

## Part III

# Approximation Algorithms

There are many practically important optimization problems that are NP-hard.

### What can we do?

- ▶ Heuristics.
- ▶ Exploit special structure of instances occurring in practise.
- ▶ Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

### Definition 2

An  $\alpha$ -approximation for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of  $\alpha$  of the value of an optimal solution.

### Minimization Problem:

Let  $\mathcal{I}$  denote the set of problem instances, and let for a given instance  $I \in \mathcal{I}$ ,  $\mathcal{F}(I)$  denote the set of feasible solutions. Further let  $\text{cost}(F)$  denote the **cost** of a feasible solution  $F \in \mathcal{F}$ .

Let for an algorithm  $A$  and instance  $I \in \mathcal{I}$ ,  $A(I) \in \mathcal{F}(I)$  denote the feasible solution computed by  $A$ . Then  $A$  is an approximation algorithm with approximation guarantee  $\alpha \geq 1$  if

$$\forall I \in \mathcal{I} : \text{cost}(A(I)) \leq \alpha \cdot \min_{F \in \mathcal{F}(I)} \{\text{cost}(F)\} = \alpha \cdot \text{OPT}(I)$$

### Maximization Problem:

Let  $\mathcal{I}$  denote the set of problem instances, and let for a given instance  $I \in \mathcal{I}$ ,  $\mathcal{F}(I)$  denote the set of feasible solutions. Further let  $\text{profit}(F)$  denote the **profit** of a feasible solution  $F \in \mathcal{F}$ .

Let for an algorithm  $A$  and instance  $I \in \mathcal{I}$ ,  $A(I) \in \mathcal{F}(I)$  denote the feasible solution computed by  $A$ . Then  $A$  is an approximation algorithm with approximation guarantee  $\alpha \leq 1$  if

$$\forall I \in \mathcal{I} : \text{cost}(A(I)) \geq \alpha \cdot \max_{F \in \mathcal{F}(I)} \{\text{profit}(F)\} = \alpha \cdot \text{OPT}(I)$$

### Why approximation algorithms?

- ▶ We need algorithms for hard problems.
- ▶ It gives a rigorous mathematical base for studying heuristics.
- ▶ It provides a metric to compare the difficulty of various optimization problems.
- ▶ Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

### Why not?

- ▶ Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

### What can we hope for?

#### Definition 3

A polynomial-time approximation scheme (PTAS) is a family of algorithms  $\{A_\epsilon\}$ , such that  $A_\epsilon$  is a  $(1 + \epsilon)$ -approximation algorithm (for minimization problems) or a  $(1 - \epsilon)$ -approximation algorithm (for maximization problems).

Many NP-complete problems have polynomial time approximation schemes.

### There are difficult problems!

The class MAX SNP (which we do not define) contains optimization problems like maximum cut or MAX-3SAT.

#### Theorem 4

*For any MAX SNP-hard problem, there does not exist a polynomial-time approximation scheme, unless  $P = NP$ .*

**MAXCUT.** Given a graph  $G = (V, E)$ ; partition  $V$  into two disjoint pieces  $A$  and  $B$  s. t. the number of edges between both pieces is maximized.

**MAX-3SAT.** Given a 3CNF-formula. Find an assignment to the variables that satisfies the maximum number of clauses.

## There are really difficult problems!

### Theorem 5

For any constant  $\epsilon > 0$  there does not exist an  $\Omega(n^{\epsilon-1})$ -approximation algorithm for the maximum clique problem on a given graph  $G$  with  $n$  nodes unless  $P = NP$ .

Note that an  $1/n$ -approximation is trivial.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore **Linear Programs** or **Integer Linear Programs** play a vital role in the design of many approximation algorithms.

### Definition 6

An **Integer Linear Program** or **Integer Program** is a Linear Program in which all variables are required to be integral.

### Definition 7

A **Mixed Integer Program** is a Linear Program in which a subset of the variables are required to be integral.

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

**Note that solving Integer Programs in general is NP-complete!**

## Set Cover

Given a ground set  $U$ , a collection of subsets  $S_1, \dots, S_k \subseteq U$ , where the  $i$ -th subset  $S_i$  has weight/cost  $w_i$ . Find a collection  $I \subseteq \{1, \dots, k\}$  such that

$$\forall u \in U \exists i \in I: u \in S_i \text{ (every element is covered)}$$

and

$$\sum_{i \in I} w_i \text{ is minimized.}$$

## IP-Formulation of Set Cover

$$\begin{array}{ll} \min & \sum_i w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i:u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \geq 0 \\ & \forall i \in \{1, \dots, k\} \quad x_i \text{ integral} \end{array}$$

## IP-Formulation of Set Cover

$$\begin{array}{ll} \min & \sum_i w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i:u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \in \{0, 1\} \end{array}$$

## Vertex Cover

Given a graph  $G = (V, E)$  and a weight  $w_v$  for every node. Find a vertex subset  $S \subseteq V$  of minimum weight such that every edge is incident to at least one vertex in  $S$ .

## IP-Formulation of Vertex Cover

$$\begin{array}{ll} \min & \sum_{v \in V} w_v x_v \\ \text{s.t.} & \forall e = (i, j) \in E \quad x_i + x_j \geq 1 \\ & \forall v \in V \quad x_v \in \{0, 1\} \end{array}$$

## Maximum Weighted Matching

Given a graph  $G = (V, E)$ , and a weight  $w_e$  for every edge  $e \in E$ . Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

$$\begin{array}{ll} \max & \sum_{e \in E} w_e x_e \\ \text{s.t.} & \forall v \in V \quad \sum_{e: v \in e} x_e \leq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

## Maximum Independent Set

Given a graph  $G = (V, E)$ , and a weight  $w_v$  for every node  $v \in V$ . Find a subset  $S \subseteq V$  of nodes of maximum weight such that no two vertices in  $S$  are adjacent.

$$\begin{array}{ll} \max & \sum_{v \in V} w_v x_v \\ \text{s.t.} & \forall e = (i, j) \in E \quad x_i + x_j \leq 1 \\ & \forall v \in V \quad x_v \in \{0, 1\} \end{array}$$

## Knapsack

Given a set of items  $\{1, \dots, n\}$ , where the  $i$ -th item has weight  $w_i$  and profit  $p_i$ , and given a threshold  $K$ . Find a subset  $I \subseteq \{1, \dots, n\}$  of items of total weight at most  $K$  such that the profit is maximized.

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq K \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

## Facility Location

Given a set  $L$  of (possible) locations for placing facilities and a set  $C$  of customers together with cost functions  $s : C \times L \rightarrow \mathbb{R}^+$  and  $o : L \rightarrow \mathbb{R}^+$  find a set of facility locations  $F$  together with an assignment  $\phi : C \rightarrow F$  of customers to open facilities such that

$$\sum_{f \in F} o(f) + \sum_c s(c, \phi(c))$$

is minimized.

In the **metric facility location** problem we have

$$s(c, f) \leq s(c, f') + s(c', f) + s(c', f') .$$

## Facility Location

$$\begin{array}{ll} \min & \sum_f x_f o(f) + \sum_c \sum_f y_{cf} s(c, f) \\ \text{s.t.} & \forall c \in C, f \in L \quad y_{cf} \leq x_f \\ & \forall c \in C \quad \sum_f y_{cf} \geq 1 \\ & \forall f \in L \quad x_f \in \{0, 1\} \\ & \forall c \in C, f \in L \quad y_{cf} \in \{0, 1\} \end{array}$$

- ▶  $y_{cf} \leq x_f$  ensures that we cannot assign customers to facilities that are not open.
- ▶  $\sum_f y_{cf} \geq 1$  ensures that every customer is assigned to a facility.

## Relaxations

### Definition 8

A linear program LP is a **relaxation** of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing  $x_i \in [0, 1]$  instead of  $x_i \in \{0, 1\}$ .

By solving a relaxation we obtain an upper bound for a maximization problem and a lower bound for a minimization problem.

## Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

**Set Cover relaxation:**

$$\begin{array}{ll} \min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i:u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \in [0, 1] \end{array}$$

Let  $f_u$  be the number of sets that the element  $u$  is contained in (the frequency of  $u$ ). Let  $f = \max_u \{f_u\}$  be the maximum frequency.

## Technique 1: Round the LP solution.

**Rounding Algorithm:**

Set all  $x_i$ -values with  $x_i \geq \frac{1}{f}$  to 1. Set all other  $x_i$ -values to 0.

## Technique 1: Round the LP solution.

### Lemma 9

*The rounding algorithm gives an  $f$ -approximation.*

**Proof:** Every  $u \in U$  is covered.

- ▶ We know that  $\sum_{i:u \in S_i} x_i \geq 1$ .
- ▶ The sum contains at most  $f_u \leq f$  elements.
- ▶ Therefore one of the sets that contain  $u$  must have  $x_i \geq 1/f$ .
- ▶ This set will be selected. Hence,  $u$  is covered.

## Technique 1: Round the LP solution.

The cost of the rounded solution is at most  $f \cdot \text{OPT}$ .

$$\begin{aligned} \sum_{i \in I} w_i &\leq \sum_{i=1}^k w_i (f \cdot x_i) \\ &= f \cdot \text{cost}(x) \\ &\leq f \cdot \text{OPT} . \end{aligned}$$

## Technique 2: Rounding the Dual Solution.

### Relaxation for Set Cover

#### Primal:

$$\begin{array}{ll} \min & \sum_{i \in I} w_i x_i \\ \text{s.t. } \forall u & \sum_{i: u \in S_i} x_i \geq 1 \\ & x_i \geq 0 \end{array}$$

#### Dual:

$$\begin{array}{ll} \max & \sum_{u \in U} y_u \\ \text{s.t. } \forall i & \sum_{u: u \in S_i} y_u \leq w_i \\ & y_u \geq 0 \end{array}$$

## Technique 2: Rounding the Dual Solution.

### Rounding Algorithm:

Let  $I$  denote the index set of sets for which the dual constraint is tight. This means for all  $i \in I$

$$\sum_{u: u \in S_i} y_u = w_i$$

## Technique 2: Rounding the Dual Solution.

### Lemma 10

*The resulting index set is an  $f$ -approximation.*

#### Proof:

Every  $u \in U$  is covered.

- ▶ Suppose there is a  $u$  that is not covered.
- ▶ This means  $\sum_{u: u \in S_i} y_u < w_i$  for all sets  $S_i$  that contain  $u$ .
- ▶ But then  $y_u$  could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

## Technique 2: Rounding the Dual Solution.

### Proof:

$$\begin{aligned} \sum_{i \in I} w_i &= \sum_{i \in I} \sum_{u: u \in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\leq \sum_u f y_u \\ &\leq f \sum_u y_u \\ &\leq f \text{cost}(x^*) \\ &\leq f \cdot \text{OPT} \end{aligned}$$

Let  $I$  denote the solution obtained by the first rounding algorithm and  $I'$  be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means  $I'$  is never better than  $I$ .

- ▶ Suppose that we take  $S_i$  in the first algorithm. I.e.,  $i \in I$ .
- ▶ This means  $x_i \geq \frac{1}{f}$ .
- ▶ Because of **Complementary Slackness Conditions** the corresponding constraint in the dual must be tight.
- ▶ Hence, the second algorithm will also choose  $S_i$ .

## Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an  $f$ -approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

where  $x^*$  is an optimum solution to the primal LP.

2. The set  $I$  contains only sets for which the dual inequality is tight.

Of course, we also need that  $I$  is a cover.

## Technique 3: The Primal Dual Method

### Algorithm 1 PrimalDual

```

1:  $y \leftarrow 0$ 
2:  $I \leftarrow \emptyset$ 
3: while exists  $u \notin \bigcup_{i \in I} S_i$  do
4:   increase dual variable  $y_i$  until constraint for some
   new set  $S_\ell$  becomes tight
5:    $I \leftarrow I \cup \{\ell\}$ 

```

## Technique 4: The Greedy Algorithm

### Algorithm 1 Greedy

```

1:  $I \leftarrow \emptyset$ 
2:  $\hat{S}_j \leftarrow S_j$  for all  $j$ 
3: while  $I$  not a set cover do
4:    $\ell \leftarrow \arg \min_{j: \hat{S}_j \neq \emptyset} \frac{w_j}{|\hat{S}_j|}$ 
5:    $I \leftarrow I \cup \{\ell\}$ 
6:    $\hat{S}_j \leftarrow \hat{S}_j - S_\ell$  for all  $j$ 

```

In every round the Greedy algorithm takes the set that covers remaining elements in the most **cost-effective** way.

We choose a set such that the ratio between cost and still uncovered elements in the set is minimized.

## Technique 4: The Greedy Algorithm

### Lemma 11

Given positive numbers  $a_1, \dots, a_k$  and  $b_1, \dots, b_k$  then

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}$$

## Technique 4: The Greedy Algorithm

Let  $n_\ell$  denote the number of elements that remain at the beginning of iteration  $\ell$ .  $n_1 = n = |U|$  and  $n_{s+1} = 0$  if we need  $s$  iterations.

In the  $\ell$ -th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j \in \text{OPT}} w_j}{\sum_{j \in \text{OPT}} |\hat{S}_j|} = \frac{\text{OPT}}{\sum_{j \in \text{OPT}} |\hat{S}_j|} \leq \frac{\text{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining  $n_\ell$  elements with cost OPT.

Let  $\hat{S}_j$  be a subset that minimizes this ratio. Hence,  
 $w_j / |\hat{S}_j| \leq \frac{\text{OPT}}{n_\ell}$ .

## Technique 4: The Greedy Algorithm

Adding this set to our solution means  $n_{\ell+1} = n_\ell - |\hat{S}_j|$ .

$$w_j \leq \frac{|\hat{S}_j| \text{OPT}}{n_\ell} = \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT}$$

## Technique 4: The Greedy Algorithm

$$\begin{aligned} \sum_{j \in I} w_j &\leq \sum_{\ell=1}^s \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT} \\ &\leq \text{OPT} \sum_{\ell=1}^s \left( \frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \dots + \frac{1}{n_{\ell+1} + 1} \right) \\ &= \text{OPT} \sum_{i=1}^k \frac{1}{i} \\ &= H_n \cdot \text{OPT} \leq \text{OPT}(\ln n + 1) . \end{aligned}$$

## Technique 5: Randomized Rounding

One round of randomized rounding:

Pick set  $S_j$  uniformly at random with probability  $1 - x_j$  (for all  $j$ ).

**Version A:** Repeat rounds until you have a cover.

**Version B:** Repeat for  $s$  rounds. If you have a cover STOP.

Otherwise, repeat the whole algorithm.

**Probability that  $u \in U$  is not covered (in one round):**

$\Pr[u \text{ not covered in one round}]$

$$\begin{aligned} &= \prod_{j:u \in S_j} (1 - x_j) \leq \prod_{j:u \in S_j} e^{-x_j} \\ &= e^{-\sum_{j:u \in S_j} x_j} \leq e^{-1} . \end{aligned}$$

**Probability that  $u \in U$  is not covered (after  $\ell$  rounds):**

$$\Pr[u \text{ not covered after } \ell \text{ round}] \leq \frac{1}{e^\ell} .$$

$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$

$= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \dots \vee u_n \text{ not covered}]$

$$\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} .$$

### Lemma 12

*With high probability  $\mathcal{O}(\log n)$  rounds suffice.*

**With high probability:**

For any constant  $\alpha$  the number of rounds is at most  $\mathcal{O}(\log n)$

with probability at least  $1 - n^{-\alpha}$ .

**Proof:** We have

$$\Pr[\#rounds \geq (\alpha + 1) \ln n] \leq ne^{-(\alpha+1) \ln n} = n^{-\alpha} .$$

## Expected Cost

► Version A.

Repeat for  $s = (\alpha + 1) \ln n$  rounds. If you don't have a cover simply take all sets.

$$E[\text{cost}] \leq (\alpha + 1) \ln n \cdot \text{cost}(LP) + \left(\sum_j w_j\right) n^{-\alpha} = \mathcal{O}(\ln n) \cdot \text{OPT}$$

If the weights are polynomially bounded (smallest weight is 1), sufficiently large  $\alpha$  and OPT at least 1.

## Expected Cost

► Version B.

Repeat for  $s = (\alpha + 1) \ln n$  rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$E[\text{cost} \mid \text{success}]$$

$$= \frac{1}{\Pr[\text{success}]} (E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}])$$

$$\leq \frac{1}{\Pr[\text{success}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1) \ln n \cdot \text{cost}(LP)$$

$$\leq 2(\alpha + 1) \ln n \cdot \text{OPT}$$

for  $n \geq 2$  and  $\alpha \geq 1$ .

Randomized rounding gives an  $\mathcal{O}(\log n)$  approximation. The running time is polynomial with high probability.

### Theorem 13 (without proof)

*There is no approximation algorithm for set cover with approximation guarantee better than  $\frac{1}{2} \log n$  unless NP has quasi-polynomial time algorithms (algorithms with running time  $2^{\text{poly}(\log n)}$ ).*

### Techniques:

- Deterministic Rounding
- Rounding of the Dual
- Primal Dual
- Greedy
- Randomized Rounding
- Local Search
- Rounding the Data + Dynamic Programming

## Scheduling Jobs on Identical Parallel Machines

Given  $n$  jobs, where job  $j \in \{1, \dots, n\}$  has processing time  $p_j$ .  
Schedule the jobs on  $m$  identical parallel machines such that the **Makespan** (finishing time of the last job) is minimized.

$$\begin{array}{ll} \min & L \\ \text{s.t.} & \forall \text{machines } i \quad \sum_j p_j \cdot x_{j,i} \leq L \\ & \forall \text{jobs } j \quad \sum_i x_{j,i} \geq 1 \\ & \forall i, j \quad x_{j,i} \in \{0, 1\} \end{array}$$

Here the variable  $x_{j,i}$  is the decision variable that describes whether job  $j$  is assigned to machine  $i$ .

## Lower Bounds on the Solution

Let for a given schedule  $C_j$  denote the finishing time of machine  $j$ , and let  $C_{\max}$  be the makespan.

Let  $C_{\max}^*$  denote the makespan of an optimal solution.

Clearly

$$C_{\max}^* \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

## Lower Bounds on the Solution

The average work performed by a machine is  $\frac{1}{m} \sum_j p_j$ .  
Therefore,

$$C_{\max}^* \geq \frac{1}{m} \sum_j p_j$$

## Local Search

A local search algorithm successively makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptually very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

## Local Search for Scheduling

**Local Search Strategy:** Take the job that finishes last in the produced schedule. Move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

## Local Search Analysis

Let  $\ell$  be the job that finishes last in the produced schedule.

Let  $S_\ell$  be its start time, and let  $C_\ell$  be its completion time.

Note that every machine is busy before time  $S_\ell$ , because otherwise we could move the job  $\ell$  and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to  $S_\ell$  the other from  $S_\ell$  to  $C_\ell$ .

The interval  $[S_\ell, C_\ell]$  is of length  $p_\ell \leq C_{\max}^*$ .

During the first interval  $[0, S_\ell]$  all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j.$$

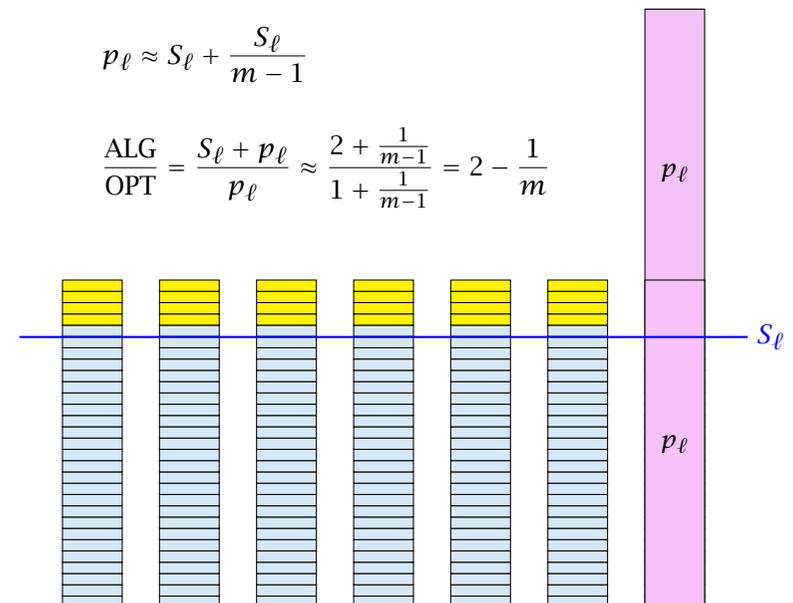
Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j = \left(1 - \frac{1}{m}\right)p_\ell + \frac{1}{m} \sum_j p_j \leq \left(2 - \frac{1}{m}\right)C_{\max}^*$$

## A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\text{ALG}}{\text{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$





- ▶ We can assume that one machine schedules  $p_1$  and  $p_n$  (the largest and smallest job).
- ▶ If not assume wlog. that  $p_1$  is scheduled on machine  $A$  and  $p_n$  on machine  $B$ .
- ▶ Let  $p_A$  and  $p_B$  be the other job scheduled on  $A$  and  $B$ , respectively.
- ▶  $p_1 + p_n \leq p_1 + p_A$  and  $p_A + p_B \leq p_1 + p_A$ , hence scheduling  $p_1$  and  $p_n$  on one machine and  $p_A$  and  $p_B$  on the other, cannot increase the Makespan.
- ▶ Repeat the above argument for the remaining machines.

## Traveling Salesman

Given a set of cities ( $\{1, \dots, n\}$ ) and a symmetric matrix  $C = (c_{ij})$ ,  $c_{ij} \geq 0$  that specifies for every pair  $(i, j) \in [n] \times [n]$  the cost for travelling from city  $i$  to city  $j$ . Find a permutation  $\pi$  of the cities such that the round-trip cost

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$$

is minimized.

## Traveling Salesman

### Theorem 15

There does not exist an  $O(2^n)$ -approximation algorithm for TSP.

### Hamiltonian Cycle:

For a given undirected graph  $G = (V, E)$  decide whether there exists a simple cycle that contains all nodes in  $G$ .

- ▶ Given an instance to HAMPATH we create an instance for TSP.
- ▶ If  $(i, j) \notin E$  then set  $c_{ij}$  to  $n2^n$  otw. set  $c_{ij}$  to 1. This instance has polynomial size.
- ▶ There exists a Hamiltonian Path iff there exists a tour with cost  $n$ . Otw. any tour has cost strictly larger than  $2^n$ .
- ▶ An  $O(2^n)$ -approximation algorithm could decide btw. these cases. Hence, cannot exist unless  $P = NP$ .

## Metric Traveling Salesman

In the metric version we assume for every triple  $i, j, k \in \{1, \dots, n\}$

$$c_{ij} \leq c_{ik} + c_{jk} .$$

It is convenient to view the input as a complete undirected graph  $G = (V, E)$ , where  $c_{ij}$  for an edge  $(i, j)$  defines the distance between nodes  $i$  and  $j$ .

## TSP: Lower Bound I

### Lemma 16

The cost  $\text{OPT}_{\text{TSP}}(G)$  of an optimum traveling salesman tour is at least as large as the weight  $\text{OPT}_{\text{MST}}(G)$  of a minimum spanning tree in  $G$ .

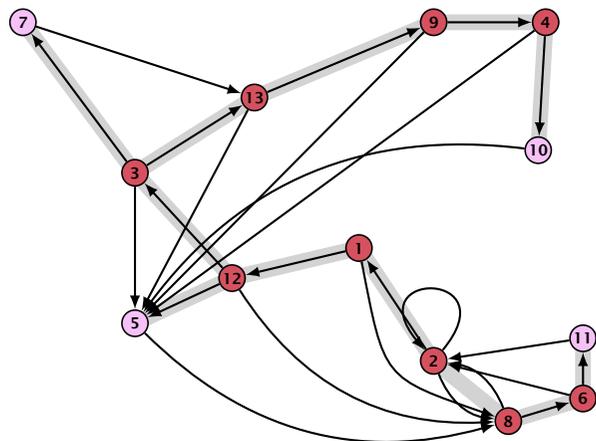
#### Proof:

- ▶ Take the optimum TSP-tour.
- ▶ Delete one edge.
- ▶ This gives a spanning tree of cost at most  $\text{OPT}_{\text{TSP}}(G)$ .

## TSP: Greedy Algorithm

- ▶ Start with a tour on a subset  $S$  containing a single node.
- ▶ Take the node  $v$  closest to  $S$ . Add it to  $S$  and expand the existing tour on  $S$  to include  $v$ .
- ▶ Repeat until all nodes have been processed.

## TSP: Greedy Algorithm



The gray edges form an MST, because exactly these edges are taken in Prim's algorithm.

## TSP: Greedy Algorithm

### Lemma 17

The Greedy algorithm is a 2-approximation algorithm.

Let  $S_i$  be the set at the start of the  $i$ -th iteration, and let  $v_i$  denote the node added during the iteration.

Further let  $s_i \in S_i$  be the node closest to  $v_i \in S_i$ .

Let  $r_i$  denote the successor of  $s_i$  in the tour before inserting  $v_i$ .

We replace the edge  $(s_i, r_i)$  in the tour by the two edges  $(s_i, v_i)$  and  $(v_i, r_i)$ .

This increases the cost by

$$c_{s_i, v_i} + c_{v_i, r_i} - c_{s_i, r_i} \leq 2c_{s_i, v_i}$$

## TSP: Greedy Algorithm

The edges  $(s_i, v_i)$  considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i, v_i} = \text{OPT}_{\text{MST}}(G)$$

which with the previous lower bound gives a 2-approximation.

## TSP: A different approach

Suppose that we are given an **Eulerian** graph  $G' = (V, E', c')$  of  $G = (V, E, c)$  such that for any edge  $(i, j) \in E'$   $c'(i, j) \geq c(i, j)$ .

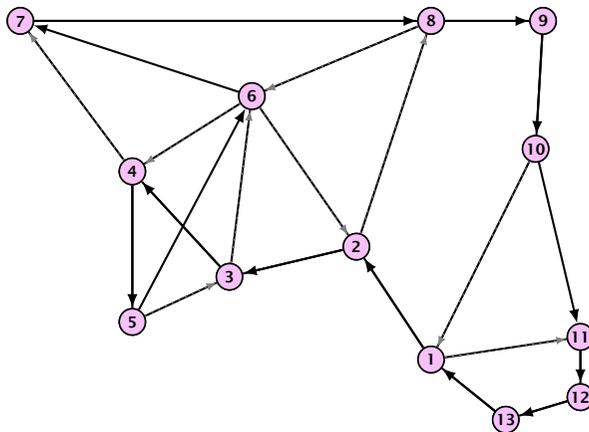
Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- ▶ Find an Euler tour of  $G'$ .
- ▶ Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- ▶ The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as **short cutting** the Euler tour.

## TSP: A different approach



## TSP: A different approach

Consider the following graph:

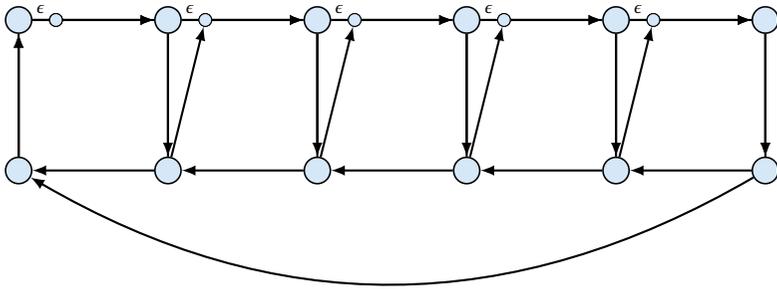
- ▶ Compute an MST of  $G$ .
- ▶ Duplicate all edges.

This graph is Eulerian, and the total cost of all edges is at most  $2 \cdot \text{OPT}_{\text{MST}}(G)$ .

Hence, short-cutting gives a tour of cost no more than  $2 \cdot \text{OPT}_{\text{MST}}(G)$  which means we have a 2-approximation.



## Tree shortcutting. Tight Example



► edges have Euclidean distance.

## 17 Rounding Data + Dynamic Programming

### Knapsack:

Given a set of items  $\{1, \dots, n\}$ , where the  $i$ -th item has weight  $w_i \in \mathbb{N}$  and profit  $p_i \in \mathbb{N}$ , and given a threshold  $W$ . Find a subset  $I \subseteq \{1, \dots, n\}$  of items of total weight at most  $W$  such that the profit is maximized (we can assume each  $w_i \leq W$ ).

$$\begin{array}{ll} \max & \sum_{i=1}^n p_i x_i \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \leq W \\ & \forall i \in \{1, \dots, n\} \quad x_i \in \{0, 1\} \end{array}$$

## 17 Rounding Data + Dynamic Programming

### Algorithm 1 Knapsack

```

1:  $A(1) \leftarrow [(0, 0), (p_1, w_1)]$ 
2: for  $j \leftarrow 2$  to  $n$  do
3:    $A(j) \leftarrow A(j-1)$ 
4:   for each  $(p, w) \in A(j-1)$  do
5:     if  $w + w_j \leq W$  then
6:       add  $(p + p_j, w + w_j)$  to  $A(j)$ 
7:     remove dominated pairs from  $A(j)$ 
8: return  $\max_{(p, w) \in A(n)} p$ 

```

The running time is  $\mathcal{O}(n \cdot \min\{W, P\})$ , where  $P = \sum_i p_i$  is the total profit of all items. This is only **pseudo-polynomial**.

## 17 Rounding Data + Dynamic Programming

### Definition 18

An algorithm is said to have pseudo-polynomial running time if the running time is polynomial when the numerical part of the input is encoded in unary.

## 17 Rounding Data + Dynamic Programming

- ▶ Let  $M$  be the maximum profit of an element.
- ▶ Set  $\mu := \epsilon M/n$ .
- ▶ Set  $p'_i := \lfloor p_i/\mu \rfloor$  for all  $i$ .
- ▶ Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}(n \sum_i p'_i) = \mathcal{O}(n \sum_i \lfloor \frac{p_i}{\epsilon M/n} \rfloor) \leq \mathcal{O}(\frac{n^3}{\epsilon}) .$$

## 17 Rounding Data + Dynamic Programming

Let  $S$  be the set of items returned by the algorithm, and let  $O$  be an optimum set of items.

$$\begin{aligned} \sum_{i \in S} p_i &\geq \mu \sum_{i \in S} p'_i \\ &\geq \mu \sum_{i \in O} p'_i \\ &\geq \sum_{i \in O} p_i - |O|\mu \\ &\geq \sum_{i \in O} p_i - n\mu \\ &= \sum_{i \in O} p_i - \epsilon M \\ &\geq (1 - \epsilon) \text{OPT} . \end{aligned}$$

## Scheduling Revisited

The previous analysis of the scheduling algorithm gave a makespan of

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where  $\ell$  is the last job to complete.

Together with the observation that if each  $p_i \geq \frac{1}{3} C_{\max}^*$  then LPT is optimal this gave a 4/3-approximation.

## 17.2 Scheduling Revisited

Partition the input into **long** jobs and **short** jobs.

A job  $j$  is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

**Idea:**

1. Find the optimum Makespan for the long jobs by brute force.
2. Then use the list scheduling algorithm for the short jobs, always assigning the next job to the least loaded machine.

We still have the inequality

$$\frac{1}{m} \sum_{j \neq \ell} p_j + p_\ell$$

where  $\ell$  is the last job (this only requires that all machines are busy before time  $S_\ell$ ).

If  $\ell$  is a long job, then the schedule must be optimal, as it consists of an optimal schedule of long jobs plus a schedule for short jobs.

If  $\ell$  is a short job its length is at most

$$p_\ell \leq \sum_j p_j / (mk)$$

which is at most  $C_{\max}^*/k$ .

Hence we get a schedule of length at most

$$(1 + \frac{1}{k})C_{\max}^*$$

There are at most  $km$  long jobs. Hence, the number of possibilities of scheduling these jobs on  $m$  machines is at most  $m^{km}$ , which is constant **if  $m$  is constant**. Hence, it is easy to implement the algorithm in polynomial time.

### Theorem 19

*The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling  $n$  jobs on  $m$  identical machines if  $m$  is constant.*

We choose  $k = \lceil \frac{1}{\epsilon} \rceil$ .

How to get rid of the requirement that  $m$  is constant?

We first design an algorithm that works as follows:

On input of  $T$  it either finds a schedule of length  $(1 + \frac{1}{k})T$  or certifies that no schedule of length at most  $T$  exists (assume  $T \geq \frac{1}{m} \sum_j p_j$ ).

We partition the jobs into **long** jobs and **short** jobs:

- ▶ A job is long if its size is larger than  $T/k$ .
- ▶ Otw. it is a short job.

- ▶ We round all long jobs down to multiples of  $T/k^2$ .
- ▶ For these rounded sizes we first find an optimal schedule.
- ▶ If this schedule does not have length at most  $T$  we conclude that also the original sizes don't allow such a schedule.
- ▶ If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most  $T$ .

There can be at most  $k$  (long) jobs assigned to a machine as otherwise their rounded sizes would add up to more than  $T$  (note that the rounded size of a long job is at least  $T/k$ ).

Since, jobs had been rounded to multiples of  $T/k^2$  going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1 + \frac{1}{k}\right)T .$$

During the second phase there always must exist a machine with load at most  $T$ , since  $T$  is larger than the average load.

Assigning the current (short) job to such a machine gives that the new load is at most

$$T + \frac{T}{k} \leq \left(1 + \frac{1}{k}\right)T .$$

**Running Time for scheduling large jobs:** There should not be a job with rounded size more than  $T$  as otherwise the problem becomes trivial.

Hence, any large job has rounded size of  $\frac{i}{k^2}T$  for  $i \in \{k, \dots, k^2\}$ . Therefore the number of different inputs is at most  $n^{k^2}$  (described by a vector of length  $k^2$  where, the  $i$ -th entry describes the number of jobs of size  $\frac{i}{k^2}T$ ). **This is polynomial.**

The schedule/configuration of a particular machine  $x$  can be described by a vector of length  $k^2$  where the  $i$ -th entry describes the number of jobs of rounded size  $\frac{i}{k^2}T$  assigned to  $x$ . There are only  $(k + 1)^{k^2}$  different vectors.

This means there are a **constant** number of different machine configurations.

Let  $\text{OPT}(n_1, \dots, n_{k^2})$  be the **number of machines** that are required to schedule input vector  $(n_1, \dots, n_{k^2})$  with Makespan at most  $T$ .

**If  $\text{OPT}(n_1, \dots, n_{k^2}) \leq m$  we can schedule the input.**

We have

$$\text{OPT}(n_1, \dots, n_{k^2}) = \begin{cases} 0 & (n_1, \dots, n_{k^2}) = 0 \\ 1 + \min_{(s_1, \dots, s_{k^2}) \in C} \text{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \neq 0 \\ \infty & \text{otw.} \end{cases}$$

where  $C$  is the set of all configurations.

Hence, the running time is roughly  $(k + 1)^{k^2} n^{k^2} \approx (nk)^{k^2}$ .

We can turn this into a PTAS by choosing  $k = \lceil 1/\epsilon \rceil$  and using binary search. This gives a running time that is exponential in  $1/\epsilon$ .

**Can we do better?**

Scheduling on identical machines with the goal of minimizing Makespan is a **strongly NP-complete** problem.

**Theorem 20**

*There is no FPTAS for problems that are strongly NP-hard.*

**More General**

Let  $OPT(n_1, \dots, n_A)$  be the number of machines that are required to schedule input vector  $(n_1, \dots, n_A)$  with Makespan at most  $T$  (**A: number of different sizes**).

If  $OPT(n_1, \dots, n_A) \leq m$  we can schedule the input.

$$OPT(n_1, \dots, n_A) = \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min_{(s_1, \dots, s_A) \in C} OPT(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \neq 0 \\ \infty & \text{otw.} \end{cases}$$

where  $C$  is the set of all configurations.

$|C| \leq (B + 1)^A$ , where  $B$  is the **number of jobs that possibly can fit on the same machine**.

The **running time** is then  $O((B + 1)^A n^A)$  because the dynamic programming table has just  $n^A$  entries.

**Bin Packing**

Given  $n$  items with sizes  $s_1, \dots, s_n$  where

$$1 > s_1 \geq \dots \geq s_n > 0 .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

**Theorem 21**

*There is no  $\rho$ -approximation for Bin Packing with  $\rho < 3/2$  unless  $P = NP$ .*

**Bin Packing**

**Proof**

- ▶ In the partition problem we are given positive integers  $b_1, \dots, b_n$  with  $B = \sum_i b_i$  even. Can we partition the integers into two sets  $S$  and  $T$  s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- ▶ We can solve this problem by setting  $s_i := 2b_i/B$  and asking whether we can pack the resulting items into 2 bins or not.
- ▶ A  $\rho$ -approximation algorithm with  $\rho < 3/2$  cannot output 3 or more bins when 2 are optimal.
- ▶ Hence, such an algorithm can solve Partition.

## Bin Packing

### Definition 22

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms  $\{A_\epsilon\}$  along with a constant  $c$  such that  $A_\epsilon$  returns a solution of value at most  $(1 + \epsilon)\text{OPT} + c$  for minimization problems.

- ▶ Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.
- ▶ However, we will develop an APTAS for Bin Packing.

## Bin Packing

Again we can differentiate between small and large items.

### Lemma 23

*Any packing of items of size at most  $\gamma$  into  $\ell$  bins can be extended to a packing of all items into  $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$  bins, where  $\text{SIZE}(I) = \sum_i s_i$  is the sum of all item sizes.*

- ▶ If after Greedy we use more than  $\ell$  bins, all bins (apart from the last) must be full to at least  $1 - \gamma$ .
- ▶ Hence,  $r(1 - \gamma) \leq \text{SIZE}(I)$  where  $r$  is the number of nearly-full bins.
- ▶ This gives the lemma.

Choose  $\gamma = \epsilon/2$ . Then we either use  $\ell$  bins or at most

$$\frac{1}{1 - \epsilon/2} \cdot \text{OPT} + 1 \leq (1 + \epsilon) \cdot \text{OPT} + 1$$

bins.

It remains to find an algorithm for the large items.

## Bin Packing

### Linear Grouping:

Generate an instance  $I'$  (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first  $k$  items belong to group 1; the following  $k$  items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

### Lemma 24

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

#### Proof 1:

- ▶ Any bin packing for  $I$  gives a bin packing for  $I'$  as follows.
- ▶ Pack the items of group 2, where in the packing for  $I$  the items for group 1 have been packed;
- ▶ Pack the items of groups 3, where in the packing for  $I$  the items for group 2 have been packed;
- ▶ ...

### Lemma 25

$$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$$

#### Proof 2:

- ▶ Any bin packing for  $I'$  gives a bin packing for  $I$  as follows.
- ▶ Pack the items of group 1 into  $k$  new bins;
- ▶ Pack the items of groups 2, where in the packing for  $I'$  the items for group 2 have been packed;
- ▶ ...

Assume that our instance does not contain pieces smaller than  $\epsilon/2$ . Then  $\text{SIZE}(I) \geq \epsilon n/2$ .

We set  $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$ .

Then  $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$  (here we used  $\lfloor \alpha \rfloor \geq \alpha/2$  for  $\alpha \geq 1$ ).

Hence, after grouping we have a constant number of piece sizes ( $4/\epsilon^2$ ) and at most a constant number ( $2/\epsilon$ ) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon) \text{OPT}(I)$$

- ▶ running time  $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$ .

### Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\text{OPT}(I) + \mathcal{O}(\log^2(\text{SIZE}(I))) .$$

Note that this is usually better than a guarantee of

$$(1 + \epsilon) \text{OPT}(I) + 1 .$$

## Configuration LP

### Change of Notation:

- ▶ Group pieces of identical size.
- ▶ Let  $s_1$  denote the largest size, and let  $b_1$  denote the number of pieces of size  $s_1$ .
- ▶  $s_2$  is second largest size and  $b_2$  number of pieces of size  $s_2$ ;
- ▶ ...
- ▶  $s_m$  smallest size and  $b_m$  number of pieces of size  $s_m$ .

## Configuration LP

A possible packing of a bin can be described by an  $m$ -tuple  $(t_1, \dots, t_m)$ , where  $t_i$  describes the number of pieces of size  $s_i$ . Clearly,

$$\sum_i t_i \cdot s_i \leq 1 .$$

We call a vector that fulfills the above constraint a **configuration**.

## Configuration LP

Let  $N$  be the number of configurations (**exponential**).

Let  $T_1, \dots, T_N$  be the sequence of all possible configurations (a configuration  $T_j$  has  $T_{ji}$  pieces of size  $s_i$ ).

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \\ & \forall j \in \{1, \dots, N\} \quad x_j \text{ integral} \end{array}$$

### How to solve this LP?

later...

We can assume that each item has size at least  $1/\text{SIZE}(I)$ .

## Harmonic Grouping

- ▶ Sort items according to size (monotonically decreasing).
- ▶ Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- ▶ I.e.,  $G_1$  is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for  $G_2, \dots, G_{r-1}$ .
- ▶ Only the size of items in the last group  $G_r$  may sum up to less than 2.

## Harmonic Grouping

From the grouping we obtain instance  $I'$  as follows:

- ▶ Round all items in a group to the size of the largest group member.
- ▶ Delete all items from group  $G_1$  and  $G_r$ .
- ▶ For groups  $G_2, \dots, G_{r-1}$  delete  $n_i - n_{i-1}$  items.
- ▶ Observe that  $n_i \geq n_{i-1}$ .

### Lemma 26

*The number of different sizes in  $I'$  is at most  $\text{SIZE}(I)/2$ .*

- ▶ Each group that survives (recall that  $G_1$  and  $G_r$  are deleted) has total size at least 2.
- ▶ Hence, the number of surviving groups is at most  $\text{SIZE}(I)/2$ .
- ▶ All items in a group have the same size in  $I'$ .

## Lemma 27

The total size of deleted items is at most  $\mathcal{O}(\log(\text{SIZE}(I)))$ .

- ▶ The total size of items in  $G_1$  and  $G_r$  is at most 6 as a group has total size at most 3.
- ▶ Consider a group  $G_i$  that has strictly more items than  $G_{i-1}$ .
- ▶ It discards  $n_i - n_{i-1}$  pieces of total size at most

$$3 \frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the smallest piece has size at most  $3/n_i$ .

- ▶ Summing over all  $i$  that have  $n_i > n_{i-1}$  gives a bound of at most

$$\sum_{j=1}^{n_r-1} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) .$$

(note that  $n_r \leq \text{SIZE}(I)$  since we assume that the size of each item is at least  $1/\text{SIZE}(I)$ ).

## Algorithm 1 BinPack

- 1: **if**  $\text{SIZE}(I) < 10$  **then**
- 2:     pack remaining items greedily
- 3: Apply harmonic grouping to create instance  $I'$ ; pack discarded items in at most  $\mathcal{O}(\log(\text{SIZE}(I)))$  bins.
- 4: Let  $x$  be optimal solution to configuration LP
- 5: Pack  $\lfloor x_j \rfloor$  bins in configuration  $T_j$  for all  $j$ ; call the packed instance  $I_1$ .
- 6: Let  $I_2$  be remaining pieces from  $I'$
- 7: Pack  $I_2$  via  $\text{BinPack}(I_2)$

## Analysis

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \leq \text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$$

### Proof:

- ▶ Each piece surviving in  $I'$  can be mapped to a piece in  $I$  of no lesser size. Hence,  $\text{OPT}_{\text{LP}}(I') \leq \text{OPT}_{\text{LP}}(I)$
- ▶  $\lfloor x_j \rfloor$  is feasible solution for  $I_1$  (even integral).
- ▶  $x_j - \lfloor x_j \rfloor$  is feasible solution for  $I_2$ .

## Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in  $I_1$ .
3. Pieces in  $I_2$  are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most  $\text{OPT}_{\text{LP}}$  many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where  $L$  is the number of recursion levels.

## Analysis

We can show that  $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$ . Hence, the number of recursion levels is only  $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$  in total.

- ▶ The number of non-zero entries in the solution to the configuration LP for  $I'$  is at most the number of constraints, which is the number of different sizes ( $\leq \text{SIZE}(I)/2$ ).
- ▶ The total size of items in  $I_2$  can be at most  $\sum_{j=1}^N x_j - \lfloor x_j \rfloor$  which is at most the number of non-zero entries in the solution to the configuration LP.

## How to solve the LP?

Let  $T_1, \dots, T_N$  be the sequence of all possible configurations (a configuration  $T_j$  has  $T_{ji}$  pieces of size  $s_i$ ).  
In total we have  $b_i$  pieces of size  $s_i$ .

### Primal

$$\begin{array}{ll} \min & \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

### Dual

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

## Separation Oracle

Suppose that I am given variable assignment  $y$  for the dual.

### How do I find a violated constraint?

I have to find a configuration  $T_j = (T_{j1}, \dots, T_{jm})$  that

- ▶ is feasible, i.e.,

$$\sum_{i=1}^m T_{ji} \cdot s_i \leq 1,$$

- ▶ and has a large profit

$$\sum_{i=1}^m T_{ji} y_i > 1$$

But this is the Knapsack problem.

## Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with  $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$  we find it, since we can obtain at least  $(1 - \epsilon)$  of the optimal profit.

The solution we get is feasible for:

### Dual'

$$\begin{array}{ll} \max & \sum_{i=1}^m y_i b_i \\ \text{s.t.} & \forall j \in \{1, \dots, N\} \quad \sum_{i=1}^m T_{ji} y_i \leq 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} \quad y_i \geq 0 \end{array}$$

### Primal'

$$\begin{array}{ll} \min & (1 + \epsilon') \sum_{j=1}^N x_j \\ \text{s.t.} & \forall i \in \{1 \dots m\} \quad \sum_{j=1}^N T_{ji} x_j \geq b_i \\ & \forall j \in \{1, \dots, N\} \quad x_j \geq 0 \end{array}$$

## Separation Oracle

If the value of the computed dual solution (which may be infeasible) is  $z$  then

$$\text{OPT} \leq z \leq (1 + \epsilon') \text{OPT}$$

### How do we get good primal solution (not just the value)?

- ▶ The constraints used when computing  $z$  **certify** that the solution is feasible for  $\text{DUAL}'$ .
- ▶ Suppose that we drop all unused constraints in  $\text{DUAL}$ . We will compute the same solution feasible for  $\text{DUAL}'$ .
- ▶ Let  $\text{DUAL}''$  be  $\text{DUAL}$  without unused constraints.
- ▶ The dual to  $\text{DUAL}''$  is  $\text{PRIMAL}$  where we ignore variables for which the corresponding dual constraint has not been used.
- ▶ The optimum value for  $\text{PRIMAL}''$  is at most  $(1 + \epsilon') \text{OPT}$ .
- ▶ We can compute the corresponding solution in polytime.

This gives that overall we need at most

$$(1 + \epsilon') \text{OPT}_{\text{LP}}(I) + \mathcal{O}(\log^2(\text{SIZE}(I)))$$

bins.

We can choose  $\epsilon' = \frac{1}{\text{OPT}}$  as  $\text{OPT} \leq \# \text{items}$  and since we have a **fully polynomial time approximation scheme (FPTAS)** for knapsack.

## 18 MAXSAT

### Problem definition:

- ▶  $n$  Boolean variables
- ▶  $m$  clauses  $C_1, \dots, C_m$ . For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- ▶ Non-negative weight  $w_j$  for each clause  $C_j$ .
- ▶ Find an assignment of true/false to the variables such that the total weight of clauses that are **satisfied** is maximum.

## 18 MAXSAT

### Terminology:

- ▶ A variable  $x_i$  and its negation  $\bar{x}_i$  are called **literals**.
- ▶ Hence, each clause consists of a set of literals (i.e., no duplications:  $x_i \vee x_i \vee \bar{x}_j$  is **not** a clause).
- ▶ We assume a clause does not contain  $x_i$  and  $\bar{x}_i$  for any  $i$ .
- ▶  $x_i$  is called a **positive literal** while the negation  $\bar{x}_i$  is called a **negative literal**.
- ▶ For a given clause  $C_j$  the number of its literals is called its **length** or **size** and denoted with  $\ell_j$ .
- ▶ Clauses of length one are called **unit clauses**.

## MAXSAT: Flipping Coins

Set each  $x_i$  independently to **true** with probability  $\frac{1}{2}$  (and, hence, to **false** with probability  $\frac{1}{2}$ , as well).

Define random variable  $X_j$  with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight  $W$  of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\ &= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\ &\geq \frac{1}{2} \sum_j w_j \\ &\geq \frac{1}{2} \text{OPT} \end{aligned}$$

## MAXSAT: LP formulation

- ▶ Let for a clause  $C_j$ ,  $P_j$  be the set of positive literals and  $N_j$  the set of negative literals.

$$C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{i \in N_j} \bar{x}_i$$

$$\begin{array}{ll} \max & \sum_j w_j z_j \\ \text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\ & \forall i \quad y_i \in \{0, 1\} \\ & \forall j \quad z_j \leq 1 \end{array}$$

## MAXSAT: Randomized Rounding

Set each  $x_i$  independently to **true** with probability  $y_i$  (and, hence, to **false** with probability  $(1 - y_i)$ ).

## Lemma 28 (Geometric Mean $\leq$ Arithmetic Mean)

For any nonnegative  $a_1, \dots, a_k$

$$\left( \prod_{i=1}^k a_i \right)^{1/k} \leq \frac{1}{k} \sum_{i=1}^k a_i$$

## Definition 29

A function  $f$  on an interval  $I$  is **concave** if for any two points  $s$  and  $r$  from  $I$  and any  $\lambda \in [0, 1]$  we have

$$f(\lambda s + (1 - \lambda)r) \geq \lambda f(s) + (1 - \lambda)f(r)$$

## Lemma 30

Let  $f$  be a concave function on the interval  $[0, 1]$ , with  $f(0) = a$  and  $f(1) = a + b$ . Then

$$\begin{aligned} f(\lambda) &= f((1 - \lambda)0 + \lambda 1) \\ &\geq (1 - \lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for  $\lambda \in [0, 1]$ .

$$\begin{aligned} \Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i \\ &\leq \left[ \frac{1}{\ell_j} \left( \sum_{i \in P_j} (1 - y_i) + \sum_{i \in N_j} y_i \right) \right]^{\ell_j} \\ &= \left[ 1 - \frac{1}{\ell_j} \left( \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \right) \right]^{\ell_j} \\ &\leq \left( 1 - \frac{z_j}{\ell_j} \right)^{\ell_j}. \end{aligned}$$

The function  $f(z) = 1 - (1 - \frac{z}{\ell})^\ell$  is concave. Hence,

$$\begin{aligned} \Pr[C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j} \\ &\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j . \end{aligned}$$

$f''(z) = -\frac{\ell-1}{\ell} \left[1 - \frac{z}{\ell}\right]^{\ell-2} \leq 0$  for  $z \in [0, 1]$ . Therefore,  $f$  is concave.

$$\begin{aligned} E[W] &= \sum_j w_j \Pr[C_j \text{ satisfied}] \\ &\geq \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \\ &\geq \left(1 - \frac{1}{e}\right) \text{OPT} . \end{aligned}$$

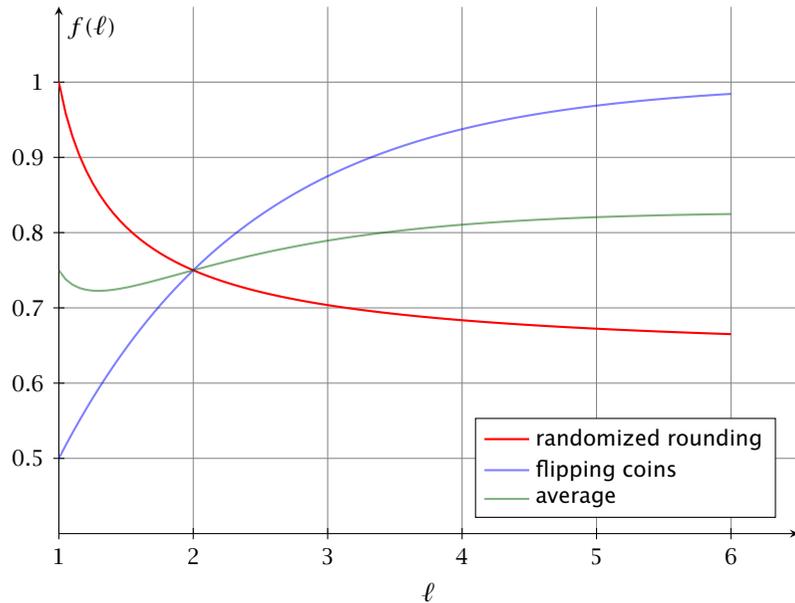
## MAXSAT: The better of two

### Theorem 31

Choosing the better of the two solutions given by randomized rounding and coin flipping yields a  $\frac{3}{4}$ -approximation.

Let  $W_1$  be the value of randomized rounding and  $W_2$  the value obtained by coin flipping.

$$\begin{aligned} E[\max\{W_1, W_2\}] &\geq E\left[\frac{1}{2}W_1 + \frac{1}{2}W_2\right] \\ &\geq \frac{1}{2} \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2} \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\ &\geq \sum_j w_j z_j \underbrace{\left[\frac{1}{2} \left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) + \frac{1}{2} \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right)\right]}_{\geq \frac{3}{4} \text{ for all integers}} \\ &\geq \frac{3}{4} \text{OPT} \end{aligned}$$



## MAXSAT: Nonlinear Randomized Rounding

So far we used **linear** randomized rounding, i.e., the probability that a variable is set to 1/true was exactly the value of the corresponding variable in the linear program.

We could define a function  $f : [0, 1] \rightarrow [0, 1]$  and set  $x_i$  to true with probability  $f(y_i)$ .

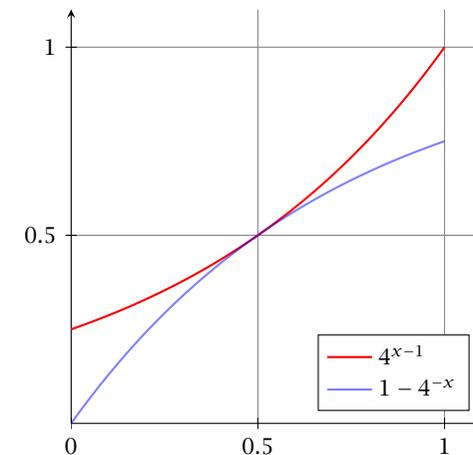
## MAXSAT: Nonlinear Randomized Rounding

Let  $f : [0, 1] \rightarrow [0, 1]$  be a function with

$$1 - 4^{-x} \leq f(x) \leq 4^{x-1}$$

### Theorem 32

*Rounding the LP-solution with a function  $f$  of the above form gives a  $\frac{3}{4}$ -approximation.*



$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i) \\
&\leq \prod_{i \in P_j} 4^{-y_i} \prod_{i \in N_j} 4^{y_i - 1} \\
&= 4^{-(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i))} \\
&\leq 4^{-z_j}
\end{aligned}$$

The function  $g(z) = 1 - 4^{-z}$  is concave on  $[0, 1]$ . Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \geq \frac{3}{4} \sum_j w_j z_j \geq \frac{3}{4} \text{OPT}$$

### Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

#### Definition 33 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

#### Lemma 34

*Our ILP-formulation for the MAXSAT problem has integrality gap at most  $\frac{3}{4}$ .*

$$\begin{array}{ll}
\max & \sum_j w_j z_j \\
\text{s.t.} & \forall j \quad \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \\
& \forall i \quad y_i \in \{0, 1\} \\
& \forall j \quad z_j \leq 1
\end{array}$$

Consider:  $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- ▶ any solution can satisfy at most 3 clauses
- ▶ we can set  $y_1 = y_2 = 1/2$  in the LP; this allows to set  $z_1 = z_2 = z_3 = z_4 = 1$
- ▶ hence, the LP has value 4.

## Facility Location

Given a set  $L$  of (possible) locations for placing facilities and a set  $D$  of customers together with cost functions  $s : D \times L \rightarrow \mathbb{R}^+$  and  $o : L \rightarrow \mathbb{R}^+$  find a set of facility locations  $F$  together with an assignment  $\phi : D \rightarrow F$  of customers to open facilities such that

$$\sum_{f \in F} o(f) + \sum_c s(c, \phi(c))$$

is minimized.

In the **metric facility location** problem we have

$$s(c, f) \leq s(c, f') + s(c', f) + s(c', f') .$$

## Facility Location

### Integer Program

$$\begin{array}{ll} \min & \sum_{i \in F} f_i y_i + \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij} \\ \text{s.t.} & \forall j \in D \quad \sum_{i \in F} x_{ij} = 1 \\ & \forall i \in F, j \in D \quad x_{ij} \leq y_i \\ & \forall i \in F, j \in D \quad x_{ij} \in \{0, 1\} \\ & \forall i \in F \quad y_i \in \{0, 1\} \end{array}$$

As usual we get an LP by relaxing the integrality constraints.

## Facility Location

### Dual Linear Program

$$\begin{array}{ll} \max & \sum_{j \in D} v_j \\ \text{s.t.} & \forall i \in F \quad \sum_{j \in D} w_{ij} \leq f_i \\ & \forall i \in F, j \in D \quad v_j - w_{ij} \leq c_{ij} \\ & \forall i \in F, j \in D \quad w_{ij} \geq 0 \end{array}$$

## Facility Location

### Definition 35

Given an LP solution  $(x^*, y^*)$  we say that facility  $i$  neighbours client  $j$  if  $x_{ij}^* > 0$ . Let  $N(j) = \{i \in F : x_{ij}^* > 0\}$ .

### Lemma 36

If  $(x^*, y^*)$  is an optimal solution to the facility location LP and  $(v^*, w^*)$  is an optimal dual solution, then  $x_{ij}^* > 0$  implies  $c_{ij} \leq v_j^*$ .

Follows from slackness conditions.

Suppose we open set  $S \subseteq F$  of facilities s.t. for all clients we have  $S \cap N(j) \neq \emptyset$ .

Then every client  $j$  has a facility  $i$  s.t. assignment cost for this client is at most  $c_{ij} \leq v_j^*$ .

Hence, the total assignment cost is

$$\sum_j c_{i_j j} \leq \sum_j v_j^* \leq \text{OPT} ,$$

where  $i_j$  is the facility that client  $j$  is assigned to.

### Problem: Facility cost may be huge!

Suppose we can partition a subset  $F' \subseteq F$  of facilities into neighbour sets of some clients. I.e.

$$F' = \biguplus_k N(j_k)$$

where  $j_1, j_2, \dots$  form a subset of the clients.

Now in each set  $N(j_k)$  we open the **cheapest** facility. Call it  $f_{i_k}$ .

We have

$$f_{i_k} = f_{i_k} \sum_{i \in N(j_k)} x_{ij_k}^* \leq \sum_{i \in N(j_k)} f_i x_{ij_k}^* \leq \sum_{i \in N(j_k)} f_i y_i^* .$$

Summing over all  $k$  gives

$$\sum_k f_{i_k} \leq \sum_k \sum_{i \in N(j_k)} f_i y_i^* = \sum_{i \in F'} f_i y_i^* \leq \sum_{i \in F} f_i y_i^*$$

Facility cost is at most the facility cost in an optimum solution.

**Problem:** so far clients  $j_1, j_2, \dots$  have a neighboring facility.  
**What about the others?**

**Definition 37**

Let  $N^2(j)$  denote all neighboring **clients** of the neighboring facilities of client  $j$ .

Note that  $N(j)$  is a set of facilities while  $N^2(j)$  is a set of clients.

**Algorithm 1** FacilityLocation

```

1:  $C \leftarrow D$  // unassigned clients
2:  $k \leftarrow 0$ 
3: while  $C \neq \emptyset$  do
4:    $k \leftarrow k + 1$ 
5:   choose  $j_k \in C$  that minimizes  $v_j^*$ 
6:   choose  $i_k \in N(j_k)$  as cheapest facility
7:   assign  $j_k$  and all unassigned clients in  $N^2(j_k)$  to  $i_k$ 
8:    $C \leftarrow C - \{j_k\} - N^2(j_k)$ 
  
```

Facility cost of this algorithm is at most OPT because the sets  $N(j_k)$  are disjoint.

**Total assignment cost:**

- ▶ Fix  $k$ ; set  $j = j_k$  and  $i = i_k$ . We know that  $c_{ij} \leq v_j^*$ .
- ▶ Let  $\ell \in N^2(j)$  and  $h$  (one of) its neighbour(s) in  $N(j)$ .

$$c_{i\ell} \leq c_{ij} + c_{hj} + c_{h\ell} \leq v_j^* + v_j^* + v_\ell^* \leq 3v_\ell^*$$

Summing this over all facilities gives that the total assignment cost is at most  $3 \cdot \text{OPT}$ . Hence, we get a 4-approximation.

In the above analysis we use the inequality

$$\sum_{i \in F} f_i y_i^* \leq \text{OPT} .$$

We know something stronger namely

$$\sum_{i \in F} f_i y_i^* + \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij}^* \leq \text{OPT} .$$

### Observation:

- ▶ Suppose when choosing a client  $j_k$ , instead of opening the cheapest facility in its neighborhood we choose a random facility according to  $x_{ij_k}^*$ .
- ▶ Then we incur connection cost

$$\sum_i c_{ij_k} x_{ij_k}^*$$

for client  $j_k$ . (In the previous algorithm we estimated this by  $v_{j_k}^*$ ).

- ▶ Define

$$C_j^* = \sum_i c_{ij} x_{ij}^*$$

to be the connection cost for client  $j$ .

### What will our facility cost be?

We only try to open a facility once (when it is in neighborhood of some  $j_k$ ). (recall that neighborhoods of different  $j_k$ 's are disjoint).

We open facility  $i$  with probability  $x_{ij_k} \leq y_i$  (in case it is in some neighborhood; otw. we open it with probability zero).

Hence, the expected facility cost is at most

$$\sum_{i \in F} f_i y_i .$$

### Algorithm 1 FacilityLocation

```
1:  $C \leftarrow D$  // unassigned clients
2:  $k \leftarrow 0$ 
3: while  $C \neq 0$  do
4:    $k \leftarrow k + 1$ 
5:   choose  $j_k \in C$  that minimizes  $v_{j_k}^* + C_{j_k}^*$ 
6:   choose  $i_k \in N(j_k)$  according to probability  $x_{ij_k}^*$ .
7:   assign  $j_k$  and all unassigned clients in  $N^2(j_k)$  to  $i_k$ 
8:    $C \leftarrow C - \{j_k\} - N^2(j_k)$ 
```

### Total assignment cost:

- ▶ Fix  $k$ ; set  $j = j_k$ .
- ▶ Let  $\ell \in N^2(j)$  and  $h$  (one of) its neighbour(s) in  $N(j)$ .
- ▶ If we assign a client  $\ell$  to the same facility as  $i$  we pay at most

$$\sum_i c_{ij} x_{ij}^* + c_{hj} + c_{h\ell} \leq C_j^* + v_j^* + v_\ell^* \leq C_\ell^* + 2v_\ell^*$$

Summing this over all clients gives that the total assignment cost is at most

$$\sum_j C_j^* + \sum_j 2v_j^* \leq \sum_j C_j^* + 2\text{OPT}$$

Hence, it is at most  $2\text{OPT}$  plus the total assignment cost in an optimum solution.

Adding the facility cost gives a 3-approximation.

### Lemma 38 (Chernoff Bounds)

Let  $X_1, \dots, X_n$  be  $n$  *independent* 0-1 random variables, not necessarily identically distributed. Then for  $X = \sum_{i=1}^n X_i$  and  $\mu = E[X]$ ,  $L \leq \mu \leq U$ , and  $\delta > 0$

$$\Pr[X \geq (1 + \delta)U] < \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^U,$$

and

$$\Pr[X \leq (1 - \delta)L] < \left( \frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^L,$$

### Lemma 39

For  $0 \leq \delta \leq 1$  we have that

$$\left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^U \leq e^{-U\delta^2/3}$$

and

$$\left( \frac{e^{-\delta}}{(1 - \delta)^{1-\delta}} \right)^L \leq e^{-L\delta^2/2}$$

## Integer Multicommodity Flows

- ▶ Given  $s_i$ - $t_i$  pairs in a graph.
- ▶ Connect each pair by a paths such that not too many path use any given edge.

$$\begin{array}{ll} \min & W \\ \text{s.t.} & \forall i \quad \sum_{p \in \mathcal{P}_i} x_p = 1 \\ & \sum_{p: e \in p} x_p \leq W \\ & x_p \in \{0, 1\} \end{array}$$

## Integer Multicommodity Flows

### Randomized Rounding:

For each  $i$  choose one path from the set  $\mathcal{P}_i$  at random according to the probability distribution given by the Linear Programming Solution.

### Theorem 40

If  $W^* \geq c \ln n$  for some constant  $c$ , then with probability at least  $n^{-c/3}$  the total number of paths using any edge is at most  $W^* + \sqrt{cW^* \ln n}$ .

## Integer Multicommodity Flows

Let  $X_e^i$  be a random variable that indicates whether the path for  $s_i-t_i$  uses edge  $e$ .

Then the number of paths using edge  $e$  is  $Y_e = \sum_i X_e^i$ .

$$E[Y_e] = \sum_i \sum_{p: e \in p} x_p^* = \sum_{p: e \in p} x_p^* \leq W^*$$

## Integer Multicommodity Flows

Choose  $\delta = \sqrt{(c \ln n)/W^*}$ .

Then

$$\Pr[Y_e \geq (1 + \delta)W^*] < e^{-W^* \delta^2/3} = \frac{1}{n^{c/3}}$$

## Repetition: Primal Dual for Set Cover

**Primal Relaxation:**

$$\begin{array}{ll} \min & \sum_{i=1}^k w_i x_i \\ \text{s.t.} & \forall u \in U \quad \sum_{i: u \in S_i} x_i \geq 1 \\ & \forall i \in \{1, \dots, k\} \quad x_i \geq 0 \end{array}$$

**Dual Formulation:**

$$\begin{array}{ll} \max & \sum_{u \in U} \gamma_u \\ \text{s.t.} & \forall i \in \{1, \dots, k\} \quad \sum_{u: u \in S_i} \gamma_u \leq w_i \\ & \gamma_u \geq 0 \end{array}$$

## Repetition: Primal Dual for Set Cover

### Algorithm:

- ▶ Start with  $y = 0$  (feasible dual solution).  
Start with  $x = 0$  (integral primal solution that may be infeasible).
- ▶ While  $x$  not feasible
  - ▶ Identify an element  $e$  that is not covered in current primal integral solution.
  - ▶ Increase dual variable  $y_e$  until a dual constraint becomes tight (maybe increase by 0!).
  - ▶ If this is the constraint for set  $S_j$  set  $x_j = 1$  (add this set to your solution).

## Repetition: Primal Dual for Set Cover

### Analysis:

- ▶ For every set  $S_j$  with  $x_j = 1$  we have

$$\sum_{e \in S_j} y_e = w_j$$

- ▶ Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e \leq f \cdot \sum_e y_e \leq f \cdot \text{OPT}$$

Note that the constructed pair of primal and dual solution fulfills **primal slackness conditions**.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

If we would also fulfill **dual slackness conditions**

$$y_e > 0 \Rightarrow \sum_{j: e \in S_j} x_j = 1$$

then the solution would be **optimal!!!**

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j: e \in S_j} x_j \leq f$$

This is sufficient to show that the solution is an  $f$ -approximation.

Suppose we have a primal/dual pair

$$\begin{array}{ll} \min & \sum_j c_j x_j \\ \text{s.t.} & \forall i \sum_j a_{ij} x_j \geq b_i \\ & \forall j x_j \geq 0 \end{array} \quad \begin{array}{ll} \max & \sum_i b_i y_i \\ \text{s.t.} & \forall j \sum_i a_{ij} y_i \leq c_j \\ & \forall i y_i \geq 0 \end{array}$$

and solutions that fulfill approximate slackness conditions:

$$\begin{aligned} x_j > 0 &\Rightarrow \sum_i a_{ij} y_i \geq \frac{1}{\alpha} c_j \\ y_i > 0 &\Rightarrow \sum_j a_{ij} x_j \leq \beta b_i \end{aligned}$$

Then

$$\begin{aligned} \sum_j c_j x_j &\leq \alpha \sum_j \left( \sum_i a_{ij} y_i \right) x_j \\ &= \alpha \sum_i \left( \sum_j a_{ij} x_j \right) y_i \\ &\leq \alpha \beta \cdot \sum_i b_i y_i \end{aligned}$$

Diagram annotations:  
 - A red box highlights  $\sum_j c_j x_j$  with the label "right hand side of j-th dual constraint".  
 - A blue box highlights  $\sum_j c_j x_j$  with the label "primal cost".  
 - A green box highlights  $\sum_i b_i y_i$  with the label "dual objective".

## Feedback Vertex Set for Undirected Graphs

- ▶ Given a graph  $G = (V, E)$  and non-negative weights  $w_v \geq 0$  for vertex  $v \in V$ .
- ▶ Choose a minimum cost subset of vertices s.t. every cycle contains at least one vertex.

We can encode this as an instance of Set Cover

- ▶ Each vertex can be viewed as a set that contains some cycles.
- ▶ However, this encoding gives a Set Cover instance of non-polynomial size.
- ▶ The  $O(\log n)$ -approximation for Set Cover does not help us to get a good solution.

Let  $\mathcal{C}$  denote the set of all cycles (where a cycle is identified by its set of vertices)

**Primal Relaxation:**

$$\begin{array}{ll} \min & \sum_v w_v x_v \\ \text{s.t.} & \forall C \in \mathcal{C} \quad \sum_{v \in C} x_v \geq 1 \\ & \forall v \quad x_v \geq 0 \end{array}$$

**Dual Formulation:**

$$\begin{array}{ll} \max & \sum_{C \in \mathcal{C}} \gamma_C \\ \text{s.t.} & \forall v \in V \quad \sum_{C: v \in C} \gamma_C \leq w_v \\ & \forall C \quad \gamma_C \geq 0 \end{array}$$

If we perform the previous dual technique for Set Cover we get the following:

- ▶ Start with  $x = 0$  and  $y = 0$
- ▶ While there is a cycle  $C$  that is not covered (does not contain a chosen vertex).
  - ▶ Increase  $\gamma_e$  until dual constraint for some vertex  $v$  becomes tight.
  - ▶ set  $x_v = 1$ .

Then

$$\begin{aligned} \sum_v w_v x_v &= \sum_v \sum_{C: v \in C} \gamma_C x_v \\ &= \sum_{v \in S} \sum_{C: v \in C} \gamma_C \\ &= \sum_C |S \cap C| \cdot \gamma_C \end{aligned}$$

where  $S$  is the set of vertices we choose.

If every cycle is short we get a good approximation ratio, but this is unrealistic.

**Algorithm 1** FeedbackVertexSet

- 1:  $y \leftarrow 0$
- 2:  $x \leftarrow 0$
- 3: **while** exists cycle  $C$  in  $G$  **do**
- 4:     increase  $\gamma_C$  until there is  $v \in C$  s.t.  $\sum_{C: v \in C} \gamma_C = w_v$
- 5:      $x_v = 1$
- 6:     remove  $v$  from  $G$
- 7:     repeatedly remove vertices of degree 1 from  $G$

**Idea:**

Always choose a short cycle that is not covered. If we always find a cycle of length at most  $\alpha$  we get an  $\alpha$ -approximation.

**Observation:**

For any path  $P$  of vertices of degree 2 in  $G$  the algorithm chooses at most one vertex from  $P$ .

**Observation:**

If we always choose a cycle for which the number of vertices of degree at least 3 is at most  $\alpha$  we get an  $\alpha$ -approximation.

**Theorem 41**

*In any graph with no vertices of degree 1, there always exists a cycle that has at most  $\mathcal{O}(\log n)$  vertices of degree 3 or more. We can find such a cycle in linear time.*

This means we have

$$y_C > 0 \Rightarrow |S \cap C| \leq \mathcal{O}(\log n) .$$

## Primal Dual for Shortest Path

Given a graph  $G = (V, E)$  with two nodes  $s, t \in V$  and edge-weights  $c : E \rightarrow \mathbb{R}^+$  find a shortest path between  $s$  and  $t$  w.r.t. edge-weights  $c$ .

$$\begin{array}{ll} \min & \sum_e c(e)x_e \\ \text{s.t.} & \forall S \in \mathcal{S} \quad \sum_{e:\delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here  $\delta(S)$  denotes the set of edges with exactly one end-point in  $S$ , and  $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$ .

## Primal Dual for Shortest Path

**The Dual:**

$$\begin{array}{ll} \max & \sum_S y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S:e \in \delta(S)} y_S \leq c(e) \\ & \forall S \in \mathcal{S} \quad y_S \geq 0 \end{array}$$

Here  $\delta(S)$  denotes the set of edges with exactly one end-point in  $S$ , and  $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$ .

## Primal Dual for Shortest Path

We can interpret the value  $\gamma_S$  as the width of a moat surrounding the set  $S$ .

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

### Algorithm 1 PrimalDualShortestPath

```
1:  $\gamma \leftarrow 0$ 
2:  $F \leftarrow \emptyset$ 
3: while there is no  $s$ - $t$  path in  $(V, F)$  do
4:   Let  $C$  be the connected component of  $(V, F)$  containing  $s$ 
5:   Increase  $\gamma_C$  until there is an edge  $e' \in \delta(C)$  such that  $\sum_{S: e' \in \delta(S)} \gamma_S = c(e')$ .
6:    $F \leftarrow F \cup \{e'\}$ 
7: Let  $P$  be an  $s$ - $t$  path in  $(V, F)$ 
8: return  $P$ 
```

### Lemma 42

*At each point in time the set  $F$  forms a tree.*

#### Proof:

- ▶ In each iteration we take the current connected component from  $(V, F)$  that contains  $s$  (call this component  $C$ ) and add some edge from  $\delta(C)$  to  $F$ .
- ▶ Since, at most one end-point of the new edge is in  $C$  the edge cannot close a cycle.

$$\begin{aligned} \sum_{e \in P} c(e) &= \sum_{e \in P} \sum_{S: e \in \delta(S)} \gamma_S \\ &= \sum_{S: s \in S, t \notin S} |P \cap \delta(S)| \cdot \gamma_S. \end{aligned}$$

If we can show that  $\gamma_S > 0$  implies  $|P \cap \delta(S)| = 1$  gives

$$\sum_{e \in P} c(e) = \sum_S \gamma_S \leq \text{OPT}$$

by weak duality.

Hence, we find a shortest path.

If  $S$  contains two edges from  $P$  then there must exist a subpath  $P'$  of  $P$  that starts and ends with a vertex from  $S$  (and all interior vertices are not in  $S$ ).

When we increased  $y_S$ ,  $S$  was a connected component of the set of edges  $F'$  that we had chosen till this point.

$F' \cup P'$  contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

### Steiner Forest Problem:

Given a graph  $G = (V, E)$ , together with source-target pairs  $s_i, t_i, i = 1, \dots, k$ , and a cost function  $c : E \rightarrow \mathbb{R}^+$  on the edges. Find a subset  $F \subseteq E$  of the edges such that for every  $i \in \{1, \dots, k\}$  there is a path between  $s_i$  and  $t_i$  only using edges in  $F$ .

$$\begin{array}{ll} \min & \sum_e c(e)x_e \\ \text{s.t.} & \forall S \subseteq V : S \in S_i \text{ for some } i \quad \sum_{e \in \delta(S)} x_e \geq 1 \\ & \forall e \in E \quad x_e \in \{0, 1\} \end{array}$$

Here  $S_i$  contains all sets  $S$  such that  $s_i \in S$  and  $t_i \notin S$ .

$$\begin{array}{ll} \max & \sum_{S : \exists i \text{ s.t. } S \in S_i} y_S \\ \text{s.t.} & \forall e \in E \quad \sum_{S : e \in \delta(S)} y_S \leq c(e) \\ & y_S \geq 0 \end{array}$$

The difference to the dual of the shortest path problem is that we have many more variables (sets for which we can generate a moat of non-zero width).

### Algorithm 1 FirstTry

- 1:  $y \leftarrow 0$
- 2:  $F \leftarrow \emptyset$
- 3: **while** not all  $s_i$ - $t_i$  pairs connected in  $F$  **do**
- 4:     Let  $C$  be some connected component of  $(V, F)$  such that  $|C \cap \{s_i, t_i\}| = 1$  for some  $i$ .
- 5:     Increase  $y_C$  until there is an edge  $e' \in \delta(C)$  s.t.  $\sum_{S \in S_i : e' \in \delta(S)} y_S = c_{e'}$
- 6:      $F \leftarrow F \cup \{e'\}$
- 7:     Let  $P_i$  be an  $s_i$ - $t_i$  path in  $(V, F)$
- 8: **return**  $\bigcup_i P_i$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S: e \in \delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S .$$

If we show that  $y_S > 0$  implies that  $|\delta(S) \cap F| \leq \alpha$  we are in good shape.

However, this is not true:

- ▶ Take a graph on  $k + 1$  vertices  $v_0, v_1, \dots, v_k$ .
- ▶ The  $i$ -th pair is  $v_0 - v_i$ .
- ▶ The first component  $C$  could be  $\{v_0\}$ .
- ▶ We only set  $y_{\{v_0\}} = 1$ . All other dual variables stay 0.
- ▶ The final set  $F$  contains all edges  $\{v_0, v_i\}, i = 1, \dots, k$ .
- ▶  $y_{\{v_0\}} > 0$  but  $|\delta(\{v_0\}) \cap F| = k$ .

### Algorithm 1 SecondTry

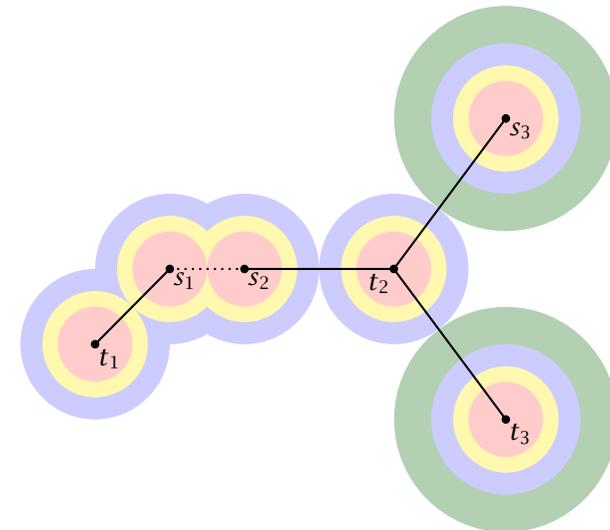
```

1:  $y \leftarrow 0; F \leftarrow \emptyset; \ell \leftarrow 0$ 
2: while not all  $s_i - t_i$  pairs connected in  $F$  do
3:    $\ell \leftarrow \ell + 1$ 
4:   Let  $C$  be set of all connected components  $C$  of  $(V, F)$ 
      such that  $|C \cap \{s_i, t_i\}| = 1$  for some  $i$ .
5:   Increase  $y_C$  for all  $C \in C$  uniformly until for some edge
       $e_\ell \in \delta(C')$ ,  $C' \in C$  s.t.  $\sum_{S: e_\ell \in \delta(S)} y_S = c_{e_\ell}$ 
6:    $F \leftarrow F \cup \{e_\ell\}$ 
7:  $F' \leftarrow F$ 
8: for  $k \leftarrow \ell$  downto 1 do // reverse deletion
9:   if  $F' - e_k$  is feasible solution then
10:    remove  $e_k$  from  $F'$ 
11: return  $F'$ 

```

The reverse deletion step is not strictly necessary this way. It would also be sufficient to simply delete all unnecessary edges in any order.

### Example



### Lemma 43

For any  $C$  in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|C|$$

This means that the number of times a moat from  $C$  is crossed in the final solution is at most twice the number of moats.

**Proof:** later...

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

- ▶ In the  $i$ -th iteration the increase of the left-hand side is

$$\epsilon \sum_{C \in \mathcal{C}} |F' \cap \delta(C)|$$

and the increase of the right hand side is  $2\epsilon|C|$ .

- ▶ Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

### Lemma 44

For any set of connected components  $C$  in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|C|$$

**Proof:**

- ▶ At any point during the algorithm the set of edges forms a forest (why?).
- ▶ Fix iteration  $i$ .  $e_i$  is the set we add to  $F$ . Let  $F_i$  be the set of edges in  $F$  at the beginning of the iteration.
- ▶ Let  $H = F' - F_i$ .
- ▶ All edges in  $H$  are necessary for the solution.

- ▶ Contract all edges in  $F_i$  into single vertices  $V'$ .
- ▶ We can consider the forest  $H$  on the set of vertices  $V'$ .
- ▶ Let  $\deg(v)$  be the degree of a vertex  $v \in V'$  within this forest.
- ▶ Color a vertex  $v \in V'$  **red** if it corresponds to a component from  $C$  (an active component). Otw. color it blue. (Let  $B$  the set of blue vertices (with non-zero degree) and  $R$  the set of red vertices)
- ▶ We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \stackrel{?}{\leq} 2|C| = 2|R|$$

- ▶ Suppose that no node in  $B$  has degree one.
- ▶ Then

$$\begin{aligned}\sum_{v \in R} \deg(v) &= \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- ▶ Every blue vertex with non-zero degree must have degree at least two.
  - ▶ Suppose not. The single edge connecting  $b \in B$  comes from  $H$ , and, hence, is necessary.
  - ▶ But this means that the cluster corresponding to  $b$  must separate a source-target pair.
  - ▶ But then it must be a red node.