Lecture 11: Approximation algorithms

- Coping with NP-completeness
- The Set Cover problem
- Approximation algorithms
- The Vertex Cover problem
- The Travelling Salesman problem
- The Knapsack problem
- An approximability hierarchy
11.1 Coping with NP-completeness

- Many problems of great practical interest are NP-complete, i.e. not likely to be fully solvable by polynomial-time algorithms.
- Nevertheless one **must** deal with these problems in practice, so what to do?
“Incomplete” approaches to NP-complete problems

1. Exponential-time algorithms (→ T-79.4101)
   - Backtracking, branch-and-bound search, A* search
   - Integer linear programming
   - Algebraic techniques (current “hot topic”)

2. Local search methods (→ T-79.4101)
   - Steepest descent (iterated, randomised, …)
   - Simulated annealing
   - Tabu search, genetic algorithms, “ant-colony” optimisation, …

3. Approximation algorithms (→ T-79.5207)
   - Polynomial-time algorithms for NP-hard optimisation problems
   - Produce feasible, but generally nonoptimal solutions
   - Solutions come with a guaranteed approximation bound
11.2 The Set Cover problem

- **Set Cover [Optimisation] Problem** (SC[O]): Given a finite base set $B$ and a collection of subsets $S = \{S_1, \ldots, S_n\}$ of $B$, determine the smallest number of sets in $S$ whose union covers all of $B$.

- A candidate solution $\{ S'_1, \ldots, S'_k \} \subseteq S$ to a given instance of the SC problem is feasible if $\bigcup_{i=1}^{k} S'_i = B$, and its cost is the number of sets picked, i.e. $k$. An optimal solution is a feasible solution with minimum cost.

- The Set Cover decision problem SC is NP-complete, so it is unlikely that an efficient algorithm for it, or the corresponding optimisation problem SC[O] exists.
A greedy approximation heuristic

- A straightforward greedy heuristic for the Set Cover optimisation problem is the following:

  Repeat until all elements in $B$ are covered:
  Pick set $S_i$ with most uncovered elements

- This does not always yield optimal solutions. (Example!)
- However, one can establish the following bound:
- **Claim.** Suppose $B$ contains $n$ elements and the optimal cover consists of $k$ sets. Then the greedy algorithm will pick at most $k \ln n$ sets.
- I.e. the greedy algorithm is guaranteed to be always at most a factor of $\ln n$ worse than the optimum. We say that it has an **approximation bound** of $\ln n$.\(^1\)

\(^1\)In fact, if P $\neq$ NP, there exists a constant $0 < c < 1$ such that no polynomial-time approximation algorithm can have an approximation bound of $c \ln n$. [Raz & Safra 1997; Alon, Moshkovitz & Safra 2006]
Proof of approximation bound

- Let $n_t$ be the number of elements still uncovered after $t$ steps of the greedy algorithms. Initially $n_0 = n$.
- These $n_t$ elements are covered by the optimal $k$ sets, so some set in the optimal cover must contain $\geq n_t/k$ of them.
- By the greedy set selection rule, then:

$$n_{t+1} \leq n_t - \frac{n_t}{k} = n_t \left(1 - \frac{1}{k}\right).$$

- By repeated application of this inequality one obtains:

$$n_t \leq n_0 \left(1 - \frac{1}{k}\right)^t < n_0(e^{-1/k})^t = ne^{-t/k}.$$

- At $t = k \ln n$, this inequality results in $n_t < ne^{-\ln n} = 1$. I.e. after this many stages of the greedy algorithm no more elements remain to be covered.
In general, an optimisation problem $\Pi = (\mathcal{I}, S, c)$ consists of:

- An infinite set of instances $\mathcal{I}$.
- For each instance $I \in \mathcal{I}$ a set of feasible solutions $S_I \subseteq S$.
  - Assume for simplicity that the sets of feasible solutions are nonempty and disjoint, i.e. $S_I \neq \emptyset$ for all $I$, and if $I \neq I'$, then $S_I \cap S_{I'} = \emptyset$.
- An objective (cost, value) function $c : S \to \mathbb{R}$.
  - Assume for simplicity that the values of the objective function are positive integers, i.e. that $c : S \to \mathbb{Z}^+$.

Assume that the problem $\Pi$ under consideration is a minimisation problem.\(^2\) Then the optimal cost for an instance $I \in \mathcal{I}$ is

$$c^*(I) = \min_{s \in S_I} c(s),$$

and a solution $s \in S_I$ is optimal if $c(s) = c^*(I)$.

\(^2\)For maximisation problems, analogous definitions can be given.
An approximation algorithm for optimisation (minimisation) problem $\Pi$ is a polynomial-time algorithm $\mathcal{A}$ that for any instance $I \in \mathcal{I}$ produces a solution $\mathcal{A}(I) \in S_I$.

The approximation ratio of $\mathcal{A}$ is defined as

$$\alpha_\mathcal{A} = \sup_{I \in \mathcal{I}} \frac{c(\mathcal{A}(I))}{c^*(I)}.$$ 

Sometimes it is useful to consider the approximation ratio of $\mathcal{A}$ as a function of input size $n$:³

$$\alpha_\mathcal{A}(n) = \max_{|I|=n} \frac{c(\mathcal{A}(I))}{c^*(I)}.$$ 

³We assume some natural notion of input size.
Approximation algorithms (3/3)

- An algorithm $\mathcal{A}$ is an $r(n)$-approximation algorithm if $\alpha_{\mathcal{A}}(n) \leq r(n)$ for all input sizes $n$.
- In other words: for any instance $I$ of size $n$ and (unknown) optimal cost $c^*(I) = k$, algorithm $\mathcal{A}$ is guaranteed to produce a solution of cost at most $r(n) \cdot k$.
- Thus, for example, the greedy set cover heuristic of Section 11.2 is an $\ln n$-approximation algorithm.
11.4 The Vertex Cover problem

- **Vertex Cover [Optimisation] Problem (VC[O])**: 
  - *Instance*: Graph $G = (V, E)$.
  - *Feasible solutions*: Subsets of vertices $S \subseteq V$ such that each edge $e \in E$ is incident to (“covered by”) at least one vertex $u \in S$.
  - *Cost (objective) function*: $c(S) = |S|$.

- The Vertex Cover decision problem is NP-complete.
- Since Vertex Cover is a special case of Set Cover (check this!), the greedy set cover heuristic yields an $\ln n$-approximation algorithm also for Vertex Cover.\(^4\)
- Moreover, there are VC instances where this bound is tight.
- However, in the case of VC there is another equally simple heuristic with a better approximation bound.

---

\(^4\)Work out carefully how the algorithm works in the case of Vertex Cover!
Maximal matchings

- Recall from Lecture 7:
  - A matching $M$ in graph $G$ is a set of edges with no shared endpoints, i.e. a 1-to-1 pairing among some subset of the vertices. A matching in $G$ is maximal if it is not a subset of a matching of larger cardinality.

- Maximal matchings are easy to generate: repeatedly pick edges that are disjoint from ones chosen already, until this is no longer possible.
Matchings and vertex covers

- **Key fact:** Given any vertex cover $S$ and any matching $M$ in a graph $G$, $|S| \geq |M|$. (Why?!) This holds in particular also for any *optimal* vertex cover $S^\ast$.

- Thus, for a given graph $G$, even though we don’t know the value of $c^\ast(G)$, we can lower bound this as

  $$c^\ast(G) = |S^\ast| \geq |M|,$$

  where $M$ is any matching in $G$.

- Consider, on the other hand, a *maximal* matching $\hat{M}$ in a graph $G$. Then a vertex cover $\hat{S}$ of $G$ is obtained by including into it both endpoints of each edge in $\hat{M}$. (Why?)

- This vertex cover has size $|\hat{S}| = 2|\hat{M}|$. 
A 2-approximation algorithm for VC

- We now obtain a very simple approximation algorithm \( A \):
  - Given an input graph \( G = (V, E) \):
    1. Find a maximal matching \( \hat{M} \subseteq E \).
    2. Return set \( \hat{S} = \{ \text{all endpoints of edges in } \hat{M} \} \).
  - The approximation ratio of this algorithm satisfies:
    \[
    \frac{c(A(G))}{c^*(G)} = \frac{\hat{S}}{S^*} \leq \frac{2|\hat{M}|}{|\hat{M}|} = 2.
    \]

- Note key ingredient in the construction: a simple auxiliary structure that can be used to:
  (a) provide a lower bound on the unknown optimal cost,
  (b) construct a feasible solution with a known upper bound on cost.
11.5 The Travelling Salesman problem

- **Travelling Salesman [Optimisation] Problem (TSP[O]):**
  - **Instance:** An \( n \times n \) distance matrix \( D = [d_{ij}] \), \( d_{ij} \in \mathbb{Z}^+ \) for \( i \neq j \), \( d_{ii} = 0 \).
  - **Feasible solutions:** Permutations (“tours”) \( \pi \) of the set \([n] = \{1, \ldots, n\}\).
  - **Cost (objective) function:** \( c(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \).

- A TSP instance \( D = [d_{ij}] \) is:
  - **symmetric** if \( d_{ij} = d_{ji} \) for all \( i, j \);
  - **metric** if \( d_{ik} \leq d_{ij} + d_{jk} \) for all \( i, j, k \) (triangle inequality holds);
  - **Euclidean** if the distances \( d_{ij} \) correspond to actual geometric distances in the plane (then of course \( d_{ij} \in \mathbb{R}^+ \) rather than \( d_{ij} \in \mathbb{Z}^+ \) as we are now assuming).

- TSP instances are usually assumed to be symmetric unless otherwise indicated, and we shall follow this convention.
Metric TSPs and minimum spanning trees (1/4)

- Consider a given symmetric TSP instance $D$ as a weighted undirected complete graph $G = (V, E, d)$ on the vertex set $V = \{1, \ldots, n\}$.

- Following the approximation algorithm scheme for Vertex Cover, what would be simple lower-bounding structures for optimal TSP tours in this case?

- Answer: Minimum-weight spanning trees of $G$!

- Consider namely any TSP tour $\pi$ on $G$. If one removes any edge from $\pi$, e.g. the last one $\pi(n) \rightarrow \pi(1)$, one is left with a spanning path $P$ of $G$.

- Because a spanning path is a spanning tree,

\[ c(\pi) \geq c(P) \geq c(T^*), \]

where $T^*$ is any minimum-weight spanning tree of $G$. 


Now how to use a spanning tree to obtain a good TSP tour? One can almost do this by “traversing twice around the tree”:

- Starting from a tree $T$ of cost $c(T)$, this yields a walk $W$ of cost $c(W) = 2c(T)$. However, the walk $W$ is not a TSP tour, because it visits some (actually all) vertices several times.
Metric TSPs and minimum spanning trees (3/4)

▶ Luckily, in a metric TSP this problem can be fixed by taking “shortcuts”: every time the walk $W$ would return to a previously visited vertex, the eventual tour skips to the next vertex, or if that is already visited, then to the next one, etc.

▶ Because of the triangle inequality, such shortcuts can never increase the length of a walk, so eventually the process results in a TSP tour $\pi$ satisfying:

$$c(\pi) \leq c(W) \leq 2c(T).$$
Metric TSPs and minimum spanning trees (4/4)

- One thus obtains the following approximation algorithm $A$:
  - Given an input graph $G = (V, E, d)$ representing a metric TSP instance:
    1. Find a minimum spanning tree $T^*$ of $G$.
    2. Construct a TSP tour $\pi$ in $G$ by traversing twice around $T^*$, taking shortcuts whenever needed.

- The approximation ratio of this algorithm satisfies:
  $$\frac{c(A(G))}{c^*(G)} = \frac{c(\pi)}{c(\pi^*)} \leq \frac{2c(T^*)}{c(T^*)} = 2.$$

- The ratio can still be improved to 1.5 by graph matching techniques (Christofides 1976).
Nonmetric TSPs (1/2)

- In the case of TSPs where the triangle inequality does not hold the approximability situation is dramatically worse: unless $P = NP$, then no polynomial-time approximation algorithm for general nonmetric TSPs can guarantee any constant-factor approximation bound.

- Indeed, for any constant $C > 0$ there is a reduction $f_C$ from the NP-complete Hamiltonian Cycle problem\(^5\) to nonmetric TSP with the following “gap expansion” property:
  - For a graph $G$ with $n$ vertices:
    1. if $G$ contains a Hamiltonian cycle, then $c^*(f_C(G)) = n$;
    2. if $G$ does not contain a Hamiltonian cycle, then $c^*(f_C(G)) \geq n + C$.
  - The reduction $f_C$ forms a TSP instance $D$ from $G$ by associating to each edge $\{i, j\}$ in $G$ the distance value $d_{ij} = 1$, and sets all other distances to $d_{ij} = C + 1$.

\(^5\)The textbook Dasgupta et al. uses the nonstandard name “Rudrata Cycle problem”.
Nonmetric TSPs (2/2)

▶ Suppose then that there would be a general polynomial-time TSP approximation algorithm $A$ with constant approximation bound $k$.
▶ Consider the following decision algorithm for Hamiltonian Cycle:
   ▶ Given a graph $G = (V, E)$ with $n$ vertices:
     1. Let $C = k \cdot n$ and compute the TSP instance $D = f_C(G)$.
     2. Compute the (presumed) $k$-approximate tour $\pi = A(D)$.
     3. If $c(\pi) \leq k \cdot n$, then $G$ contains a Hamiltonian cycle, otherwise not.
   ▶ Note that by the (presumed) approximation bound, the tour $\pi$ produced in step 2 of the algorithm always has cost either $c(\pi) \leq k \cdot n$ or $c(\pi) \geq n + C > k \cdot n$.
▶ In case the presumed polynomial-time $k$-approximation algorithm $A$ for TSP really existed, the above decision algorithm for Hamiltonian Cycle would also run in polynomial time, implying $P = NP$. 
11.6 The Knapsack problem

- Recall from Lecture 9:
- **Knapsack [Optimisation] Problem** (Knapsack[O]):
  - *Instance:* A family of $n$ items ("weights") $\{w_1, \ldots, w_n\}$ and their values $\{v_1, \ldots, v_n\}$, knapsack weight bound $W$. (All positive integers.)
  - *Feasible solutions:* Sets of items $K \subseteq [n] = \{1, \ldots, n\}$ satisfying $\sum_{i \in K} w_i \leq W$.
  - *Value (objective) function:* $v(K) = \sum_{i \in K} v_i$. 


Approximability of the Knapsack problem

- The Knapsack (decision) problem is NP-complete, so an efficient algorithm is unlikely. Thus approximations, or other incomplete techniques, need to be considered.

- Since Knapsack [Optimisation] is a maximisation problem, the notion of approximation ratio for an approximation algorithm \( \mathcal{A} \) is defined as:

\[
\alpha_{\mathcal{A}} = \sup_{I \in \mathcal{I}} \frac{v^*(I)}{v(\mathcal{A}(I))}.
\]

- In the case of Knapsack, it is possible to obtain for any constant \( \varepsilon > 0 \) a polynomial-time approximation algorithm \( \mathcal{A}_\varepsilon \) satisfying

\[
v(\mathcal{A}_\varepsilon(I)) \geq (1 - \varepsilon)v^*(I)
\]

for all instances \( I \), i.e.

\[
\alpha_{\mathcal{A}_\varepsilon} \leq \frac{1}{1 - \varepsilon} \sim 1 + \varepsilon.
\]
A polynomial time approximation scheme (1/3)

- The approximation scheme for Knapsack combines the dynamic programming approach from Lecture 9 with a rescaling technique.
- Let the total value of the available items be $V = \sum_i v_i$. Using dynamic programming, it is straightforward to find an exactly optimal combination of items in time $O(nV)$.
- The problem with this approach is that $V$ may be an exponentially large number, in terms of its bit representation size.
- But we can trade time for accuracy! Idea:
  1. rescale large numbers to reasonable size
  2. cut off fractional part
  3. run efficient algorithm on small integers
  4. rescale back to large numbers (if necessary)
  5. check loss of accuracy
A polynomial time approximation scheme (2/3)

Let $\varepsilon > 0$ be the desired approximation error. The corresponding approximation algorithm $A_{\varepsilon}$ is the following:

1. Discard any item with weight $> W$.
2. Let $v_{\text{max}} = \max_i v_i$.
3. Rescale values as $\hat{v}_i = \left\lfloor v_i \cdot \frac{n}{\varepsilon v_{\text{max}}} \right\rfloor$.
4. Run the dynamic programming algorithm with values $\hat{v}_i$.
5. Output the resulting choice of items.

For the analysis, note first that since all the rescaled values $\hat{v}_i$ are at most $n/\varepsilon$, the dynamic programming method runs in time $O(n^3/\varepsilon)$, i.e. polynomial time in the parameters $n$ and $1/\varepsilon$. 
A polynomial time approximation scheme (3/3)

Now suppose the optimal solution to the original problem is to pick some subset of items $K$, with total value $V^*$.

The rescaled value of this same assignment is:

$$\sum_{i \in K} \hat{v}_i = \sum_{i \in K} \left[ v_i \cdot \frac{n}{\varepsilon v_{\text{max}}} \right] \geq \sum_{i \in K} \left( v_i \cdot \frac{n}{\varepsilon v_{\text{max}}} - 1 \right) \geq V^* \cdot \frac{n}{\varepsilon v_{\text{max}}} - n.$$

Therefore the optimal selection of items for the reduced problem, call it $\hat{K}$, has a rescaled value of at least this much. In terms of the original problem, selection $\hat{K}$ thus has value:

$$\sum_{i \in \hat{K}} v_i \geq \sum_{i \in \hat{K}} \hat{v}_i \cdot \frac{\varepsilon v_{\text{max}}}{n} \geq \left( V^* \cdot \frac{n}{\varepsilon v_{\text{max}}} - n \right) \cdot \frac{\varepsilon v_{\text{max}}}{n} = V^* - \varepsilon v_{\text{max}}$$

$$\geq V^* (1 - \varepsilon).$$
11.7 An approximability hierarchy

- We have seen that NP-complete optimisation problems can have broadly different approximability characteristics:
  - A polynomial time approximation scheme (approximability class PTAS): Knapsack
  - Polynomial time, constant factor approximations (approximability class APX): Vertex Cover, Metric TSP
  - No constant factor approximations, unless P = NP: general TSP

- There is currently a broad scientific literature on these topics, containing a large number of nontrivial combinatorial approximation algorithms, and some rather deep general theory on approximability lower bounds.\(^6\)

---

\(^6\)See e.g. the textbook V. Vazirani, “Approximation Algorithms” (Springer-Verlag 2001) and the course T-79.5207.