6331 - Algorithms, Spring 2014, CSE, OSU
Greedy algorithms II

Instructor: Anastasios Sidiropoulos
Greedy algorithms

- Fast
- Easy to implement
- At every step, the algorithm makes a choice that seems "locally" optimal.
When is greedy applicable?

- Many problems cannot be solved using a greedy algorithm.
When is greedy applicable?

- Many problems cannot be solved using a greedy algorithm.
- Some problems can be solved *approximately* using a greedy algorithm.
Consider some **optimization problem** $\Pi$.

- Given some input $X$.
- Compute a solution $\phi$ for $X$, that **optimizes** some objective function $f(\phi)$. 
Consider some **optimization problem** $\Pi$.

- Given some input $X$.
- Compute a solution $\phi$ for $X$, that optimizes some objective function $f(\phi)$.
  - **Minimization problems**: Compute $\phi$ that **minimizes** $f(\phi)$.
  - **Maximization problems**: Compute $\phi$ that **maximizes** $f(\phi)$. 
Examples of optimization problems

- **Activity-Selection**: Given activities with start and finish times, compute a maximum-size subset of compatible activities.
Examples of optimization problems

- **Activity-Selection**: Given activities with start and finish times, compute a *maximum*-size subset of compatible activities.

- **Going to the grocery store**: Given start and destination points on a map, compute a route of *minimum* length.
Examples of optimization problems

- **Activity-Selection**: Given activities with start and finish times, compute a maximum-size subset of compatible activities.
- **Going to the grocery store**: Given start and destination points on a map, compute a route of minimum length.
- ...
Approximation algorithms

Let $\Pi$ be a minimization problem.
Approximation algorithms

Let \( \Pi \) be a **minimization** problem.

For any input \( \phi \), let \( OPT(\phi) \) be the minimum cost of a solution for \( \phi \).
Approximation algorithms

Let $\Pi$ be a **minimization** problem.

For any input $\phi$, let $OPT(\phi)$ be the minimum cost of a solution for $\phi$.

Suppose that we have an algorithm $\mathcal{A}$ for $\Pi$, such that for any input $\phi$, computes a solution with cost at most $\beta \cdot OPT(\phi)$. 
Approximation algorithms

Let $\Pi$ be a \textbf{minimization} problem.

For any input $\phi$, let $OPT(\phi)$ be the minimum cost of a solution for $\phi$.

Suppose that we have an algorithm $A$ for $\Pi$, such that for any input $\phi$, computes a solution with cost at most $\beta \cdot OPT(\phi)$.

Then, we say that $A$ is a $\beta$-approximation algorithm.
Approximation algorithms

Let $\Pi$ be a maximization problem.
Let $\Pi$ be a maximization problem.

For any input $\phi$, let $OPT(\phi)$ be the minimum cost of a solution for $\phi$. 
Approximation algorithms

Let $\Pi$ be a \textbf{maximization} problem.

For any input $\phi$, let $OPT(\phi)$ be the minimum cost of a solution for $\phi$.

Suppose that we have an algorithm $A$ for $\Pi$, such that for any input $\phi$, computes a solution with cost \textbf{at least} $OPT(\phi)/\beta$. 
Approximation algorithms

Let $\Pi$ be a maximization problem.

For any input $\phi$, let $OPT(\phi)$ be the minimum cost of a solution for $\phi$.

Suppose that we have an algorithm $A$ for $\Pi$, such that for any input $\phi$, computes a solution with cost at least $OPT(\phi)/\beta$.

Then, we say that $A$ is a $\beta$-approximation algorithm.
The Bin-Packing problem

Suppose you are preparing for a trip.
The Bin-Packing problem

Suppose you are preparing for a trip.

- You need to pack your things into suitcases.
The Bin-Packing problem

Suppose you are preparing for a trip.
  ▶ You need to pack your things into suitcases.
  ▶ Each suitcase can weight at most 100 lbs.
The Bin-Packing problem

Suppose you are preparing for a trip.

- You need to pack your things into suitcases.
- Each suitcase can weight at most 100 lbs.
- Each suitcase costs $50 to carry.
The Bin-Packing problem

Suppose you are preparing for a trip.

- You need to pack your things into suitcases.
- Each suitcase can weight at most 100 lbs.
- Each suitcase costs $50 to carry.
- How can you partition your things into suitcases, so that you minimize the amount of money spent? I.e., minimize the number of suitcases.
The Bin-Packing problem

**Given:** \( n \) items, of size \( s_1, \ldots, s_n \in (0, 1] \).

**Compute:** A partition of the items into \( n \) bins of size at most 1. 
I.e., compute a partition \( B_1 \cup \ldots \cup B_k = \{1, \ldots, n\} \), for some \( k > 0 \), such that for each \( i \in \{1, \ldots, k\} \), we have

\[
\sum_{j \in B_i} s_j \leq 1.
\]

**Goal:** Minimize the number of bins, i.e. \( k \).
How easy is Bin-Packing?

- Fundamental problem in Computer Science.
How easy is Bin-Packing?

- Fundamental problem in Computer Science.
- There is no known polynomial-time algorithm for Bin-Packing!
How easy is Bin-Packing?

- Fundamental problem in Computer Science.
- There is no known polynomial-time algorithm for Bin-Packing!
- We believe that no such algorithm exists!
How easy is Bin-Packing?

- Fundamental problem in Computer Science.
- There is no known polynomial-time algorithm for Bin-Packing!
- We believe that no such algorithm exists!
- More in later lectures . . .
A greedy algorithm for Bin-Packing

**Greedy-Bin-Packing**\((s, n)\) \hspace{1em} k = 1
\[
B_1 = \emptyset
\]
for \(i = 1\) to \(n\)
\[
\text{if } s_i \text{ fits in } B_k
\]
\[
B_k = B_k \cup \{i\}
\]
else
\[
k = k + 1
\]
\[
B_k = \{i\}
\]
Analysis

Let $k_{OPT}$ be the cost of the optimum solution.
Analysis

Let $k_{OPT}$ be the cost of the optimum solution. We have

$$k_{OPT} \geq \sum_{i=1}^{n} s_i$$

In the computed solution, at least $k - 1$ bins are more than half-full.
Let $k_{OPT}$ be the cost of the optimum solution. We have

$$k_{OPT} \geq \sum_{i=1}^{n} s_i$$

In the computed solution, at least $k - 1$ bins are more than half-full. Thus

$$\sum_{i=1}^{n} s_i > \frac{k - 1}{2}$$
Analysis

Let $k_{OPT}$ be the cost of the optimum solution. We have

$$k_{OPT} \geq \sum_{i=1}^{n} s_i$$

In the computed solution, at least $k - 1$ bins are more than half-full. Thus

$$\sum_{i=1}^{n} s_i > \frac{k - 1}{2}$$

So

$$k < 2k_{OPT} + 1 \leq 2k_{OPT}$$
Analysis

In other words, Greedy-Bin-Packing is a 2-approximation algorithm for the Bin-Packing problem.
Remarks

- We are referring to $k_{OPT}$ in the analysis, even though we cannot compute it efficiently!
Remarks

- We are referring to $k_{OPT}$ in the analysis, even though we cannot compute it efficiently!
- Is Greedy-Bin-Packing optimal?
The Max-Cut problem

**Given:** A graph $G = (V, E)$.

**Solution:** A partition $V = S \cup S'$.

**Goal:** Minimize the number of edges between $S$ and $S'$ (i.e. with one end-point in $S$, and one end-point in $S'$).
A greedy algorithm for Max-Cut

Greedy-Max-Cut

Start with an arbitrary partition $V = S \cup S'$. 
while $\exists v \in V$ with more neighbors on the same side
    move $v$ to the other side
Analysis

- Running time:
Analysis

- Running time: At most $|E|$ iterations.
Analysis

- Running time: At most $|E|$ iterations. Why?
Analysis

- Running time: At most $|E|$ iterations. Why? After every iteration, the number of cut edges increases by at least 1.
Analysis

- Running time: At most $|E|$ iterations. Why? After every iteration, the number of cut edges increases by at least 1.
- Every vertex has at least half of its incident edges in the cut.
Analysis

- Running time: At most $|E|$ iterations. Why? After every iteration, the number of cut edges increases by at least 1.
- Every vertex has at least half of its incident edges in the cut.
- Greedy-Max-Cut is a 2-approximation for Max-Cut.