Randomized and Approximation Algorithms

Lecture Algorithm Design

Stefano Leonardi, Rebecca Reiffenhäuser

December 18, 2020

Part 1
A Simple 2-Approximation

Reminder: Randomized Algorithms

- Make random decisions by flipping coins, drawing random numbers, etc. - Alternative: deterministic algorithms can always show the same behavior for the same input, but still *have access* to randomness if the input itself is randomized.
- We measure the quality of approximation not by approximative ratio $\frac{OPT}{ALG}$, but instead $\mathbb{E}\left[\frac{OPT}{ALG}\right]$.
- This is used because, while randomness can in the worst case lead to very bad results ($\frac{OPT}{ALG}$ would be usually very high for these algorithms), it can also lead to good ones - the expectation evens this out!
- **In general: more powerful than deterministic algorithms!**
  - Therefore: Even when there is a proven impossibility to $\alpha-$approximate a problem P deterministically (and in polynomial time), this often works with a randomized algorithm!
  - Randomness *breaks structures* that keep deterministic methods from performing well!

Chris Grocery Service

Chris' crazy working hours and his descent from southern Germany have given him a business idea: Since in Bavaria, the shops close very early, he will open a grocery service for people who never get off work early enough to do their own shopping. He has been offered to rent rooms in a set B of office buildings. His employees will get the ordered groceries for everyone and stash them in a special room in some of those buildings for the customers to then pick up.

To save costs, Chris will not rent a room (and equip it with a refrigerator...) for each $b \in B$, but wants to do so for as few as possible.

For every pair (b_1, b_2) from B, these two might or might not be connected by a street. If there is a street connecting two buildings, Chris assumes all people working on that street can easily pick up their things in both buildings b_1 and b_2.

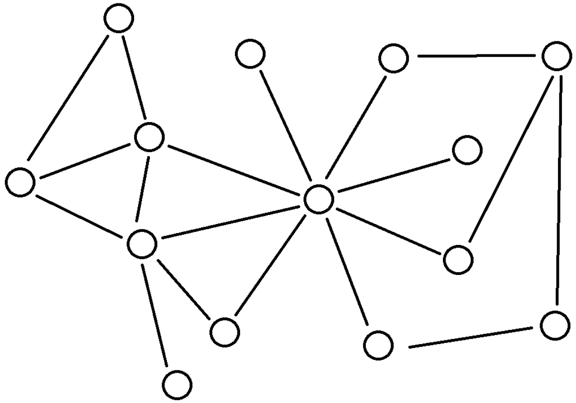
We want to find a good approximation algorithm.

Modeling the Problem

Each street is represented by an edge (b_1, b_2) in the graph G on vertex set B, meaning that people between (or in) building b_1 and b_2 can pick up their groceries easily at either b_1 or b_2.

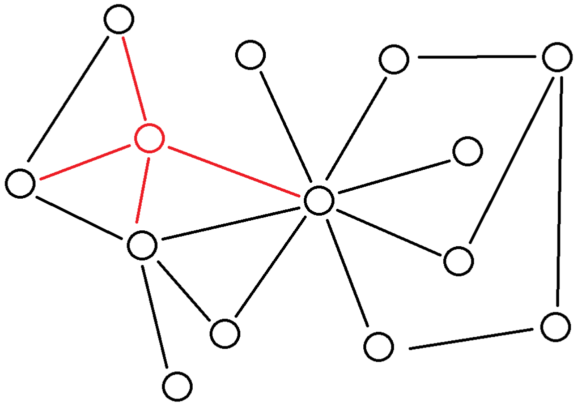
Modeling the Problem

Each street is represented by an edge (b_1, b_2) in the graph G on vertex set B, meaning that people between (or in) building b_1 and b_2 can pick up their groceries easily at either b_1 or b_2.



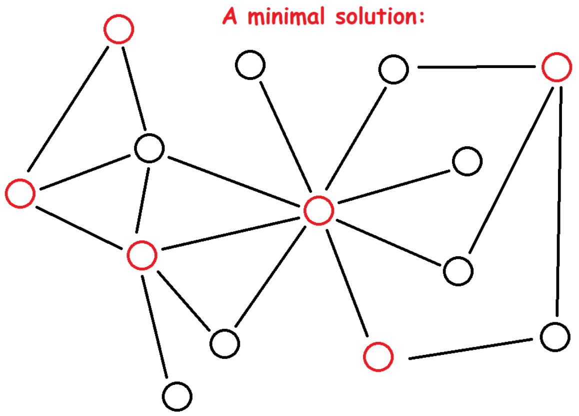
Modeling the Problem

Each street is represented by an edge (b_1, b_2) in the graph G on vertex set B, meaning that people between (or in) building b_1 and b_2 can pick up their groceries easily at either b_1 or b_2.



Modeling the Problem

Each street is represented by an edge (b_1, b_2) in the graph G on vertex set B, meaning that people between (or in) building b_1 and b_2 can pick up their groceries easily at either b_1 or b_2.

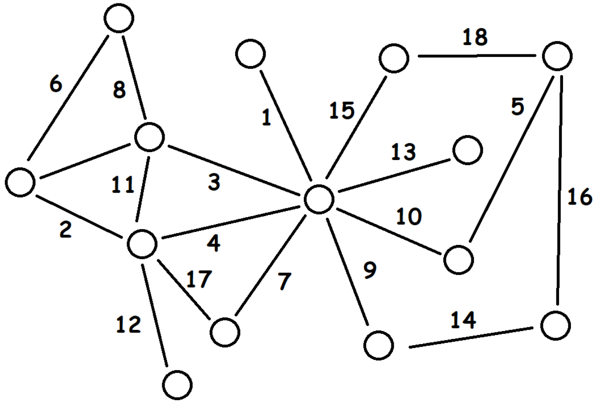


Algorithm Idea

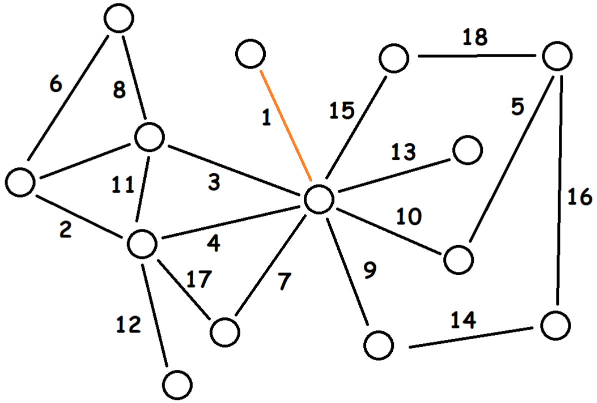
Chris has the following idea for a randomized algorithm computing a good subset $B' \subseteq B$ of buildings to rent a room in, such that all customers can get their groceries:

- Fix some order $e_1, e_2, \ldots, e_m$ of all edges in the edge set E of G, and set $B' = \emptyset$.
- Add to B' all isolated vertices, i.e. the ones without any incident edges.
- For every edge $e_1, e_2, \ldots,$ check if one of its endpoints is already contained in B'.
  If not, flip a fair coin deciding which of the endpoints to choose, and add this endpoint to B'.

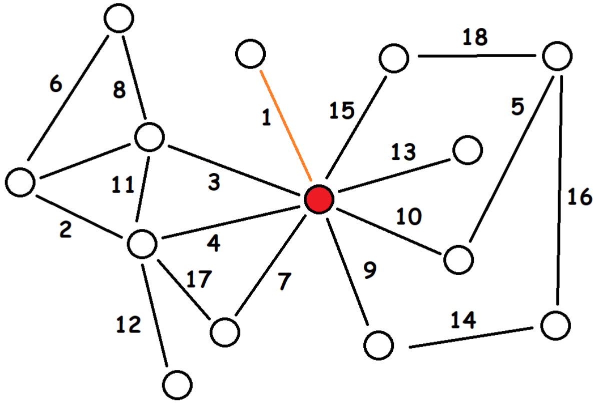
Visualization: Algorithm



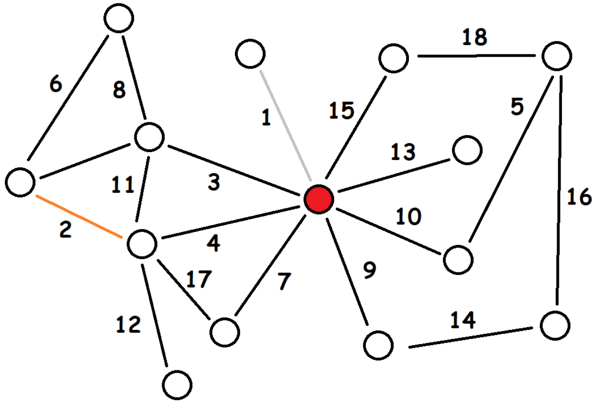
Visualization: Algorithm



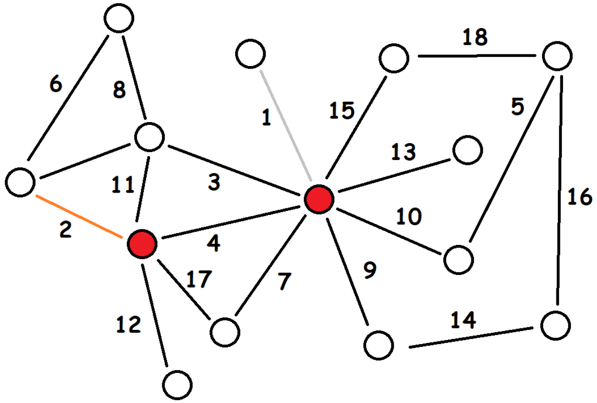
Visualization: Algorithm



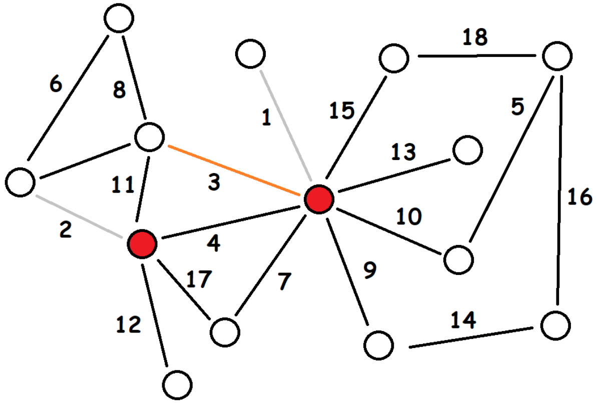
Visualization: Algorithm



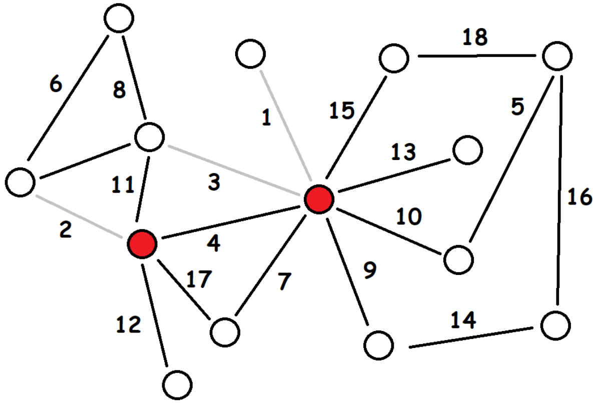
Visualization: Algorithm



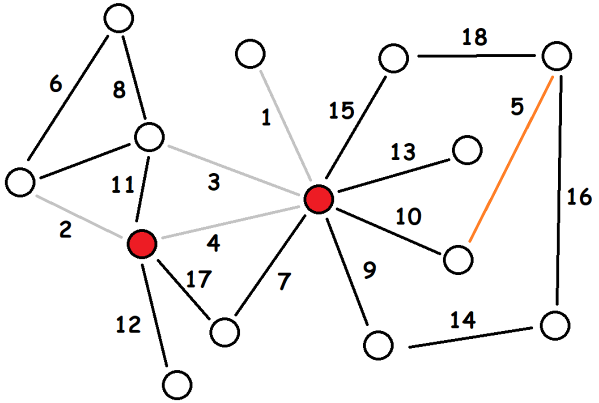
Visualization: Algorithm



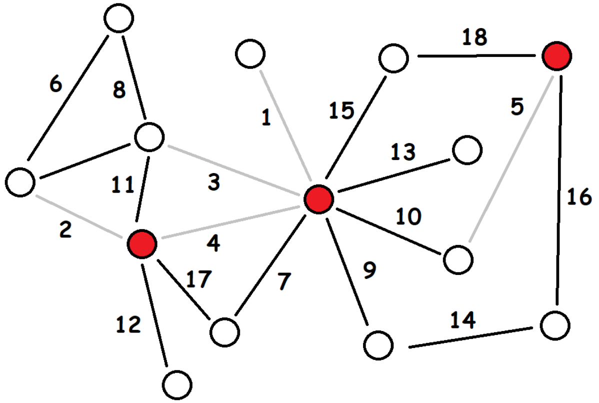
Visualization: Algorithm



Visualization: Algorithm



Visualization: Algorithm



Gameplan

a) We will show that in expectation, this algorithm will output a feasible solution where Chris has to rent at most twice as many rooms as in the optimal one.

b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.

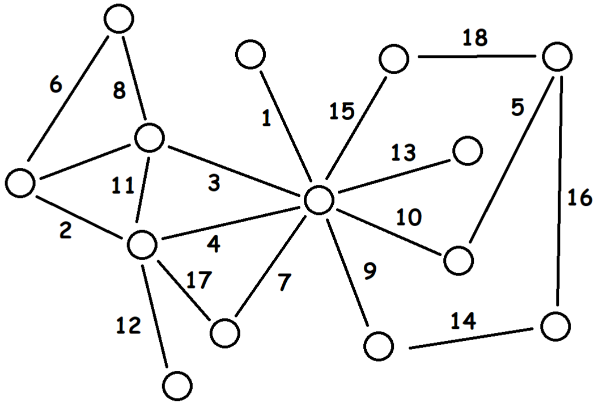
Proof of 2-Approximation

Let $v_1, \dots, v_k$ be the vertices in an optimal solution OPT, but without the isolated ones (these are always in both OPT and B'). Fix some $v \in OPT$. The algorithm will go through all edges incident to v_i in a fixed order (and in between, possibly also some other edges that are not incident to v). Let w.l.o.g. $E_v = \{e_1, \dots, e_l\}$ be all the edges incident to v in the order the algorithm considers them in.

We would like to count how many vertices are added to B' because of edges in E_v.

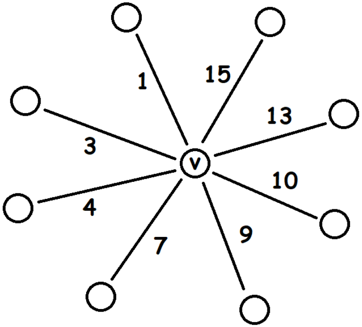
Proof of 2-Approximation

Recall: $E_v = \{e_1, \ldots, e_l\}$. So we look only at the following situation:



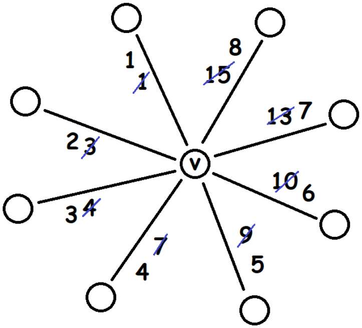
Proof of 2-Approximation

Recall: $E_v = \{e_1, \ldots, e_l\}$. So we look only at the following situation:



Proof of 2-Approximation

Recall: $E_v = \{e_1, \ldots, e_l\}$. So we look only at the following situation:

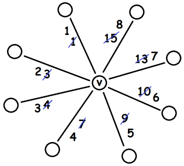


Proof of 2-Approximation

The algorithm will first look at e_1, and either do nothing (because an endpoint of it has already been chosen), or flip a coin. In the first case, no vertex is added because of e_1. In the second case, one vertex is added. If this vertex is v, no other vertex will ever be added again because of any edge in E_v (because v is an endpoint of them all). This happens with probability $1/2$.

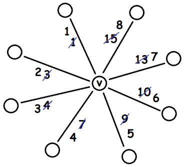
Proof of 2-Approximation

The algorithm will first look at e_1, and either do nothing (because an endpoint of it has already been chosen), or flip a coin. In the first case, no vertex is added because of e_1. In the second case, one vertex is added. If this vertex is v, no other vertex will ever be added again because of any edge in E_v (because v is an endpoint of them all). This happens with probability $1/2$.



Proof of 2-Approximation

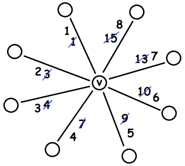
The algorithm will first look at e_1, and either do nothing (because an endpoint of it has already been chosen), or flip a coin. In the first case, no vertex is added because of e_1. In the second case, one vertex is added. If this vertex is v, no other vertex will ever be added again because of any edge in E_v (because v is an endpoint of them all). This happens with probability $1/2$.



All in all, we can depict the expected number n_v of vertices added when considering an edge from E_v as follows:

Proof of 2-Approximation

The algorithm will first look at e_1, and either do nothing (because an endpoint of it has already been chosen), or flip a coin. In the first case, no vertex is added because of e_1. In the second case, one vertex is added. If this vertex is v, no other vertex will ever be added again because of any edge in E_v (because v is an endpoint of them all). This happens with probability $1/2$.



All in all, we can depict the expected number n_v of vertices added when considering an edge from E_v as follows:

$$\mathbb{E}\left[n_v\right] \leq \sum_{i \in \{1,\ldots,l\}} \mathbb{P}[v \text{ not chosen before considering } e_i] \cdot 1$$

Proof of 2-Approximation

Expected number n_v added while considering edges from E_v:

$$\mathbb{E}\left[n_v\right] \leq \sum_{i \in \{1,\ldots,l\}} \mathbb{P}[v \text{ not chosen before considering } e_i] \cdot 1$$

Note that if any e_i is not considered by the algorithm because its other endpoint (not v) has been added in the meantime, this just shortens the sum since we can ignore this edge completely - for example, if this holds for e_1, then e_2 will just take its place. However, the above still poses an upper bound.

Proof of 2-Approximation

Expected number n_v added while considering edges from E_v:

$$\mathbb{E}[n_v] \leq \sum_{i \in \{1,\ldots,l\}} \mathbb{P}[v \text{ not chosen before considering } e_i] \cdot 1$$

Now we can rewrite

$$\mathbb{E}[n_v] \leq \underbrace{1 \cdot 1}_{v \text{ not picked!}} + \underbrace{\frac{1}{2} \cdot 1}_{v \text{ picked for } e_1 \text{w. pr. } \frac{1}{2}} + \underbrace{\frac{1}{4} \cdot 1}_{v \text{ picked for } e_1 \text{ or } e_2 \text{ w. pr. } \frac{1}{4}} \ldots + \frac{1}{2^{l-1}} \cdot 1$$

This yields

$$\mathbb{E}[n_v] \leq \sum_{i=1}^{l} \frac{1}{2^{i-1}} \leq 2$$

Therefore, for every E_v and $v \in OPT$ (v not isolated), the expected number of added edges is at most two. This implies, with $E = \bigcup_{v \in OPT} E_v$:

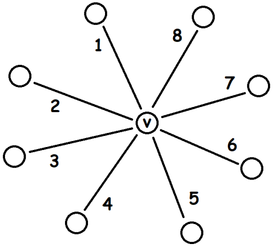
$$|OPT| \geq \frac{1}{2}|B'|$$

Proof of Worst-Case Performance

Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.

Proof of Worst-Case Performance

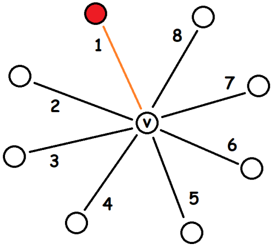
Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.



Consider the star-shaped graph with center v and outer vertices $v_1, \ldots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \ldots, \lceil c \rceil\}\}$.

Proof of Worst-Case Performance

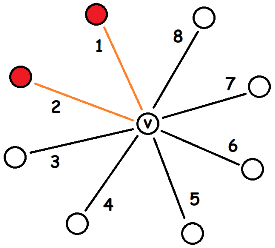
Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.



Consider the star-shaped graph with center v and outer vertices $v_1, \dots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \dots, \lceil c \rceil\}\}$.

Proof of Worst-Case Performance

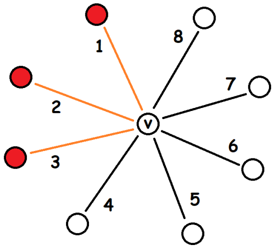
Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.



Consider the star-shaped graph with center v and outer vertices $v_1, \ldots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \ldots, \lceil c \rceil\}\}$.

Proof of Worst-Case Performance

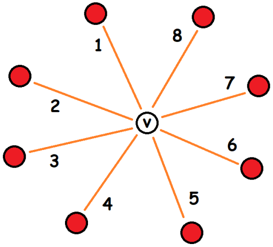
Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.



Consider the star-shaped graph with center v and outer vertices $v_1, \ldots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \ldots, \lceil c \rceil\}\}$.

Proof of Worst-Case Performance

Gameplan Reminder: b) Randomized algorithms usually perform badly in the worst case-approximation. We prove that indeed, for every constant $c \geq 1$, the algorithm might produce a B' with $|B'| \geq c|OPT|$.



Consider the star-shaped graph with center v and outer vertices $v_1, \ldots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \ldots, \lceil c \rceil\}\}$.

Proof of Worst-Case Performance

Consider the star-shaped graph with center v and outer vertices $v_1, \ldots, v_{\lceil c \rceil}$, i.e. the edge set is given by $E = \{(v, v_i) | i \in \{1, \ldots, \lceil c \rceil\}\}$.

With probability $(\frac{1}{2})^{\lceil c \rceil} > 0$, the algorithm never adds v, but all of the other vertices, while the optimal solution consists only of v.

Part 2
Approximation and LPs

Reminder: LPs and ILPs

ILP (Integer Linear Program)

- great to model any type of *discrete* problem with constraints
- of the form $\max_{x \in \mathcal{Z}^n} \left\{c^T x | Ax \leq b,\, x \geq 0\right\}$
- Usually not solvable in polynomial time!

LP (Linear Program)

- models *continuous/fractional* problems
- of the form $\max \left\{c^T x | Ax \leq b,\, x \geq 0\right\}$
- Solvable in polynomial time! (Ellipsoid method)

Reminder: LPs and ILPs

ILP (Integer Linear Program)

- great to model any type of *discrete* problem with constraints
- of the form $\max_{x \in \mathcal{Z}^n} \left\{ c^T x | Ax \leq b, \, x \geq 0 \right\}$
- Usually not solvable in polynomial time!

LP (Linear Program)

- models *continuous/fractional* problems
- of the form $\max \left\{ c^T x | Ax \leq b, \, x \geq 0 \right\}$
- Solvable in polynomial time! (Ellipsoid method)

Problems in CS often discrete:

Reminder: LPs and ILPs

ILP (Integer Linear Program)

- great to model any type of *discrete* problem with constraints
- of the form $\max_{x \in \mathcal{Z}^n} \left\{ c^T x | Ax \leq b, \; x \geq 0 \right\}$
- Usually not solvable in polynomial time!

LP (Linear Program)

- models *continuous/fractional* problems
- of the form $\max \left\{ c^T x | Ax \leq b, \; x \geq 0 \right\}$
- Solvable in polynomial time! (Ellipsoid method)

Problems in CS often discrete:
ILPs are not practical, but LPs cannot fully capture them.

Reminder: LPs and ILPs

ILP (Integer Linear Program)

- great to model any type of *discrete* problem with constraints
- of the form $\max_{x \in \mathcal{Z}^n} \left\{c^T x | Ax \leq b, \, x \geq 0\right\}$
- Usually not solvable in polynomial time!

LP (Linear Program)

- models *continuous/fractional* problems
- of the form $\max \left\{c^T x | Ax \leq b, \, x \geq 0\right\}$
- Solvable in polynomial time! (Ellipsoid method)

Problems in CS often discrete:
ILPs are not practical, but LPs cannot fully capture them.
Still: Often, LPs can be used to obtain *approximate* solutions!

Mikele's Wedding Problem

Mikele is very interested in fashion and owns a whole bunch of nice outfits, collectively forming the set O. A set F of his friends all want to borrow one of Mikele's outfits to go to a wedding. This should be doable, since $|F| \leq |O|$. However, due to different restrictions like size, body form, or adjustements that have to be made to an outfit to really look wedding-appropriate, there will be some extra effort $w(f, o) \geq 0$ required when any friend f wants to wear outfit o.

Goal: Dress everyone nicely with minimum effort!

Mikele's Wedding Problem: Recap

Friends F

Outfits O

$|F| \leq |O|$

Efforts $w(f, o) \geq 0$ for all $f \in F$, $o \in O$

Find an outfit for each friend, i.e. a set S of $|F|$ pairs s.t. $\sum_{(f,o)\in S} w(f, o)$ is minimized.

Steps to an Approximation

a) Model the problem as an integer linear program, and relax this to a corresponding LP.

b) Usually, the optimal solution to a problem's LP-relaxation is better than that of the original ILP. The factor by which both can differ is called the *integrality gap*. Find out the integrality gap g for Mikele's problem, and state a proof.

c) Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

a) Finding an ILP

- Start with the objective: We want minimum weight for assigned outfits
- Whenever there's a *yes/no choice*: make a variable! (here: indicating whether (f, o) is in the solution.)
- Make sure no double assignment happens
- Make sure everyone has something to wear

$$min \sum_{f \in F, o \in O} w(f, o) x_{f,o} \qquad \text{s.t.}$$

a) Finding an ILP

- Start with the objective: We want minimum weight for assigned outfits
- Whenever there's a *yes/no choice*: make a variable! (here: indicating whether (f, o) is in the solution.)
- Make sure no double assignment happens
- Make sure everyone has something to wear

$$min \sum_{f \in F, o \in O} w(f, o) x_{f,o} \qquad \text{s.t.}$$

$$\sum_{f \in F} x_{f,o} \leq 1 \qquad \forall o \in O,$$

a) Finding an ILP

- Start with the objective: We want minimum weight for assigned outfits
- Whenever there's a *yes/no choice*: make a variable! (here: indicating whether (f, o) is in the solution.)
- Make sure no double assignment happens
- Make sure everyone has something to wear

$$
\begin{aligned}
&min \sum_{f \in F, o \in O} w(f, o) x_{f,o} && \text{s.t.} \\
&\sum_{f \in F} x_{f,o} \leq 1 && \forall o \in O, \\
&\sum_{i \in O} x_{f,o} = 1 && \forall f \in F,
\end{aligned}
$$

a) Finding an ILP

- Start with the objective: We want minimum weight for assigned outfits
- Whenever there's a *yes/no choice*: make a variable! (here: indicating whether (f, o) is in the solution.)
- Make sure no double assignment happens
- Make sure everyone has something to wear

$$
\begin{aligned}
&min \sum_{f \in F, o \in O} w(f,o) x_{f,o} && \text{s.t.} \\
&\sum_{f \in F} x_{f,o} \leq 1 && \forall o \in O, \\
&\sum_{i \in O} x_{f,o} = 1 && \forall f \in F, \\
&x_{f,o} \in \{0,1\} && \forall f \in F, o \in O.
\end{aligned}
$$

b) Stating the LP

Recall: ILPs generally not solvable in polytime!
Simply remove the integer requirement.

$$
\begin{aligned}
&min \sum_{f\in F, o\in O} w(f,o)x_{f,o} && \text{s.t.}\\
&\sum_{f\in F} x_{f,o} \leq 1 && \forall o \in O,\\
&\sum_{i\in O} x_{f,o} = 1 && \forall f \in F,\\
&x_{f,o} \geq 0 && \forall f \in F, o \in O.
\end{aligned}
$$

b) Find the Integrality Gap

Recall: The integrality gap is the (max.) factor by which the optimal LP solution can be *better* than the ILP optimum.

Strategy: LP solution has **fractional** assignments! If the weight of such a fractional solution is better than that of **any** integral one, it must be because splitting outfits between people somehow reduces the cost.

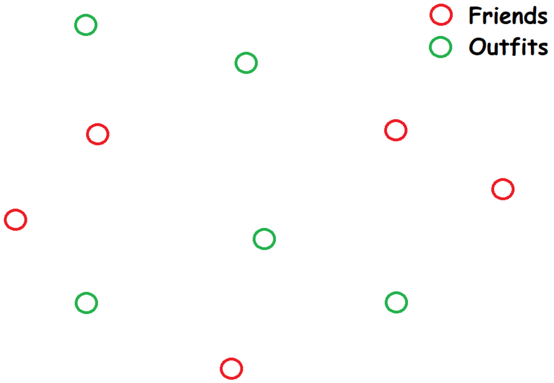
- seems unlikely in our problem: how? But let's see.
- look at such a fractional solution and try to see why the same isn't possible with an integral one...

Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take.

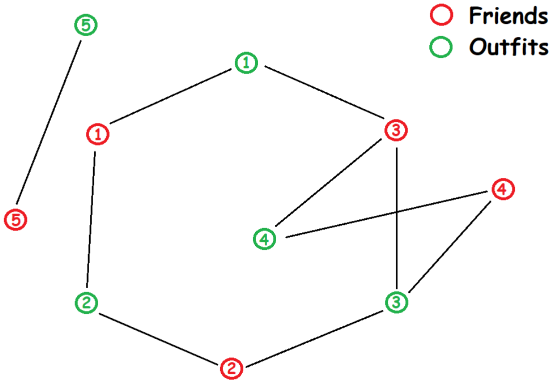
Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take. This sounds an awful lot like a graph structure! Indeed:



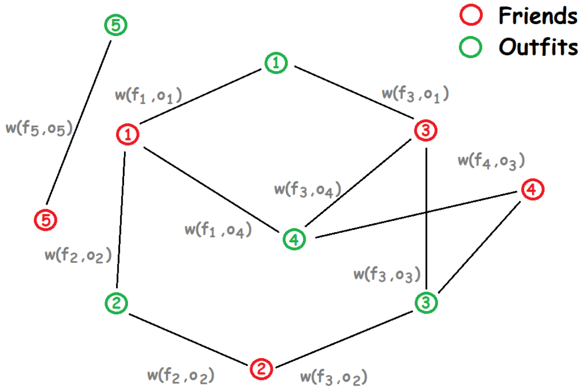
Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take. This sounds an awful lot like a graph structure! Indeed:



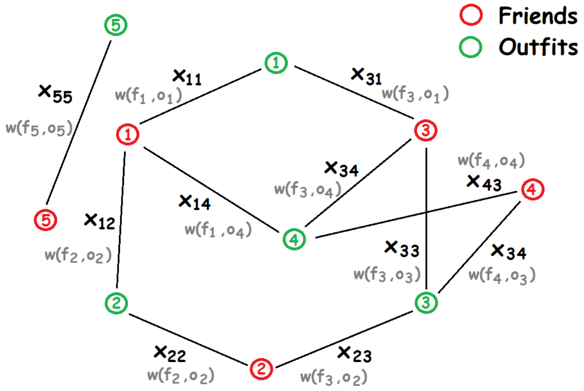
Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take. This sounds an awful lot like a graph structure! Indeed:



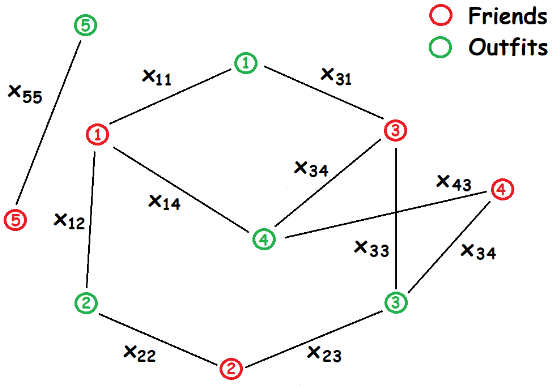
Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take. This sounds an awful lot like a graph structure! Indeed:



Visualization of the Solution

Note: The solution is given by the values of $x_{f,o}$, telling us, in case of the ILP, *how much of* any couple (f, o) we take. This sounds an awful lot like a graph structure! Indeed:

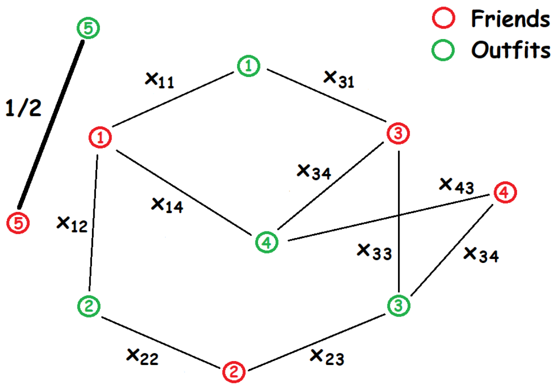


Analysis

Assumption: No integral solution is as good as this one!
Let's start looking at a path...

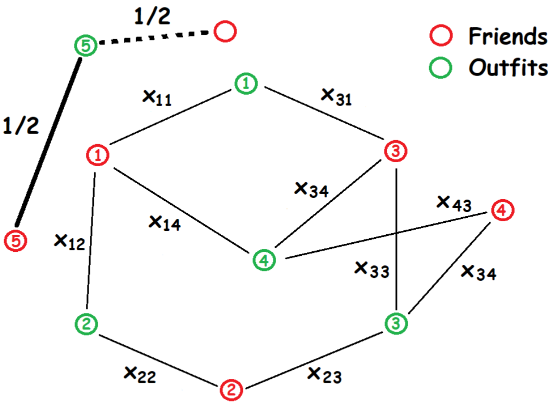
Analysis

Assumption: No integral solution is as good as this one!
Let's start looking at a path...



Analysis

Assumption: No integral solution is as good as this one!
Let's start looking at a path...

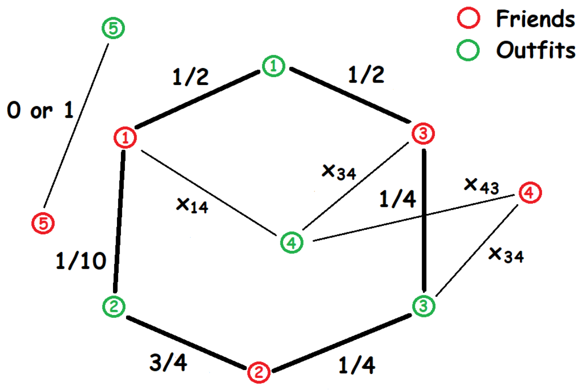


Analysis

Assumption: No integral solution is as good as this one!
Now for a cycle...

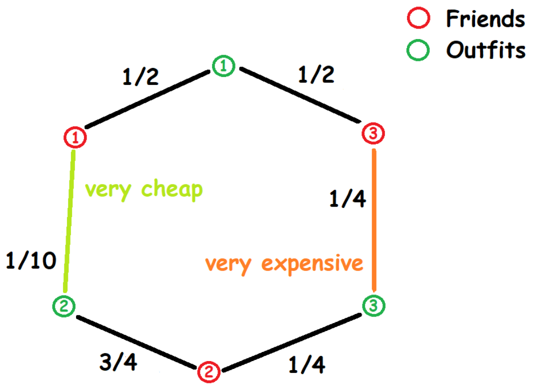
Analysis

Assumption: No integral solution is as good as this one!
Now for a cycle...



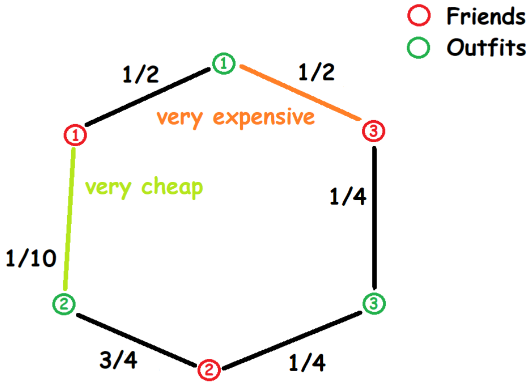
Analysis

Assumption: No integral solution is as good as this one!
Now for a cycle...



Analysis

Assumption: No integral solution is as good as this one!
Now for a cycle...



Analysis

Assume that the LP-solution x is better than that of the ILP, x^I. Then it follows that there exists no integral solution with the same value as x which is feasible.

However, consider a maximal path (or cycle) in the graph defined by x where every $x_{f,o}$ is non-integer. We prove it is feasible to shift ϵ weight into any direction, when ϵ is chosen to be at most the minimum distance between any $x_{f,o}$ on the path and the nearest integer, and one of these shift-directions does not increase the overall weight.

Denote the path P as $(o_1, f_1), (f_1, o_2), (o_2, f_2), \ldots, (f_{k-1}, o_k)$. (If it is a cycle, we will have $o_1 = o_k$)

Analysis

Now, consider the two modified solutions, consisting of the original x, together with the following adjustments to edges on the path:

A: $x_{o_1,f_1}+\epsilon,\ x_{f_1,o_2}-\epsilon,\ x_{o_2,f_2}+\epsilon,\ldots,x_{f_{k-1},o_k}-\epsilon$

B: $x_{o_1,f_1}-\epsilon,\ x_{f_1,o_2}+\epsilon,\ x_{o_2,f_2}-\epsilon,\ldots,x_{f_{k-1},o_k}+\epsilon$

We have that one of them is at least as good as x, i.e. $min\{w(A1),w(A2)\}\leq w(x)$, since

$$w(A)=w(x)+\epsilon w(A^+)-\epsilon w(A^-) \quad \text{and} \quad w(B)=w(x)+\epsilon w(A^-)-\epsilon w(A^+)$$

where we denote the enlarged x_{fo} in A as A^+, and analogously for the other case. Also, both A and B are feasible whenever ϵ is small enough.

Analysis

Let us choose ϵ to be the maximum-possible one that results in a feasible assignment for the better solution from $\{A, B\}$. Then, the new solution is

- feasible,
- no worse than x, and
- does not contain the non-integer path P any more since at least one of its edges is now integer.

We repeat this process until the whole solution is integer. Because in every step, there is no loss in weight, the resulting integer assignment has the same weight as x, proving an integrality gap of $g = 1$ (i.e, there is **no gap**!).

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The algorithm will do exactly what we described before:

- Starting from any non-integer x_{fo} in the current solution, find a maximum non-integer path P
- Find the shifting direction that is better in terms of weight by computing both alternating edge-sums
- Shift into this direction until one of the x_{fo} in P becomes 0 or 1

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The algorithm will do exactly what we described before:

- Starting from any non-integer x_{fo} in the current solution, find a maximum non-integer path P (doable in time $O(|F|)$)
- Find the shifting direction that is better in terms of weight by computing both alternating edge-sums
- Shift into this direction until one of the x_{fo} in P becomes 0 or 1

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The algorithm will do exactly what we described before:

- Starting from any non-integer x_{fo} in the current solution, find a maximum non-integer path P (doable in time $O(|F|)$)
- Find the shifting direction that is better in terms of weight by computing both alternating edge-sums ($O(|F|)$)
- Shift into this direction until one of the x_{fo} in P becomes 0 or 1

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The algorithm will do exactly what we described before:

- Starting from any non-integer x_{fo} in the current solution, find a maximum non-integer path P (doable in time $O(|F|)$)
- Find the shifting direction that is better in terms of weight by computing both alternating edge-sums ($O(|F|)$)
- Shift into this direction until one of the x_{fo} in P becomes 0 or 1 ($O(|F|)$ again)

The Algorithm

Recall question c: Give a polynomial-time algorithm that, from any given optimal LP-solution, computes a feasible integer assignment that approximates OPT up to a factor of g.

The algorithm will do exactly what we described before:

- Starting from any non-integer x_{fo} in the current solution, find a maximum non-integer path P (doable in time $O(|F|)$)
- Find the shifting direction that is better in terms of weight by computing both alternating edge-sums ($O(|F|)$)
- Shift into this direction until one of the x_{fo} in P becomes 0 or 1 ($O(|F|)$ again)

Given the fact that every such consideration of a path will make at least one entry of x integer, and integer ones are never considered in any path again, we will need no more than $O(|F||O|)$ such steps, which is clearly polynomial.