Recall

Many practical problems are NP-complete — no one knows a polynomial time algorithm, nor can we prove that none exists.

This lecture: What to do with NP-hard optimization problems.

1. Efficient exhaustive search (backtracking, branch-and-bound).
   Exponential time in the worst case, but can be useful.

2. Heuristics W https://en.wikipedia.org/wiki/Heuristic_(computer_science)

   - there might be no guarantee on run-time nor on quality of solution.

   - local search — start with some solution and try to improve it via small "local" changes. hill climbing, simulated annealing

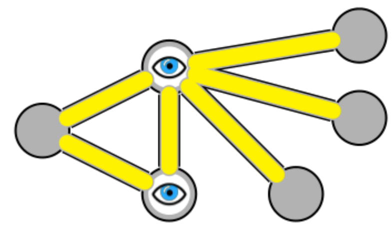
   - particle swarm, evolutionary algorithms

3. Approximation algorithms — today's topic

   polynomial time and a guarantee on the quality of the solution
   e.g. for a minimization problem, might guarantee a solution $\leq 2 \cdot \min$

Approximation algorithms for Vertex Cover

Recall A *vertex cover* is a set $S \subseteq V$ such that every edge $(u,v) \in E$ has u or v (or both) in S.



Fschwarzentruber

W https://en.wikipedia.org/wiki/Vertex_cover

Optimization problem: find a minimum size vertex cover.

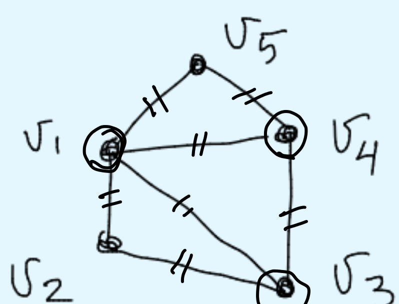
Recall that the decision version is NP-complete.

Greedy Algorithm 1

C := ∅
repeat
  C := C ∪ {vertex of maximum degree}
  remove covered edges
until no edges remain

Note that this is a polynomial time algorithm

Example



$C = \{v_1, v_4, v_3\}$

$|C| = 3$ seems optimum (min. size)

Note: Alg. runs in polynomial time

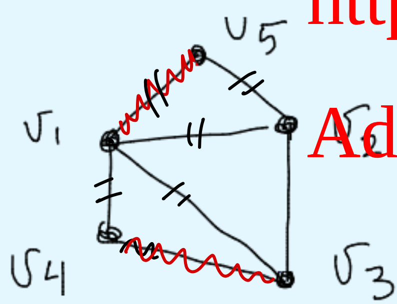
Approximation algorithms for Vertex Cover

Greedy Algorithm 2

```
C := ∅    F := E      // F is uncovered edges
while F ≠ ∅
   pick e = (u,v) from F
   add u and v to C
   remove edges incident to u from F
   remove edges incident to v from F
```

Note that this is a polynomial time algorithm

Example



$C = \{v_1, v_5, v_3, v_4\}$

$|C| = 4$ not optimum.

Which is better, Algorithm 1 or Algorithm 2?

on this example, Alg. 1 is better.

Ex. Find an example where Alg. 2 is better.

Approximation algorithms for Vertex Cover

Greedy Algorithm 2

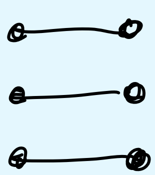
```
C := ∅    F := E     // F is uncovered edges
while F ≠ ∅
   pick e = (u,v) from F
   add u and v to C
   remove edges incident to u from F
   remove edges incident to v from F
```


Analysis of approximation factor

Let C = vertex cover found by Algorithm 2.
Let C_{OPT} = a minimum vertex cover.

Claim. $|C| \le 2 \cdot |C_{OPT}|$

Proof.
Note that the edges chosen form a <u>matching</u> M (no two edges are incident)



$|C| = 2|M|$

Any vertex cover must have at least one vertex from each edge in M

$\therefore |M| \le |C_{OPT}|$ so $|C| \le 2 \cdot |C_{OPT}|$ $\boxtimes$

Approximation algorithms for Vertex Cover

We say that Algorithm 2 has ***approximation factor*** 2
because it produces a vertex cover of size ≤ 2 · optimum

FACT: Algorithm 1 has approximation factor Θ(log n).
It is worse than Algorithm 2.

Recall that Vertex Cover and Independent Set are closely related.
However:

FACT: Independent Set has no good approximation algorithm unless P = NP.

CS 466 covers this

Summary of Lecture 22, Part 1

Approximation algorithms for Vertex Cover

What you should know

- what is an approximation algorithm
- what does approximation factor mean
- some NP-complete problems have good approximation algorithms and some do not (unless P = NP)

Next:

Approximation algorithm for Travelling Salesman Problem in the Plane

Travelling Salesman Problem

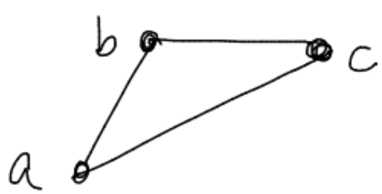
W https://en.wikipedia.org/wiki/Travelling_salesman_problem

Given a graph G, weights on edges, number k, does G have a TSP tour of length ≤ k

Euclidean TSP. For the complete graph on points in the plane, with weight = Euclidean distance.

FACT: even Euclidean TSP is NP-complete.

key property of Euclidean case: triangle inequality



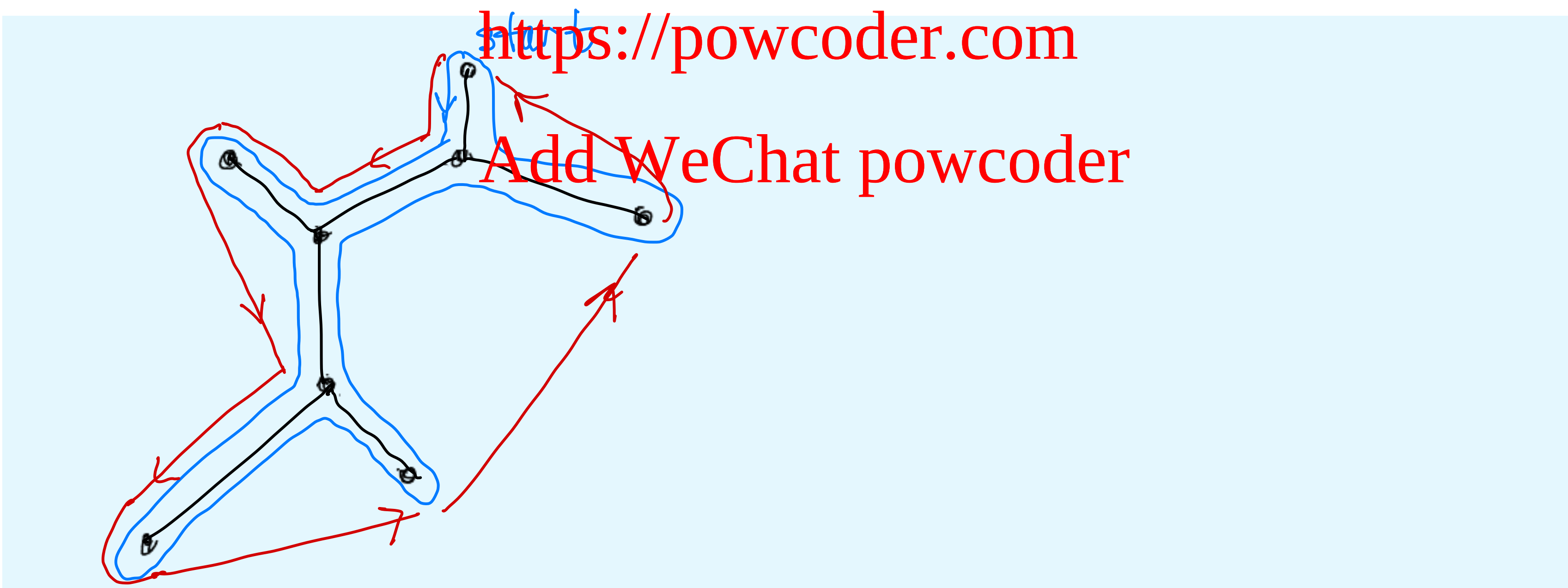
$w(a,c) \le w(a,b) + w(b,c)$

Approximation algorithm for Euclidean TSP

compute MST (min. spanning tree) black

take a tour by walking around it blue
(we visit every vertex but maybe more than once)

take shortcuts to avoid revisiting vertices red
note: by the triangle inequality, the short-cuts are shorter



This algorithm takes poly time.

Approximation algorithm for Euclidean TSP

Let t = length of tour found by this algorithm.
Let t_{TSP} = length of minimum TSP tour

Claim. $t \leq 2\, t_{TSP}$

This means that in polynomial time we can find a tour within 2 times the optimum.

Proof of Claim.
Let t_{MST} = length of MST

$t_{MST} \leq t_{TSP}$ because deleting one edge of optimum TSP tour (decreasing length) gives some spanning tree, so MST is even less in length.

$t \leq 2\, t_{MST}$ because blue tour has length $2\, t_{MST}$ and then we take shortcuts (using triangle inequality)

Thus $t \leq 2\, t_{TSP}$ $\boxtimes$

We say that the algorithm has ***approximation factor*** 2 because it finds a tour of length at most 2 times the optimum, i.e. $t \leq 2\, t_{TSP}$

FACT: the factor of 2 can be improved for this problem. For any $\varepsilon > 0$ there is an algorithm that finds a tour of length $\leq (1+\varepsilon)\, t_{TSP}$
But as $\varepsilon \to 0$, the run time becomes exponential.

CS 466 covers this

Summary of Lecture 22

good approximation algorithms for Vertex Cover and Euclidean TSP.

What you should know

- what is an approximation algorithm

- what does approximation factor mean

- some NP-complete problems have good approximation algorithms and some do not (unless P = NP)