DESIGN AND ANALYSIS OF ALGORITHMS (DAA 2018)

Juha kärkkäinen

Based on slides by Veli Mäkinen

NP-hardness & approximability

Week V

NP-hardness

Definitions, reductions, examples

Book Chapter 34

DECISION VS OPTIMIZATION PROBLEM

Decision problem is a problem with yes/no answer.

- Hamiltonian Cycle Problem: Given a graph, is there a cycle that visits every vertex exactly once.

Optimization problem seeks a solution with a minimal or maximal value.

- Traveling Salesperson Problem: Given a weighted graph, find a Hamiltonian cycle with the smallest total weight.

Optimization problems have decisions versions:

- Traveling Salesperson Problem: Given a weighted graph and a value W, is there a Hamiltonian cycle with a total weight ≤W.

Obviously, if we can solve the optimization problem, we can solve the decision version, but the opposite is usually true too (blackboard).

Complexity classes are usually defined for decision problems.

- Hard decision version implies hard optimization version.

COMPLEXITY CLASSES P AND NP

P = problems that can be *solved* in $O(n^k)$ time

k constant

n input length, when *encoded*

NP = problems that can be *verified* in $O(n^k)$ time

k constant

n input length + *proof* length, when *encoded*

Next week

NP stands for *nondeterministic polynomial time*: The problems can be "solved" using the following nondeterministic algorithm:

1. Nondeterministically ”guess” an optimal solution/proof/certificate
   - For example, guess a list of edges for Hamiltonian cycle
2. Verify the solution/proof/certificate in polynomial time.
3. Return ”yes” if verified and ”no” otherwise
   - Every ”yes” instances must have a certificate that can verified (co-NP = problems with polytime verification of ”no” instances)

NP-HARD AND NP-COMPLETE

NP-hard = problems s. t. a polynomial time algorithm for it implies polynomial time algorithm for every NP problem

- Proof by reduction from any *NP-complete* problem
- Optimization problem is NP-hard if its decision version is

NP-complete = NP-hard problems that are in NP

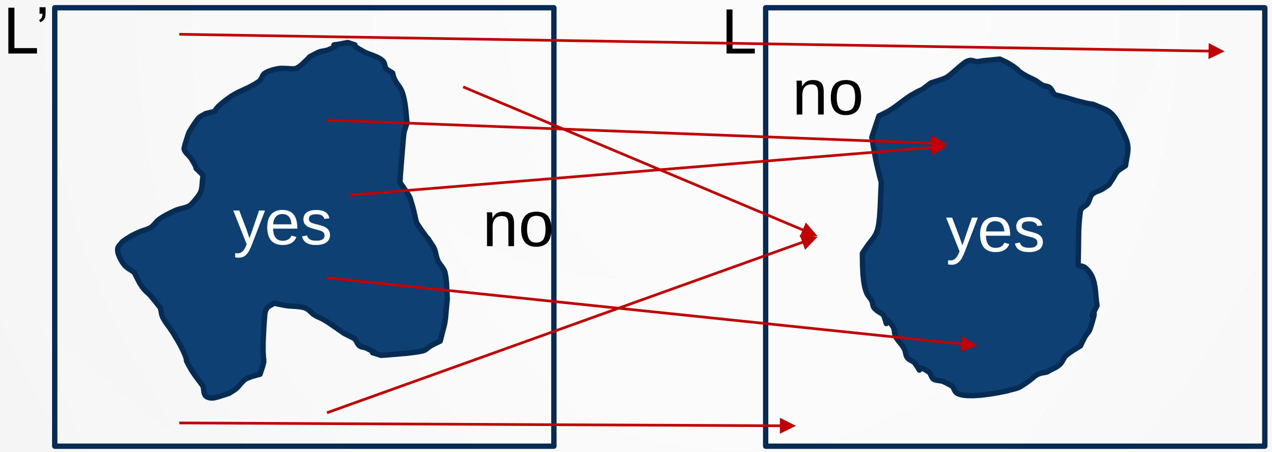
- Proof by reduction from any other NP-complete problem plus polynomial time verification

The unproven but generally accepted conjecture P≠NP implies

- NP contains problems that have no polynomial time algorithm
- No NP-hard problem has a polynomial time algorithm

REDUCTIONS

We denote L'$\leq_p$L if any input x' to decision problem L' can be converted in polynomial time (i.e. O(n^k) time) to an input x of L such that L'(x')=L(x)∈ $\{yes, no\}$.



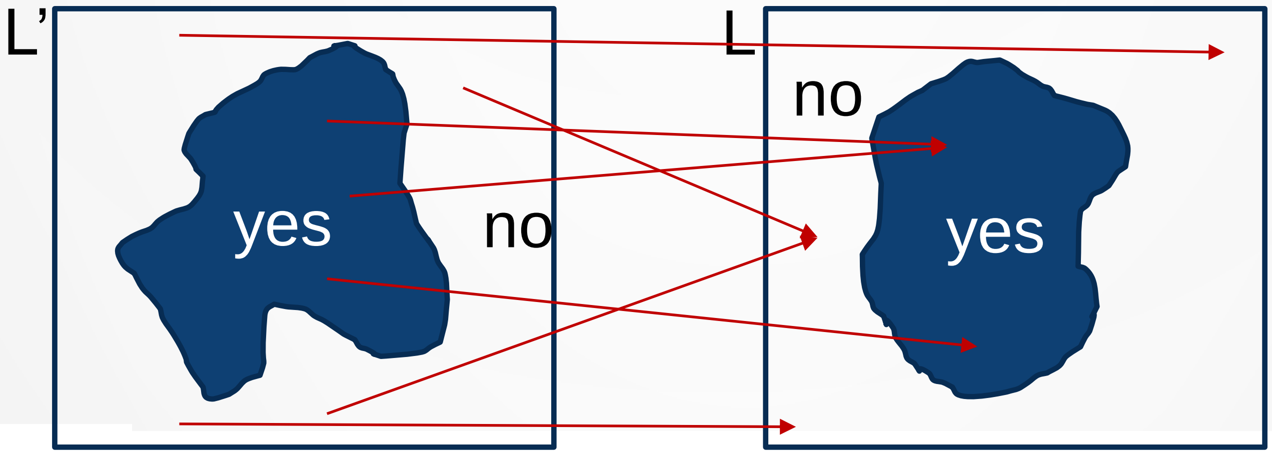
DEFINITIONS

Definition: Problem L is *NP-complete* if

1. L$\in NP$
2. L'$\leq_p$L for every L' $\in NP$.

Definition: Problem L is *NP-hard* if 2. holds for its *decision version* L^{dec}:

L^{dec}(x)="Is L(x)<t for given t?" [L minimization problem]



TOOL TO PROVE NP-HARDNESS

Lemma: Problem L is NP-hard if there is NP-complete problem L'' such that $L'' \leq_p L^{dec}$, where L^{dec} is the decision version of L.

Proof.

L'' is NP-complete ➔ $L' \leq_p L''$ for every $L' \in NP$ (by def.)

$L'' \leq_p L^{dec}$ ➔ $L' \leq_p L'' \leq_p L^{dec}$ for every $L' \in NP$

transitivity ➔ $L' \leq_p L^{dec}$ for every $L' \in NP$.

□

Corollary. We just need to show one problem NP-complete directly from definition. Then we can reduce all other problems from that.

BOOLEAN SATISFIABILITY: SAT

Decide if a boolean formula ϕ is true or not, where ϕ is composed of

n boolean variables $x_1, x_2, x_3, \ldots, x_n$.

m boolean connectives:

$\wedge$ (and), $\vee$ (or), $\neg$ $(negation)$, $\Rightarrow$ $(implication)$,

$\Leftrightarrow$ (iff)

Parentheses

THEOREM (Cook-Levin): SAT is NP-complete.

We shall prove this next week.

Now we use this fact to show other problems NP-complete.

EXAMPLE REDUCTION SAT $\leq_p$ 3CNF

3CNF is like SAT but ϕ is in a *3-Conjunctive Normal Form*:

Each *clause* (formula in parenthesis) contains 3 variables or their negations (*literals*) connected by two or's $\vee$.

– No variable can appear twice in the same clause.

Clauses are connected by and's $\wedge$.

$E.g.\, \phi^{3CNF} = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg\, x_1 \vee x_2 \vee x_3)$

To show 3CNF NP-complete, we first need to show it is in NP and then show that we can convert any ϕ into *3CNF* ϕ^{3CNF} in polynomial time such that ϕ=true iff ϕ^{3CNF}=true.

The proof works in several phases, converting the formula closer and closer to 3CNF form.

EXAMPLE REDUCTION SAT $\leq_p 3\text{CNF}$

3CNF is in NP:

We need a solution/proof/certificate for any ”yes” instance and a polynomial time algorithm for verifying the certificate.

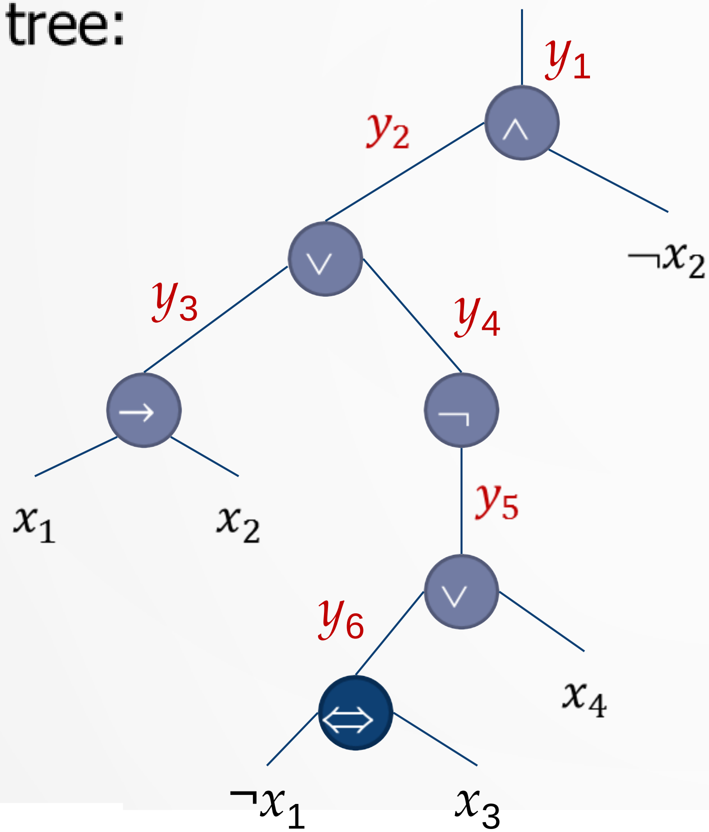
- For 3CNF, the certificate is an assignment of truth values to variables s.t. the formula is satisfied.
- In simple cases like this, writing down the actual algorithm is not required… but for the sake of practice:
  - Read the assignments to variables.
  - Read the 3CNF and evaluate a clause at a time. Return false if any clause evaluates false. Otherwise return true.
  - (exact details left as exercise).

EXAMPLE REDUCTION SAT $\leq_p$ 3CNF

We will work out the conversion through an example:

$$\Phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \Leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

Parse tree:



$$\begin{aligned}\Phi' = y_1 &\wedge \underbrace{(y_1 \Leftrightarrow (y_2 \wedge \neg x_2))}_{\Phi'_1} \\ &\wedge (y_2 \Leftrightarrow (y_3 \vee y_4)) \\ &\wedge (y_3 \Leftrightarrow (x_1 \rightarrow x_2)) \\ &\wedge (y_4 \Leftrightarrow \neg y_5) \\ &\wedge (y_5 \Leftrightarrow (y_6 \vee x_4)) \\ &\wedge (y_6 \Leftrightarrow (\neg x_1 \Leftrightarrow x_3))\end{aligned}$$

EXAMPLE REDUCTION SAT $\leq_p$ 3CNF

$\Phi_1' = y_1 \Leftrightarrow (y_2 \wedge \neg x_2)$

truth table

y_1	y_2	x_2	$y_1 \Leftrightarrow (y_2 \wedge \neg x_2)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

back to boolean formula

3DNF (D=disjunctive)

$\neg\Phi_1' = (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge y_2 \wedge x_2)$

De Morgan ➔ $\Phi_1'' = (y_1 \vee \neg y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2)$

3CNF

EXAMPLE REDUCTION SAT $\leq_p$ 3CNF

We are done, except for a special case:

If we have less than three literals after converting Φ_i' to Φ_i'', we need to add *dummy* variables to have the clauses in 3CNF.

E.g. $y_1 = (y_1 \vee a \vee b) \wedge (y_1 \vee a \vee \neg b) \wedge (y_1 \vee \neg a \vee b) \wedge (y_1 \vee \neg a \vee \neg b)$

- Any assignment of a and b makes y_1 decisive on one clause, others evaluate to true.

Let Φ''' be the 3CNF after these conversions.

Finally, all conversion steps can be done in polynomial time:

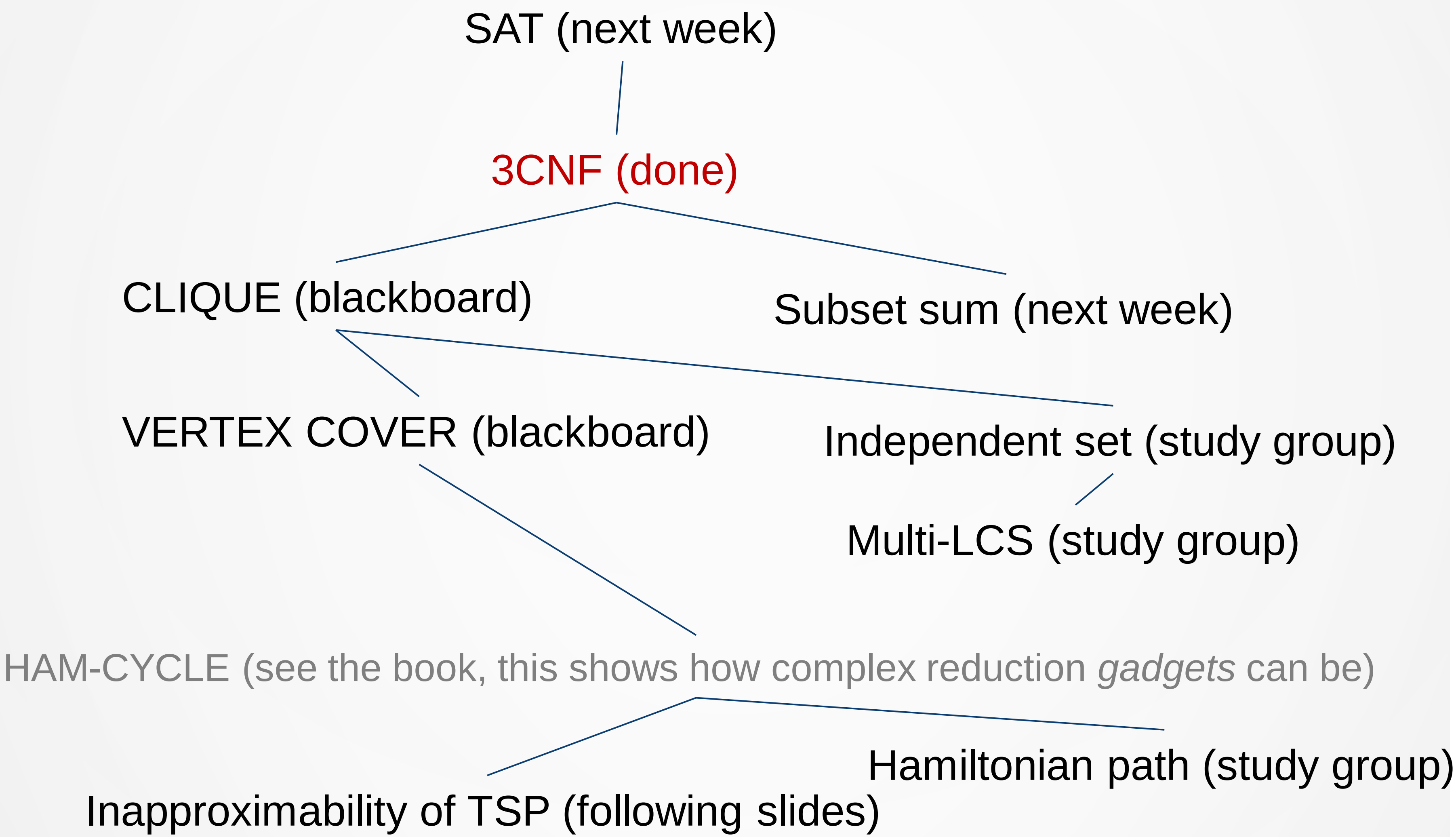
Each *connective* in Φ → at most 1 variable and 1 clause in Φ'.

Each clause in Φ' → at most 8 clauses in Φ''.

Each clause in Φ'' → at most 4 clauses in Φ'''. □

$$\begin{aligned}\Phi' = y_1 &\wedge (y_1 \Leftrightarrow (y_2 \wedge \neg x_2)) \\ &\wedge (y_2 \Leftrightarrow (y_3 \vee y_4)) \\ &\wedge (y_3 \Leftrightarrow (x_1 \rightarrow x_2)) \\ &\wedge (y_4 \Leftrightarrow \neg y_5) \\ &\wedge (y_5 \Leftrightarrow (y_6 \vee x_4)) \\ &\wedge (y_6 \Leftrightarrow (\neg x_1 \Leftrightarrow x_3))\end{aligned}$$

REDUCTION TREE COVERED IN THIS COURSE



SOME NP-HARD PROBLEMS

Max-Clique: Given a graph G, find the maximum clique (fully connected subgraph) in G.

CLIQUE: Does a graph G contain a clique of size k.

Min-Vertex-Cover: Given a graph G, find the smallest set V’ of vertices s.t. every edge in G is incident to a vertex in V’.

VERTEX-COVER: Does a graph G have a vertex cover of size ≤k.

Approximability

Definitions, examples

Book Chapter 35

APPROXIMATION ALGORITHMS

We will see in study groups and exercises that many important optimization problems are NP-hard.

However, it turns out that one can sometimes find good enough results in polynomial time.

Consider a minimization problem whose optimal solution has cost OPT. A *c-approximation algorithm* returns an answer that is at most c*OPT, where c>1.

*Maximization problem ➔ (1/c)*OPT.*

There are O(1) approximations, O(log n) approximations, etc.

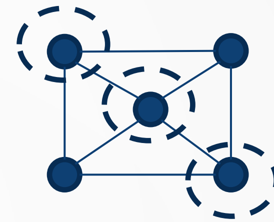
$(1+\epsilon)OPT$ = approximation scheme, $\epsilon > 0$.

PTAS = pol. in n for fixed ϵ, e.g. $O(n^{1/\epsilon})$.

FPTAS = pol. in n and in $1/\epsilon$, e,.g. $O((1/\epsilon)^{100} n^c)$, for constant c.

VERTEX COVER 2-APPROXIMATION

Minimum vertex cover: Find minimum size subset of vertices in an undirected graph G=(V,E) such that each edge is *incident* to at least one vertex in this subset.

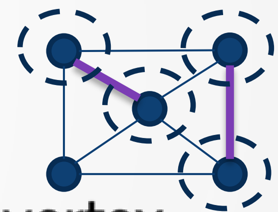


OPT=3

Approximation algorithm:

Repeat until |E|=0:

1. Add u and v to solution C for any (u,v)∈ E.
2. Remove all edges incident to u or to v.

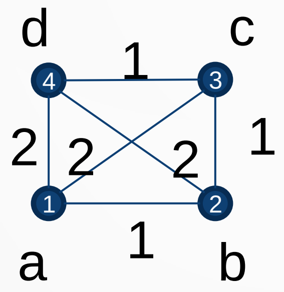


Theorem: The above algorithm is a 2-approximation to vertex cover.

Proof. Clearly C is a vertex cover. Let A be the set of edges selected at line 1. One of the endpoints of an edge in A needs to belong to an arbitrary vertex cover. No vertex is added twice in C: OPT $\geq |A|$ & $|C| = 2|A| \leq 2 * OPT$. □

METRIC TSP 2-APPROXIMATION

Traveling salesperson problem (TSP) asks to find a minimum cost *Hamiltonian cycle* (one visiting each vertex exactly ones) in a *complete* undirected graph G=(V,E). Cost of a path is the sum of costs of edges. Metric TSP is a variant where $c(u,w) \leq c(u,v) + c(v,w)$ for all v, for all edges (u,w).



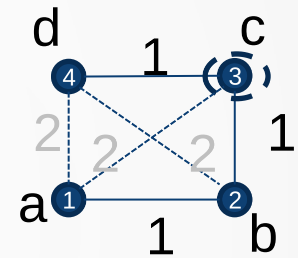
METRIC TSP 2-APPROXIMATION

Spanning tree is a tree on V with edges $E' \subseteq E$. Minimum spanning tree has smallest cost of edges.

Approximation algorithm:

H= list of vertices in the preorder of T, where T is a *minimum spanning tree* of G=(V,E).

Return the cycle *induced* by H



H	cost
c	
d	1
b	2
a	1
c +	2
	6

Theorem. The above algorithm is a 2-approximation for metric TSP.

Proof.

TSP is a spanning tree, after one edge is removed

$C(T)=\sum_{(u,v)\in T} c(u,v) \leq OPT_{TSP}$

Consider a *full walk* over T, where each edge visited twice (once going down, once going up). Cost of full walk is 2C(T).

The cycle induced by H has at most the cost of the full walk:

– Full walk (c,d),(d,c),(c,b),(b,a),(a,b),(b,c)

– Cycle induced by H (c,d), (d,b), (b,a), (a,c)

□

preorder makes shortcuts to full walk and by triangle inequality the cost is less

INAPPROXIMABILITY

Some problems are hard to approximate well.
An example is general TSP (without triangle inequality).

Theorem. If $P \neq NP$, then for any constant $c \geq 1$, there is no polynomial time c-approximation algorithm for general TSP.

Proof. Reduction from Hamiltonian Cycle:

- Let $G=(V,E)$ be the Hamiltonian Cycle instance.
- Let $G'=(V,E')$ be the complete graph on V.
- Let w be the edge cost function:
  $w(e)=1$ if $e \in E$ and $w(e)=c|V|+1$ otherwise.
- Then a Hamiltonian cycle in G has cost $|V|$ in G' and any other cycle has cost at least $c|V|+1+|V|-1 = (c+1)|V| > c|V|$.
- Thus any c-approximation algorithm would have to find a Hamiltonian cycle if it exists.