理论计算机科学导论

上海交通大学
陶表帅
2022年10月24日

前情提要

回顾&思考

- 判定问题$f:\{0,1\}^* \to \{0,1\}$
  - 0代表no，1代表yes
  - 例子：停机问题、SAT问题、独立集问题
- 计算机能做什么？
- 计算机能否计算f？
  - 有些问题不可被计算：停机问题
- 如果能的话，能多快计算f？
  - P vs NP

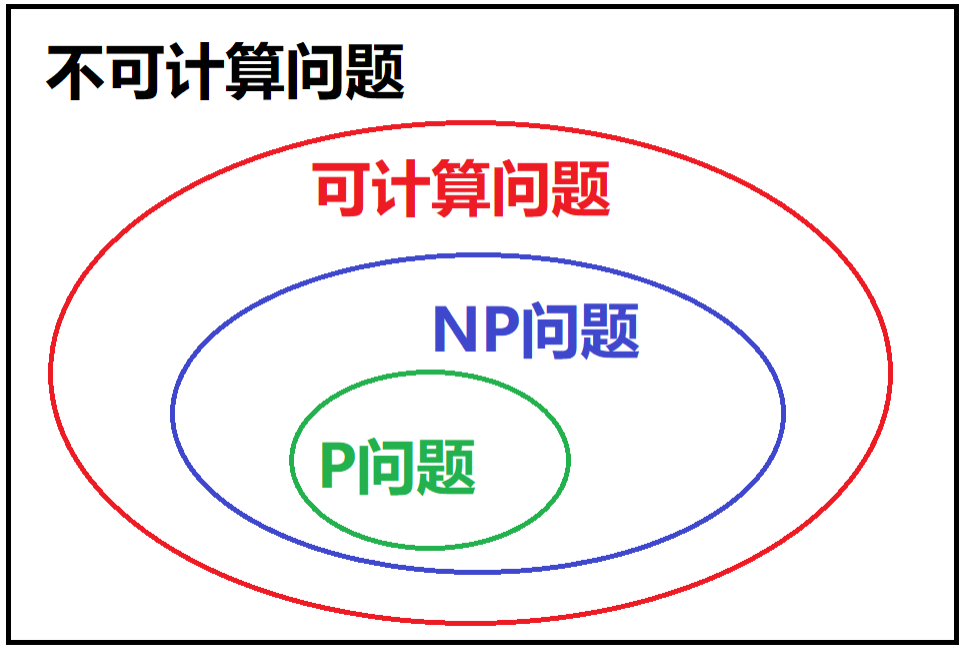
按照“难度”划分判定问题

- “**最难**”：根本无法计算
- “**其次**”：可以被计算，但时间可能很长……
  - NP：（确定型）图灵机能在多项式时间内**验证**
  - NP-complete：NP中的“最难的”问题
- “**最易**”：可以被快速计算
  - P：多项式时间内可以被**判定**
  - P是否等于NP?

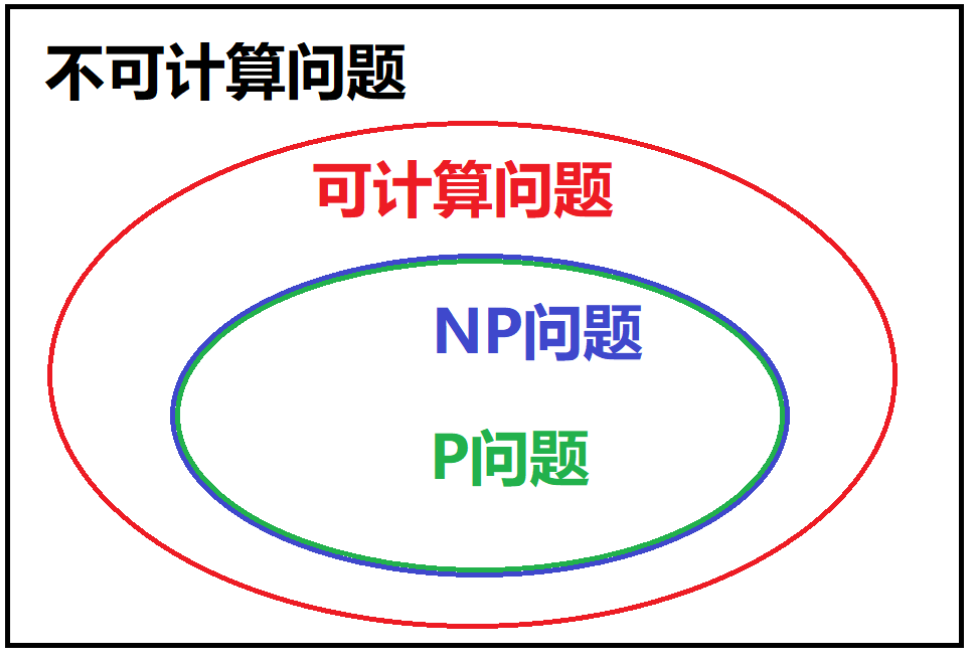
P和NP

- 对一个判定问题 f，什么时候$f \in NP$呢？
- 存在一个多项式时间的**图灵机（验证器）** g，输入是x和y。
  - x：原问题的输入。
  - y：某个多项式规模的选项（$|x|$的多项式）。
- 该**图灵机（验证器）**能做到
  - $f(x) = yes$时，存在一个y，使得 $g(x,y) = yes$。
  - $f(x) = no$时，对于所有的y，$g(x,y) = no$。
- 对比 $f \in P$ 的定义
  - 存在一个多项式时间的**图灵机（判定器）** g，输入是x
  - $f(x) = yes$时，$g(x) = yes$。
  - $f(x) = no$时，$g(x) = no$。

按照“难度”划分判定问题



P ≠ NP的情况



P = NP的情况

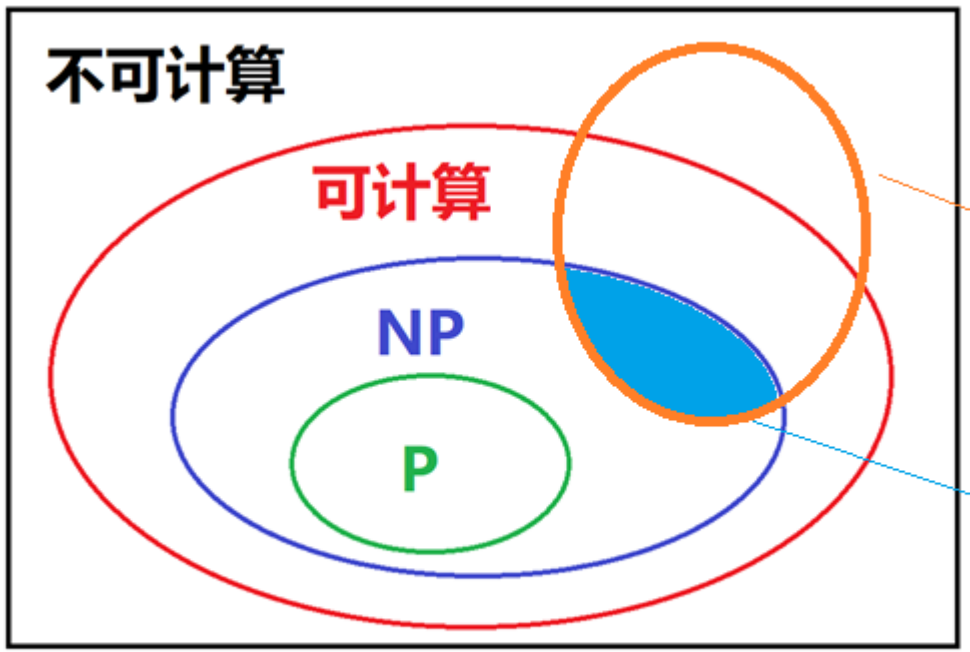
NP-hard和NP-complete

- NP-hard问题
  - 我们称一个问题f是NP-hard，如果对于**所有**NP问题g，$g \leq_? f$。
  - 在这门课中，我们可以用$\leq_T^P$定义。
  - 在复杂度理论中，常见的是使用$\leq_k$定义。
- NP-complete问题
  - 如果f是NP-hard，
  - 并且$f \in \mathbf{NP}$，
  - 我们称f是NP-complete。

NP-complete问题

- NP里面“最难”的问题
- [Cook-Levin定理] SAT 问题是 NP-complete的。
- 上节课：
  - 3SAT问题是NP-complete的
  - 独立集问题（independent set）是NP-complete的
  - 还有许多……
- 这些NP-complete问题都是NP里面“最难” 的问题，他们两两之间都“一样难”
- 如果一个NP-complete的问题存在**多项式时间复杂度**的判定算法，那么所有NP问题都存在，则我们有**P = NP**。

P ≠ NP的情况



NP-hard问题

NP-complete问题

本节课

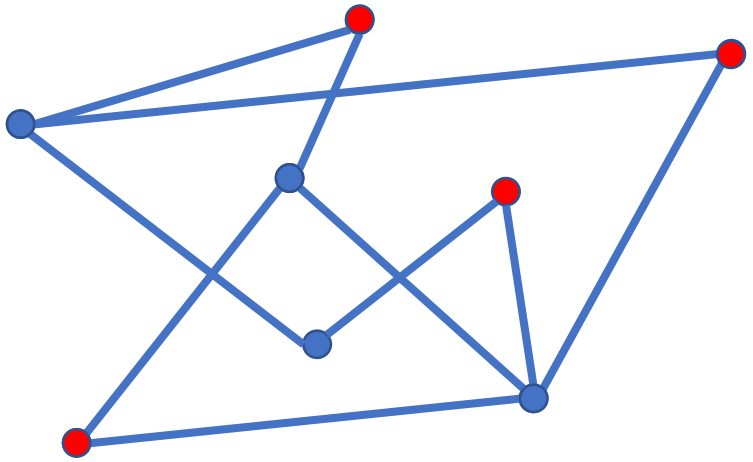
优化问题、近似算法

这节课：

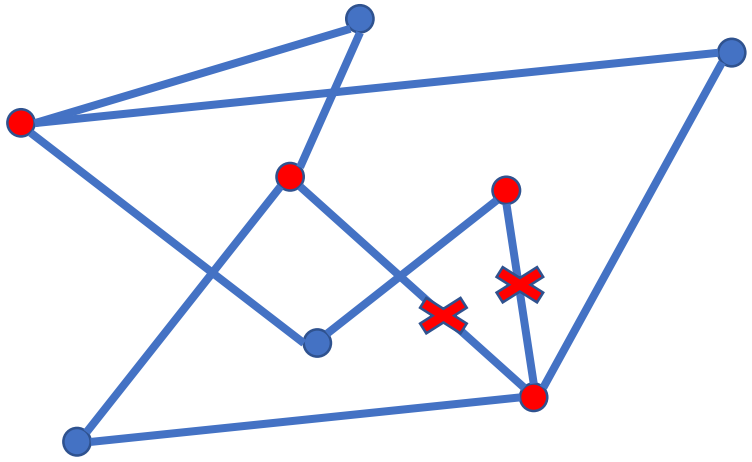
- 优化问题
  - 最大化问题、最小化问题
- NP-hard的优化问题
- 如何处理NP-hard优化问题？
- 近似算法

优化问题 vs 判定问题：独立集

- 判定问题(decision problem)：
  - 输入$G = (V, E)$以及$k \in \mathbb{Z}^+$；输出该图是否存在一个k个点的独立集合。
- 优化问题(optimization problem)：
  - 输入$G = (V, E)$；输出该图最大的独立集合。



独立集



非独立集

优化问题 vs 判定问题：3SAT

- 判定问题(decision problem)：
  - 输入一个3-CNF布尔表达式；输出是否有一种$x_1 \sim x_n$的赋值，使得这个CNF最后答案是true。
- 优化问题(optimization problem)：
  - 输入一个3-CNF布尔表达式；输出一组$x_1 \sim x_n$的赋值，使得尽可能多的项的值为true。

$$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_5)$$

优化问题 vs 判定问题：上升子串

- 判定问题(decision problem)：
  - 给你一串数字，问你其中是否存在长度为k的上升子串。
- 优化问题(optimization problem)：
  - 给你一串数字，输出最长的上升子串。

NP-hard优化问题

- 很多**判定问题**可以被描述成**优化问题**
- **优化问题**往往至少不比**判定问题**容易
- **优化问题**的答案比**判定问题**的答案更有信息量
  - 【独立集】如果知道最大的独立集有多大，肯定能知道是否存在大小为k的独立集
  - 【3SAT】如果知道ϕ最多能被满足多少项，肯定能知道ϕ能否被满足
  - 【上升子串】如果知道最长上升子串的长度，肯定能知道是否存在长度为k的上升子串。
- 故而，一个NP-hard的**判定问题**所对应的**优化问题**往往也是“难的”

NP-hard优化问题

- **【定义（不严谨）】**一个优化问题是**NP-hard**的，如果存在一个**多项式时间复杂度**算法解决该问题能推出**P = NP**。
- 【独立集】如果能有**多项式时间复杂度**算法找到最大的独立集，则该算法能被用于判定该图是否存在大小为k的独立集
  - 独立集判定 $\leq_T^P$ 最大独立集
- 【3SAT】如果能有多项式时间复杂度算法找到最多能满足多少项，则该算法能被用于判定ϕ能否被满足
  - 3SAT判定 $\leq_T^P$ 3SAT最大化满足项
- **最大独立集**和**最大3SAT（Max-3SAT）**都是NP-hard的优化问题！

近似算法

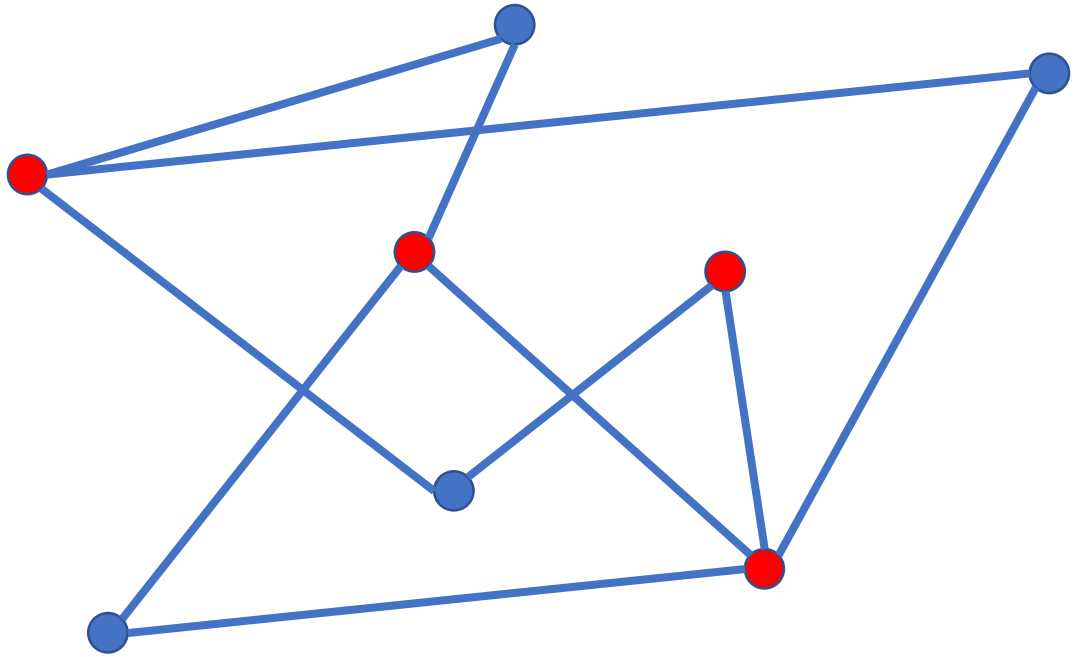
- 在P ≠ NP的假设下，NP-hard优化问题无法在**多项式时间**内被解决
- 如何处理NP-hard优化问题？
- 近似算法（approximation algorithm）
  - 输出“**近似**”最优解的算法
- 以独立集为例：倘若有一个算法，当一个图的最大独立集大小为K时，总能输出一个大小至少为$0.5K$的独立集，那么该算法被称作是一个0.5-近似算法

近似算法案例—

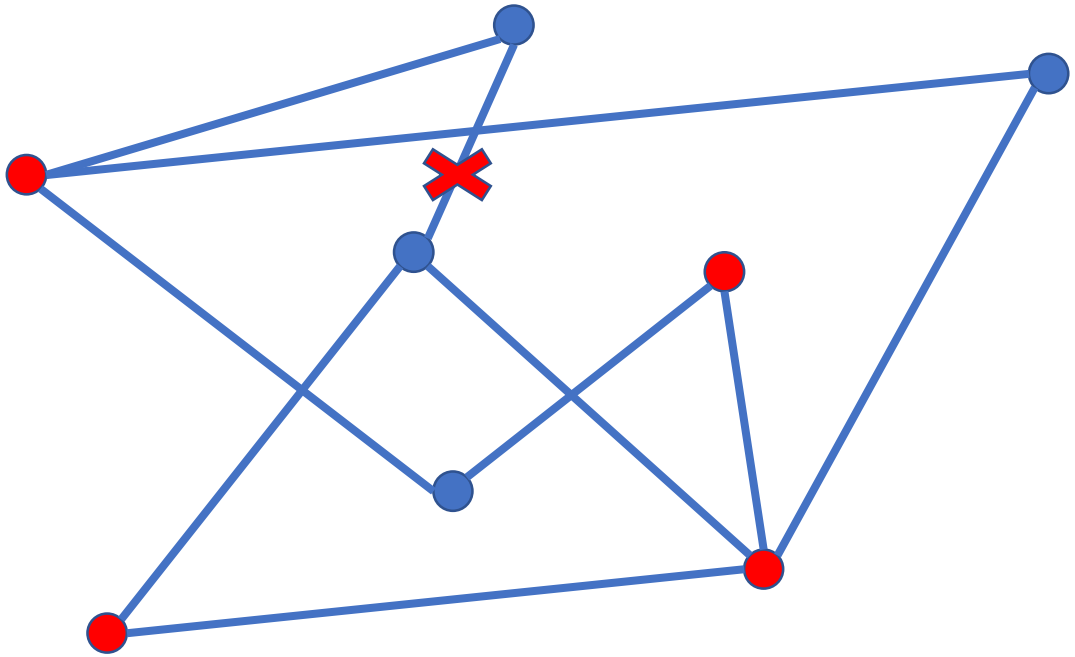
点集覆盖问题

点集覆盖问题（vertex cover）

- 给定一个**无向图**$G = (V, E)$，一个**顶点子集** $S \subseteq V$ 被称作一个点集覆盖（vertex cover）如果它包括了**每条边的至少一个端点**。



点集覆盖



不是点集覆盖

点集覆盖问题

- 判定问题

- 给定一个**无向图**$G = (V, E)$和一个**整数**k，输出该图是否存在一个大小为k的**点集覆盖**。

点集覆盖是NP-complete的

- **【定理】点集覆盖是NP-complete的。**
- **【证明】**（作业）

点集覆盖问题，优化版本

- 最大化点集覆盖?
- 最小化点集覆盖?

点集覆盖问题，优化版本

- **【定理】**最小化点集覆盖问题是NP-hard的。

预告

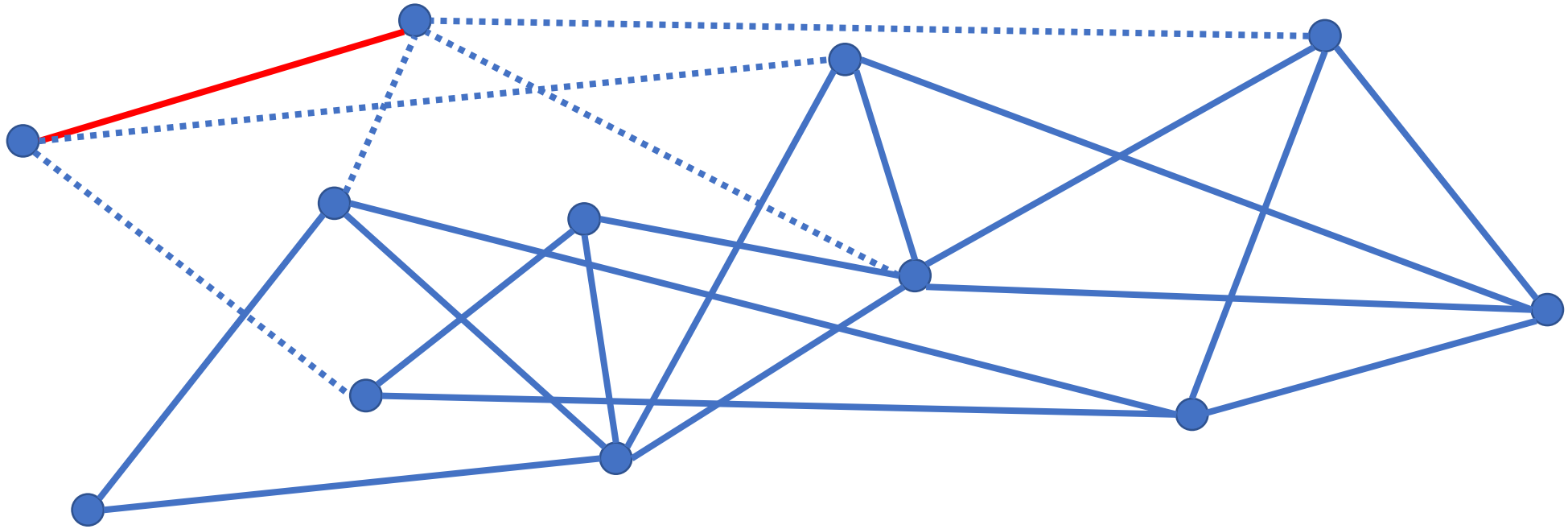
- 我们将给出一个点集覆盖的**2-近似算法**!
- 如果最小的点集覆盖大小为K，我们算法找到的点集覆盖最多用$2K$个点!

极大匹配（Maximal Matching）

- 给定一个**无向图**$G = (V, E)$，一个**匹配**（matching）是一个两两不共顶点的边集。
- 一个匹配M是**极大的**（maximal）当没有更多的边能加进M使得M仍然是一个匹配。
- 找一个极大匹配非常简单：一条边一条边加进来，直到加不了……

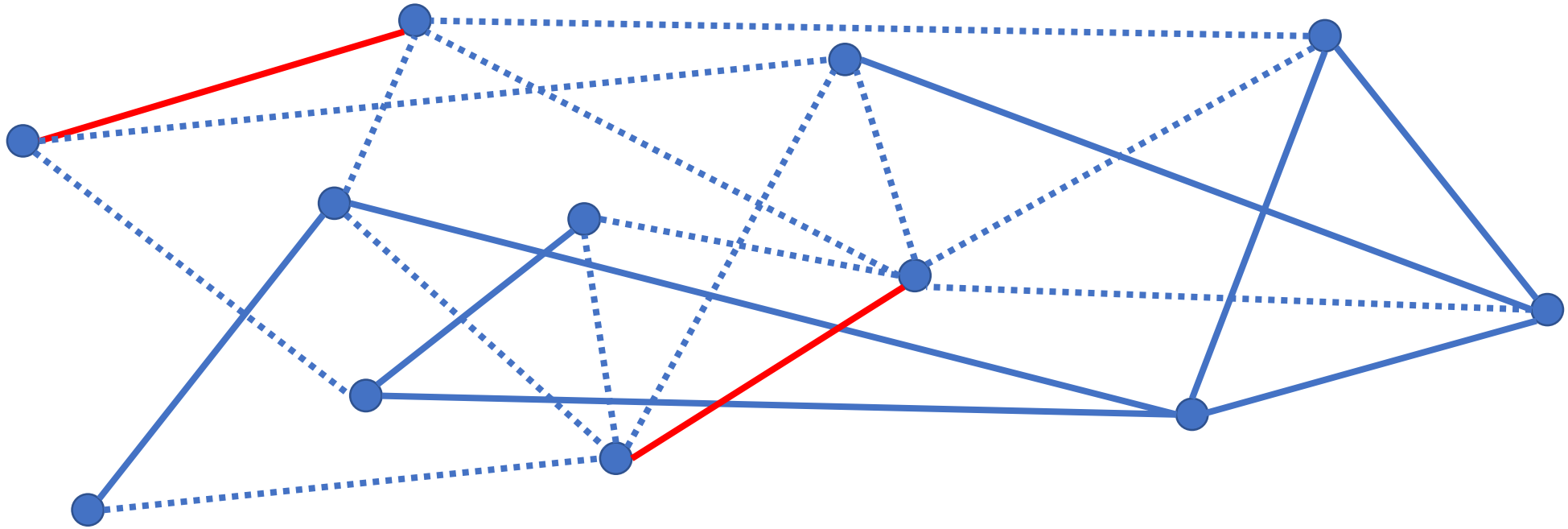
寻找极大匹配

- 一条边一条边加进来，直到加不了……



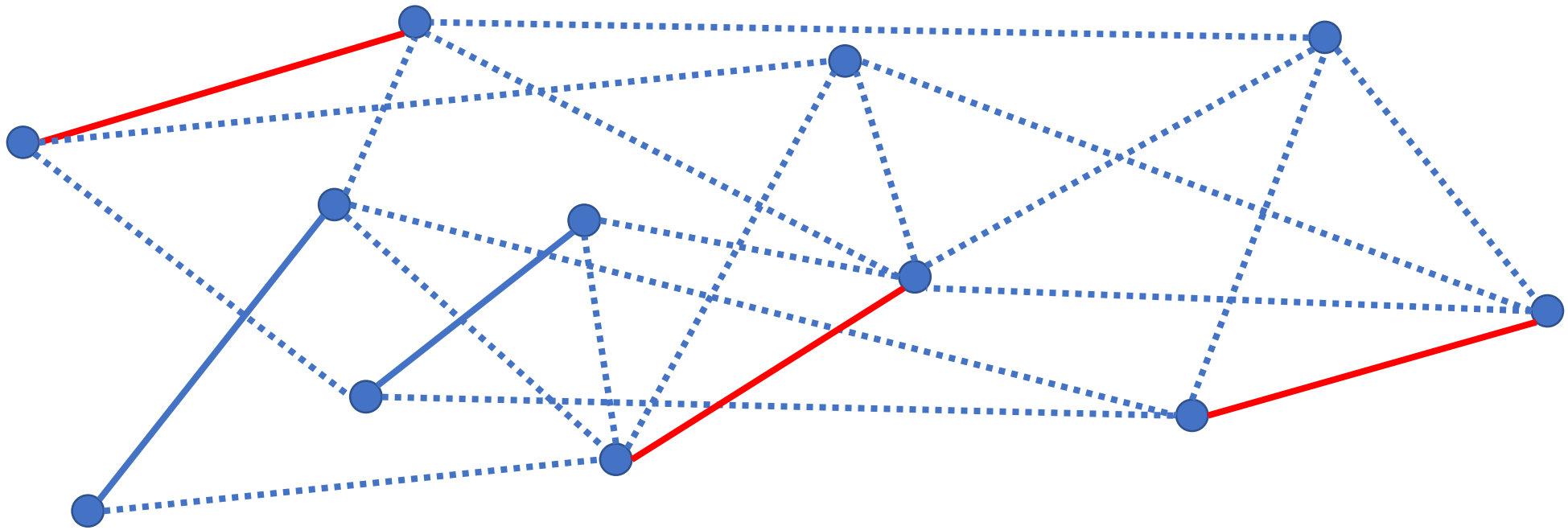
寻找极大匹配

- 一条边一条边加进来，直到加不了……



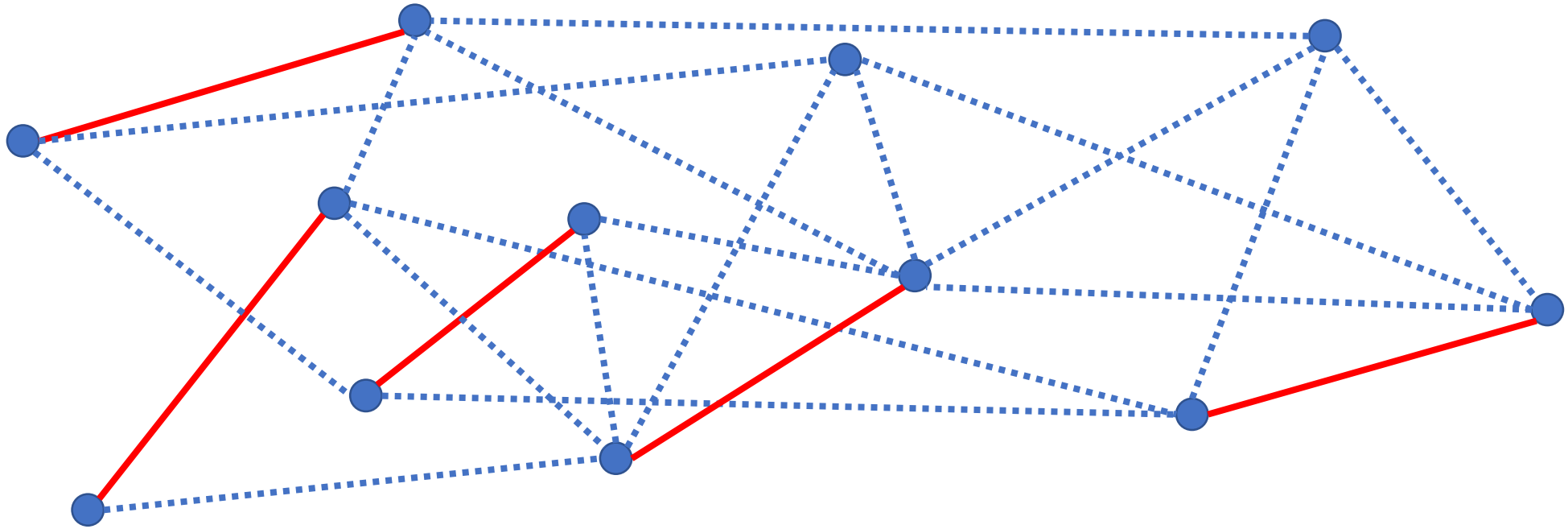
寻找极大匹配

- 一条边一条边加进来，直到加不了……



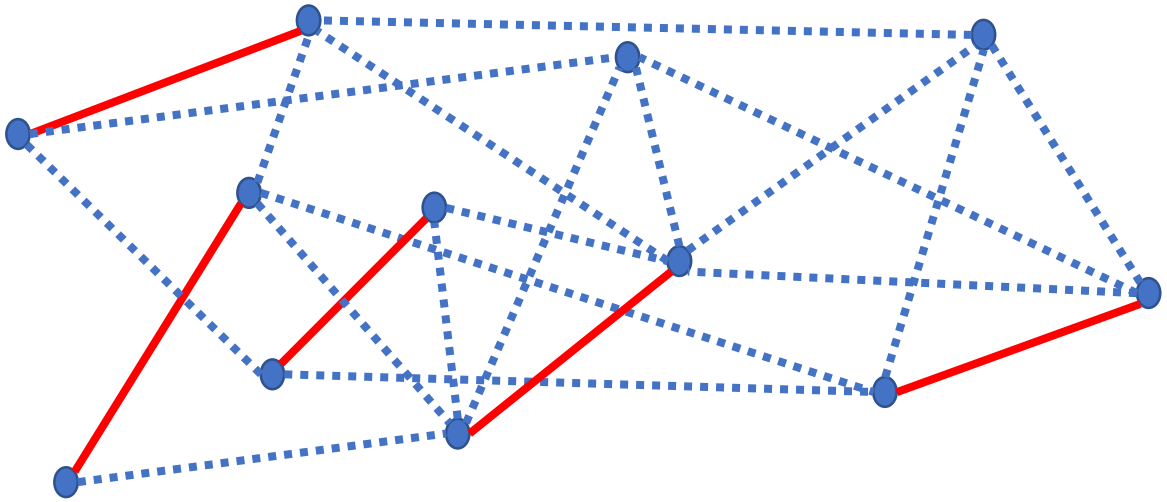
寻找极大匹配

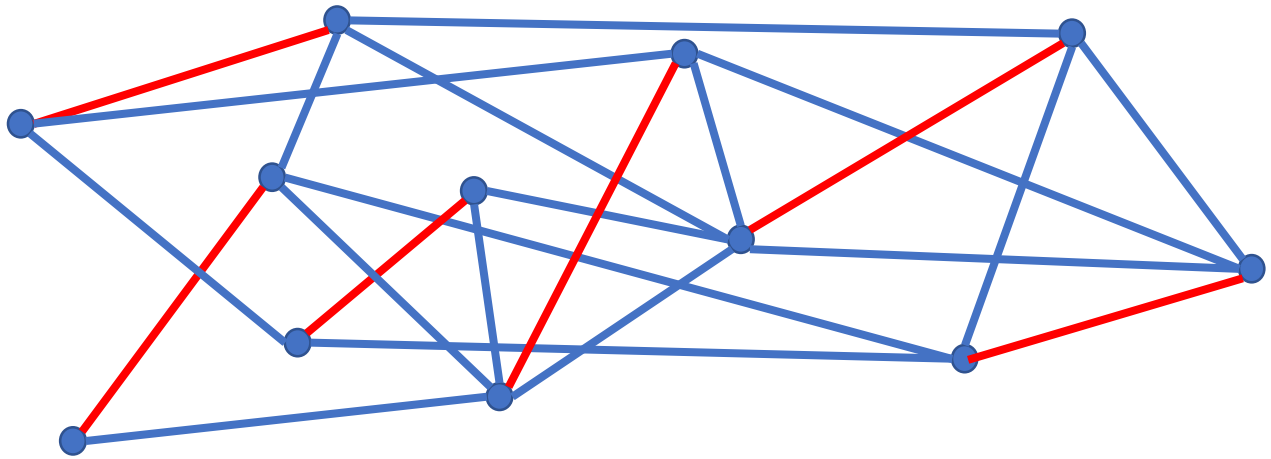
- 一条边一条边加进来，直到加不了……



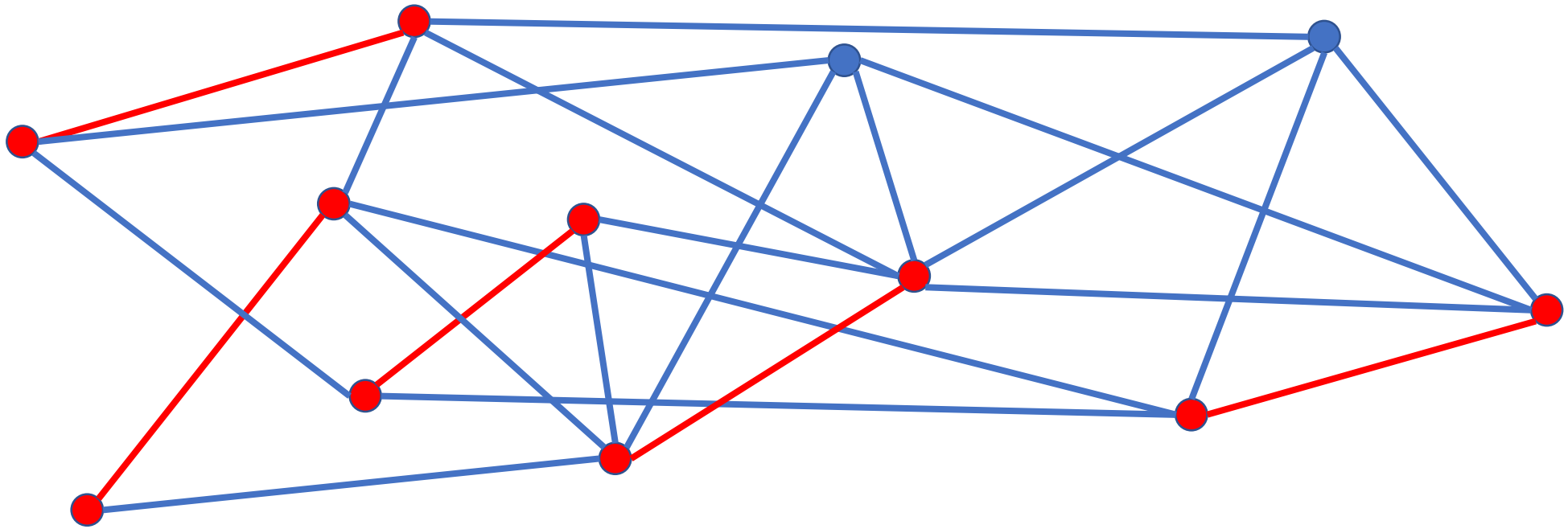
极大 vs 最大

- 题外话：注意**极大**（maximal）和**最大**（maximum）是两个不同的概念!
- **极大**匹配不一定**最大**!





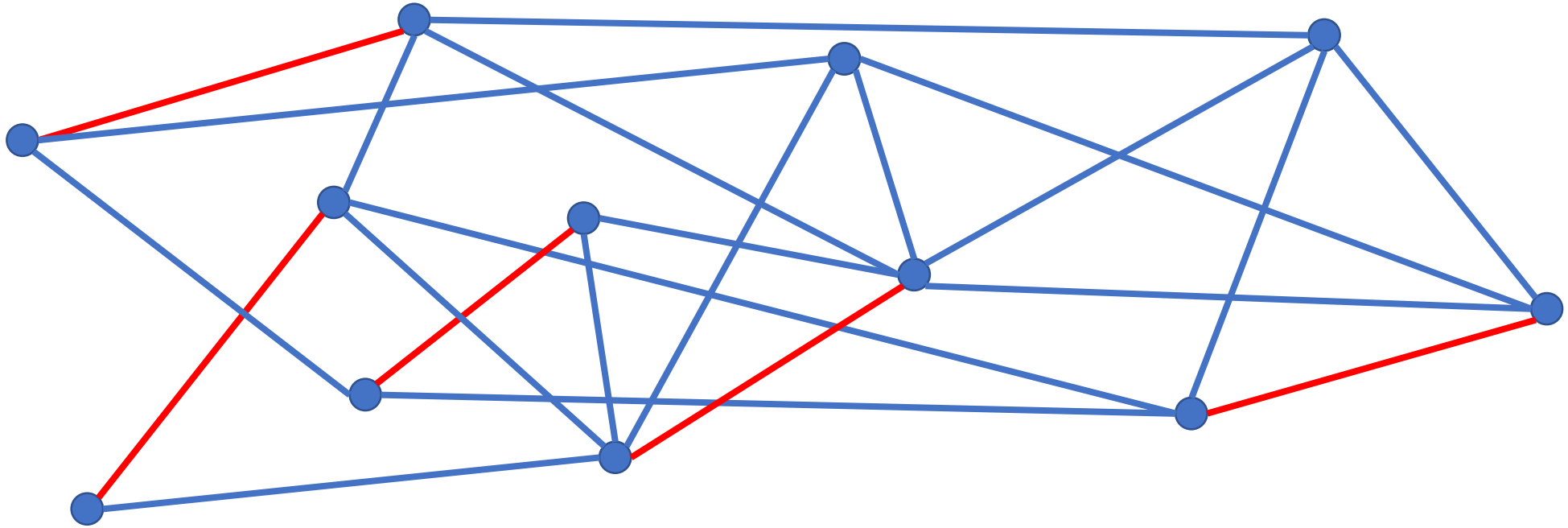
【引理1 】极大匹配中所有端点构成一个点集覆盖。



【证明】设 $M \subseteq E$ 为任意一个极大匹配。

- 对于任意的边$e = (u, v)$，端点u, v中至少有一个在M里面：否则，$M \cup \{e\}$ 仍然是一个匹配，和M的极大性矛盾!
- 根据点集覆盖的定义，这已经说明M 是一个点集覆盖!

【引理2】 对于任意的极大匹配 M, 任意一个点集覆盖大小至少为 $|M|$。



【证明】

- M当中的边至少必须被覆盖
- 一个顶点最多只能覆盖M当中的一条边
- 光是去覆盖M中的边就至少需要$|M|$个点

一个2-近似算法

Algorithm 1:

- Find a maximal matching M
- Let S be the endpoints of all edges in M
- Output S

给定一个无向图$G = (V, E)$，设

- $OPT(G)$ 为最小点集覆盖的大小
- $S(G)$ 为Algorithm 1输出的点集覆盖大小

【定理】对于任意无向图 $G = (V, E)$，有$S(G) \leq 2 \cdot OPT(G)$

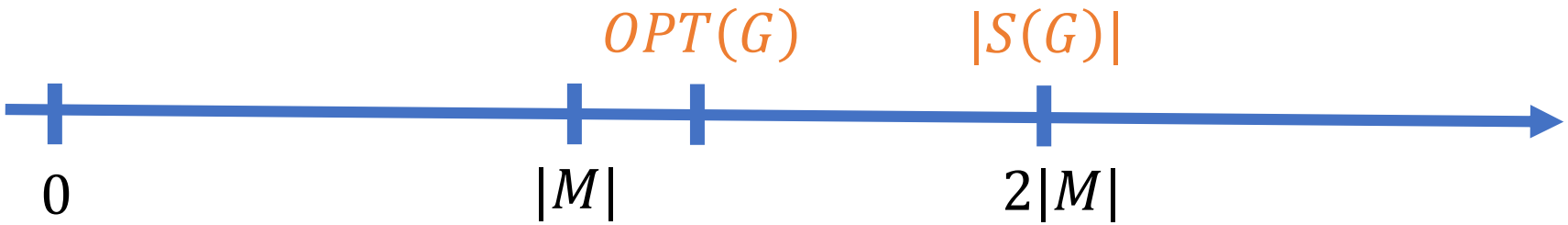
$\forall G: |S(G)| \leq 2 \cdot OPT(G)$

【引理1 】极大匹配中所有端点构成一个点集覆盖。

- $\Longrightarrow S(G)$ 是一个点集覆盖（算法输出的解一定**合法**）
- 显而易见， $|S(G)| = 2|M|$

【引理2】 对于任意的极大匹配 M, 任意一个点集覆盖大小至少为$|M|$。

- $\Longrightarrow OPT(G) \geq |M|$



——些“观后感”

近似算法定义

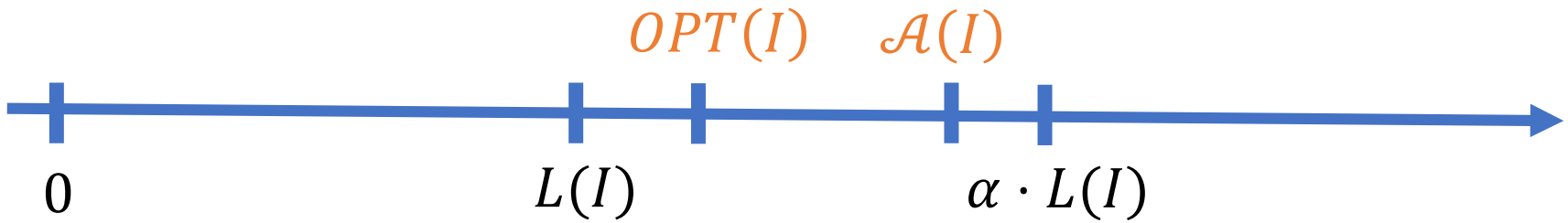
- 考虑一个**最小化问题**的算法$\mathcal{A}$。对于输入I，设$\mathcal{A}(I)$为算法输出值，$OPT(I)$为I的最优解（最小值）。$\mathcal{A}$是一个α-近似算法当

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \leq \alpha$$

- 对于**最大化问题**，$\mathcal{A}$是一个α-近似算法当

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \geq \alpha$$

一个设计近似算法的常见思路



- 寻找$OPT(I)$ 的一个容易被计算的**下界**$L(I)$
- 设计算法$\mathcal{A}$ 并找到 α 使得 $\forall I$: $\mathcal{A}(I) \leq \alpha \cdot L(I)$

算法改进

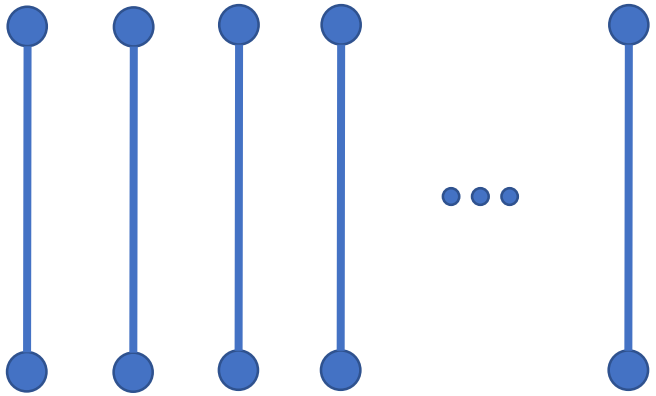
Algorithm 1:

- Find a maximal matching M
- Let S be the endpoints of all edges in M
- Output S

【问题】我们能做到比2-近似更好吗？

- **思路1**：同样的算法，我们能分析的更好吗？即，我们能否通过证明其实这个算法能做到比2-近似更好？
- **思路2**：有没有更聪明的算法？

思路1行不通……

$G =$ 

- 假设 G 有 $2n$ 个顶点和 n 条边，如上所示
- $OPT(G) = n$
- $\mathcal{A}(G) = 2n$

思路2“很可能”行不通……

- 不可近似性
- 类似这种风格的结论：如果问题f有α-近似算法，那么P = NP。

- [Khot, Minzer & Safra, 2017] 对于任意$\epsilon > 0$，一个多项式时间复杂度的 $(\sqrt{2} - \epsilon)$-近似算法的存在意味着 $\mathbf{P} = \mathbf{NP}$.
- [Khot & Regev, 2008] 假设 Unique Game Conjecture成立，对于任意$\epsilon > 0$，一个多项式时间复杂度的 $(2 - \epsilon)$-近似算法的存在意味着 $\mathbf{P} = \mathbf{NP}$.

当我们成功设计出一个α-近似算法后……

我们有两个思路去提升**近似比** α:

- 思路1：同样算法，更好的分析
- 思路2：更好的算法

- 想要否定思路1，找一个例子说明该算法在该例子下只能做到α近似。这种例子被称作tight example。
- 想要否定思路2，证明不可近似性。

近似算法案例二

Max-3SAT

英文预警

- Boolean formula：布尔表达式
- Clause：布尔表达式中的一“项”
- Variable：变量x_i
- Literal：一个正变量x_i或逆变量$\neg x_i$构成的一个“**单词**”

$$\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_5)$$

a 3-CNF Boolean formula with **3 clauses**, **5 variables**, and **9 literals**

Max-3SAT

[Max-3SAT]

- Input: a 3-CNF Boolean formula ϕ
- Output: an assignment satisfying maximum number of clauses

Assumption:

1. Each clause contains exactly 3 literals
2. Each clause contains 3 distinct variables

What if we assign values randomly?

- For each x_i, assign
  - $x_i = \text{true}$ with probability 0.5;
  - $x_i = \text{false}$ with probability 0.5.
- What is the probability that a clause is satisfied?
- What is the number of satisfied clauses in expectation?

Linearity of Expectation

- **Theorem**. Let
  - $X_1, \ldots, X_n$ be n random variables **that may be dependent**, and
- We have

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i]\,.$$

Notation

- n: number of variable
- m: number of clauses

- ϕ contains $3m$ literals, and $n \leq 3m$

Max-3SAT Random Assignment

- For each $i = 1, \ldots, m$, define random variable

$$Y_i = \begin{cases} 1, & \text{if } i\text{th clause is satisfied} \\ 0, & \text{otherwise} \end{cases}$$

- We have $\mathbb{E}[Y_i] = 1 \times \Pr(Y_i = 1) + 0 \times \Pr(Y_i = 0) = \frac{7}{8}$.
- $Y = \sum_{i=1}^{m} Y_i$: total number of satisfied clauses
- We want to compute $\mathbb{E}[Y]$.
- By Linearity of Expectation:

$$\mathbb{E}[Y] = \mathbb{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbb{E}[Y_i] = \frac{7}{8}m.$$

A $\frac{7}{8}$-Approximation Algorithm?

- m is clearly an upper bound to $\mathbf{OPT}$.
- If we can satisfied $\geq \frac{7}{8}m$ clauses, we get a $\frac{7}{8}$-Approximation Algorithm!

Let's try to assign value to x_1

- We have

$$\begin{aligned}&\mathbb{E}[Y]\\ &= \mathbb{E}[Y|x_1 = \text{true}] \cdot \Pr(x_1 = \text{true}) + \mathbb{E}[Y|x_1 = \text{false}] \cdot \Pr(x_1 = \text{false})\\ &= \frac{1}{2} \cdot \mathbb{E}[Y|x_1 = \text{true}] + \frac{1}{2} \cdot \mathbb{E}[Y|x_1 = \text{false}]\end{aligned}$$

- which implies

$$\mathbb{E}[Y|x_1 = \text{true}] + \mathbb{E}[Y|x_1 = \text{false}] = 2 \cdot \mathbb{E}[Y].$$

- Thus, either $\mathbb{E}[Y|x_1 = \text{true}] \geq \mathbb{E}[Y]$ or $\mathbb{E}[Y|x_1 = \text{false}] \geq \mathbb{E}[Y]$.
- The two conditional expectations can be computed in $O(m)$ time.
- We can assign value to x_1 with larger conditional expectation!

Example

- $\phi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$
- Assigning $x_1 = \text{true}$ results in
  - $\phi = \text{true} \wedge \text{true} \wedge (\neg x_2 \vee x_4)$
  - $\mathbb{E}[Y|x_1 = \text{true}] = 1 + 1 + \frac{3}{4} = 2.75$
- Assigning $x_1 = \text{false}$ results in
  - $\phi = (x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge \text{true}$
  - $\mathbb{E}[Y|x_1 = \text{false}] = \frac{3}{4} + \frac{3}{4} + 1 = 2.5$
- We shall assign $x_1 = \text{true}$.

Continue for x_2…

- After assigning some value for x_1:
- $x_1 = v_1$ where $v_1 \in \{\text{true}, \text{false}\}$
- We assign value for x_2 by comparing
- $\mathbb{E}[Y|x_1 = v_1, x_2 = \text{true}]$,$\mathbb{E}[Y|x_1 = v_1, x_2 = \text{false}]$
- Assign $x_2 = v_2 \in \{\text{true}, \text{false}\}$ with larger conditional expectation.

An Approximation Algorithm

1. for $i = 1, \ldots, n$:
2. compute $\mathbb{E}[Y|x_1 = v_1, \ldots, x_{i-1} = v_{i-1}, x_i = \text{true}], \mathbb{E}[Y|x_1 = v_1, \ldots, x_{i-1} = v_{i-1}, x_i = \text{false}]$
3. assign $x_i = v_i \in \{\text{true}, \text{false}\}$ with the larger conditional expectation
4. endfor

Expectation Monotonicity

In each iteration:

$$\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$$

$$= \frac{1}{2}\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{true}] + \frac{1}{2}\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{false}]$$

Thus, either

- $\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{true}] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$, or
- $\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = \text{false}] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$

The algorithm always choose $x_i = v_i \in \{\text{true}, \text{false}\}$ with larger expectation:

$$\mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}, x_i = v_i] \geq \mathbb{E}[Y|x_1 = v_1, \dots, x_{i-1} = v_{i-1}]$$

The conditional expectation for Y is non-decreasing!

Expectation Monotonicity

- The conditional expectation for Y is non-decreasing!
- Thus, $\mathbb{E}[Y|x_1 = v_1, \ldots, x_n = v_n] \geq \mathbb{E}[Y] = \frac{7}{8}m$.
- $\mathbb{E}[Y|x_1 = v_1, \ldots, x_n = v_n]$ is already deterministic.
  - With assignment $x_1 = v_1, \ldots, x_n = v_n$, this is exactly the number of satisfied clauses!
- We have a $\frac{7}{8}$-approximation algorithm!
- Running Time: $O(mn)$

Possible Improvements?

- Can this algorithm do better than $\frac{7}{8}$-approximation?
- No…
- Can you come up with a tight example?

Possible Improvements?

- Exist other better algorithms?
- Assuming **P ≠ NP**, no…
- [Håstad, 2001] A polynomial time $\left(\frac{7}{8}+\varepsilon\right)$-approximation algorithm for any $\varepsilon > 0$ implies P = NP.

Story for Maximum Independent Set?

Maximum Independent Set

- For any $\varepsilon > 0$, Maximum Independent Set/Clique is NP-hard to approximate to within factor $(|V|^{1-\varepsilon})$.
  - [Håstad, 1999], [Khot, 2001] and [Zuckerman, 2006]
- Can you give a $|V|$-approximation algorithm?
- An $O\left(\frac{|V|(\log\log|V|)^2}{(\log|V|)^3}\right)$-approximation algorithm…
  - [Feige, 2004]