Greedy Algorithms

CPSC 320 2024S

Optimization problems

- This unit deals with **optimization problems**:
  - We have a problem with several valid solutions
  - There is an **objective function** that tells us how good (or bad) each solution is
  - We are looking for the valid solution that minimizes or maximizes the value of the objective function

Optimization problems

Examples:

- Given a set of intervals, find the largest set of intervals that don’t overlap.
  - The objective function is the cardinality of the set
- Given a set of jobs with deadlines, order the jobs so as to minimize their total lateness
  - Objective function is the total lateness
- Given a weighted connected graph, find a spanning tree with the smallest total edge weight
  - Objective function is the sum of the weights of the edges in the spanning tree

Defining greedy algorithms

- A greedy algorithm proceeds by:
  - Making a choice based on a simple, local criterion
  - Solving the subproblem that results from that choice
  - Combining the choice and the subproblem choice
- We can think of a greedy algorithm as making a **sequence of** (locally “good”) **choices.**
- There is no precise definition of “greedy.”

Defining greedy algorithms

Examples of choice:

- Choosing the interval with the earliest finish time.
- Choosing the job with the earliest deadline.
- Choosing the item needed further in the future.
- Choosing the smallest weight edge that does not create a cycle.

Defining greedy algorithms

Does a greedy algorithm always give the correct solution?

- Sometimes yes, sometimes no.
- There are some classes of problems (e.g., matroids) for which there exists a greedy algorithm that always returns the correct solution.
- There are other problems where no one knows any greedy algorithm with this property (e.g., weighted interval scheduling).
- There are some problems where greedy doesn't give an optimal solution, but it's provably *close* to optimal in some sense (e.g., traveling salesperson).

Proving a greedy algorithm correct

Method 1: “greedy stays ahead”

- Essentially a proof by induction
- You compare the list of choices made by the greedy algorithm, to a similar list of choices made by an optimal solution
- Show that at each stage, the greedy choice is *at least as good* as the choice in the optimal solution
- Example: algorithm for interval scheduling problem (4.1)

Proving a greedy algorithm correct

Method 2: exchange arguments

- Prove that if **O** is an optimal solution, and **G** is the greedy solution, then you can modify **O** slightly to get **O'** such that:
  - **O'** is more similar to **G** than **O** was
  - **O'** is at least as good a solution as **O**
- Then describe how you can repeatedly modify **O** until it is *the same solution* as **G**, without ever decreasing the solution quality
- Examples of what "more similar to" might mean:
  - Has more edges in common with
  - Selects more of the same jobs
  - Has fewer elements out of order compared with the greedy solution

Worked example: greedy stays ahead (from 2023W2)

On a busy day of the final exam period, several computer science exams are being written in different rooms of OSBO. You have n bundles of exams that some of the invigilating TAs will need to bring back to ICICS for scanning, and each bundle has b_i exam booklets. However, there are a few rules you need to follow:

- Each TA can carry no more than m exams.
- The exams are expected at the CS building in a particular order, so they need to be transported back in the order listed (bundle 1, bundle 2, ..., bundle n).
- You can't split any of the exam bundles among multiple TAs, since some exams may end up getting misplaced. (You can assume that no bundle contains more than m exams.)

You want to distribute the exams to TAs such that you transport the exams using as few TAs as possible. For example, suppose you have bundles of size $[40, 40, 150, 25, 100]$ and $m = 200$, then an optimal distribution is $[40, 40], [150, 25], [100]$ – that is, giving bundles 1 and 2 to the first TA, bundles 3 and 4 to a second TA, and bundle 5 to a third TA. (For this instance, the optimal distribution is not unique: other optimal distributions are $[40], [40, 150], [25, 100]$ and $[40, 40], [150], [25, 100]$.)

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. More formally (but still not using pseudocode): if the sum of all bundles is <= *m*, give all bundles to a single TA. Otherwise, let *j* be the first index such that the first *j* bundles have sum greater than *m*; give *j-1* bundles to the first TA and recurse on the remaining bundles.

- **Clicker Q:** On the example [40, 40, 150, 25, 100] and *m* = 200, which optimal solution does the greedy algorithm produce?

A. [40, 40], [150, 25], [100]	B. [40], [40, 150], [25, 100]
C. [40], [40, 150], [25, 100]	D. None of these

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. More formally (but still not using pseudocode): if the sum of all bundles is <= *m*, give all bundles to a single TA. Otherwise, let *j* be the first index such that the first *j* bundles have sum greater than *m*; give *j-1* bundles to the first TA and recurse on the remaining bundles.

- **Clicker Q:** On the example [40, 40, 150, 25, 100] and *m* = 200, which optimal solution does the greedy algorithm produce?

A. [40, 40], [150, 25], [100]	B. [40], [40, 150], [25, 100]
C. [40], [40, 150], [25, 100]	D. None of these

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. We want a **greedy stays ahead lemma** that tells us something about how a partial solution produced by the greedy algorithm is “better” than a partial solution produced by some optimal solution.

Must-have properties for our lemma:

- Should be inductive (i.e., be about partial solutions)
- Should imply (if true) that the greedy algorithm is optimal
- Should actually be true for our algorithm

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. Which one is the **best** greedy stays ahead lemma?

A. The greedy solution transports all exam booklets using no more TAs than the optimal solution
B. The *i*th TA in the greedy solution is carrying no more than *m* exam booklets
C. The *i*th TA in the greedy solution is carrying more exam booklets than the *i*th TA in the optimal solution
D. The first *i* TAs in the greedy solution are carrying at least as many total booklets as the first *i* TAs in the optimal solution

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. Which one is the **best** greedy stays ahead lemma?

A. The greedy solution transports all exam booklets using no more TAs than the optimal solution
B. The *i*th TA in the greedy solution is carrying no more than *m* exam booklets
C. The *i*th TA in the greedy solution is carrying more exam booklets than the *i*th TA in the optimal solution
D. The first *i* TAs in the greedy solution are carrying at least as many total booklets as the first *i* TAs in the optimal solution

Worked example: greedy stays ahead (from 2023W2)

Optimal greedy algorithm is to give each TA as many exam booklets as they can carry before assigning any booklets to the next TA. Sketch of a proof that algorithm is optimal:

- Use the following lemma: The first *i* TAs in the greedy solution are carrying at least as many total booklets as the first *i* TAs in the optimal solution
  - Prove by ✨induction✨
- Suppose the greedy solution uses *k* TAs to transport all Σb_i exams. Optimal can't use $j < k$ TAs, because the first *j* TAs in greedy are transporting fewer than Σb_i exams, and lemma tells us that the first *j* TAs in optimal can't be transporting more than that. Therefore, greedy uses no more TAs than optimal, and is thus optimal.

Worked example: exchange argument (2023W2)

You work for a small business that plows driveways. Your city has experienced a major snowstorm so your boss, Mr. Plow, has lots of work to do! He has n clients who need some snow plowing done, and each job will take one hour to complete. All of the clients would prefer to have their driveways plowed earlier rather than later. More precisely, if client i is Mr. Plow's jth client, then client i's satisfaction is $s_i = n - j$.

Some clients are more important to Mr. Plow than others (perhaps they are paying more, are more likely to recommend Mr. Plow to others, or are more likely to be repeat customers in the future). Mr. Plow assign eachs client i a weight w_i. You want to schedule Mr. Plow's clients across the n days such that you maximize the weighed sum of satisfaction, that is, $\sum_{1 \leq i \leq n} w_i s_i$.

Worked example: exchange argument (2023W2)

You work for a small business that plows driveways. Your city has experienced a major snowstorm so your boss, Mr. Plow, has lots of work to do! He has n clients who need some snow plowing done, and each job will take one hour to complete. All of the clients would prefer to have their driveways plowed earlier rather than later. More precisely, if client i is Mr. Plow's jth client, then client i's satisfaction is $s_i = n - j$.

Some clients are more important to Mr. Plow than others (perhaps they are paying more, are more likely to recommend Mr. Plow to others, or are more likely to be repeat customers in the future). Mr. Plow assign eachs client i a weight w_i. You want to schedule Mr. Plow's clients across the n days such that you maximize the weighed sum of satisfaction, that is, $\sum_{1 \leq i \leq n} w_i s_i$.

- Optimal greedy algorithm is to complete projects in decreasing order of w_i

Worked example: exchange argument (2023W2)

Optimal algorithm = decreasing w_i. Sketch of a proof that the algorithm is optimal:

- Let *O* denote an optimal solution $o_1, o_2, \ldots, o_n$ and assume *O* is different from our greedy solution *G*
  - We need to define a way to make a single “exchange” to *O* to make it “closer to” *G.* We define a single “difference” as an inversion (pair of elements that appear in a different order in *O*/*G*)
- Consider indices $p < q$ where $o_p <= o_q$. We show that we can swap o_p and o_q and get a solution with >= weighted satisfaction, by ✨algebra✨
- We can repeatedly swap inversions until the optimal solution **becomes** the greedy schedule, all without decreasing weighted satisfaction. QED.