Algorithm Problem Solving (APS): Greedy Method

Niema Moshiri
UC San Diego SPIS 2019
Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - \( C = \{1\text{¢ (penny)}, 5\text{¢ (nickel)}, 10\text{¢ (dime)}, 25\text{¢ (quarter)}\} \)
Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - \( C = \{1\text{¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}\} \)

- **Input:** A non-negative integer \( x \) (in cents, not dollars)
Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - $C = \{1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)\}$

- **Input:** A non-negative integer $x$ (in cents, not dollars)

- **Output:** A selection of coins in $C$ summing to $x$
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?
  - I selected a coin that wasn’t in C!
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?
  - I selected a coin that wasn’t in C!
- The issue: my solution is incorrect
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in C
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in C
  - The coins I selected don’t sum to 42!
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies.
- You would probably be annoyed with me, but why?
  - All coins I selected were in C.
  - The coins I selected don’t sum to 42!
- The issue: my solution is incorrect.
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in C
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies

- You would probably be annoyed with me, but why?
  - All coins I selected were in C
  - The sum of my coins equals 42
Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in C
  - The sum of my coins equals 42
- The issue: your problem formulation was not specific!
Optimization Problems

- In many problems, we may have many (even infinite) possible solutions
Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions
- In all problems, we *must* define the precise definition of *correctness*
Optimization Problems

● In many problems, we may have *many* (even infinite) possible solutions

● In all problems, we *must* define the precise definition of *correctness*

● We can also choose to define an *objective function* to optimize
Optimization Problems

- In many problems, we may have many (even infinite) possible solutions.
- In all problems, we must define the precise definition of correctness.
- We can also choose to define an objective function to optimize.
- A solution satisfying the definition of correctness is correct.
Optimization Problems

- In many problems, we may have many (even infinite) possible solutions.
- In all problems, we must define the precise definition of correctness.
- We can also choose to define an objective function to optimize.
- A solution satisfying the definition of correctness is correct.
- A correct solution optimizing the objective function is optimal.
Revisiting the Change Problem (USA Currency)

- \( C = \{1\cent \text{ (penny)}, 5\cent \text{ (nickel)}, 10\cent \text{ (dime)}, 25\cent \text{ (quarter)}\} \)
Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}

- **Input:** A non-negative integer x (in cents, not dollars)
Revisiting the Change Problem (USA Currency)

- $\mathcal{C} = \{1\text{¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}\}$

- **Input:** A non-negative integer $x$ (in cents, not dollars)

- **Output:** A selection of coins in $\mathcal{C}$ summing to $x$
Revisiting the Change Problem (USA Currency)

- \( C = \{1\text{¢ (penny)}, 5\text{¢ (nickel)}, 10\text{¢ (dime)}, 25\text{¢ (quarter)}\} \)

- **Input:** A non-negative integer \( x \) (in cents, not dollars)

- **Output:** A selection of coins in \( C \) summing to \( x \) such that the number of selected coins is minimized
Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if $C = \{1\text{¢}, 2\text{¢}, 3\text{¢}, 4\text{¢}\}$ and $x = 5\text{¢}$
Multiple Optimal Solutions

● In some problems, there may be multiple equally-optimal solutions

○ Imagine if \( C = \{1\text{¢}, 2\text{¢}, 3\text{¢}, 4\text{¢}\} \) and \( x = 5\text{¢} \)

○ \([1\text{¢}, 4\text{¢}] \) and \([2\text{¢}, 3\text{¢}] \) are equally-optimal solutions
Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if \( C = \{1¢, 2¢, 3¢, 4¢\} \) and \( x = 5¢ \)
    - \([1¢, 4¢]\) and \([2¢, 3¢]\) are equally-optimal solutions
  - You should be happy receiving any such solution
Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if $\mathbf{C} = \{1\text{¢}, 2\text{¢}, 3\text{¢}, 4\text{¢}\}$ and $x = 5\text{¢}$
  - $[1\text{¢}, 4\text{¢}]$ and $[2\text{¢}, 3\text{¢}]$ are equally-optimal solutions
- You should be happy receiving any such solution
  - If not, you need to fix your objective function!
Revisiting the Change Problem (USA Currency)

- \( C = \{1\text{¢ (penny)}, 5\text{¢ (nickel)}, 10\text{¢ (dime)}, 25\text{¢ (quarter)}\} \)
- Imagine I owe you 42¢. How should I choose the coins to give you?

Let’s solve the problem!
Revisiting the Change Problem (USA Currency)

Algorithm change_USA(x,C):

  change ← empty list

  For each coin c in C (descending order):

    While x >= c:

      Add c to change

      x ← x - c

  Return change
Algorithm change_USA(x, C):

```plaintext
change ← empty list

For each coin c in C (descending order):

While x >= c:

Add c to change

x ← x - c

Return change
```

Does this work for any arbitrary currency?
Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
  - **Exhaustive:** Simply looking at every possible solution
Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
  - **Exhaustive**: Simply looking at every possible solution
- When we try to cleverly search for an optimal solution more quickly:
Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
  - **Exhaustive:** Simply looking at every possible solution
- When we try to cleverly search for an optimal solution more quickly:
  - **Global:** We can look at entire solutions at a time
Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
  - **Exhaustive**: Simply looking at every possible solution

- When we try to cleverly search for an optimal solution more quickly:
  - **Global**: We can look at entire solutions at a time
  - **Local**: We can break solutions into parts and optimize part-by-part
Local Search: The Greedy Method

- **Greedy Method**: Selecting the best possible choice at each step
Local Search: The Greedy Method

- **Greedy Method**: Selecting the best possible choice at each step

- Note that this does not always work!!!
Local Search: The Greedy Method

● **Greedy Method**: Selecting the best possible choice at each step

● **Note that this does not always work!!!**
  ○ We often skip what’s immediately best to improve in the long-run
Local Search: The Greedy Method

- **Greedy Method**: Selecting the best possible choice at each step

- **Note that this does not always work!!!**
  - We often skip what’s immediately best to improve in the long-run
  - **Example**: Buying vs. leasing a car
Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step

- **Note that this does not always work!!!**
  - We often skip what’s immediately best to improve in the long-run
  - **Example:** Buying vs. leasing a car

- Thus, it’s important to prove the correctness of a Greedy Algorithm
Revisiting the Change Problem

- \( \mathcal{C} = \{1¢, 3¢, 4¢\} \)
Revisiting the Change Problem

- $C = \{1¢, 3¢, 4¢\}$

- Imagine I owe you 6¢. How should I choose the coins to give you?
Revisiting the Change Problem

- $\mathcal{C} = \{1\text{¢}, 3\text{¢}, 4\text{¢}\}$

- Imagine I owe you 6¢. How should I choose the coins to give you?
  - The greedy algorithm would return [4¢, 1¢, 1¢]
Revisiting the Change Problem

- $C = \{1\$\,c, 3\$\,c, 4\$\,c\}$

- Imagine I owe you 6\$. How should I choose the coins to give you?
  - The greedy algorithm would return $[4\$, 1\$, 1\$]
  - The optimal solution is $[3\$, 3\$]
Revisiting the Change Problem

- \( C = \{1\,\text{¢}, 3\,\text{¢}, 4\,\text{¢}\} \)

- Imagine I owe you 6¢. How should I choose the coins to give you?
  - The greedy algorithm would return \([4\,\text{¢}, 1\,\text{¢}, 1\,\text{¢}]\)
  - The optimal solution is \([3\,\text{¢}, 3\,\text{¢}]\)
  - **Our greedy algorithm doesn’t work for all possible currencies!!!**
Immediate Benefit vs. Opportunity Cost
Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit**: How much do I gain from this choice?
- **Opportunity Cost**: How much is the future restricted by this choice?
Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
- **Opportunity Cost:** How much is the future restricted by this choice?
- **Greedy:** Take the best immediate benefit and ignore opportunity costs
Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
- **Opportunity Cost:** How much is the future restricted by this choice?
- **Greedy:** Take the best immediate benefit and ignore opportunity costs
  - Optimal when immediate benefit outweighs opportunity costs
Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
  - You charge a flat rate, regardless of the length of the event
Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
  - You charge a flat rate, regardless of the length of the event
  - Thus, you want to schedule as many events as possible
Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
  - You charge a flat rate, regardless of the length of the event
  - Thus, you want to schedule as many events as possible
  - However, events cannot overlap
Example: The Event Scheduling Problem

- **Input:** All $n$ possible events $E = [(start_1, end_1), ..., (start_n, end_n)]$
Example: The Event Scheduling Problem

- **Input:** All $n$ possible events $E = [(start_1, end_1), ..., (start_n, end_n)]$

- **Output:** A non-overlapping subset of $E$ maximizing its size
Example: The Event Scheduling Problem

• **Input:** All $n$ possible events $E = [(start_1, end_1), ..., (start_n, end_n)]$

• **Output:** A non-overlapping subset of $E$ maximizing its size

• If we wanted to design a greedy algorithm, what would we optimize?
Example: The Event Scheduling Problem

● **Input:** All $n$ possible events $E = [(\text{start}_1, \text{end}_1), \ldots, (\text{start}_n, \text{end}_n)]$

● **Output:** A non-overlapping subset of $E$ maximizing its size

● If we wanted to design a greedy algorithm, what would we optimize?
  ○ Shortest duration?
  ○ Earliest start time?
  ○ Fewest conflicts?
  ○ Earliest end time?
Counterexample: Shortest Duration
Counterexample: Shortest Duration
Counterexample: Shortest Duration
Counterexample: Shortest Duration
Counterexample: Shortest Duration

[Diagram showing a timeline with two bars: the first bar covers 0 to 8, and the second bar covers 8 to 16.]
Counterexample: Earliest Start Time
Counterexample: Earliest Start Time
Counterexample: Earliest Start Time
Counterexample: Earliest Start Time

![Diagram showing earliest start times with two overlapping tasks.]
Counterexample: Earliest Start Time
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts
Counterexample: Fewest Conflicts

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Counterexample: Fewest Conflicts
Counterexample: Earliest End Time
Counterexample: Earliest End Time

I can’t think of one!
Counterexample: Earliest End Time

We still need to prove it’s correct!!!

I can’t think of one!
Example: The Event Scheduling Problem

Algorithm $\text{schedule}(E)$:

Sort $E$ in ascending order of end time

$\text{curr\_time} \leftarrow \text{negative infinity}$

$\text{events} \leftarrow \text{empty list}$

For each event $(\text{start}, \text{end})$ in $E$:

If $\text{start} \geq \text{curr\_time}$:

Add $(\text{start}, \text{end})$ to $\text{events}$

$\text{curr\_time} \leftarrow \text{end}$

Return $\text{events}$
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
Proofs: The Exchange Argument

● Common approach for proving greedy algorithms
  ○ Let $g$ be the first greedy choice
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let \( g \) be the first greedy choice
  - Let \( S \) be any optimal solution that does not include \( g \)
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let $g$ be the first greedy choice
  - Let $S$ be any optimal solution that does not include $g$
  - Create $S'$ by exchanging a choice in $S$ with $g$ and show that
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let $g$ be the first greedy choice
  - Let $S$ be any optimal solution that does not include $g$
  - Create $S'$ by exchanging a choice in $S$ with $g$ and show that $S'$ is a valid solution
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let $g$ be the first greedy choice
  - Let $S$ be any optimal solution that does not include $g$
  - Create $S'$ by exchanging a choice in $S$ with $g$ and show that
    - $S'$ is a valid solution
    - $S'$ is just as good, or better than, $S$
Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let $g$ be the first greedy choice
  - Let $S$ be any optimal solution that does not include $g$
  - Create $S'$ by exchanging a choice in $S$ with $g$ and show that
    - $S'$ is a valid solution
    - $S'$ is just as good, or better than, $S$

Let’s try to prove our algorithm!