Overview and Greedy Approximation

Shao-Heng Ko

Advanced Algorithms Study Group, IIS, Academia Sinica

Oct 18, 2019

Today’s Agenda

- Motivation and Overview
  - Why study advanced algorithms
  - What to study: approximation, randomized, streaming, online
- Greedy Approximation (Set Cover, Vertex Cover)
  - Lecture 01 of [ETHZAA19]
  - Real exercise!
- Get to know each other

The first question

- What makes a good algorithm? [5P/each]

What are the qualities of a good algorithm?

Answer Follow · 7 Request

3 Answers

Daniel R. Page, Theoretical Computer Scientist, CS PhD
Answered Jun 28, 2018

I don't think there exists just one list (i.e., using "the").

Here are some qualities that make for a "good" algorithm:

- **Efficient** (with respect to a given problem)
- **Space-efficient** (does not utilize more memory than necessary), a lot of the time the battle is between finding an efficient algorithm that is space-efficient. Remember that you can re-use memory, but time is not reusable.
- **Stable:** does the algorithm produce solutions in a manner that is consistent. This may be that the calculations performed are numerically stable, or perhaps that the algorithm follows a similar execution path in a program for all instances so that the time taken for a given input (of a given size) does not differ greatly.
- **Effective:** is the algorithm correct, or is this more of a heuristic that is allowed to not yield optimal/correct answers sometimes?
- **Simple and/or elegant:** makes for an easier to understand algorithm and tends to be easier to implement and use. Furthermore, if it is "simple" and utilizes well-known techniques, it can cut down the time needed to check/utilize/implement a given algorithm.

What makes a "good" algorithm?

- A good algorithm should produce the correct outputs for any set of legal inputs.
- A good algorithm should execute efficiently with the fewest number of steps as possible.
- A good algorithm should be designed in such a way that others will be able to understand it and modify it to specify solutions to additional problems.

2 Motivation: "Fast. Reliable. Cheap. Choose two."

Some desirable properties for an algorithm are:

1. runs in polynomial time,
2. finds exact optimal solution,
3. robust: works for any input instance of a problem.

- runs in polynomial time/space
- finds exact optimal solution
- robust: works for any input instance of a problem

- Impossible for NP-hard problems T____T

- runs in polynomial time/space
- finds ~~exact~~ approximate optimal solution
- robust: works for any input instance of a problem

approximation algorithms!

Why are they better than heuristics? [5p]

Definition 1.1 (α-approximation). *An algorithm $\mathcal{A}$ is an α-approximation algorithm for a* minimization *problem with respect to cost metric c if for any problem instance I and for some optimum solution OPT,*

$$c(\mathcal{A}(I)) \leq \alpha \cdot c(OPT(I))$$

Maximization problems are defined similarly with $c(OPT(I)) \leq \alpha \cdot c(\mathcal{A}(I))$.

- runs in polynomial ~~time~~/space unfixed/unbounded
  finds exact optimal solution
- robust: works for any input instance of a problem

Las Vegas randomized algorithms!

- Some examples? [5p/each]

- Quicksort with randomized pivots
- Finding someone in the world

- runs in polynomial time/space
- finds ~~exact~~ approximate optimal solution
- robust: almost always works for any input instance of a problem

Monte Carlo (approximation) algorithms!

- Q: what exactly does “almost always” mean? [10p]
- Q: is it equivalent to “works for almost all inputs”? [10p]

- runs in polynomial time/~~space~~ limited
- finds ~~exact~~ approximate optimal solution
- robust: almost always works for any input instance of a problem

streaming algorithms!

- data model: inputs come sequentially without random access

Evaluation metrics:

- # of passes, space, accuracy, time

- Q: which alphabet (in A-E) appears the most?

- A

- C

- A

- D

- E

- A

- C

- D

- C

- E

- B

- A

- B

- D

- C

- B

- E

- B

- A

- D

- E

- Q: which alphabet (in A-E) appears the most?

[10p]-right answer

[-10p]-wrong answer

- ACADEACDCEBABDCBEBADE

- Sometimes, for real-world problems, we cannot afford waiting for all inputs to arrive in the future.

- To break-up or not break-up?
- To study advanced algorithms or not?

online algorithms!

- Need to give solutions whenever an input arrives

Definition 11.1 (α-competitive online algorithm). *Let σ be an input sequence, c be a cost function, $\mathcal{A}$ be the online algorithm and OPT be the optimal offline algorithm. Then, denote $c_{\mathcal{A}}(\sigma)$ as the cost incurred by $\mathcal{A}$ on σ and $c_{OPT}(\sigma)$ as the cost incurred by OPT on the same sequence. We say that an online algorithm is α-competitive if for any input sequence σ, $c_{\mathcal{A}}(\sigma) \leq \alpha \cdot c_{OPT}(\sigma)$.*

Remark We do not assume that the optimal offline algorithm has to be computationally efficient.

Happiness=2P

Definition 11.2 (Ski rental problem). *Suppose we wish to ski every day but we do not have any skiing equipment initially. On each day, we could:*

- *Rent the equipment for a day* \$1P
- *Buy the equipment (once and for all)* \$10P

In the toy setting where we may break our leg on each day (and cannot ski thereafter), let d be the (unknown) total number of days we ski. What is the best online strategy for renting/buying?

- Day 1
- Q: Buy or not buy?

- Day 2
- Q: Buy or not buy?

- Day 3
- Q: Buy or not buy?

- Day 4
- Q: Buy or not buy?

- Day 5

- Q: Buy or not buy?

- Day 6

- Q: Buy or not buy?

- Day 7
- Q: Buy or not buy?

- Day 8

- Q: Buy or not buy?

- Day 9

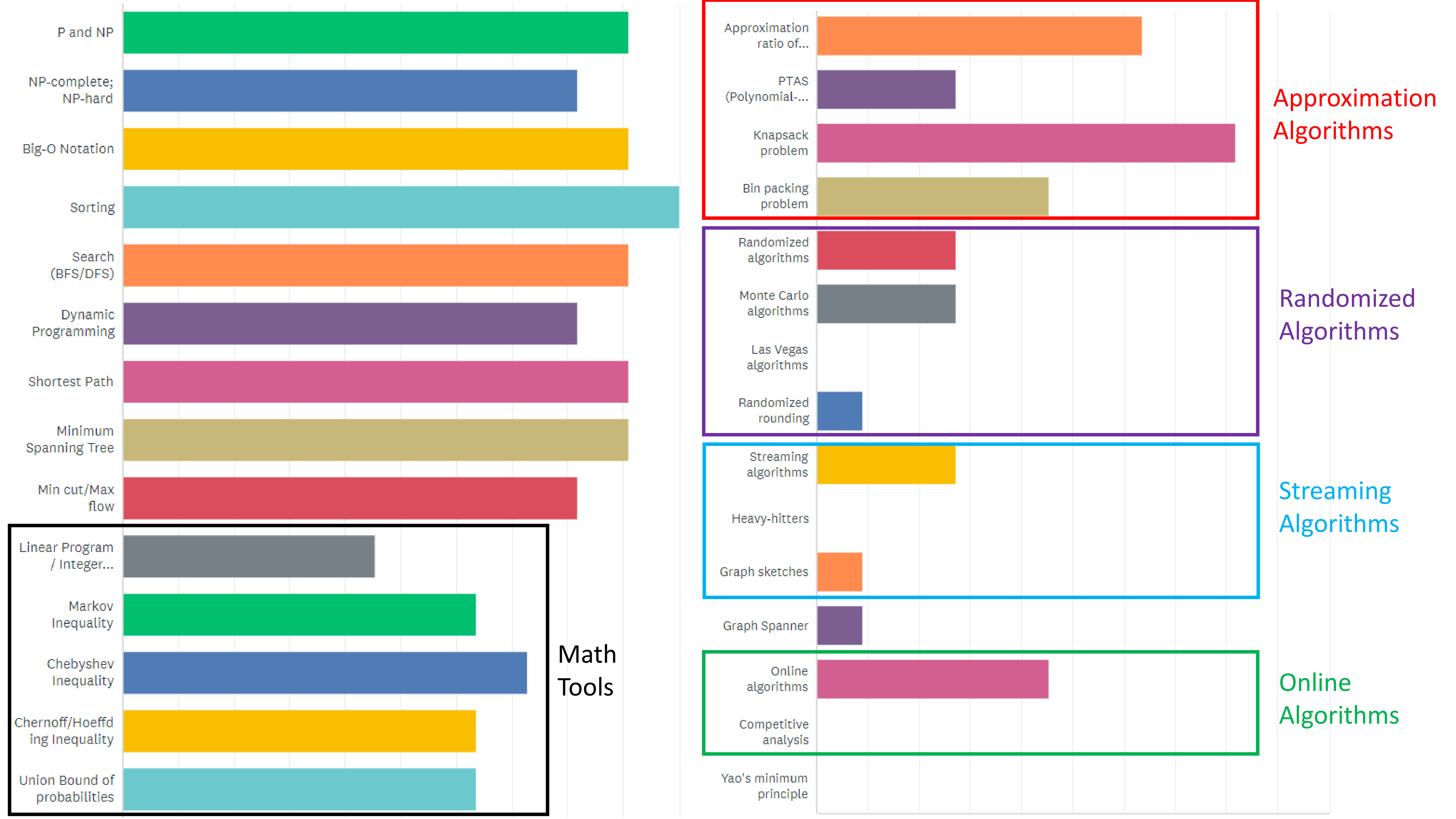
- Q: Buy or not buy?

- Day 10
- Q: Buy or not buy?

- Day 11
- Q: Buy or not buy?

- Question:

- What’s the difference between streaming and online algorithms? [10p]


P and NP
NP-complete; NP-hard
Big-O Notation
Sorting
Search (BFS/DFS)
Dynamic Programming
Shortest Path
Minimum Spanning Tree
Min cut/Max flow
Linear Program / Integer...
Markov Inequality
Chebyshev Inequality
Chernoff/Hoeffding Inequality
Union Bound of probabilities
Math Tools
Approximation ratio of...
PTAS (Polynomial-...
Knapsack problem
Bin packing problem
Approximation Algorithms
Randomized algorithms
Monte Carlo algorithms
Las Vegas algorithms
Randomized rounding
Randomized Algorithms
Streaming algorithms
Heavy-hitters
Graph sketches
Streaming Algorithms
Graph Spanner
Online algorithms
Competitive analysis
Online Algorithms
Yao's minimum principle

Greedy Approximation

- What do we usually think about greedy algorithms? [2p/each]

Greedy Approximation

- It is intuitive
- It is straightforward
- It is stupid
- It is not high-quality research…?

- Greedy is very effective for probably more problems than people think

Greedy Approximation

- What are some greedy algorithms we learn in undergraduate algorithms course? [5p/each]

- Dijkstra's algorithm for shortest path
- Prim's algorithm for minimum spanning tree
- Kruskal's algorithm for minimum spanning tree
- Huffman coding

Greedy Approximation

- It is better if we can perform theoretical analysis to show that greedy is indeed effective for some problem.

- Material: Lecture 1 in [AA19]

Ground rules

- Generally, we can directly use [AA19] lecture note
- Search for visualization slides online
- Leader reads the material thoroughly in advance
  - Then go through it efficiently with high-level comprehension

- Be skeptical when reading [AA19]
  - There are typoes/mistakes
  - It's great if we can find and correct them [5p/each]

Exercise Activity / Real applications

- It is expected that the leader of each meeting provide either an exercise activity or a real application of the theory material.
- Today there is an exercise activity.
- Sheng-Hao will talk about influence maximization in the next meeting, an application for submodular optimization.

A Covering practice

- We need to cover later meetings with people (leaders). Topics are listed in the next page.
- Let's first ignore schedule problems (we'll take care of it)
- We use points to order everyone XD
- (Discussion) How's this different from Minimum Set Cover? Important practice for future research

- **PTAS/FPTAS/FPRAS/APX, Knapsack (L2)**
- **Bin Packing (L2-L3)**
- **DNF Counting and Graph Coloring (L4)**
- **LP/ILP, Randomized Rounding (L5)**
- **Multi-commodity Routing (L5)**
- **Probabilistic Tree Embedding (L6)**
- **Frequent Elements, Approximate Counting (L7)**
- **Distinct Elements, Moment Estimators (L7-L8)**
- **Graph Sketches (L9)**
- **Graph Spanners (L10)**
- **Graph Sparsifiers (L11)**