CSE202
Design and Analysis of Algorithms

Week 14 — Approximation Algorithms for NP-hard problems

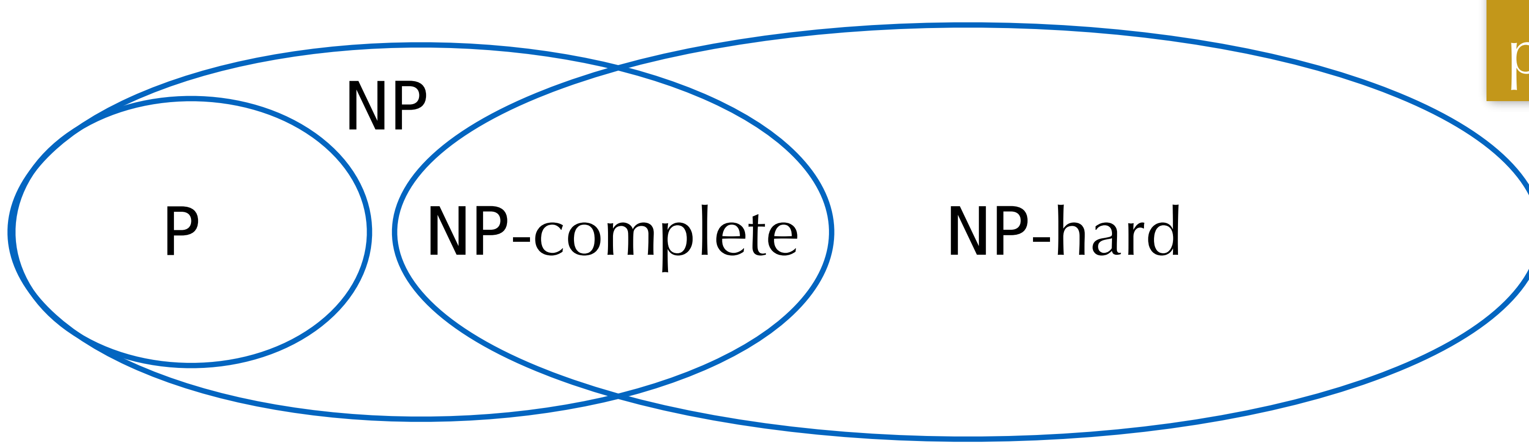
P, NP, NP-complete, NP-hard

Recall:

P: decision problems computable in polynomial time;
NP: decision problems with a verifier in **P**;
B in **NP** is **NP**-complete if for all A in NP, $A \leq B$.

Also useful: **Def**: B is NP-hard if for all A in NP, $A \leq B$.

If P≠NP,



Exercise:
picture when P=NP

Decision vs. Optimization

[14, 18, 15, 8, 10, 14, 12, 9, 13, 16]

14+15+10+14=53

Recall Problem SubsetSum:

Input: $L = (x_1, \ldots, x_\ell) \in \mathbb{N}^\ell$, k in $\mathbb{N}$

Output: yes iff there exists $A \subset \{1, \ldots, \ell\}$ s.t. $\sum_{i \in A} x_i = k$.

Optimization variants:

1. Is there (A, m), s.t.
2. Maximize m s.t.
3. Find corresponding A.

$$\sum_{i \in A} x_i = m \leq k?$$

ρ-approximation algorithm: finds M s.t. $M \geq \rho\, m_{\max}$ $(\rho < 1)$

($M \leq \rho\, m_{\min}$ $(\rho > 1)$ for a minimization problem).

Approximation Schemes

$$M \geq (1 - \varepsilon) m_{\max}$$

Def. Polynomial Time Approximation Scheme (PTAS):

runs in polynomial time for a given ε.

Def. Fully Polynomial Time Approximation Scheme (FPTAS):

runs in time polynomial also in $1/\varepsilon$.

NP-complete problems are not equal for approximation

1. Subset-Sum;
2. k-SAT;
3. Traveling Salesman.

I. Approximation for Subset-Sum

Exact Algorithm

```
def subset_sum(X,t):
    X = sorted(X)
    n = len(X)
    S = [0]
    for i in range(n):
        S = merge(S,X[i],t)
    return S[-1]
```

$(S \cup (S + X_i)) \cap [0,t]$
in time $O(|S|)$
(works as in MergeSort)

Correctness: all subsets with sum $\leq t$ are considered.

Complexity:

$$O\left(\sum_{i=1}^{n} |S_i|\right) = O(\min(2^n, nt))$$

These sets have size typically exponential in the bit-size of the input

Exercise:
with 7 more lines of code,
return a corresponding subset

Approximation

```python
def approx_subset_sum(X,t,epsilon):
    X = sorted(X)
    n = len(X)
    S = [0]
    for i in range(n):
        S = merge(S,X[i],t)
        S = filter(S,epsilon/n)
    return S[-1]
```

```python
def filter(S,delta):
    res = [S[0]]
    for i in range(1,len(S)):
        if S[i]>(1+delta)*res[-1]:
            res.append(S[i])
    return res
```

in time $O(|S|)$.

```
>>> L=[14,18,15,8,10,14,12,9,13,16]
>>> subset_sum_sol(L,53)
[14, 12, 10, 9, 8] --> 53
>>> approx_subset_sum_sol(L,53,.1)
[14, 12, 10, 9, 8] --> 53
>>> approx_subset_sum_sol(L,53,.5)
[15, 14, 14, 10] --> 53
>>> approx_subset_sum_sol(L,53,.9)
[18, 15, 9, 8] --> 50
```

Lemma. For any 0<a≤b,

$$|\mathrm{filter}(S,\delta) \cap [a,b]| \leq 1 + \ln_{1+\delta}(b/a).$$

Thm. This algorithm returns a (1-ε)-approximation of the optimum, in time $O(\varepsilon^{-1} n^3 \log n)$.

FPTAS

Proof

Notation:

$S_i^{(e)}$: S_i exact algo.

$S_i^{(a)}$: S_i approx algo.

Proof of approximation in 3 steps:

1. $\forall u \in L, \exists v \in \text{filter}(L, \delta),\ v \leq u < v(1+\delta)$.
2. $\forall s \in S_i^{(e)}, \exists r \in S_i^{(a)},\ r \leq s \leq r(1+\delta)^i$.
3. $s_{-1}^{(e)} \leq s_{-1}^{(a)}(1+\delta)^n$ and $(1+\varepsilon/n)^{-n} \geq 1-\varepsilon$.

Detailed proof on the blackboard.

Complexity:

$$|S_i^{(a)}| \leq |S_i^{(a)} \cap [0, X_i)| + |S_i^{(a)} \cap [X_i, \infty)|$$

$$\leq |S_{i-1}^{(a)}| + 1 + \log_{1+\delta} \frac{X_i + \max S_{i-1}^{(a)}}{X_i} \leq |S_{i-1}^{(a)}| + 1 + \log_{1+\delta}(i)$$

$$\sum_{i=1}^{n} |S_i^{(a)}| \leq \sum_{i=1}^{n} i(1 + \log_{1+\delta}(i)) \leq n^2 \left(1 + \frac{\log n}{\log(1+\varepsilon/n)}\right) = O(\varepsilon^{-1} n^3 \log n)$$

for $\varepsilon = o(n)$.

II. Max-SAT and its Approximation

Max-k-SAT is NP-complete for $k \geq 2$

Problem Max-k-SAT:

Input: k-SAT formula ϕ, $\ell \in \mathbb{N}$

Output: yes iff there exists a truth assignment satisfying $\geq \ell$ clauses in ϕ.

Clearly in NP: solution easy to check.

$$3\text{-SAT} \leq \text{Max-3-SAT} \quad (\text{take } \ell = |\phi|)$$

$$\boxed{3\text{-SAT} \leq \text{Max-2-SAT}}$$

Reduction: $\phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_m \vee b_m \vee c_m)$

$$\mapsto \bigwedge_{i=1}^{m} \Big((a_i \vee e_i) \wedge (a_i \vee \bar{e}_i) \wedge (b_i \vee f_i) \wedge (b_i \vee \bar{f}_i) \wedge (c_i \vee g_i) \wedge (c_i \vee \bar{g}_i)$$
$$\wedge (\bar{a}_i \vee \bar{b}_i) \wedge (\bar{a}_i \vee \bar{c}_i) \wedge (\bar{b}_i \vee \bar{c}_i)$$
$$\wedge (d_i \vee h_i) \wedge (d_i \vee \bar{h}_i) \wedge (a_i \vee \bar{d}_i) \wedge (b_i \vee \bar{d}_i) \wedge (c_i \vee \bar{d}_i) \Big),$$
$$\ell = 11m.$$

Exercise: Check the possibilities.

The Optimum is Not Harder

Problem Max-k-SAT (optimum variant):

Input: k-SAT formula ϕ

Output: max num. clauses in ϕ that can be satisfied at once.

Binary search reduces to $\log m$ solutions to the threshold (decision) problem.

Problem Max-k-SAT (value):

Input: k-SAT formula ϕ

Output: assignment maximizing num. satisfied clauses in ϕ.

Use recursively
$$\mathrm{opt}(\phi) = \max(\mathrm{opt}(\phi|_{x_1=0}), \mathrm{opt}(\phi|_{x_1=1})).$$

Random Assignments

$\phi = C_1 \wedge \cdots \wedge C_m$ a conjunction of m clauses in $x_1, \ldots, x_n$

n_i : num. variables in C_i ($n_i = 3$ for 3-SAT)

Z_i : 1 if C_i is satisfied, 0 otherwise

$Z := Z_1 + \cdots + Z_m$ num. satisfied clauses.

Lemma. For a uniform random assignment of the variables, $\mathbb{E}(Z_i) = 1 - 2^{-n_i}$.

For 3-SAT, linearity of expectation yields:

1. There exists an assignment satisfying at least $(7/8)m$ clauses;
2. random assignment expected within a factor 7/8 of the optimal.

(Deterministic) Approximation Algorithm

$$\mathbb{E}[Z] = \frac{1}{2}\mathbb{E}[Z \mid x_1 = 0] + \frac{1}{2}\mathbb{E}[Z \mid x_1 = 1]$$

$$\implies \max\left(\mathbb{E}[Z \mid x_1 = 0], \mathbb{E}[Z \mid x_1 = 1]\right) \geq \mathbb{E}[Z].$$

can be computed in polynomial time

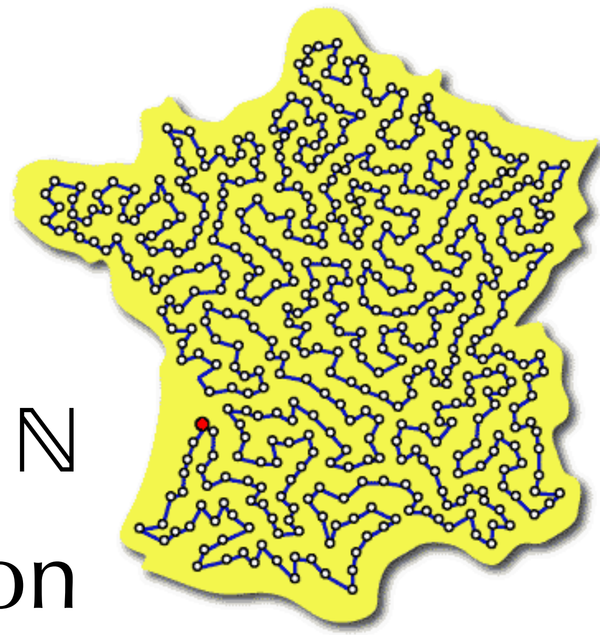
Algorithm: choose the values of the variables one by one, by greedily going for the higher expectation.

(Beyond our scope)
Max-k-sat NP-hard to approximate for any $\rho > 1 - 2^{-k}$.

Polynomial time, approx. factor $\rho = 1 - 2^{-k}$.

III. Approximate Traveling Salesman

Inapproximability



Recall Problem TSP:

Input: $n \times n$ matrix M of positive integers, k in $\mathbb{N}$

Output: yes iff there exists a cyclic permutation σ s.t. $\displaystyle\sum_{1 \le i \le n} M_{i,\sigma(i)} \le k$.

Problem Hamiltonian Cycle: TSP with $M_{i,j} \in \{1, \infty\}$, $k = n$.

3-SAT $\le$ Hamiltonian Cycle $\le$ TSP

Not proved here

$\infty \mapsto 1 + 2\varepsilon n$

For any $\varepsilon > 0$, approximating the TSP within a factor $1 + \varepsilon$ is NP-hard.

Proof:

$(n-1) + (1 + 2\varepsilon n) = (1 + 2\varepsilon)n$.

Metric TSP

Same problem as TSP, when

$$M_{i,j} = M_{j,i},\ M_{i,j} \leq M_{i,k} + M_{k,j} \quad \text{for all } i, j, k\,.$$

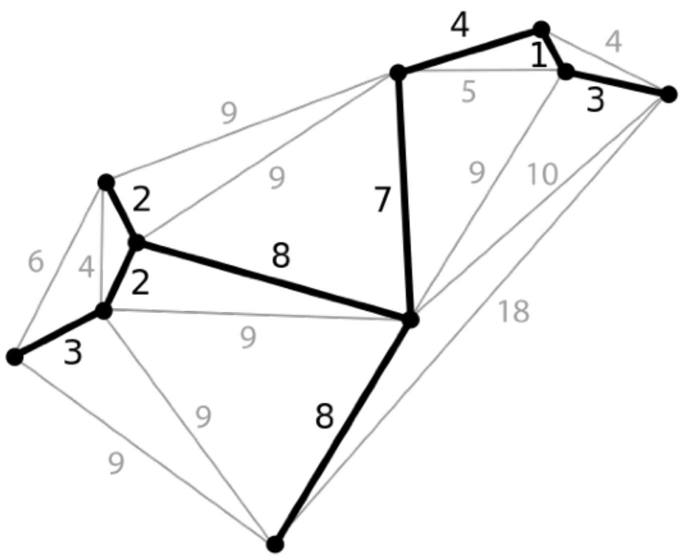
Metric TSP is NP-complete:

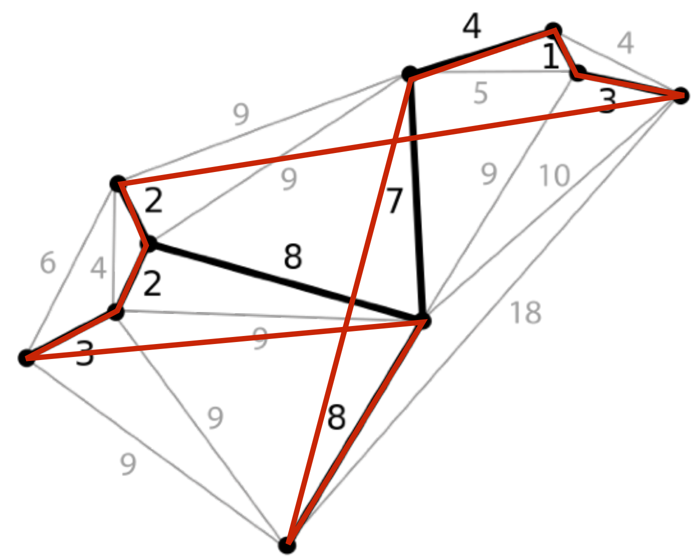
Clearly in NP: solution easy to check.

Hamiltonian Cycle ≤ Metric TSP:

$$M_{i,i} = 0,\ M_{i,j} = 1 \text{ if } (i, j) \in E,\ 2 \text{ otherwise.}$$

2-Approximation to the Metric TSP





1. compute a minimum spanning tree

$$\ell(T) \leq \ell_{\text{opt}}$$

2. visit nodes in depth-first order

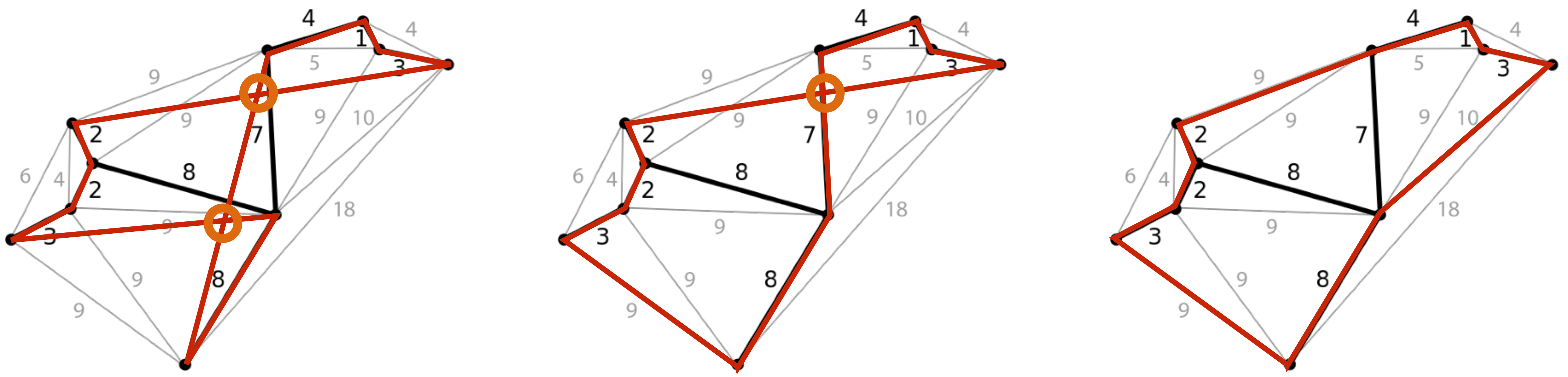
$$\ell \leq 2\ell(T)$$

More is known:
. PTAS when metric is Euclidean
. Current best approximation factor: 3/2
. No polynomial approx ≤123/122 if P≠NP

New (Jul. 2020)
$3/2 - 10^{-36}$

Heuristics

In a Euclidean setting and dim 2, get rid of crossings:

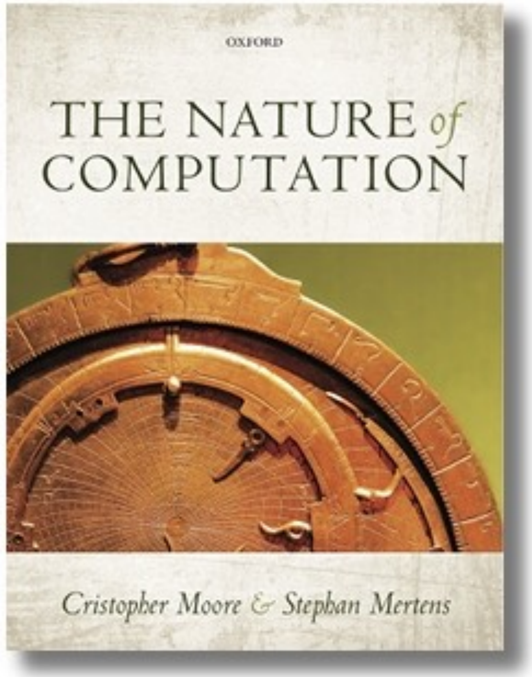


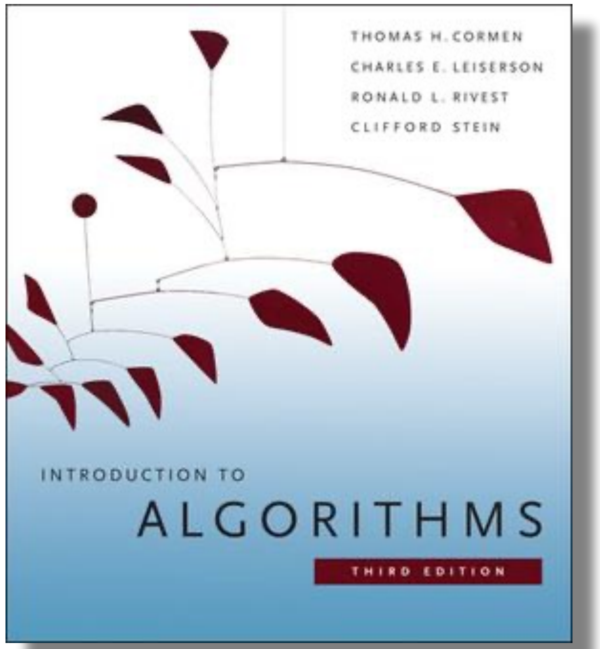
With more heuristics, problems with 10,000 cities are routinely solved within a few percents.

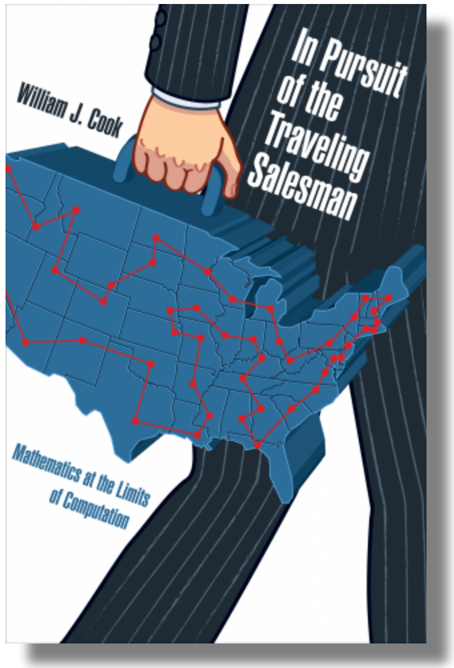
References for this lecture

The slides are designed to be self-contained.

They were prepared using the following
books that I recommend if you want to learn more:







Next

Assignment: a heuristic for the metric TSP

Next tutorial: exercises

Week Jan. 18—Jan. 22: final exam
(details not known yet)

Feedback

Moodle for the slides, tutorials and exercises.

Questions or comments: Bruno.Salvy@inria.fr

The End