CS 445
Analysis of algorithms 2
Approximation algorithms
Éric Schost
eschost@uwo.ca
Why approximation algorithms

Suppose

• you are to solve an optimization problem $\max\{\cdots\}$ or $\min\{\cdots\}$.

• but (the decision version of) this optimization problem is \textbf{NP}-complete.

Then, you should give up trying to give a polynomial-time algorithm.

\textbf{Workarounds:}

• find an \textit{approximate} solution;

• use \textit{non-deterministic} algorithms.
Measuring the approximation

An approximation algorithm computes a value $V_{\text{output}}$ which is not quite the optimum $V_{\text{opt}}$, but hopefully close to it.

For maximization problems, the situation is

$$? \leq V_{\text{output}} \leq V_{\text{opt}}.$$  

We will be interested in figuring out the largest $\lambda \leq 1$ for which we can ensure

$$\lambda V_{\text{opt}} \leq V_{\text{output}} \leq V_{\text{opt}}.$$  

It is customary to consider $\varepsilon = 1 - \lambda$, so that

$$0 \leq \frac{V_{\text{opt}} - V_{\text{output}}}{V_{\text{opt}}} \leq \varepsilon \leq 1.$$  

$\varepsilon$ is the approximation ratio. The smaller, the better.
Measuring the approximation

An approximation algorithm computes a value $V_{\text{output}}$ which is not quite the optimum $V_{\text{opt}}$, but hopefully close to it.

For minimization problems, the situation is

$$V_{\text{opt}} \leq V_{\text{output}} \leq ?$$

We will be interested in figuring out the smallest $\lambda \geq 1$ for which we can ensure

$$V_{\text{opt}} \leq V_{\text{output}} \leq \lambda V_{\text{opt}}.$$ 

It is customary to consider $\varepsilon = 1 - 1/\lambda$, so that

$$0 \leq \frac{V_{\text{output}} - V_{\text{opt}}}{V_{\text{output}}} \leq \varepsilon \leq 1.$$ 

$\varepsilon$ is the approximation ratio. The smaller, the better.
Load balancing
Load balancing

**Setup.** We consider

- some workers $W_1, \ldots, W_m$,
- some jobs $j_1, \ldots, j_n$, which take $t(j_1), \ldots, t(j_n)$ time units.

Each worker $W_i$ will be assigned some jobs $J_i$, so that it will work for

$$T_i = \sum_{j \in J_i} t(j).$$

We want to **minimize** $\max_i T_i$.

**Prop:** the (decision version of) this problem is **NP**-complete.

**Proof:** $3SAT \leq PARTITION \leq LOAD\text{-}BALANCING$
**Greedy algorithm**

For $\ell = 1, \ldots, n$, do

- assign $j_\ell$ to the least loaded worker.

**Prop.** The greedy algorithm is $1/2$-approximated.

**First remark:** we have $\sum_{\ell \leq m} t(j_\ell) \leq m V_{\text{opt}}$, where $V_{\text{opt}}$ is the minimum.

Start from

$$\sum_{i \leq m} T_i \leq m \max_i T(i).$$

Since $\sum_{i \leq m} T_i = \sum_{\ell \leq n} t(j_\ell)$, we take the min over all assignments and we are done.
Greedy algorithm

For $\ell = 1, \ldots, n$, do

- assign $j_\ell$ to the least loaded worker.

**Prop.** The greedy algorithm is $1/2$-approximated.

**Second remark:** At any step of the algorithm, the minimum load is at most $V_{opt}$.

- At any step, the sum of the loads is at most $\sum_\ell t(j_\ell)$, which is $\leq mV_{opt}$.
- The sum is at least $m$ times the minimum.
Greedy algorithm

For $\ell = 1, \ldots, n$, do

- assign $j_\ell$ to the least loaded worker.

Prop. The greedy algorithm is $1/2$-approximated.

Third remark: for any assignment and any task $j_\ell$, we have

$$t(j_\ell) \leq \max_i T(i).$$

So the optimal has

$$t(j_\ell) \leq V_{\text{opt}}.$$
Proof

Let $T_1, \ldots, T_m$ be the assignment computed by the greedy algorithm, and suppose that:

- the maximum load is $T_i$;
- the last job assigned to $W_i$ is $j_\ell$.

This means that at the moment $j_\ell$ was assigned, $W_i$ had load $T_i - t(j_\ell)$.

- Because of the greediness, this was the minimal load at that moment, so $T_i - t(j_\ell) \leq V_{opt}$.
- We also have $t(j_\ell) \leq V_{opt}$.
- So $T_i \leq 2V_{opt}$. 
Vertex covering
Vertex covering

A vertex cover in a non-oriented graph is a subset $S$ of the vertices such that each edge has an extremity in $S$.

**Question:** find a cover as small as possible.
Difficulty

Recall the decision version of the problem:

- Given a graph $G$ and a threshold $k$, is there a vertex cover of size $\leq k$?

is $\text{NP}$-complete.

Conclusion: there is little hope that we can give a polynomial-time algorithm for finding the optimal.

Approximation algorithms:

- some greedy approaches do rather well,
- while some others do not.
Naive algorithm, version 2

- while there are still edges in the graph
  - pick a vertex, put it in the cover under construction;
  - remove it from the graph, together with all edges that contain it.
Bad performance

If you are not lucky, you may be very far from the optimal:

![Graph example](image)

On this example:

- you may choose all vertices but one
- the optimal has a single vertex

so the approximation ratio is close to 1.
A better greedy strategy?

- while there are still edges in the graph
  - pick the vertex connected to the **highest number** of edges, put it in the cover under construction;
  - remove it from the graph, together with all edges that contain it.

This would be optimal on the previous example. However, it is **not** optimal here:
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Approximation ratio
Imagine now a similar graph with $n$ vertices on the top row.

How many vertices on the bottom row?

$$n + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor + \cdots + \left\lfloor \frac{n}{n} \right\rfloor \simeq n \log(n)$$

So the approximation ratio is no better than $\simeq \log(n)$. 
Upper bound

Prop. The output $V_{\text{greedy}}$ of the algorithm satisfies

$$|V_{\text{greedy}}| \leq |V_{\text{opt}}| \log(n) \quad \text{or} \quad \frac{|V_{\text{greedy}}| - |V_{\text{opt}}|}{|V_{\text{greedy}}|} \leq 1 - \frac{1}{\log(n)}.$$ 

For a vertex $v$, let edge($v$) be the set of all edges that have $v$ has an extremity.

For a set of edges $E$, let $V_E = \{ \text{vertices which belong to an edge in } E \}$. 

Prop. $|E| \leq |V_{\text{opt}}| \max_{x \in V_E} |\text{edge}(x)|$. 

- Any edge in $E$ is in some edge($x$), for an $x$ that belongs to both $V_{\text{opt}}$ and $V_E$. 
- So $E \subset \bigcup_{x \in V_E \cap V_{\text{opt}}} \text{edge}(x)$. 
- So $|E| \leq \sum_{x \in V_E \cap V_{\text{opt}}} |\text{edge}(x)|$. 
- There are $\leq |V_{\text{opt}}|$ terms in the sum, and each is $\leq \max_{x \in V_E} |\text{edge}(x)|$. 
Upper bound

Step 1. Let

- \( E_i \) be the edges not covered (= not removed) yet at the beginning of \( i \)th step; 
  (so \( E_1 = |E| \)).

- \( x_i \) be the vertex that we remove at step \( i \).

So we have

\[
\begin{align*}
|E_1| & = |\text{edge}(x_1)| + |E_2| \\
|E_2| & = |\text{edge}(x_2)| + |E_3| \\
& \vdots \\
|E_i| & = |\text{edge}(x_i)| + |E_{i+1}| \\
|E_i| & = |\text{edge}(x_i)| + |\text{edge}(x_{i+1})| + \cdots + |\text{edge}(x_{j-1})| + |E_j|
\end{align*}
\]
Upper bound

Step 2. Recall $|E_i| \leq |V_{\text{opt}}| \max_{x \in V_{E_i}} |\text{edge}(x)|$.

Since $x_i$ is the element of $V_{E_i}$ with the largest set of edges, we get

$$|E_i| \leq |V_{\text{opt}}| |\text{edge}(x_i)|.$$ 

Step 3. $|\text{edge}(x_1)| \geq |\text{edge}(x_2)| \cdots$ (greedy algorithm!)

Step 4: Combine those to get $T_{(\ell+1)k} \leq \frac{1}{2} T_{\ell k}$, with $k = |V_{\text{opt}}|$.

Step 5: Conclude that we finish in $\leq k \log(n)$ steps.
Naive algorithm, version 1

- while there are still edges in the graph
  - pick one edge, put both extremities in the cover under construction;
  - remove it from the graph, together with its extremities, and all edges connected to them.
Approximation ratio

Prop. The output $V'_{\text{greedy}}$ of the algorithm satisfies

\[ |V'_{\text{greedy}}| \leq 2|V_{\text{opt}}| \quad \text{or} \quad \frac{|V'_{\text{greedy}}| - |V_{\text{opt}}|}{|V'_{\text{greedy}}|} \leq \frac{1}{2}. \]

Proof.

• At each step, we find at least one element of $V_{\text{opt}}$.

• So the number of steps is at most $|V_{\text{opt}}|$.

• The total number of vertices we select is $2 \times$ the number of steps, so at most $2|V_{\text{opt}}|$.
Traveling Salesman Problem
The traveling salesman problem

Problem (optimization version)

- Given a complete symmetric graph $G$, with vertices $v_1, \ldots, v_n$ and with weights $w(v_i, v_j)$ on its edges, find a cycle

$$
(v_{\sigma_1}, v_{\sigma_2}, \ldots, v_{\sigma_n}, v_{\sigma_1})
$$

that visits all vertices, and such that

$$
\sum_{i=1}^{n} w(v_{\sigma_i}, v_{\sigma_{i+1}}) + w(v_{\sigma_n}, v_{\sigma_1})
$$

is minimal.
The traveling salesman problem

Problem (decision version)

• Given a complete symmetric graph $G$, with vertices $v_1, \ldots, v_n$ and with weights $w(v_i, v_j)$ on its edges, and given an integer $k$, is there a cycle

$$(v_{\sigma_1}, v_{\sigma_2}, \ldots, v_{\sigma_n}, v_{\sigma_1})$$

that visits all vertices, and such that

$$\sum_{i=1}^{n} w(v_{\sigma_i}, v_{\sigma_{i+1}}) + w(v_{\sigma_n}, v_{\sigma_1}) \leq k?$$
The Hamiltonian cycle problem

Closely related question:

**HamiltonianCycle**

- Given a symmetric graph $G$, with vertices $v_1, \ldots, v_n$, is there a cycle
  
  $$(v_{\sigma_1}, v_{\sigma_2}, \ldots, v_{\sigma_n}, v_{\sigma_1})$$

  that visits all vertices?

**Prop.** **HamiltonianCycle** is **NP**-complete.

- **NP**-ness is easy (it amounts to check that a list of vertices is a cycle)
- **NP**-hardness is difficult. Reduction from 3SAT.

The reduction uses tricks similar to the reduction from 3SAT to **MAXINDEPENDENTSET**, to turn boolean formula questions into graph problems.
**NP-completeness**

**Prop.** The optimization version of TSP is **NP**-complete.

- **NP**-ness is easy (just verify that a set of vertices is a cycle and compute its weight).
- **NP**-hardness uses reduction from HAMILTONIANCycle.

**What to do:** given an instance of HAMILTONIANCycle, we turn it into an instance of TSP.

- we keep the same set of vertices;
- if there is an edge between \((v_i, v_j)\), we put \(w(v_i, v_j) = 1\);
- otherwise, we put \(w(v_i, v_j) = 2\).

Then, there is a cycle of length \(\leq n + 1\) in the new graph if and only if there is a Hamiltonian cycle in the input graph.
Non-approximability

Prop. Suppose that TSP has an approximation threshold $< 1$. Then $P=NP$.

(which is very unlikely.)

Proof.

- Suppose there is an $\varepsilon$-algorithm for TSP, with $\varepsilon < 1$. We are going to solve \textsc{HamiltonianCycle} in polynomial time (which proves that $P=NP$).
- Given an instance of \textsc{HamiltonianCycle}, we construct an instance of TSP as before:
  - we put weight 1 for non-connected cities;
  - we put weight $|V|/(1 - \varepsilon) > 1$ for connected cities.
- We apply our approximation algorithm to this graph . . .
Recall what $\varepsilon$-approximate means: for any graph, the output of the algorithm satisfies

$$\frac{w_{\text{output}} - w_{\text{optimal}}}{w_{\text{output}}} \leq \varepsilon,$$

or equivalently

$$w_{\text{output}} \leq \frac{w_{\text{optimal}}}{1 - \varepsilon}.$$
Non-approximability

Claim.

• Hamiltonian cycles in the old graph correspond to cycles of weight $|V|$ on the new graph.

• Any other cycle in the new graph will have weight $> |V|/(1 - \varepsilon)$.

Proof: obvious!

Output of the approximate algorithm:

• either we get a cycle of weight $w_{\text{output}} = |V|$. 
  there is a Hamiltonian cycle in the old graph

• or we get a cycle of weight $w_{\text{output}} > |V|/(1 - \varepsilon)$. In that case, $w_{\text{optimal}} > |V|$. 
  there is no Hamiltonian cycle in the old graph