1}$ strings.

$\Rightarrow$ with i queries we can "cover" at most $\sum_{j=1}^{i} (n+1)^{j-1} = \frac{(n+1)^{i+1}-1}{n} \leq (n+1)^{i+1}$

We need $(n+1)^{i+1} \geq 2^n$, i.e. $i+1 \geq \frac{\log_2(2^n)}{\log_2(n+1)} = \frac{n}{\log_2(n+1)}$ ■

Lemma: Let $\mathcal{D}$ be the set of deterministic algorithms operating on S.
Let $\mathcal{F} = \{ f : S \to \mathbb{R} \}$ such that $\forall s \in S \; \exists f \in \mathcal{F} : s = \arg\max f$ ← i.e. s is ~~the only~~ a unique optimum of f
If $\forall f \in \mathcal{F} : |\{ f(s) \mid s \in S \}| \leq k$
then $\mathcal{D}\text{-BBC}(\mathcal{F}) \geq \lceil \log_k(|S|) \rceil - 1$.

- Now, how do we extend this to randomized algorithms?
  - Yao's (minimax) principle! It builds on the fact that randomized algos are prob. distributions over deterministic algos. It relates the best worst-case complexity of a randomized algo to the best worst-case complexity of a deterministic algorithm over random inputs.
  - Formally (Formulation from Motwani / Raghavan: Randomized Algorithms (book))

Follows from von Neumann's minimax theorem for 0-sum games

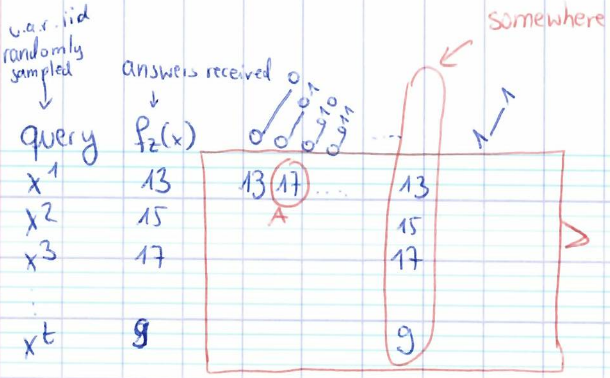
> Let Π be a problem with a finite set of instances (of a fixed size) $\mathcal{I}$, permitting a finite set of deterministic algorithms $\mathcal{A}$.
> Let p be a prob. distribution over $\mathcal{I}$
> q 〃 $\mathcal{A}$
> Then $$\min_{A \in \mathcal{A}} \mathbb{E}_p[T_A(I_p)] \leq \max_{I \in \mathcal{I}} \mathbb{E}[T_{A_q}(I)]$$
> where I_p is an instance chosen from $\mathcal{I}$ according to p
> A_q rand. algo. $\mathcal{A}$ q
> and $T_A(I)$ is the running time of A on I.

Corollary (Lemma above + Yao's principle + uniform distribution)
Under the same conditions as above, $\mathcal{R}\text{-BBC}(\mathcal{F}) \geq \lceil \log_k(|S|) \rceil - 1$
where $\mathcal{R}$ is the set of all randomized algorithms.

Proof: For each $s \in S$ select an instance f_s such that $s = \arg\max f_s$. Consider the uniform distribution over $\{ f_s \mid s \in S \}$
Consider an optimal det. decision tree. Average depth of a node is $\geq \log_k(|S|) - 2$. Hence, at least $\log_k(|S|) - 1$ queries needed.

2-color Mastermind (upper bound)

- Can we design an $o(n)$ strategy to solve 2-color Mastermind?
- Erdős & Rényi (and several others, independently): Yes!
- **Theorem:** The $\mathcal{R}\text{-BBC}(\text{2-color Mastermind}) = \Theta\left(\frac{n}{\log n}\right)$
- Surprisingly simple strategy: Sample $C\frac{n}{\log n}$ strings u.a.r. (iid) then check which secret codes are possible
  ⇒ with high probability only 1 code left.

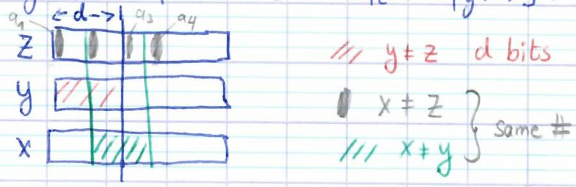


Huge matrix, 2^n columns, 1 for each string.
For each string y we can compute $f_y(x^i)$
Only strings y with $f_y(x^i) = f_z(x^i)$ can be the secret code

Ⓐ: $f_{0\_01}(x^1) = 17 \neq f_z(x^1)$, the answer that we received $\Rightarrow$ 0_01 cannot be the secret code

For a given z and y and randomly chosen x, what is $Pr[f_z(x) = f_y(x)]$?

Set $d := H(z,y)$



$$\underset{= n - a_1 - a_2 - a_3 - a_4}{f_z(x)} = f_y(x) = \underbrace{f_y(z)}_{} + a_1 + a_2 - a_3 - a_4 = n - d + a_1 + a_2 - a_3 - a_4$$

$\Rightarrow d = 2a_1 + 2a_2$; i.e $a_1 + a_2 = d/2$ $\Rightarrow$ x disagrees with y in half the bits in which y disagrees with z

$$Pr[\underbrace{f_z(x) = f_y(x)}_{y \text{ is consistent with } x} \mid H(z,y) = d] = \begin{cases} 0 : d \text{ odd} \\ \binom{d}{d/2} / 2^d & \text{otherwise} \end{cases}$$

$$Pr[\exists y \neq z : y \text{ consistent with randomly sampled } x^1, \ldots, x^t]$$
$$\leq \sum_{y \neq z} \prod_{i=1}^{t} Pr[y \text{ consistent with } x^i]$$
$$\leq \sum_{\substack{d=1 \\ d \text{ odd}}}^{n} \binom{n}{d} \cdot \left(\frac{\binom{d}{d/2}}{2^d} \right)^t \leq \ldots \leq n \cdot 2^{-3t/4} \text{ if } t \geq (1+o(1)) \frac{2n}{\log n}$$
$$= o(1)$$

Lemma: When ~~the function F~~ $\mathcal{F}$ satisfies $Pr[T \leq t] \geq c$ for some constant c, there exists an algorithm A whose ~~worst-case~~ running time T on any problem instance $f \in \mathcal{F}$, then the Black-box complexity of $\mathcal{F}$ is $O(t)$.

<u>Proof:</u> Simply run the algorithm for t steps. If it did not find an optimal solution, start all over again. Expected number of restarts $\leq 1/c = O(1)$, hence overall worst-case expected run time $\leq \frac{1}{c} \cdot t = O(t)$. ■