Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 14: Approximability

Aalto University
School of Science
Department of Computer Science

Spring 2017

Agenda

- Approximation algorithms
- Approximation and complexity
- Nonapproximability results

(C. Papadimitriou: *Computational Complexity*, Chapter 13, 299–322)

1. Approximation Algorithms

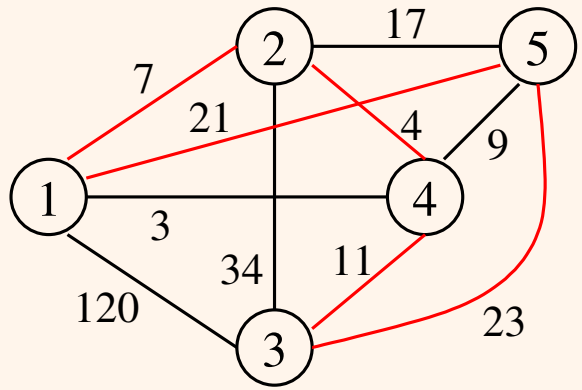
- Once **NP**-completeness of a problem has been established, techniques for solving the problem only approximatively are usually explored.
- When dealing with optimisation problems, often heuristic (search) algorithms are used.
- Such algorithms are valuable in practice even if usually nothing can be proved about their worst-case (or expected) performance.
- In some (fortunate) cases, the solutions returned by a polynomial-time heuristic algorithm are guaranteed to be “not too far from the optimum”.

Definition

An optimisation problem Π comprises an infinite set of *instances* such that for each instance x, there is a set of *feasible solutions* $F(x)$, and each solution $s \in F(x)$ has an associated a positive integer *cost* $c(s)$. The task is to find a feasible solution of optimum cost, defined as $\mathrm{OPT}(x) = \min_{s \in F(x)} c(s)$ (or $\max_{s \in F(x)} c(s)$ if Π is a maximisation problem).

Example (TSP)

- Instance x:
  A complete edge-weighted graph, given as distance matrix
- All $n!$ tours are feasible solutions
- Tour s has cost $c(s) =$ sum of distances along s



Definition

Let M be an algorithm which given any instance x returns a feasible solution $M(x) \in F(x)$. We say that M is an *ε-approximation algorithm*, where $0 \leq \varepsilon \leq 1$, iff for all inputs x,

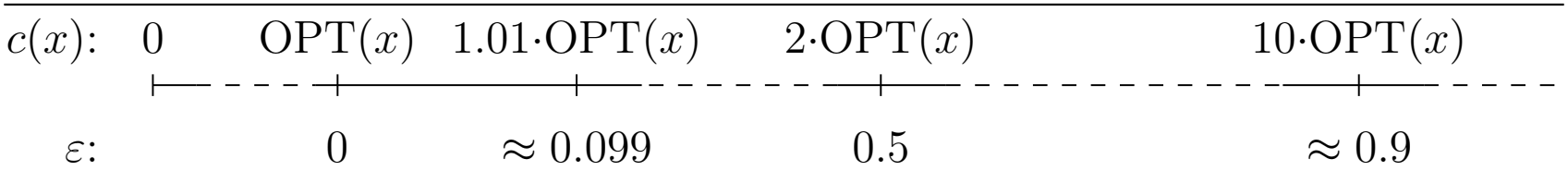
$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \varepsilon.$$

Note: ε-approximation means that the relative error is at most ε.

- For a minimisation problem

$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} = \frac{c(M(x)) - \mathrm{OPT}(x)}{c(M(x))} \leq \varepsilon$$

and hence, $c(M(x)) \leq \frac{1}{1-\varepsilon}\mathrm{OPT}(x)$ i.e. $(1-\varepsilon)c(M(x)) \leq \mathrm{OPT}(x)$



Definition

Let M be an algorithm which given any instance x returns a feasible solution $M(x) \in F(x)$. We say that M is an *ε-approximation algorithm*, where $0 \leq \varepsilon \leq 1$, iff for all inputs x,

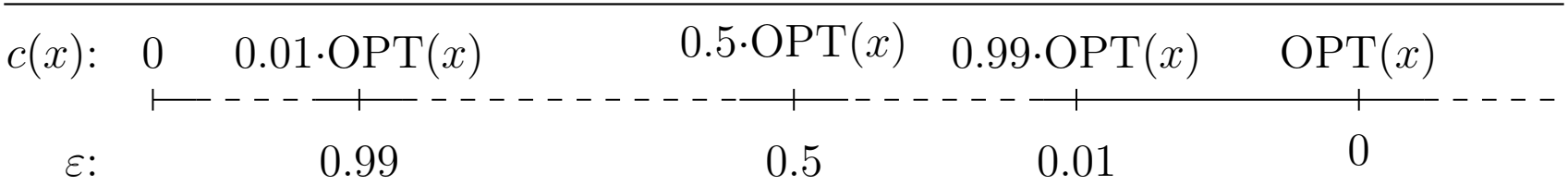
$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} \leq \varepsilon.$$

Note: ε-approximation means that the relative error is at most ε.

- For a maximisation problem

$$\frac{|c(M(x)) - \mathrm{OPT}(x)|}{\max\{\mathrm{OPT}(x), c(M(x))\}} = \frac{\mathrm{OPT}(x) - c(M(x))}{\mathrm{OPT}(x)} \leq \varepsilon$$

and hence, $c(M(x)) \geq (1-\varepsilon)\mathrm{OPT}(x)$.



Approximation Thresholds

- For an optimisation problem A, we are interested in *determining the smallest ε for which there is a polynomial-time ε-approximation algorithm for A*.
- Sometimes no such smallest ε exists, but there are approximation algorithms that achieve arbitrarily small error ratios.
- The *approximation threshold* of A is the greatest lower bound (infimum, **inf**) of all $\varepsilon > 0$ for which A has a polynomial-time ε-approximation algorithm.
- This quantity ranges from 0 (arbitrarily close approximation is possible) to 1 (essentially no approximation is possible).
- If $\mathbf{P} = \mathbf{NP}$, then for all optimisation problems in $\mathbf{NP}$ the approximation threshold is zero.

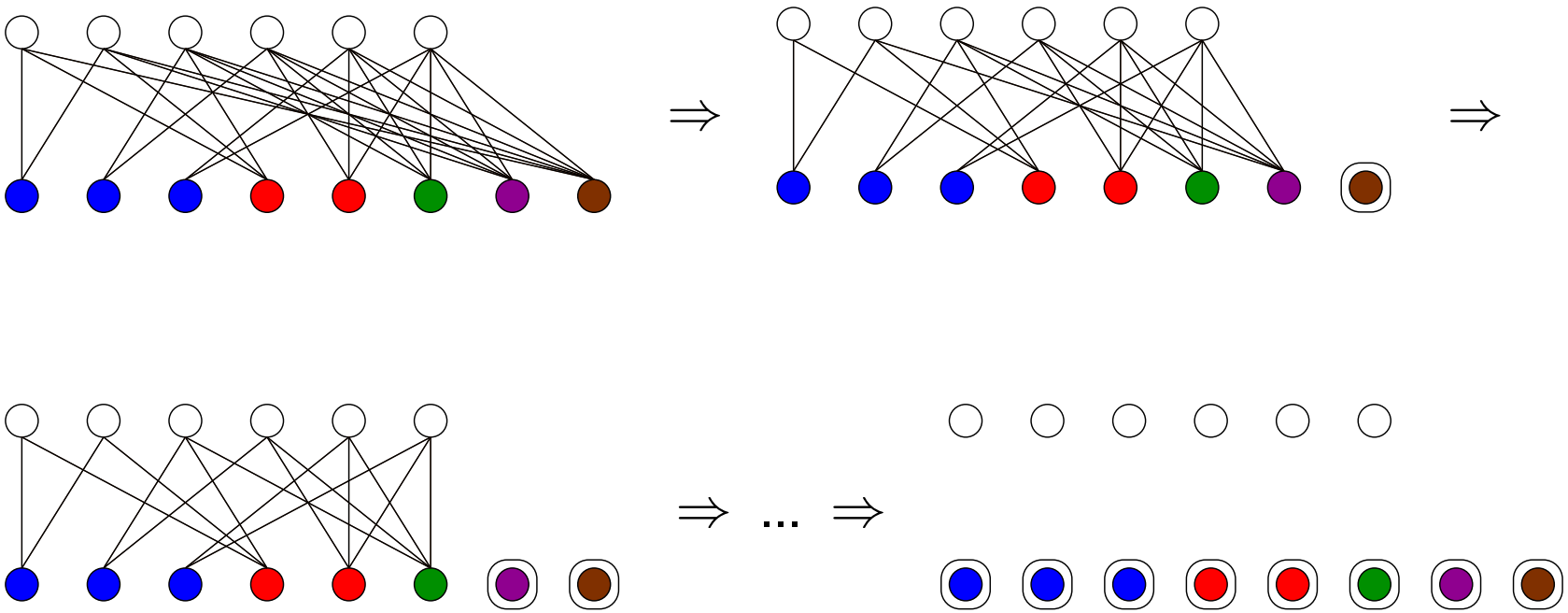
Case: Vertex Cover

- VERTEX COVER is a minimisation problem where we seek the smallest set of vertices $C \subseteq V$ in a graph $G = (V, E)$ such that for each edge in E at least one of its endpoints is in C.
- What is a plausible heuristic for obtaining a "good" vertex cover?
- A first try: If a vertex v has high degree, then it is probably a good idea to add it to the cover.
- The resulting "greedy" algorithm:

  Start with $C = \emptyset$
  While there are still edges left in G:
    choose a vertex $v \in V$ with the largest degree
    delete v (and related edges) from G and add v to C

- This is *not* an ε-approximation algorithm for any $\varepsilon < 1$ (in the worst-case its error ratio grows as $\log n$ where n is the number of vertices in the graph).

Vertex Cover: Greedy Algorithm Example



- To get an approximation algorithm for VERTEX COVER a less "greedy" approach needs to be taken such as:

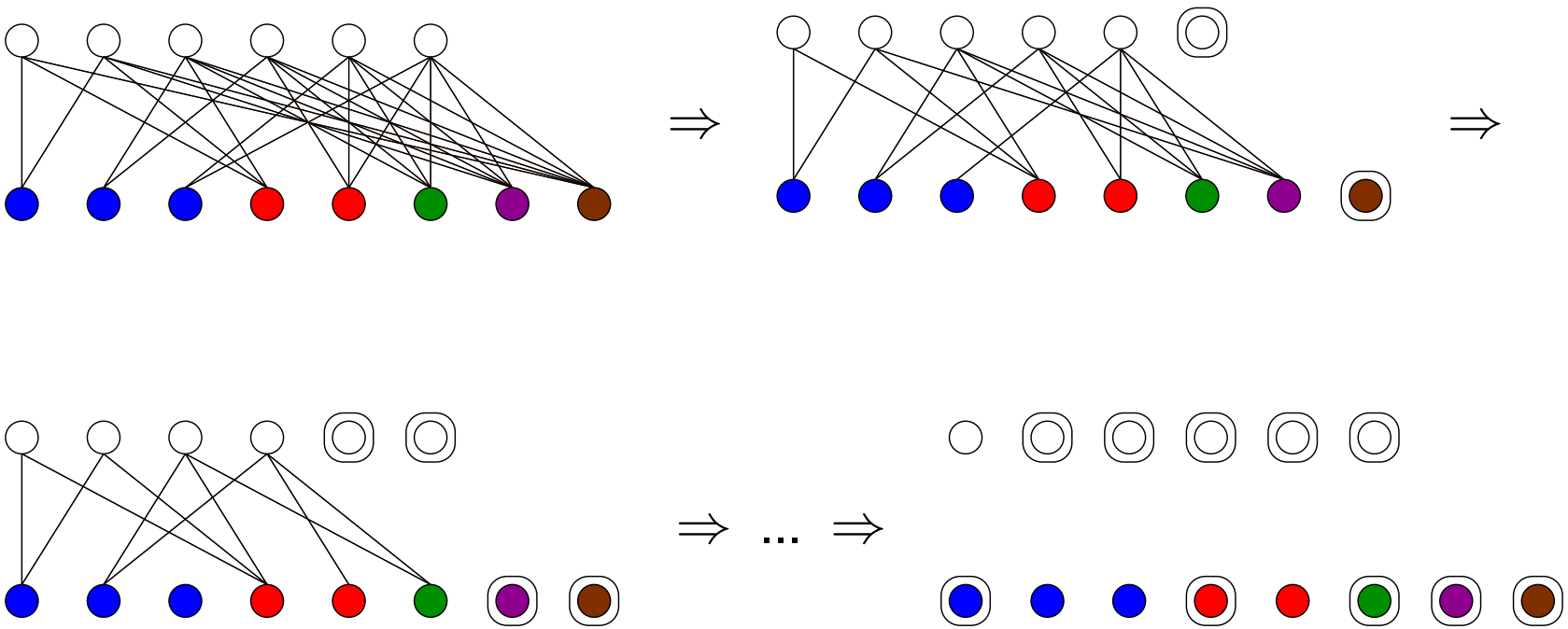
  Start with $C = \emptyset$
  While there are still edges left in G
    choose any edge $\{u, v\}$
    add both u and v to C and delete them from G.

- How far off the optimum can C be?
  - C "has" $\frac{1}{2}|C|$ edges of G (no two of which share a vertex).
  - Also the optimum cover must contain at least one vertex from each such edge.
  - Hence, $\mathrm{OPT}(G) \geq \frac{1}{2}|C|$ and, thus, $\frac{|C|-\mathrm{OPT}(G)}{|C|} \leq \frac{|C|-\frac{1}{2}|C|}{|C|} = \frac{1}{2}$.

Theorem

The approximation threshold of VERTEX COVER is at most $\frac{1}{2}$.

Vertex Cover: Less-Greedy Algorithm Example



Case: Maximum Satisfiability

- Recall: MAXSAT and even MAX2SAT are $\mathbf{NP}$-complete.
- Consider first the k-MAXGSAT problem (maximum generalised satisfiability): we are given a set of Boolean expressions $\Phi = \{\phi_1, \ldots, \phi_m\}$ over n variables, where each expression is a general Boolean expression involving at most k of the n variables ($k > 0$ is a *fixed* constant). The task is to find a truth assignment that satisfies the most expressions.
- A successful approximation algorithm is based on choosing for a variable always the truth value that maximises the expected number of satisfied expressions.

The expected number of satisfied expressions:

- Suppose we pick one of the 2^n truth assignments at random. How many expressions in Φ should we expect to satisfy?
- Each expression $\phi_i \in \Phi$ involves $r \leq k$ Boolean variables.
- We can easily calculate the number t_i of truth assignments (out of 2^r truth assignments) that satisfy ϕ_i (as r is a constant).
- Thus, a random truth assignment will satisfy ϕ_i with probability $p(\phi_i) = \frac{t_i}{2^r}$
- The expected number of satisfied expressions is then $\mathrm{E}(\Phi) = \sum_{i=1}^{m} p(\phi_i)$ (by linearity of expectation)

Example

	$x \vee y \vee z$	$x \Leftrightarrow y$	$y \wedge z$	$\neg x \vee \neg z$	E
$p(\phi_i)$	$\frac{7}{8}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$2\frac{3}{8}$

- If we set x_1 to **true** in all expressions of Φ, a set of expressions $\Phi[x_1 = \textbf{true}]$ involving variables $x_2, \ldots, x_n$ results. We can calculate again $\mathrm{E}(\Phi[x_1 = \textbf{true}])$ (and $\mathrm{E}(\Phi[x_1 = \textbf{false}])$ similarly).
- Now

$$\mathrm{E}(\Phi) = \frac{1}{2}(\mathrm{E}(\Phi[x_1 = \textbf{true}]) + \mathrm{E}(\Phi[x_1 = \textbf{false}]))$$

Example

	$x \vee y \vee z$	$x \Leftrightarrow y$	$y \wedge z$	$\neg x \vee \neg z$	E
$p(\phi_i)$	$\frac{7}{8}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$2\frac{3}{8}$
$p(\phi_i[x = \textbf{false}])$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	1	$2\frac{1}{2}$
$p(\phi_i[x = \textbf{true}])$	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$2\frac{1}{4}$

- Hence, if we modify Φ by setting x_1 equal to the truth value t that yields the largest $\mathrm{E}(\Phi[x_1 = t])$, we end up with an expression set with expectation at least as large as the original.
- The approximation algorithm:

  Set $\Phi' = \Phi$ and then for $i = 1$ to n

  compute $\mathrm{E}(\Phi'[x_i = \mathbf{true}])$ and $\mathrm{E}(\Phi'[x_i = \mathbf{false}])$

  choose the truth value t that yields the largest $\mathrm{E}(\Phi'[x_i = t])$

  set $\Phi' = \Phi'[x_i = t]$

Example

	$x \vee y \vee z$	$x \Leftrightarrow y$	$y \wedge z$	$\neg x \vee \neg z$	E
$p(\phi_i)$	$\frac{7}{8}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$2\frac{3}{8}$
$p(\phi_i[x = \mathbf{false}])$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	1	$2\frac{1}{2}$
$p(\phi_i[x = \mathbf{true}])$	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$2\frac{1}{4}$
$p(\phi_i[x = \mathbf{false}, y = \mathbf{false}])$	$\frac{1}{2}$	1	0	1	$2\frac{1}{2}$
$p(\phi_i[x = \mathbf{false}, y = \mathbf{true}])$	1	0	$\frac{1}{2}$	1	$2\frac{1}{2}$
$p(\phi_i[x = \mathbf{false}, y = \mathbf{false}, z = \mathbf{false}])$	0	1	0	1	2
$p(\phi_i[x = \mathbf{false}, y = \mathbf{false}, z = \mathbf{true}])$	1	1	0	1	3

- In the end, all variables have been given values and all expressions are either **true** or **false**. Moreover we know that at least $\mathrm{E}(\Phi)$ have been satisfied ($c(M(\Phi)) \geq \mathrm{E}(\Phi)$).
- The optimum is at most the number l of expressions that can be individually satisfied ($p(\phi_i) > 0$), i.e., $\mathrm{OPT}(\Phi) \leq l$. Then

$$\frac{\mathrm{OPT}(\Phi) - c(M(\Phi))}{\mathrm{OPT}(\Phi)} = 1 - \frac{c(M(\Phi))}{\mathrm{OPT}(\Phi)} \leq 1 - \frac{\mathrm{E}(\Phi)}{\mathrm{OPT}(\Phi)} \leq 1 - \frac{lp(\phi_i)}{\mathrm{OPT}(\Phi)} \leq 1 - \frac{lp(\phi_i)}{l} = 1 - p(\phi_i)$$

  where $p(\phi_i)$ is the smallest positive probability and, hence, $\mathrm{E}(\Phi) \geq lp(\phi_i)$.
- For every satisfiable expression ϕ_i involving at most k variables $p(\phi_i)$ is at least 2^{-k}.
- Hence, the approximation threshold for k-MAXGSAT is at most $1 - 2^{-k}$.

- In MAXSAT the input is a set of clauses and the probability of satisfaction is at least $\frac{1}{2}$ and hence $\varepsilon = \frac{1}{2}$.
- If we restrict the clauses to have at least k distinct literals, the probability that a random truth assignment satisfies a clause is $1 - 2^{-k}$ and $\varepsilon = 2^{-k}$.

Theorem

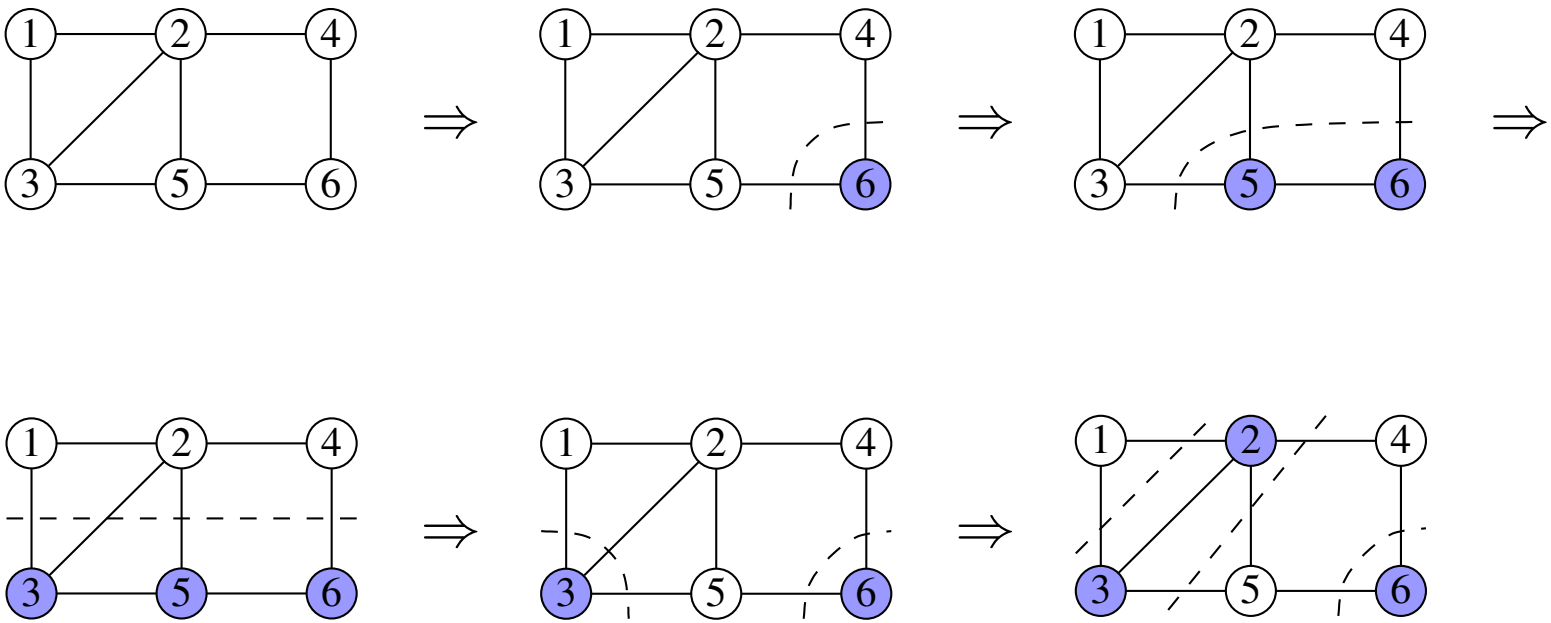
The approximation threshold for k-MAXGSAT is at most $1 - 2^{-k}$.
The approximation threshold for MAXSAT is at most $\frac{1}{2}$
and when each clause has at least k distinct literals, the approximation threshold is at most 2^{-k}.

Case: Maximum Cut

- In MAX CUT we want to partition the vertices of a graph $G = (V, E)$ into two sets S and $V - S$ such that there are as many edges as possible between S and $V - S$.
- An approximation algorithm for MAX CUT based on *local improvement*:
  Start from any partition of the vertices of G and repeat the following step: If the cut can be made larger by adding a single vertex to S or by deleting a single vertex from S, then do so. If no such improvement is possible, stop and return the cut thus obtained.
- Such local improvement algorithms can be developed for just about any optimisation problem.
- Sometimes such algorithms work well in practice but usually very little can be proved about their performance.
- MAX CUT is an exception:

Theorem

The approximation threshold for MAX CUT is at most $\frac{1}{2}$.



Case: The Travelling Salesperson Problem

TSP cannot be approximated!

Theorem

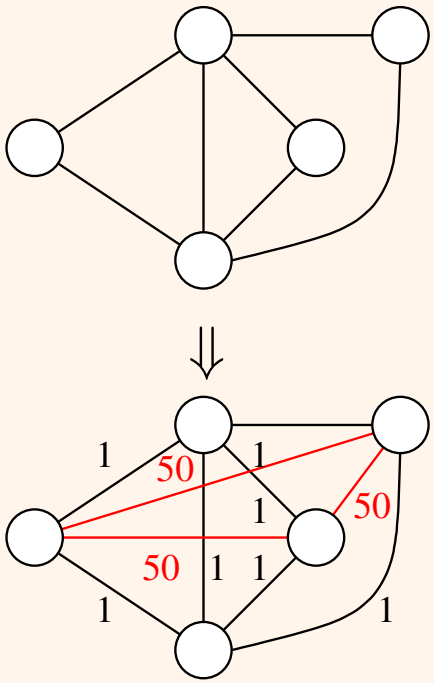
Unless $\mathbf{P} = \mathbf{NP}$*, the approximation threshold for TSP is one.*

Proof sketch

- Suppose that there is a polynomial time ε-approximation algorithm for TSP for some $\varepsilon < 1$.
- Using this algorithm we derive a polynomial-time algorithm for the $\mathbf{NP}$-complete problem HAMILTON CYCLE, which implies $\mathbf{P} = \mathbf{NP}$.
- Given a graph $G = (V, E)$ the algorithm for HAMILTON CYCLE constructs an instance of TSP with $|V|$ cities such that the distance between cities i and j is one if there is an edge between i and j in G and $\frac{|V|}{1-\varepsilon}$ otherwise.

Example

With $\varepsilon = 0.9$



Proof sketch—cont'd

- Then we run the polynomial time ε-approximation algorithm for TSP on this instance x of TSP. There are two cases:
  - If the algorithm returns a tour of total cost $|V|$, there is a Hamilton cycle for G.
  - Otherwise we say that there is no Hamilton cycle for G.
- Also in the latter case the decision is correct because then the tour returned contains at least one edge of the cost $\frac{|V|}{1-\varepsilon}$ and the total cost of the tour returned $c(M(x)) > \frac{|V|}{1-\varepsilon}$.
- As this is a polynomial time ε-approximation algorithm: $c(M(x)) \leq \frac{1}{1-\varepsilon}\mathrm{OPT}(x)$, i.e.,

$$\mathrm{OPT}(x) \geq (1-\varepsilon)c(M(x)) > \frac{(1-\varepsilon)|V|}{(1-\varepsilon)} = |V|$$

and there cannot be any Hamilton cycle for G.

- If all distances are either 1 or 2, there is a polynomial-time $\frac{1}{7}$-approximation algorithm.
- If the distances satisfy triangle inequality $d_{i,j} + d_{j,k} \geq d_{i,k}$, there is a polynomial-time $\frac{1}{3}$-approximation algorithm.

Case: Knapsack

- An instance of the KNAPSACK problem comprises a set of n items, each item i with *value* v_i and *weight* w_i, together with a weight limit W. (All parameters are positive integers). The task is to find a subset S of the items that satisfies the weight constraint $\Sigma_{i \in S} w_i \leq W$ and maximises $\Sigma_{i \in S} v_i$.
- KNAPSACK has a pseudopolynomial algorithm.
- For KNAPSACK polynomial-time approximability has no limits.

Theorem

The approximation threshold for KNAPSACK is zero.

- A polynomial-time ε-approximation algorithm for any $\varepsilon > 0$ can be derived from the following exact dynamic programming -type pseudopolynomial algorithm.
- Let $v_{\max} = \max\{v_1, \ldots, v_n\}$.
- For each $i = 0, 1, \ldots, n$ and $0 \leq V \leq nv_{\max}$, define the quantity $W(i, V)$: the minimum weight attainable by selecting among the first i items so that their total value is exactly V.
- We start with $W(0, 0) = 0$ and $W(0, V) = \infty$ for all $V \neq 0$.
- Each $W(i, V)$ with $i > 0$ can be computed by tabulating for increasing i and all V:

$$W(i, V) = \min\{W(i-1, V), W(i-1, V - v_i) + w_i\}$$

- In the end, we pick the largest V such that $W(n, V) \leq W$.
- Each entry can be computed in constant number of steps and there are $(n+1)(nv_{\max}+1)$ entries. Hence, the algorithm runs in $O(n^2 v_{\max})$ time.

Example (Knapsack: Pseudopolynomial Algorithm)

Weight limit 10, items $(760, 7), (830, 5), (700, 4), (895, 6)$

v_i	w_i	i \ $W(i,v)$	0	700	760	830	895	1460	1530	1590	1595	1655	1725	2290	2355	2425	2485	3185
895	6	4	0	4	7	5	6	11	9	12	10	13	11	16	17	15	18	22
700	4	3	0	4	7	5		11	9	12				16				
830	5	2	0		7	5				12								
760	7	1	0		7													
		0	0															

v

Optimal value: 1595 with items $(700, 4)$ and $(895, 6)$

Knapsack: Approximation Algorithm

- The algorithm *allows trading off accuracy for speed*.
- Given an instance of KNAPSACK $x = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$ we can define the approximate instance $x' = (w_1, \ldots, w_n, W, v'_1, \ldots, v'_n)$ where the new values are $v'_i = 2^b \lfloor \frac{v_i}{2^b} \rfloor$ (the old values with their b least significant bits replaced by zeros) where b is a parameter depending on ε.
- The time required to solve x' is $O(\frac{n^2 v_{\max}}{2^b})$ because we can ignore the trailing zeros in the v_is.
- The solution S' of x' obtained can be different from the optimal solution S of x but it can be shown that for $c(S') = \Sigma_{i \in S'} v'_i$ holds:

$$\sum_{i \in S} v_i \geq \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n2^b.$$

where $v_i \geq v'_i$ and $v'_i \geq v_i - 2^b$.

- Hence,

$$\frac{\mathrm{OPT}(x) - c(S')}{\mathrm{OPT}(x)} \leq \frac{\sum_{i \in S} v_i - (\sum_{i \in S} v_i - n2^b)}{\mathrm{OPT}(x)} \leq \frac{n2^b}{v_{\max}}$$

  where $\mathrm{OPT}(x) \geq v_{\max}$.

- Now given any $\varepsilon > 0$, let's truncate the last $b = \lfloor \log \frac{\varepsilon v_{\max}}{n} \rfloor$ bits of the values. Then $\frac{n2^b}{v_{\max}} \leq \frac{n\frac{\varepsilon v_{\max}}{n}}{v_{\max}} = \varepsilon$ and we arrive at an ε-approximation algorithm with running time $O(\frac{n^2 v_{\max}}{2^b}) = O(\frac{n^2 v_{\max}}{\frac{\varepsilon v_{\max}}{n}}) = O(\frac{n^3}{\varepsilon})$.
- Thus, there is a polynomial-time ε-approximation algorithm for any $\varepsilon > 0$ and the approximation threshold is zero.

Example (Knapsack: Approximation Algorithm Example)

Weight limit 10, items $(760, 7), (830, 5), (700, 4), (895, 6)$

Approximation (in base 10)

v_i	w_i	i	$W(i,v)$								
8	6	4	0	4	5	11	9	11	16	15	22
7	4	3	0	4	5	11	9		16		
8	5	2	0	7	5		12				
7	7	1	0	7							
		0	0								
			0	7	8	14	15	16	22	23	30

v

Obtained value: 1530 with items $(830, 5)$ and $(700, 4)$

Approximation Schemes

Definition

A *polynomial-time approximation scheme* for an optimisation problem A is an algorithm which, for each $\varepsilon > 0$ and instance x of A, returns a solution with a relative error of at most ε in time $p_\varepsilon(|x|)$ where p_ε is a polynomial depending on ε.

- In case of KNAPSACK, the time bound p_ε depends polynomially on $\frac{1}{\varepsilon}$ and the respective scheme is then called *fully polynomial*.
- For BIN PACKING, there is an approximation scheme where the time bound p_ε depends on $\frac{1}{\varepsilon}$ exponentially.

2. Approximation and Complexity

- A polynomial-time approximation scheme for an optimisation problem is the next best thing to a polynomial-time exact algorithm for the problem.
- For **NP**-complete optimisation problems an important question is whether such a scheme exists.
- One can use approximation preserving reductions (e.g. "L-reductions") to order optimisation problems by difficulty.

L-reductions

- An *L-reduction* from optimisation problem A to optimisation problem B is a pair of functions (R, S), both computable in logarithmic space, that satisfy the following two properties:
  (i) If x is an instance of A with optimum cost $\mathrm{OPT}(x)$, then $R(x)$ is an instance of B with optimum cost that satisfies

  $$\mathrm{OPT}(R(x)) \leq \alpha \mathrm{OPT}(x) \qquad \text{where } \alpha \text{ is a positive constant.}$$

  (ii) If s is any feasible solution of $R(x)$, then $S(s)$ is a feasible solution of x such that

  $$|\mathrm{OPT}(x) - c(S(s))| \leq \beta |\mathrm{OPT}(R(x)) - c(s)|$$

  where is β is another positive constant.
- Notice: S returns a feasible solution of x which is not much more suboptimal than the given solution of $R(x)$. By the second condition, if s is an optimum solution of $R(x)$, then $S(s)$ must be the optimum solution of x.

L-reductions compose

Proposition

If (R, S) is an L-reduction from problem A to problem B and (R', S') is an L-reduction from problem B to problem C, then their composition $(R \cdot R', S \cdot S')$ is an L-reduction from A to C.

Proposition

If there is an L-reduction (R, S) from A to B with constants α and β and there is a polynomial-time ε-approximation algorithm for B, then there is a polynomial-time $\frac{\alpha\beta\varepsilon}{1-\varepsilon}$-approximation algorithm for A.

Corollary

If there is an L-reduction (R, S) from A to B and there is a polynomial-time approximation scheme for B, then there is a polynomial-time approximation scheme for A.

Optional: Class MAXSNP

- *Fagin's theorem* characterises **NP** in terms of existential second order logic (expressions of the form $\exists P \phi$ where ϕ is first-order).
- We will consider a strict fragment of **NP** which we formalise next.
- $\mathbf{MAXSNP_0}$ is the class of optimisation problems A defined by

$$\max_{S \subseteq V^r} |\{(x_1, \ldots, x_k) \in V^k \mid \phi(G_1, \ldots, G_m, S, x_1, \ldots, x_n)\}|$$

where ϕ is a quantifier-free formula in (first-order) predicate logic involving relations $G_1, \ldots, G_m$ over finite V (forming the input) and relation S.

Example

MAX CUT $\in \mathbf{MAXSNP_0}$ as it can be stated as

$$\max_{S \subseteq V} |\{(x,y) \in V \times V : (E(x,y) \vee E(y,x)) \wedge S(x) \wedge \neg S(y)\}|$$

where the input is V (the set of vertices) and E (the edge relation of a graph).

Optional: MAXSNP-Completeness

Definition

MAXSNP is the class of all optimisation problems that are L-reducible to a problem in $\mathbf{MAXSNP}_0$.
A problem A in **MAXSNP** is **MAXSNP**-complete iff all problems in **MAXSNP** L-reduce to A.

Proposition

If a **MAXSNP***-complete problem has a polynomial-time approximation scheme, then all problems in* **MAXSNP** *have a polynomial-time approximation scheme.*

Theorem

MAX3SAT is **MAXSNP***-complete.*

Optional: Further MAXSNP-complete problems

Theorem

The following problems are **MAXSNP***-complete:*

(a) *INDEPENDENT SET*

(b) *VERTEX COVER*

(c) *MAX NAESAT*

(d) *MAX CUT*

3. Nonapproximability

Motivation

- Do $\mathbf{MAXSNP}$-complete problems have polynomial-time approximation schemes?
  Answer: No, unless $\mathbf{P} = \mathbf{NP}$.
- This (non-trivial) result is based on an alternative characterisation of $\mathbf{NP}$ using *restricted verifiers*.

Verifiers

- A relation R is *polynomially balanced* if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.
- Machine M is a *verifier* for L if L can be written as

$$L = \{x \mid (x, y) \in R \text{ for some } y\}$$

where R is a polynomially balanced relation decided by M.

Theorem (*The weak verifier version of Cook's theorem.*)

A language $L \in \mathbf{NP}$ iff it has a deterministic log-space verifier.

Restricted verifiers

A.k.a. “probabilistically checkable proof (PCP) systems”.

- An *(r,q)-restricted verifier* is a randomised machine that on input x of length n:
  - (i) uses at most $r(n)$ random bits and
  - (ii) queries at most $q(n)$ symbols of y when verifying $(x,y) \in R$.
- Given a random bit string z of at most $r(n)$ bits, the verifier:
  1. computes $Q(x,z)$, a set of $k \leq q(n)$ indices,
  2. chooses k symbols $y_1, \ldots, y_k$ from y according to indices in $Q(x,z)$,
  3. and performs a polynomial-time computation using as input x, z, and $(y_1, \ldots, y_k)$, and answers “yes” or “no”.

The PCP characterisation of NP

Definition

An (r, q)-restricted verifier *decides* a relation R iff for each input x and alleged certificate y,

1. $(x, y) \in R$ implies for all random strings the verifier says "yes" and
2. $(x, y) \notin R$ implies at least half of random strings make the verifier say "no".

By a very non-trivial proof the following can be shown:[1]

Theorem (The PCP Theorem, 1992)

Every language in $\mathbf{NP}$ *has an* $(O(\log n), O(1))$*-restricted verifier. (And vice versa.)*

[1] This major theorem was the culmination of a long sequence of partial results. Contributors include S. Arora, U. Feige, S. Goldwasser, C. Lund, L. Lovász, R. Motwani, S. Safra, M. Sudan, M. Szegedy.

Nonapproximability Results

The PCP Theorem has immediate and somewhat unexpected consequences on approximability thresholds.

Theorem

If MAX3SAT has a polynomial-time approximation scheme, then $\mathbf{P} = \mathbf{NP}$.

Some further corollaries:

- If $\mathbf{P} \neq \mathbf{NP}$, then no $\mathbf{MAXSNP}$-complete problem has a polynomial-time approximation scheme.
- If $\mathbf{P} \neq \mathbf{NP}$, then the approximation threshold of INDEPENDENT SET and CLIQUE is one (J. Håstad 1999).

Learning Objectives

- The concept of a polynomial-time ε-approximation algorithm and approximation threshold
- Examples of polynomial-time ε-approximation algorithms
- The concept of an approximation scheme
- The concepts of L-reductions and **MAXSNP**-completeness
- The concept of randomised verifiers, the PCP theorem, and related nonapproximability results