计算机问题求解

Qiu. Hu Compile.

2017.7

CONTENTS

第 1 周：从 P 到 NP: “难”与“易”的界限在哪里？

教学目的：正确理解 P, NP 和 NP-Completeness 的概念。理解问题和问题大小的定义，建立算法复杂性分析的理论基础。理解多项式规约的意义并通过示例了解规约的基本技术。

阅读材料：*Christopher Moore, etc.: The Nature of Computation, Oxford University Press 2011*; 第 4, 5 章

1. 复杂性？
2. 问题固有的复杂度与算法的复杂度？

问题固有的复杂度指问题的解空间大小，而算法复杂度依赖于求解的方法，包括模型，数据结构，以及硬件条件等。

3. 什么是 P？

> P is the class of problems for which an algorithm exists that solves instances of size n in time $O(n^c)$ for some constant c.

> TIME($f(n)$) is the class of problems for which an algorithm exists that solves instances of size n in time $O(f(n))$.

$$\mathrm{P} = \bigcup_{c>0} \mathrm{TIME}(n^c).$$

简而言之，P 问题就是这样一类问题，在实例大小为 n 时，存在$O(n^c)$时间的算法解决（c 为常数）。

4. 判定问题 & 函数问题？

判定问题： 是否存在一条欧拉回路（经过连通图 G 每条边的回路）？

函数问题： 构造一条欧拉回路。

5. 为什么 Poly 函数概念如此重要？Poly function’s robustness 意味着什么？

The beauty of the definition of P is that it is extremely robust to changes in how we measure running time, and what model of computation we use.

无论我们采取什么样的时间度量方式和计算模型，P 的定义都很鲁棒。

P 对于输入格式也很鲁棒

P 对于我们如何实现算法的诸多细节也很鲁棒，最多改变一个多项式。

6. 数学领域？计算机技术领域？
7. NP 的非对称特征，对于“是否存在 xxx？”这样的问题，判断 yes instance 是容易的，但是判定其 no instance 非常难，几乎需要穷举每个 instance，看是否都是 no。

> A decision problem is in NP if, whenever the answer for a particular instance is “yes,” there is a simple proof of this fact.

8. 为什么P ∈ NP？

当 answer 为 yes 的时候，P 显然有多项式的算法来构造出解，从而证明了该事实。

P ≠ NP -- finding needle is harder than checking them.

9. 归约

如果存在多项式时间算法将 A 的实例转化为 B 的实例，则称 A 可以归约到 B，写作$A \leq B$。这种情况下，B 至少有 A 那么难，如果可以多项式时间解决 B，那么肯定可以多项式时间解决 A。

10. NP 定义之一：存在 witness

> NP is the class of problems A of the following form:
>
> x is a yes-instance of A if and only if there exists a w such that (x, w) is a yes-instance of B,
>
> where B is a decision problem in P regarding pairs (x, w), and where $|w| = \text{poly}(|x|)$.

w 相当于一个 witness，见证 x 是 A 的一个 yes-instance。

给定输入 x，想知道是否有 needle w 存在（needle 可以是路径，着色等等），B 是检查 w 是否是合法 needle 的问题，如 Hamilton path 中，x 是图，w 是路径，B(x, w)是检查 w 是否是 x 中合法 Hamilton path 的问题。w 需要为 poly(|x|)的，表示一根 needle 的数据单元不超过 poly(|x|)。

11. |w|=poly(n), 2^poly(n) 个 witnesses，一个一个运行 B，有

$\text{NP} \subseteq \text{EXP}$. $\quad\quad$ $\text{EXP} = \text{TIME}(2^{\text{poly}(n)})$,

we believe there is a constant $\alpha > 0$ such that, for any $g(n) = o(2^{n^\alpha})$,

$$\text{NP} \not\subseteq \text{TIME}(g(n)).$$

12. NP 定义之二：逻辑定义

> NP is the class of properties A of the form
>
> $$A(x) = \exists w : B(x, w)$$
>
> where B is in P, and where $|w| = \text{poly}(|x|)$.

$A(x)$ is true if x is a yes-instance of A.

$B(x, w)$ is the property that w is a valid witness for x.

注意：$\overline{A(x)} = \overline{\exists w : B(x, w)} = \forall w : \overline{B(x, w)}$.，凸显了非对称性。

13. NP 定义之三： N 表示 nondeterministic

> NP is the class of problems for which a nondeterministic program exists that runs in time poly(n), such that the input is a yes-instance if and only if there exists a computation path that returns "yes."

*推广如下：

> NTIME($f(n)$) is the class of problems for which a nondeterministic program exists that runs in time $O(f(n))$, such that the input is a yes-instance if and only if there exists a computation path that returns "yes."

在此定义中，NP 定义为这样一类问题，即存在一个“非确定”程序，在 poly(n)时间内运行完，判定输入为 yes-instance 当且仅当存在一个返回 yes 的计算路径(computation path)。

(a) 什么是 nondeterministic？

of computer. The N in NP stands for "nondeterministic," and a nondeterministic computer is one that can make several possible choices at each step. Imagine a programming language in which we have the

(b) 什么是 computation path？

~~function of the running time.~~ This gives nondeterministic computers the amazing, and unrealistic, ability to search an exponentially large space of candidate solutions exhaustively in polynomial time. In the "needle in a haystack" metaphor, they can examine all the blades of hay in parallel. If a needle exists, they can find it in no time at all.

根据“非确定性”的概念，某条计算路径可以理解为，对每一步做一个分支，沿着某个步序列从上往下的计算过程？

13. 三种定义一致么？最关键的地方/区别在哪？

第一种定义强调有一个 witness，

第二种采用逻辑形式，与第一种类似，

第三种引入“非确定性程序”的概念。

对于 N 的观点不同？

14. NP-Completeness

最难？在算法复杂度意义下的“最”？在所有问题的范围内？

NPC 能解，则 NP 能解，其他的也都能解。

15. 什么是 NPC？

A problem B in NP is NP-*complete* if, for any problem A in NP, there is a polynomial-time reduction from A to B.

这类问题都没有找到多项式时间算法，如果其中一个存在 poly 算法，那么所有都存在 poly 算法，因为他们可互相归约。

16. 归约 – 衡量问题难度的手段。

问题归约指的是，将 A 问题的 instance 转化为 B 问题的 instance，如果 B 问题可解，则 A 问题可解。

A polynomial-time reduction from A to B is a function f, computable in polynomial time, such that if x is an instance of A, then $f(x)$ is an instance of B. Moreover, $f(x)$ is a yes-instance of B if and only if x is a yes-instance of A.

可以多步归约，归约具有可传递性。

17. NP-hard？

有这样一种问题，所有 NP 问题都可以归约到这种问题，我们称之为 NP-hard 问题。

如果一个问题既是 NP 问题又是 NP-Hard 问题，则它是 NP-Complete (NPC)问题。

直觉上难度递增：P<=NP<=NP-Complete<=NP-Hard.

18. 一些 NP 问题

GRAPH k-COLORING
Input: A graph G
Question: Is there a proper k-coloring of G?

SAT
Input: A CNF Boolean formula $\phi(x_1,\ldots,x_n)$
Question: Is ϕ satisfiable?

Both ϕ and ϕ' are Boolean formulas with a special structure, called CNF for *conjunctive normal form*. This means that each one is the AND (conjunction) of a series of *clauses*, and each clause is the OR of a set of *literals*, where a literal is either a variable or its negation. ~~Thus there are multiple ways to satisfy each~~

k-SAT
Input: A CNF Boolean formula ϕ, where each clause contains k variables
Question: Is ϕ satisfiable?

k-SAT 归约到 3-SAT，做拆分，引入 dummy variables：

More interestingly, if $k > 3$ we can break a k-SAT clause into a chain of 3-SAT clauses, using dummy variables as the chain links. For instance, we can break a 5-SAT clause down like this:

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \Leftrightarrow (x_1 \vee x_2 \vee z_1) \wedge (\overline{z}_1 \vee x_3 \vee z_2) \wedge (\overline{z}_2 \vee x_4 \vee x_5). \quad (4.7)$$

INTEGER PARTITIONING
Input: A list $S = \{x_1, \ldots, x_\ell\}$ of positive integers
Question: Is there a *balanced partition*, i.e., a subset $A \subseteq \{1, \ldots, \ell\}$ s.t. $\sum_{i \in A} x_i = \sum_{i \notin A} x_i$?

SUBSET SUM
Input: A set $S = \{x_1, \ldots, x_\ell\}$ of positive integers and an integer t
Question: Does there exist a subset $A \subseteq \{1, \ldots, \ell\}$ such that $\sum_{i \in A} x_i = t$?

INDEPENDENT SET
Input: A graph G and an integer k
Question: Does G have an independent set of size k or more?

VERTEX COVER
Input: A graph G and an integer k
Question: Does G have a vertex cover of size k or less?

CLIQUE
Input: A graph G and an integer k
Question: Does G have a clique of size k or more?

MAX INDEPENDENT SET
Input: A graph G
Output: An independent set S of maximal size

CIRCUIT SAT
Input: A Boolean circuit C
Question: Is C satisfiable?

INDEPENDENT SET (exact version)
Input: A graph $G = (V, E)$ and an integer k
Question: Does G's largest independent set have size exactly k?

EXACT SPANNING TREE
Input: A weighted graph G and two integers ℓ and u
Question: Is there a spanning tree T with weight w such that $\ell \leq w \leq u$?

MAX-k-SAT
Input: A k-SAT formula ϕ and an integer ℓ
Question: Is there a truth assignment that satisfies ℓ or more clauses in ϕ?

MAX CUT (s-t version)
Input: A weighted graph G, two vertices s, t and an integer k
Question: Does there exist a cut in G of weight at least k that separates s and t?

$$y = x_1 \wedge x_2 \iff (x_1 \vee \overline{y}) \wedge (x_2 \vee \overline{y}) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y).$$

19. 作业

4.1 Ask the oracle. Suppose that we have access to an oracle, as in Problem 1.10, who can answer the decision problem SAT. Show that, by asking her a polynomial number of questions, we can find a satisfying assignment if one exists. Hint: first describe how setting a variable gives a smaller SAT formula on the remaining variables. Show that this is true of SUBSET SUM as well. This property is called *self-reducibility*, and we will meet it again in Chapters 9 and 13.

4.12 Same-sum subsets. Suppose that S is a list of 10 distinct integers, each ranging from 0 to 100. Show that there are two distinct, disjoint sublists $A, B \subset S$ with the same total. (Note that, unlike in INTEGER PARTITIONING, we don't demand that $A \cup B = S$.) That doesn't make it easy to find one!

和最小 45，最大 955，有 911 种情况。10 个元素子集有 1024 种。鸽笼原理。

4.14 When greed works. A sequence $a_1, \ldots, a_n$ of integers is called *superincreasing* if each element a_i is strictly greater than the sum of all previous elements. Show that SUBSET SUM can be solved in polynomial time if S is superincreasing. Hint: be greedy. What does your algorithm do in the case $S = \{1, 2, 4, 8, \ldots\}$?

从大到小填，遇到能填的必须填，因为以后就没机会了。

5.1 From circuits to NAESAT. Find a direct reduction from CIRCUIT SAT to NAE-3-SAT. Hint: consider the 3-SAT clause $(\overline{x}_1 \vee \overline{x}_2 \vee y)$ in our description of an AND gate. Does it ever happen that all three literals are true?

$$\begin{aligned}\phi = {} & (x_1 \vee \overline{y}_1) \wedge (x_2 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1) \\ & \wedge (\overline{x}_1 \vee y_2) \wedge (\overline{x}_2 \vee y_2) \wedge (x_1 \vee x_2 \vee \overline{y}_2) \\ & \wedge (y_1 \vee y_3) \wedge (\overline{y}_1 \vee \overline{y}_3) \\ & \wedge (y_2 \vee \overline{z}) \wedge (y_3 \vee \overline{z}) \wedge (\overline{y}_2 \vee \overline{y}_3 \vee z) \\ & \wedge z .\end{aligned}$$

figure 5.2 加 dummy 变量，证明三个 literal 都不会相等。

5.3 Easy majority? Consider MAJORITY-OF-3 SAT, in which each clause contains three literals and at least two of them must be true for it to be satisfied. Either show that this problem is in P by reducing it to 2-SAT, or show that it is NP-complete by reducing 3-SAT to it.

$(x \wedge y \wedge z) = (x \vee y) \wedge (y \vee z) \wedge (x \vee z)$ 即，n 个子句的 MAJORITY-3-SAT 可以表示为 3n 个子句的 2-SAT。

第 2 周：P=NP？为什么这会成为计算机科学理论中最根本的问题？

教学目的：探讨 P =? NP 对于计算机科学，甚至对于认识论的深刻影响。理解为什么证明 P != NP 非常困难，而且只要 P != NP 就一定存在一些问题在 P 和 NP-complete 之间难以确定。

阅读材料：*Christopher Moore, etc.: The Nature of Computation, Oxford University Press 2011*; 第 6 章

1. coNP？

反 NP，判断 no-instance 很容易，判断 yes-instance 需要检查所有。

2. P 是否等于 NP 这个问题还处于开放状态，但是主流观点倾向于不成立，否则后果太严重。

三个后果：The great collapse / As below, so above / The demise of creativity

3. 两种问题：$\prod, \sum$

$\Pi_2\mathsf{P}$ is the class of properties A of the form

$$A(x) = \forall y : \exists z : B(x, y, z),$$

where B is in P, and where $|y|$ and $|z|$ are polynomial in $|x|$.

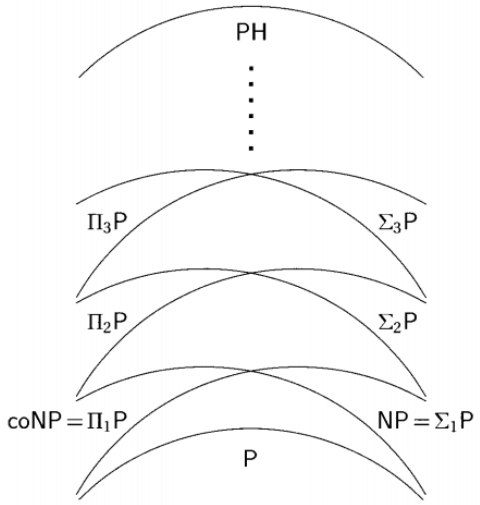
$\Sigma_k\mathsf{P}$ is the class of properties A of the form

$$A(x) = \exists y_1 : \forall y_2 : \exists y_3 : \cdots : \mathsf{Q} y_k : B(x, y_1, \ldots, y_k), \tag{6.1}$$

where B is in P, $|y_i| = \text{poly}(|x|)$ for all i, and $\mathsf{Q} = \exists$ or $\forall$ if k is odd or even respectively. Similarly, $\Pi_k\mathsf{P}$ is the class of properties A of the form

$$A(x) = \forall y_1 : \exists y_2 : \forall y_3 : \cdots : \mathsf{Q} y_k : B(x, y_1, \ldots, y_k), \tag{6.2}$$

where B is in P, $|y_i| = \text{poly}(|x|)$ for all i, and $\mathsf{Q} = \forall$ or $\exists$ if k is odd or even respectively.



多项式塔，其中

$$\mathsf{PH} = \bigcup_{k=0}^{\infty} \Sigma_k\mathsf{P} = \bigcup_{k=0}^{\infty} \Pi_k\mathsf{P}.$$

$\mathsf{P} \subseteq (\mathsf{NP} \cap \mathsf{coNP})$.

NP != coNP

4. 量词的递增 -> Chess Problem

SMALLEST BOOLEAN CIRCUIT
Input: A Boolean circuit C that computes a function f_C of its input
Question: Is C the smallest circuit that computes f_C?

A 选择电路 C’
B 给出反例 x
A 再选择电路 C’ ……
博弈…

5. 为什么我们倾向于相信在 polynomial hierarchy 中每层的 complexity class 是不一样的？
如果相同，则从最内层逐层重复吸收(absorbing)过程，最后的结论就是不管哪个层上的问题均属于 P！

6. the demise of creativity，P=NP 的另一重大后果就是，知识的获取变得更加容易了。

7. SHORT PROOF 是 NPC，为什么？

SHORT PROOF
Input: A statement S and an integer n given in unary
Question: Does S have a proof of length n or less?

as a function of n. Note also that SHORT PROOF is NP-complete, since S could be the statement that a given SAT formula is satisfiable.

8. 问题的 Lower Bound？为什么证明其比证明 Upper Bound 要困难？

One of the overarching themes of computer science is that lower bounds on a problem's complexity are much harder to prove than upper bounds. To prove that a problem is in P, all we have to do is exhibit a polynomial-time algorithm that solves it. But to prove that it is not in P, we have to reason about *all possible* polynomial-time algorithms simultaneously, and prove that each one fails in some way.

9. 关乎信息而不是关乎计算？

This "Twenty Questions" argument is about *information*, not computation. No matter how powerful an algorithm is computationally, it can't solve the problem if it doesn't have enough information to determine the answer. Since information is much easier to quantify and bound than computational power, this lets us prove a lower bound on the number of comparisons we need, without even thinking about what the algorithm will do with this information once it has it.

对于排序问题来说，即使计算力再强大，如果没有元素间比较的信息，仍然无法解决，而比较其实不需要太多计算。

10. But Are the Boxes Really Black? 如何让我们放弃使用 Lower Bound 思想来解决 P=?NP 问题？

~~bit at a time, by slotting elements into an array.~~ It takes advantage of the fact that when a and b are strings or integers, the relation $a < b$ is not a "black box"—it has internal structure that we can understand. ~~This~~

基排序无需比较，利用了数字的内部结构信息。
在很多 NP 问题，NPC 问题之前，我们并非那么的无助，有很多问题有一些良好的结构，我们可以利用起来。

but they are exponentially faster than a naive exhaustive search. Thus while we think of NP-complete problems as searching for needles in a haystack, in reality we are not quite so helpless—they correspond to highly structured haystacks, and good algorithms can exploit this structure.

11. diagonalization 途径来攻克 P=?NP
给定一个“计算能力上限”，设法构造出一个问题，并证明该问题不可能在给定的计算能力下解决。
对角线方法还可以证明图灵停机问题(Halt Problem)。

12. PREDICT

PREDICT(Π, x)

Input: A program Π and an input x

Output: If Π halts within $f(|x|)$ steps when given x as its input, return its output $\Pi(x)$. Otherwise, answer "don't know."

CATCH22(Π)

Input: A program Π

Output: If Π halts within $f(|\Pi|)$ steps when given its own source code as input, return the *negation* of its output, $\overline{\Pi(\Pi)}$. Otherwise, answer "don't know."

几个关键点：a) Predict 可以通过模拟预测任何可以在指定时间之内得到解的问题；b) 不可能不通过全程模拟就得以在快于指定时间内得到预测结果，这是通过此问题的一个特例 Catch-II 体现的；c) 考虑到模拟的额外开支，Predict 不可能在指定时间内得到解；d) 我们可以建立一个递增的复杂度层次结构。

13. The way ahead

a) P=NP，但是诸如 3-SAT 或 SHORT PROOF 的最快算法需要$O(n^{100})$的时间复杂度，我们无法完成搜索。但即便如此，checking proofs 和 finding solutions 只是计算量的差别，而不是两种问题。b) P=NP 可能有一个非构造性的证明

14. 作业

6.1 This far and no farther. Consider the version of INDEPENDENT SET given at the end of Section 4.2.4:

INDEPENDENT SET (exact version)

Input: A graph $G = (V, E)$ and an integer k

Question: Does G's largest independent set have size k?

Or similarly, the problem CHROMATIC NUMBER from Problem 4.22:

CHROMATIC NUMBER

Input: A graph G and an integer k

Question: Is k the smallest integer such that G is k-colorable?

Show that for both these problems, the property $A(x)$ that x is a yes-instance can be expressed as follows, where B and C are in P:

$$A(x) = (\exists y : B(x, y)) \wedge (\forall z : C(x, z)) .$$

Equivalently, show that there are two properties A_1, A_2 in NP such that $A(x)$ holds exactly if $A_1(x)$ is true but $A_2(x)$ is false. Thus A can be thought of as the difference between two NP properties, and the class of such properties is called DP. Now show that

$$\mathsf{DP} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P},$$

and give some intuition for why this inclusion should be proper.

设 A 为 exact-k 独立集，x 为 graph，y 为大小为 k 的集合，B(x, y): 检查 y 是否为独立集的问题。z: 大小超过 k 的集合。C(x, z): 是否不存在 z 为独立集。则得。

$\sum_2 P$: $\exists y \forall z \ B(x, y, z)$

$\prod_2 P$: $\forall z \exists y \ B(x, y, z)$

$A(x) = \exists y \forall z [B(x, y) \wedge C(x, z)]$

6.2 Mate in k. Argue that "mate in k moves" problems in Generalized Chess are in Σ_kP even on a board of *exponential* size, as long as there are only a polynomial number of pieces. Hint: you just need to check that the moves can be described with a polynomial number of bits, and that we can check in polynomial time, given the initial position and the sequence of moves, whether White has won.

Mate in k 问题可以表示为 $\mathrm{A(x)} = \exists y_1 \forall y_2 \exists y_3 \forall y_4 \ldots Q y_k \ \ B(x, y_1, \ldots, y_k)$
其中，x 是棋盘，$y_k \ \ (k \ \ is \ \ odd)$表示白子的 Move，$y_k \ \ (k \ \ is \ \ even)$表示黑子的 Move，则 A(x)表示存在白子的 moves: $y_1, y_3, \ldots, y_{2i+1} \ \ (2i+1 \leq k)$，无论黑子怎么下，白子都赢。所以 mate in k 问题在 E_2P 中，且有$|y_i|$=poly(|x|)，下子位置的表示是|x|的多项式。

6.3 Constructible time. Prove that the function $f(n) = 2^n$ is time constructible. Prove also that if $f(n)$ and $g(n)$ are time constructible, then so are $f(n) + g(n)$, $f(n) \times g(n)$, and $f(g(n))$, where for the last one we assume that $f(n) \geq n$.

time-constructible: a function f is called time-constructible if there exists a Turing machine M which, given a string 1^n, outputs the binary representation of f(n) in O(f(n)) time
fully time-constructible: A function f is called fully time-constructible if there exists a Turing machine M which, given a string 1^n consisting of n ones, stops after exactly f(n) steps.
space-constructible: a function f is space-constructible if there exists a Turing machine M which, given a string 1^n consisting of n ones, outputs the binary (or unary) representation of f(n), while using only O(f(n)) space.
图灵机。

6.11 The oracle says yes or no. Let's explore the class P^{NP} in more detail. First, show that P^{NP} contains both NP and coNP. Then, show that $\mathsf{P}^{\mathsf{NP}} \subseteq \Sigma_2\mathsf{P} \cap \Pi_2\mathsf{P}$ and also that $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2\mathsf{P}$.

Hint: let Π be the polynomial-time program that makes calls to the NP oracle. An input x is a yes-instance if there is a run of Π which returns "yes," which receives "yes" and "no" answers from the oracle at the appropriate places.

P^{NP} is the class of problems solvable in polynomial time with an oracle for some NP-complete problem.即如果存在一个解决 NPC 问题的 oracle，则 P^{NP} 问题可以在多项式时间内解决。
如有此 oracle，NP 问题可在常数时间内解决，故 NP 属于 P^{NP}，同理 coNP 属于 P^{NP}。
$\exists^{\mathrm{P}}\mathcal{C} := \{\exists^p L \mid p \text{ is a polynomial and } L \in \mathcal{C}\}$
$\forall^{\mathrm{P}}\mathcal{C} := \{\forall^p L \mid p \text{ is a polynomial and } L \in \mathcal{C}\}$
$\sum_2 P = \exists^P \prod_1 P = \exists^P coNP$
$\prod_2 P = \forall^P \sum_1 P = \forall^P NP$
$P^{NP} = \Delta_2 P \subseteq \sum_2 P \quad P^{NP} = \Delta_2 P \subseteq \prod_2 P$ 所以： $P^{NP} = \Delta_2 P \subseteq \sum_2 P \cap \prod_2 P$

第 3 周：对计算问题理解的新视角 - 为解难题寻找出路

教学目的：理解解决“难”问题的途径背后的基本思想，引导学生了解如何从思想出发考虑解决问题的技术，由此引出近似算法基本概念。

阅读材料：*Juraj Hromkovic: Theoretical Computer Science, Springer-Verlag 2004*，第 7 章

1. 难题精确算法是指数复杂度的，除了小规模以外没法用，往哪里想办法？
降低指数的级别(2^n to 1.5^n)？ 重新考虑问题本身？ 近似解？ 随机算法？

2. concept 与 technology, algorithm, strategy 的不同？
concept 更强调一种理念，一种思想，而不是具体的操作技巧，步骤或者算法。

3. 几个 concepts:

a) 我们衡量的复杂度是最坏情况下的复杂度，但是最坏情况往往在应用中不会出现，我们可以按照复杂度分为不同类，如果问题的典型输入属于简单的一类，那么可以很好解决。

b) 一些指数级算法并没有那么耗时。

c) 有时我们可以降低对问题解的要求。采用随机算法或者近似算法。随机算法将确定性的控制转换为随机的，牺牲了得正确解的保证，换来复杂度的降低。近似算法常用来解决优化问题，有时并不一定要求最优解，我们也可以接受差不太远的可行解。

4. 伪多项式算法？整数值问题(Integer problem)？背包问题与整数值问题？

Definition 7.1. *Let $\mathcal{U}$ be an integer problem and let A be an algorithm that solves $\mathcal{U}$. We say that A is a* **pseudopolynomial algorithm for $\mathcal{U}$**, *if there exists a polynomial p of two variables, such that*

$$\text{Time}_A(x) \in O(p(|x|, \text{MaxInt}(x)))$$

for all problem instances x of $\mathcal{U}$.

伪多项式算法，是输入 size 以及输入的最大数值的多项式。

整数值输入包括两点：“数值”和“大小(size)”。

背包问题可以将输入表示为：$I = (w_1,w_2,\ldots,w_n,c_1,c_2,\ldots,c_n,b)$ 属于$(N-\{0\})^{2n+1}$。

the “problem size” is $|I|$, and the value of some w_i or c_i **may** be in $\Theta(2^I)$。所以是“pseudo”的。

5. 分割可解的问题输入意义在于，使得 NP 问题在一定程度上在多项式时间可解。

6. 动态规划方法解背包问题。

Feasible solutions: For every $I = (w_1, w_2, \ldots, w_n, c_1, \ldots, c_n, b)$

$$\mathcal{M}(I) = \left\{ T \subseteq \{1,\ldots,n\} \,\middle|\, \sum_{i \in T} w_i \leq b \right\}$$

is the set of feasible solutions.

最大化 $cost(T, I) = \sum_{i \in T} c_i.$

由 SET(i)推出 SET(i+1)：

$$\begin{aligned} \text{SET}_{i+1} = \text{TRIPLE}_i \cup \{&(k + c_{i+1}, W_{i,k} + w_{i+1}, T_{i,k} \cup \{i+1\}) \mid \\ &(k, W_{i,k}, T_{i,k}) \in \text{TRIPLE}_i \text{ and } W_{i,k} + w_{i+1} \leq b\} \end{aligned}$$

Algorithm 3.2.2.2 ((DPKP)).

Input: $I = (w_1, w_2, \ldots, w_n, c_1, c_2, \ldots, c_n, b) \in (\mathbb{N} - \{0\})^{2n+1}$, n a positive integer.

$O(1)$ ← Step 1: $TRIPLE(1) := \{(0,0,\emptyset)\} \cup \{(c_1, w_1, \{1\}) \mid \text{if } w_1 \leq b\}$.

Step 2: **for** $i = 1$ **to** $n-1$ **do**
begin $SET(i+1) := TRIPLE(i)$;
for every $(k, w, T) \in TRIPLE(i)$ **do**
if $w + w_{i+1} \leq b$ **then**
$SET(i+1) := SET(i+1) \cup \{(k+c_{i+1}, w+w_{i+1}, T \cup \{i+1\})\}$;
Set $TRIPLE(i+1)$ as a subset of $SET(i+1)$ containing exactly one triple (m, w', T') for every achievable profit m in $SET(i+1)$ by choosing a triple with the minimal weight for the given m
end

$O(n \cdot Max\text{-}Int(I))$ ← Step 3: Compute $c := \max\{k \in \{1, \ldots, \sum_{i=1}^{n} c_i\} \mid (k, w, T) \in TRIPLE(n)$ for some w and $T\}$.

Output: The index set T such that $(c, w, T) \in TRIPLE(n)$.

N-1次计算$TRIPLE$(i+1),
每次的代价是
$O|TRIPLE(i+1)|$,

注意：$|TRIPLE(i)| \leq \sum_{i=1}^{n} c_i \leq n \cdot Max\text{-}Int(I)$

所以：$O(n^2 \cdot Max\text{-}Int(I))$

Since $n \leq |I|$, the time complexity of DPKP on I is in $O(|I|^2 \cdot Max\text{-}Int(I))$.

可以归一化到[0,1]，这样的话 MaxInt(I)可以看做常量，则 DPKP on I is $O(|I|^2)$，平方算法。

7. 如果 P != NP，Strongly NP-hard 问题不可能有伪多项式算法。

Theorem 7.7. *Let $\mathcal{U}$ be a strongly* NP*-hard integer problem. If* P $\neq$ NP*, then there exists no pseudopolynomial algorithm for $\mathcal{U}$.*

8. h-value-bounded subproblem of U

Definition 7.2. *Let $\mathcal{U}$ be an integer problem and let h be a mapping from* $\mathbb{N}$ *to* $\mathbb{N}$*. The* ***h*-value-bounded subproblem of $\mathcal{U}$, Value(h)-$\mathcal{U}$,** *is the subproblem of $\mathcal{U}$ whose instances are all instances x of $\mathcal{U}$ satisfying*

$$\text{MaxInt}(x) \leq h(|x|).$$

9. 近似算法基本知识

Definition 7.10. *Let $\mathcal{U} = (\Sigma_I, \Sigma_O, L, \mathcal{M}, cost, goal)$ be an optimization problem.*

We say that A is a **consistent algorithm for $\mathcal{U}$** *if, for every $x \in L$, the output $A(x)$ of the computation of A on x is a feasible solution for x (i.e., $A(x) \in \mathcal{M}(x)$).*

Let A be a consistent algorithm for $\mathcal{U}$. For every $x \in L$, we define the **approximation ratio $R_A(x)$ of A on x** *as*

$$R_A(x) = \max\left\{\frac{cost(A(x))}{\text{Opt}_{\mathcal{U}}(x)}, \frac{\text{Opt}_{\mathcal{U}}(x)}{cost(A(x))}\right\},$$

where $\text{Opt}_{\mathcal{U}}(x)$ is the cost of an optimal solution for the instance x of $\mathcal{U}$.

For any positive real number $\delta > 1$, we say that A is a **δ-approximation algorithm for $\mathcal{U}$** *if*

$$R_A(x) \leq \delta$$

for every $x \in L$.

10. 图中最小点覆盖问题

Minimum Vertex Cover Problem (MIN-VCP)

Input: A graph $G = (V, E)$.
Constraints: $\mathcal{M}(G) = \{S \subseteq V \mid$ every edge of E is incident to at least one vertex of $S\}$.
Cost: For every $S \in \mathcal{M}(G)$, $cost(S, G) = |S|$.
Goal: *minimum*.

(选择最少的点覆盖所有边)

11. 什么是匹配？它和点覆盖有什么关系？

Given a graph G = (V, E), a matching M in G is a set of pairwise *non-adjacent edges*; that is, no two edges share a common vertex.

通过找匹配来间接解 Min_VCP 问题：

```
while E' ≠ ∅ do
  begin take an arbitrary edge {u,v} from E';
    C := C ∪ {u,v};
    A := A ∪ {{u,v}};
    E' := E' − {all edges incident to u or v};
  end
```

a) 为什么是匹配？匹配直观地去覆盖边，顺便覆盖两端点相关的边。

b) 为什么循环结束时 C 是可行解？循环结束时 E'为空表示边都被覆盖，所以是可行解。

c) 为什么我们需要知道解的误差有多大？显然不是精确解，所以需要知道多大程度上能用。

d) 为什么这是一个多项式算法？每次 E'至少减少 1 条边，所以迭代次数最多为边数 m 次，而边数 m<=n^2，所以整个 VCA 算法复杂度为 O(m)，即为多项式算法。

12. 我们并不知道最优解 Opt(x)，如何知道误差率的上限？

我们可以推出 Opt(x)的上限或者下限，从而做出不等式得到误差率上限。

13. TSP 问题？为什么 TSP 问题不可能有多项式时间的 d-近似算法？（任意 d）

TSP 问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。图论中的一个等价形式是：给定一个加权完全图（顶点表示城市，边表示道路，权重就会是道路的成本或距离），求一个权值最小的哈密尔顿回路

Hamilton 路（圈/回路）：经过图 G 中每个点仅一次的路（圈/回路）称为 Hamilton 路（圈/回路）。

a Hamiltonian cycle[8] (the Hamiltonian cycle problem, HC) is NP-complete.

Lemma 7.8. TSP *is strongly* NP*-hard.*

反证法：如果存在 poly time 的 d-近似算法 A 解 TSP 问题，则存在算法 B 可解 HC 问题。

1. B constructs an instance $(K_{|V|}, c)$ of TSP, where

$$K_{|V|} = (V, E'), \text{ with } E' = \{\{u,v\} \mid u,v \in V, u \neq v\},$$
$$c(e) = 1, \quad \text{if} \quad e \in E, \text{and}$$
$$c(e) = (d-1)\cdot|V| + 2, \quad \text{if} \quad e \notin E.$$

2. B simulates the work of A on the input $(K_{|V|}, c)$. If the feasible solution computed by A is a Hamiltonian cycle of cost exactly $|V|$, then B accepts its input G. Otherwise, A rejects G.

(G = (V, E))

然后推导，证明如下结果，于是 B 以多项式时间解决了 NP-hard 的 HC 问题，P=NP，悖论。

$G = (V, E) \in \text{HC} \Leftrightarrow$ the output of B is a solution of cost $|V|$.

14. 增加限定条件定义子问题仍然会有很好的效果，能否描述一个具有有效近似算法的 TSP 有价值的子问题？

follows we consider the metric TSP, Δ-TSP, that allows only TSP instances that satisfy the triangle inequality (Example 2.40). The triangle inequality is a natural restriction that is satisfied[10] in many applications. Next we show a polynomial-time 2-approximation algorithm for Δ-TSP.

Algorithm SB

Input: A complete graph $G = (V, E)$ with the weight function $c : E \to \mathbb{N}^+$, that satisfies the triangle inequality

$$c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$$

for all pair-wise different vertices $u, v, w \in V$.

Phase 1. SB computes a minimal spanning tree[11] T of G with respect to c.
Phase 2. SB chooses an arbitrary vertex v from V and performs the depth-first search in T from v. Doing this, SB enumerates the vertices of T in the order in which they are visited. Let H be the resulting sequence of vertices that corresponds to this enumeration.
Output: The Hamiltonian cycle $\overline{H} = H, v$.

15. 什么是 Local Search？这里的 Local 是什么意思？

Local search is an algorithm design technique for optimization problems. The idea of this technique is to first compute a feasible solution α for a given input x, and then improve α by small (local) changes of α. What the term "small"

local 表示局部做微小改变产生另一个可行解（邻居）。

16. MAX-CUT 问题，选尽可能多的边来将点集分成两份。

Maximum Cut Problem (MAX-CUT)

Input: A graph $G = (V, E)$.
Constraints:

$$\mathcal{M}(G) = \{(V_1, V_2) \mid V_1 \cup V_2 = V, V_1 \neq \emptyset \neq V_2, \text{and} V_1 \cap V_2 = \emptyset\}.$$

Costs: For every cut $(V_1, V_2) \in \mathcal{M}(G)$,

$$cost((V_1, V_2), G) = |E \cap \{\{u, v\} \mid u \in V_1, v \in V_2\}|.$$

Goal: *maximum*.

Algorithm LS-CUT

$S = \emptyset$.
while there exists $v \in V$ such that
$cost(S \cup \{v\}, V - (S \cup \{v\})) > cost(S, V - S)$, or
$cost(S - \{v\}, (V - S) \cup \{v\}) > cost(S, V - S)$ **do**
begin
move v on the opposite side of the cut;
end
Output: $(S, V - S)$

Local Search 方法解 MAX-CUT 问题。这里的 neighborhood 是指什么？
是“将点 v 移动到对面集合中去”？

LS-CUT 是 *2-approximation* 的。下面是证明过程。

Proof. Obviously, the algorithm LS-CUT computes a feasible solution for MAX-CUT.

It remains to show that

$$R_{\text{LS-CUT}}(G) \leq 2$$

for every graph $G = (V, E)$. Let (Y_1, Y_2) be the output of LS-CUT. Since (Y_1, Y_2) is a local minimum with respect to moving a vertex from one side to the other side, every vertex $v \in Y_1$ $[Y_2]$ has at least as many edges to the vertices in Y_2 $[Y_1]$ as the number of edges from v to vertices[14] in Y_1 $[Y_2]$. This simple counting argument assures that the cut (Y_1, Y_2) contains at least half of all edges in G. Since $\text{Opt}_{\text{MIN-CUT}}(G)$ cannot exceed $|E|$,

$$R_{\text{LS-CUT}}(G) = \frac{\text{Opt}_{\text{MIN-CUT}}(G)}{cost((Y_1, Y_2))} \leq \frac{|E|}{|E|/2} = 2.$$

17. Local Search 解最小生成树的例子？为什么它能有效地得到精确解？一般情况下，局部算法被归于 heuristic，为什么？

由原图，选择一条权重最大的边，如果删除后仍是生成树，将其删除，如果不是，选择权重次大的，如此往复，直到边数达到 n-1。

只要边数大于 n-1，则总能够找到边删除，每次优先找最大的删除，所以保证得到的生成树是所有合法生成树中权重总和最小的。

heuristics，启发式方法，是指依据有限的知识（或“不完整的信息”）在短时间内找到问题解决方案的一种技术。Local Search 本身也是根据不完整信息来逐渐得到问题解。

18. 作业

Exercise 7.13. Construct, for any positive integer n, a graph G_n, such that the optimal vertex cover has the cardinality n and the algorithm VCA can compute a vertex cover of the cardinality $2n$.

二部图 / 二分图。

Exercise 7.16. Find, for any positive integer $n \geq 3$, a weight function c_n for the complete graph K_n of n vertices, such that there exist at least two different weights on the edges of K_n and the algorithm SB always computes an optimal solution.

Exercise 7.21. Let H be a Hamiltonian tour in a graph G. Remove three edges $\{a, b\}$, $\{c, d\}$ and $\{e, f\}$ such that $|\{a, b, c, d, e, f\}| = 6$ and H visits these 6 vertices in the order a, b, c, d, e, f. Draw all possible triples of edges whose addition to H results again in a Hamiltonian tour. How many possibilities are there, if k edges forming a matching are removed for $k \geq 3$?

见作业③。对于 k，有$2^{k-1}(k-1)! - 1$种情况。

Exercise 7.23. Prove that LS-CUT is a polynomial-time algorithm.

a) 如果有一个点度为 d，却只有不到 1/2 的度在 V_1,V_2 之间，那么我们能够在多项式时间找到它。

b) 初始$|E(V_1, V_2)| = 0$，而$|E(V_1, V_2)| <= m <= n^2$，且每次 Local Search 使之最少增加 1，所以是 polynomial time 的算法。

第 4 周：经典数据结构的拓展 - 面向应用

教学目的：理解数据结构与算法设计是统一的问题求解的两个侧面，以树为例子学习结合应用问题拓展数据结构的思想与技术。

阅读材料：*Dinesh Mehta, etc. (Ed.): Handbook of Data Structure and Applications, Chapman and Hall/CRC 2005*，第 3，15，21 章

1. 树作为最重要的数据结构之一，有着许多重要特性：层次结构，divide and conquer。
2. 这种层次结构即便对于一般图算法来说也很有意义。
3. BST 为例，解题时数据结构与算法有无明确界限？

确定了数据结构，往往确定了一类与之相关的算法，因为此类算法在此数据结构上最高效。

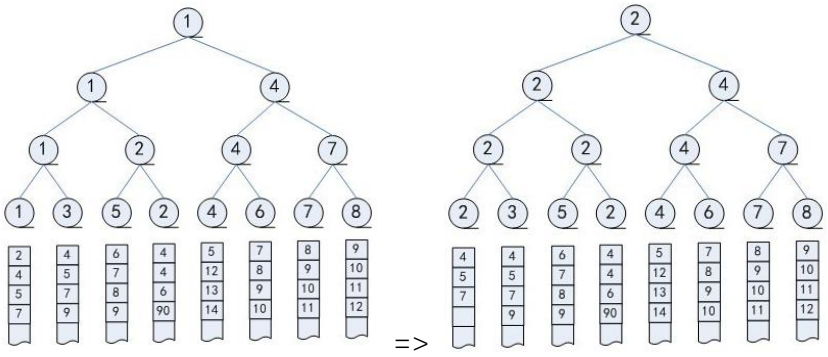
4. Balancing 针对什么需要？要求的不同技术考虑的主要因素是什么？

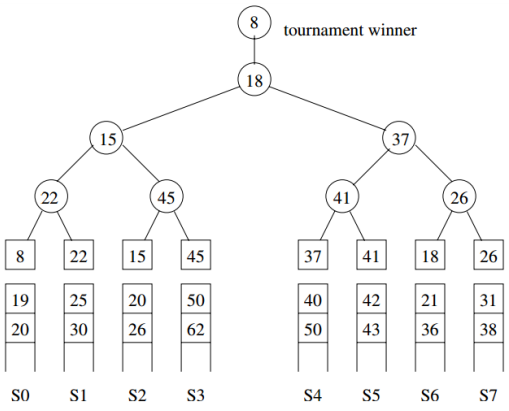
针对降低查询/搜索时间的需要，主要因素应该是搜索时间尽可能小。

5. 胜者树？败者树？（统称 Tournament Trees）

A winner tree is a complete binary tree in which each node represents the smaller of its two children. The root represents the smallest node in the tree。可用于 k 路归并。

记住以前比过的信息，下次只需要沿着某叶子往上比较一次就行了。





<= 败者树

在败者树中，用父结点记录其左右子结点进行比赛的败者，胜者参加下一轮的比赛。败者树的根结点记录的是败者，需要加一个结点来记录整个比赛的胜利者。

败者树简化了重构。败者树的重构只是与该结点的父结点的记录有关，而胜者树的重构还与该结点的兄弟结点有关。

如何记住胜者？ => 随着上升而记住

```
void Adjust (LoserTree &ls, int s) {
   // 沿从叶子结点 b[s]到根结点 ls[0]的路径调整败者树
   t = (s+k)/2;      // ls[t]是 b[s]的双亲结点
   while (t>0) {
      if (b[s].key > b[ls[t]].key) s←→ls[t];     // s 指示新的胜者
      t = t/2;
   }
   ls[0] = s;
}// Adjust
```

胜者产生后，新来的元素从下往上调整的过程。

6. 为什么似乎树是“变型”最多的？

树具有层次结构，这种层次结构在客观世界中广泛存在，可以应用到许多领域。

7. 从搜索树到 B 树，如何从 $\log_2 N$ 到 $\log_B N$ 的？

将二叉树增加分支数。has to be at least half full，保证存储效率。

8. 两个插入算法

Algorithm *InsertNotFull*(x, k, v)
Input: an in-memory page x of a B-tree, the key k and the value v of a new object.
Prerequisite: page x is not full.
Action: Insert the new object into the sub-tree rooted by x.

1. **if** x is a leaf page
   (a) Insert the new object into x, keeping objects in sorted order.
   (b) $DiskWrite(x)$.
2. **else**
   (a) Find the child pointer $x.child[i]$ whose key range contains k.
   (b) $w = DiskRead(x.child[i])$.
   (c) **if** w is full
      i. $y = AllocatePage()$.
      ii. Locate the middle object o_j stored in w. Move the objects to the right of o_j into y. If w is an index page, also move the child pointers accordingly.
      iii. Move o_j into x. Accordingly, add a child pointer in x (to the right of o_j) pointing to y.
      iv. $DiskWrite(x)$, $DiskWrite(y)$, $DiskWrite(w)$.
      v. If $k < o_j.key$, call $InsertNotFull(w, k, v)$; otherwise, call $InsertNotFull(y, k, v)$.
   (d) **else**
      $InsertNotFull(w, k, v)$.
   (e) **end if**
3. **end if**

Algorithm *Insert*($root$, k, v)
Input: *root* pageID of a B-tree, the key k and the value v of a new object.
Prerequisite: The object does not exist in the tree.
Action: Insert the new object into the B-tree.

1. $x = DiskRead(root)$.
2. **if** x is full
   (a) $y = AllocatePage()$, $z = AllocatePage()$.
   (b) Locate the middle object o_i stored in x. Move the objects to the left of o_i into y. Move the objects to the right of o_i into z. If x is an index page, also move the child pointers accordingly.
   (c) $x.child[1] = y.pageID$, $x.child[2] = z.pageID$.
   (d) $DiskWrite(x)$, $DiskWrite(y)$, $DiskWrite(z)$.
3. **end if**
4. $InsertNotFull(x, k, v)$.

9. 删除节点

1. 如果x是叶子页面，直接删除；
2. 如果x中没有k，读入含k的新页面y
   a. 如果y中对象数大于half-full，直接删除，否则
   b. 移动对象或者合并节点
3. 如果x中含k
   a. 如果k的前驱（或后继）子节点对象数大于half-full，则删除其中末位（或首位）元素，用其值替代k位置的值，如需要则递归
   b. 如果两者皆为half-full，则删除并合并

10. B+树与 B 树两个核心区别：

First, all objects in the B+-tree are kept in leaf nodes. 插入删除查找全都下放到叶子节点。

Second, all leaf nodes are linked together as a double-linked list. 允许高效顺序查找，同时保证随机二分查找。

树结构方便找“块”，顺序方便检查连续的对象。

11. B 树扩展到空间有哪些困难？ -- 属性的描述

12. R 树？MBR(Minimum Bounding Rectangle)体现了计算机问题求解哪些 concept？

R 树的核心思想是聚合距离相近的节点并在树结构的上一层将其表示为这些节点的最小外接矩形，这个最小外接矩形就成为上一层的一个节点。R 树的“R”代表“Rectangle（矩形）”。因为所有节点都在它们的最小外接矩形中，所以跟某个矩形不相交的查询就一定跟这个矩形中的所有节点都不相交。叶子节点上的每个矩形都代表一个对象，节点都是对象的聚合，并且越往上层聚合的对象就越多。Guttman 原始 R 树论文中的定义如下：

(1) Every leaf node contains between m and M index records unless it is the root
(2) For each index record (I, *tuple-identifier*) in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple
(3) Every non-leaf node has between m and M children unless it is the root
(4) For each entry (I, *child-pointer*) in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node
(5) The root node has at least two children unless it is a leaf
(6) All leaves appear on the same level

13. 为什么说 R 树源于 B 树？

14. 启发式的数据结构。

由于“划分矩形集到容量大于 2 的桶中，使得随机查询的期望的矩形相交数最小”是一个 NP-hard 问题，所以我们无法高效得知如何建立最优 R 树，但是因此也提出了许多启发式的算法，并在 2 维数据上取得了很好地效果。

15. heuristic 插入与 heuristic 分割

Algorithm ChooseLeaf Select a leaf node in which to place a new index entry *E*

CL1 [Initialize] Set *N* to be the root node

CL2 [Leaf check] If *N* is a leaf, return *N*.

CL3. [Choose subtree] If *N* is not a leaf, let *F* be the entry in *N* whose rectangle *FI* needs least enlargement to include *EI*. Resolve ties by choosing the entry with the rectangle of smallest area

CL4. [Descend until a leaf is reached.] Set *N* to be the child node pointed to by *Fp* and repeat from CL2

<= 插入

Split：

策略 1：brute-force, 穷尽一切可选项；
策略 2：选两个“种子”矩形分别给两个子集，然后按照“特定”顺序往每个子集内分配其它对象
策略 3：改变选“种子”的方法，后面加入的对象则没有“特定”顺序

16. 作业

a) 假设输入一个所有元素均不同的序列,其元素取自全序集,设计算法找出序列中第二大的元素,并分析其时间代价。

采用败者树，做到 $\log_2 N$ 的复杂度。

b) Describe a modified version of the B-tree insertion algorithm so that each time we create an overflow because of a split of a node *v*, we redistribute keys among all of *v*'s siblings, so that each sibling holds roughly the same number of keys (possibly cascading the split up to the parent of *v*). What

c) 写一个算法，在 R-树中加入一个矩形，可能需要分隔节点。讨论预期的代价。

见作业④。

第 5 周：传统算法的变形 – 追求更高效率

教学目的：理解算法课程的目的不是罗列用于不同问题的特定算法。以网络流问题及相关算法为例，理解对效率不断的追求，同时拓宽问题模型的适用性。

阅读材料：*Krishnaiyan Thulasiraman, etc. (Ed.): Handbook of Graph Theory, Combinatorial Optimization, and Algorithm, Chapman and Hall/CRC 2016*，第 4，5 章

1. 网络流模型？最大流算法形式化如下，关键限制条件。

Maximize v

subject to

$$\sum_{\{j:(i,j)\in A\}}^{t} x_{ij} - \sum_{\{j:(j,i)\in A\}}^{t} x_{ji} = \begin{cases} v, & i = s \\ 0, & i \neq s \text{ or } t \\ -v, & i = t \end{cases} \qquad i \in N \qquad (4.1)$$

$$0 \leq x_{ij} \leq u_{ij} \text{ for all } (i,j) \in A \qquad (4.2)$$

关键限制条件：容量限制，守恒条件。

2. 最大流最小割定理：

Theorem 4.2 (Max-flow min-cut theorem) *The maximum value of the flow from a source node s to a sink node t in a capacitated network equals the minimum capacity among all $s-t$ cuts.*

任何流的值不超过网络中任意割的容量，即$f_v \leq c(K)$，也即$f_v \leq f_v^* \leq c(\tilde{K}) \leq c(K)$。

如果流值达到割的容量，即$f_v = c(K)$，那么$f_v^* = c(\tilde{K})$，该流就是最大流，该割就是最小割。

3. 网络最大流求解算法：Ford-Fulkerson 算法

输入 给出一张边的容量为c的图$G = (V, E)$，源点s以及汇点t。

输出 输出在网络f中从s到t的最大流。

1. 对于图的每一条边(u, v)，在开始时流量$f(u, v) \leftarrow 0$。
2. 当在G_f中存在一条从s到t的路径p使对于每一条边$(u, v) \in p$都有$c_f(u, v) > 0$，：
    1. 找出$c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
    2. 对于每一条边$(u, v) \in p$
        1. $f(u, v) \leftarrow f(u, v) + c_f(p)$ (*在路径中“发送”流*)
        2. $f(v, u) \leftarrow f(v, u) - c_f(p)$ (*这个流有可能在之后被“发送”回来*)

4. 整性定理

Theorem 4.3 (Integrality theorem) *If all arc capacities are integer, then the maximum flow problem always has an integer maximum flow.*

The integrality theorem does not imply that every optimal solution of the maximum flow problem is integer. (The maximum flow problem might have non-integer solutions and, most often, it has such solutions. The integrality theorem shows that the problem always has at least one integer optimal solution.)

5. 通用增广路算法

Algorithm: Generic augmenting path

$x := 0$;

while $G(x)$ contains a directed path from node s to node t do

{

identify an augmenting path P from node s to node t;

$\delta(P) = \min\{r_{ij} : (i,j) \in P\}$;

augment $\delta(P)$ units of flow along path P and update $G(x)$;

}

The generic augmenting path algorithm solves the maximum flow problem in $O(nmU)$ time.

6. 改进的关键？ *U 可能很大！甚至到 2^n。*需要去除容量大小的影响。

7. 缺陷：a) 标号顺序任意，每次求出的增广路也任意，可能效率不高。b) 不一定能终止 c) forgetfulness。做出改进的三个算法如下：

particularly noteworthy: (1) the maximum capacity augmenting path algorithm, which always augments flow along the augmenting path with the maximum residual capacity and can be implemented to run in $O(m^2 \log U)$ time; (2) the capacity-scaling algorithm, which uses a scaling technique on arc capacities and can be implemented to run in $O(nm \log U)$ time; and (3) the shortest augmenting path algorithm, which augments flow along a shortest path (as measured by the number of arcs) in the residual network and runs in $O(n^2 m)$ time. Next,

(1) 每次找最大（残余）容量的增广路进行增广。

Therefore, the running time is $O(Am \log U)$, where A is the time to obtain the maximum capacity augmenting path in the residual network.

(2) 降低每次 iteration 开销，Capacity Scaling。

we augment flow along a path with a sufficiently large residual capacity instead of a path with the maximum augmenting capacity because we can obtain a path with a sufficiently large residual capacity fairly easily in $O(m)$ time. To define the capacity scaling algorithm,

Algorithm: Capacity scaling

$x := 0$;

$\Delta := 2^{\log U}$;

while $\Delta \geq 1$ do

  while $G(x, \Delta)$ contains a path from node s to node t do

    {

    identify a path P in $G(x, \Delta)$;

    augment $\min\{r_{ij} : (i,j) \in P\}$ units of flow along P and update $G(x, \Delta)$;

    }

  $\Delta := \Delta/2$;

The capacity-scaling algorithm solves the maximum flow problem within O(mlogU) augmentations and runs in $O(m^2 \log U)$ time.

Δ是什么？起什么作用？ 一个阈值，控制选的增广路残余容量不要太小。在Δ循环内，既然 bottleneck 至少是Δ，则每次 augment，流的值至少增加Δ。

(3) 每次找最短增广路进行增广。

Theorem 4.10 *The shortest augmenting path algorithm runs in $O(n^2 m)$ time.*

基于增广路思想的算法时间复杂度小结：

	# Augment Iterations	Time to Find an Augmenting Path	Best Running Time
Generic [6,7]	$O(nU)$	$O(m)$	$O(nmU)$
Capacity scaling [8,9]	$O(m \log U)$	$O(n)$	$O(nm \log U)$
Shortest [8,10]	$O(nm)$	$O(\log n)$	$O(nm \log n)$

8. 仍然不是传统意义上的多项式算法？阅读材料提出一种不同于 augment 思想的算法。

9. Preflow-Push 算法。

```
Start with labeling:   h(s) = n, h(t) = 0, h(v) = 0, for all other v
Start with preflow f:  f(e) = c(e) for e = (s, v), f(e) = 0, for all other edges e

While there is a node (other than t) with positive excess
  Pick a node v with excess(v) > 0
  If there is an edge (v, w) in Ef such that push(v, w) can be applied
    Push(v, w)
  Else
    Relabel(v)
```

Push(v, w): Applies if excess(v) > 0, h(w) < h(v), (v, w) in E_f
q = min(excess(v), c_f(v,w))
Add q to f(v, w)

Relabel(v): Applies if excess(v) > 0, for all w s.t (v, w) in E_f, h(w) >= h(v)
Increase h(v) by l

算法运行时间：

Theorem 4.16 *Generic preflow-push algorithm runs in $O(n^2m)$ time.*

变种 FIFO Preflow Push 算法，复杂度为：

Theorem 4.17 *The FIFO preflow-push algorithm runs in $O(n^3)$ time.*

单位容量最大流：

Theorem 4.19 *The unit capacity maximum flow algorithm establishes a maximum flow the in unit capacity simple networks in $O(\sqrt{nm})$ time.*

最快算法：The fastest strongly polynomial time algorithm is due to *A faster deterministic maximum flow algorithm* by King et.al.

10. 最小费用流？与最大流有何异同？

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{5.1}$$

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i), \text{ for all } i \in N, \tag{5.2}$$

$$0 \le x_{ij} \le u_{ij}, \text{ for all } (i,j) \in A \tag{5.3}$$

We associate with each node i ∈ N a number b(i) that indicates its supply or demand depending upon whether b(i) > 0 or b(i) < 0。

<= 另一种表示形式

$$\text{Minimize} \sum_{e\in A} c(e)f(e)$$

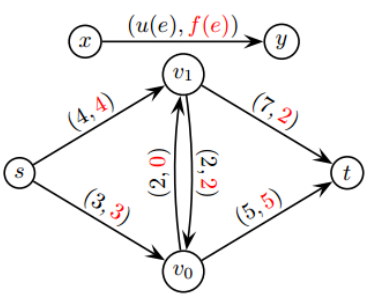
subject to

$$\sum_{e\in N^+(v)} f(e) - \sum_{e\in N^-(v)} f(e) = b(v) \quad \text{for all } v \in V$$

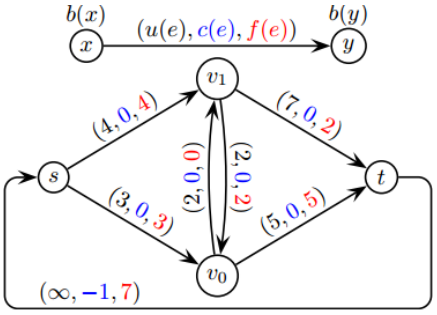
$$l(e) \leqslant f(e) \leqslant u(e) \quad \text{for all } e \in A$$

11. 问题的转换

maximum bound on the amount we can send though an edge. We can formulate this problem as a minimum cost flow problem in the following manner. We set $b(v) = 0$ for all $v \in V$, $c(e) = 0$ for all $e \in A$ and keep the maximum bound $u(e)$. We further introduce an arc (t, s) with cost $c(t, s) = -1$ and a flow bound $u(t, s) = \infty$. Then the minimum cost flow maximizes the flow on the arc (t, s); but since any flow on arc (t, s) must travel from node s to t through the other arcs of network, we will maximize the flow from s to t in the original network. Figure 2 summarizes the transformation on an example.



Network flow graph (maximum flow in red)



Network flow graph (minimum cost flow in red/blue)

Figure 2: Example of a maximum flow problem and its mapping to the minimum cost flow model

12. 如何将以下 Circulation with Demands 问题转化为最大流问题？

In this setting, we say that a *circulation* with demands $\{d_v\}$ is a function f that assigns a nonnegative real number to each edge and satisfies the following two conditions.

(i) *(Capacity conditions)* For each $e \in E$, we have $0 \le f(e) \le c_e$.

(ii) *(Demand conditions)* For each $v \in V$, we have v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Now, instead of considering a maximization problem, we are concerned with a *feasibility problem*: We want to know whether there *exists* a circulation that meets conditions (i) and (ii).

13. Much of the power of the problems on network flow model has essentially nothing to do with the fact that it *models traffic in a network*. Rather, it lies in the fact that many *problems with a nontrivial combinatorial search component* can be solved in polynomial time because they can be *reduced to* the problem of finding a minimum cost flow or a maximum flow in a directed graph.

14. 作业

Consider the following problem. You are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k.

The goal is to delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s-t flow in $G' = (V, E - F)$ is as small as possible subject to this.

Give a polynomial-time algorithm to solve this problem.

【Solution】删除割边，寻找最小割为$O(n^3)$，每次最大流减 1，所以时间复杂度为$O(|E| * n^3)$.

A car manufacturer has several manufacturing plants and produces several car models at each plant that it then ships to geographically dispersed retail centers throughout the country. Each retail center requests a specific number of cars of each model. The firm must determine the production plan of each model at each plant and a shipping pattern that satisfies the demands of each retail center and minimizes the overall cost of production and transportation.

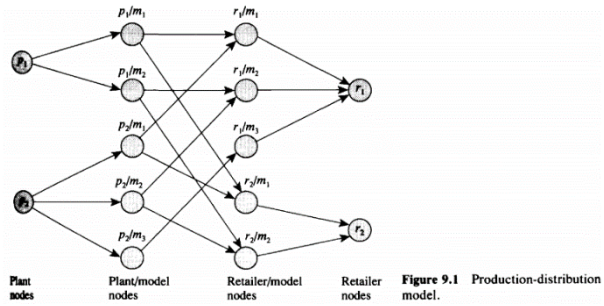


Figure 9.1 Production-distribution model.

【Solution】
如左图具有两个工厂，3 种模型。

设计 4 种节点:
(1) plant nodes, 代表工厂
(2) plant/model nodes 工厂生产的模型
(3) retailer/model nodes 零售商需求
(4) retailer nodes 代表零售商

设计三种边：
1. Production arcs. These arcs connect a plant node to a plant/model node; the cost of this arc is the cost of producing the model at that plant. We might place lower and upper bounds on these arcs to control for the minimum and maximum production of each particular car model at the plants.
2. Transportation arcs. These arcs connect plant/model nodes to retailer/model nodes; the cost of such an arc is the *total cost of shipping one car from the manufacturing plant to the retail center.* The transportation arcs might have *lower or upper bounds* imposed on their flows to model contractual agreements with shippers or capacities imposed on any distribution channel.
3. Demand arcs. These arcs connect retailer/model nodes to the retailer nodes. These arcs have *zero costs* and *positive lower bounds* which equal the demand of that model at that retail center.
Clearly, the production and shipping schedules for the automobile company correspond in a one-to-one fashion with the feasible flows in this network model. Consequently, a minimum cost flow would yield an optimal production and shipping schedule. 一种调度对应一个可行流，最小费用流对应最优调度。

第 6 周：限制问题 – 通过改变问题空间寻求“难”问题的解法

教学目的：理解问题空间的大小会直接影响问题的难度。通过限制条件压缩解空间是很自然的想法。本章将第 3 周提到的“解子问题”的思想技术化。

阅读材料：*Teofilo Gonzalez (Ed.): Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC 2007*，第 3 章

1. SubProblem? By a subproblem of a problem P we mean restricting the solution space for P by disallowing a subset of the feasible solutions.

2. Even when using the approximation ratio as the only evaluation criteria for an algorithm, it is not at all clear how to select a subproblem that can be solved quickly and from which a best possible solution could be generated.

3. Steiner 树

Steiner tree problem consists of a metric graph $G = (V, E, W)$ and a subset of vertices $T \subseteq V$. The problem is to find a tree that includes all the vertices in T plus some other vertices in the graph such that the sum of the weight of the edges in the tree is least possible. The Steiner tree problem in an NP-hard problem.

4. 网络中的最小斯坦纳树问题

MINIMUM STEINER PROBLEM IN NETWORKS:

Given: A connected network $N = (V,E,\ell)$, and a set $K \subseteq V$ of terminals.
Find: A Steiner minimum tree for K in N. That is, a Steiner tree T for K such that $\ell(T) = \min\{\ell(T') \mid T' \text{ is a Steiner tree for } K \text{ in } N\}$.

两点之间最短路问题即为两个点的 Steiner Tree 问题，最小生成树问题就是包含所有点的 Steiner Tree 问题。

5. 特例与子问题的区别？

特例是固定某些条件，而子问题是限制解空间？

6. 一个最小 Steiner 树中任意两个“分支”节点之间的通路应该满足什么性质？

7. 为什么引入距离图？

For set of terminals N, the minimum cost of a Steiner tree in G equals the minimum cost of a Steiner tree in the distance network of G (induced by V).

【诱导子图】

Formally, let G = (V, E) be any graph, and let S ⊂ V be any subset of vertices of G. Then the induced subgraph G[S] is the graph whose vertex set is S and whose edge set consists of all of the edges in E that have both endpoints in S.

也就是说，子图 G‘=（V’，E’），原图 G=（V，E）。V’是 V 的子集，且 E’中保留了所有 V 中对应的边。

8. 枚举算法

Algorithm 5.2 (ENUMERATION ALGORITHM)
Input: A network $N = (V,E,\ell)$ and a terminal set $K \subseteq V$.
Output: A Steiner minimum tree T for K.

(1) Compute the distance network $D(N) = (V,E_D,\ell_D)$ and store for each edge $\{u,w\}$ of E_D a shortest u-w path in N.
(2) Compute for all subsets S of $V \setminus K$ of size $|S| \leq k-2$ a minimum spanning tree for the subnetwork of $D(N)$ induced by $K \cup S$.
(3) Transform the shortest of the spanning trees encountered in step (2) into a Steiner tree T of the network N.

基本思想：Each Steiner minimum tree T corresponds to a minimum spanning tree in the subnetwork of D(N) induced by the set K ∪ S. (S 是 T 中分支节点的集合)

复杂度：$\mathcal{O}(n^2 \log n + nm + \min\{n^{k-2},2^{n-k}\} \cdot k^2)$

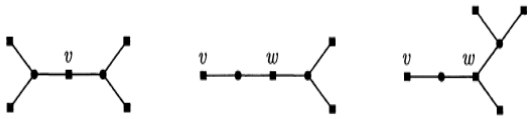
第二步 all subsets 总共：at most $\Sigma_{i=0}^{k-2}\binom{n-k}{i} \leq \min\{n^{k-2},2^{n-k}\}$

9. Steiner Tree 的动态规划算法

DREYFUS-WAGNER ALGORITHM
Input: A network $N = (V,E,\ell)$ and a terminal set $K \subseteq V$.
Output: The length of a Steiner minimum tree T for K.
{ *Initialization* }
for all $v,w \in V$ **do**
  compute $p(v,w)$;
for all $\{x,y\} \subseteq K$ **do**
  $s(\{x,y\}) := p(x,y)$;
{ *Perform recursion according to Lemma 5.5* }
for $i = 2$ **to** $k-1$ **do**
  for all $X \subseteq K$ with $|X| = i$ and all $v \in V \setminus X$ **do**
    $s_v(X \cup \{v\}) := \min_{\emptyset \neq X' \subseteq X} \{s(X' \cup \{v\}) + s((X \setminus X') \cup \{v\})\}$;
  for all $X \subseteq K$ with $|X| = i$ and all $v \in V \setminus X$ **do**
    $s(X \cup \{v\}) := \min\{\min_{w \in X}\{p(v,w) + s(X)\}, \min_{w \in V \setminus X}\{p(v,w) + s_w(X \cup \{w\})\}\}$;

递归 1：v 在 steiner 树中至少是 2 次

递归 2：如果 v 是叶结点，在 Steiner 树中以 v 为起点的最长无分支通路的另一端 w 可能是 terminal, 也可能不是（那一定是分支结点）。否则可以归入(5.1), 即 v=w。



复杂度为 $\mathcal{O}(3^k n + 2^k n^2 + n^2 \log n + nm)$

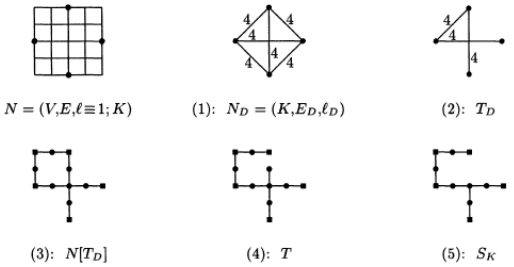
10. Steiner Tree 的最小生成树逼近算法

Algorithm: MST-STEINER
Input: A graph $G = (V, E, w)$ and a terminal set $L \subset V$.
Output: A Steiner tree T.
1: Construct the metric closure G_L on the terminal set L.
2: Find an MST T_L on G_L.
3: $T \leftarrow \emptyset$.
4: **for** each edge $e = (u, v) \in E(T_L)$ in a depth-first-search order of T_L **do**
4.1: Find a shortest path P from u to v on G.
4.2: **if** P contains less than two vertices in T **then**
Add P to T;
else
Let p_i and p_j be the first and the last vertices already in T;
Add subpaths from u to p_i and from p_j to v to T.
5: Output T.

Theorem 1: The MST-STEINER algorithm finds a 2-approximation of an SMT of a general graph in $O(|V||L|^2)$ time, where V is the vertex set and L is the terminal set. Furthermore, the ratio is asymptotically tight.

距离图表示的 MST 解法：

Algorithm 6.3 (MST-ALGORITHM)
Input: Network $N = (V,E,\ell;K)$.
Output: Steiner tree S_K for N.
(1) Compute the distance network $N_D = (K,E_D,\ell_D)$.
(2) Compute a minimum spanning tree T_D in N_D.
(3) Transform T_D into a subnetwork $N[T_D]$ of N by replacing every edge of T_D by the corresponding shortest path.
(4) Compute a minimum spanning tree T for the subnetwork $N[T_D]$.
(5) Transform T into a Steiner tree S_K for N by successively deleting leaves which are no terminals.



$N = (V,E,\ell \equiv 1;K)$ (1): $N_D = (K,E_D,\ell_D)$ (2): T_D <= 例子

(3): $N[T_D]$ (4): T (5): S_K

11. 作业

Modify the Dreyfus-Wagner algorithm in such a way that it computes not only the length of a Steiner minimum tree for T, but also the tree itself.

计算p(v, w)时，采用 floyd-warshell 算法保存最短路径。

对每个 X', v 建立一颗临时树，达到最小 Steiner 树时，将其赋予当前最优树。

Modify step (3) of the MST-ALGORITHM as follows:

(3') Compute the set S of all vertices which are contained in the shortest paths corresponding to the edges in T_D. Let $N[T_D]$ denote the subnetwork induced by the set S.

Let S'_K denote the Steiner tree which is computed by the modified algorithm, while S_K denotes the Steiner tree computed by the original algorithm. Is it true that $\ell(S'_K) \leq \ell(S_K)$? Prove it or give a counterexample.

见作业⑥

Show that, assuming $\mathcal{P} \neq \mathcal{NP}$, there cannot exist a constant $B \in \mathbb{N}$ and an algorithm $\mathcal{A}$ that returns for every network $N = (V,E,\ell;K)$ in polynomial time a Steiner tree $S_\mathcal{A}$ such that $\ell(S_\mathcal{A}) \leq \ell(S_{\text{opt}}) + B$, where S_{opt} is a Steiner minimum tree in N.

见作业⑥，扩大 k 倍，$l(S_A)$，$l(S_{opt})$相应扩大 k 倍，算法 A 必须找到误差小于 B/k 的解。

第 7 周：贪心策略运用

教学目的：理解最基本的算法策略可以在解难题时产生明显效果，同时在新的背景下，简单的贪心策略可以演化为有效的新策略。

阅读材料：*David Williamson, etc.: The Design of Approximation Algorithm, Cambridge University Press 2011*，第 2 章

1. MakeSpan 调度问题，目标：多个任务，多个机器，如何调度任务到机器上执行，使得总执行时间（第一个任务开始执行到最后一个任务执行完）最短。

Input: Positive integers $p_1, p_2, \ldots, p_n$ and an integer $m \geq 2$ for some $n \in \mathbb{N} - \{0\}$.
{p_i is the processing time of the ith job on any of the m available machines}.

Constraints: For every input instance $(p_1, \ldots, p_n, m)$ of MS,
$\mathcal{M}(p_1, \ldots, p_n, m) = \{S_1, S_2, \ldots, S_m \mid S_i \subseteq \{1, 2, \ldots, n\}$ for $i = 1, \ldots, m$, $\bigcup_{k=1}^{m} S_k = \{1, 2, \ldots, n\}$, and $S_i \cap S_j = \emptyset$ for $i \neq j\}$.
{$\mathcal{M}(p_1, \ldots, p_n, m)$ contains all partitions of $\{1, 2, \ldots, n\}$ into m subsets. The meaning of $(S_1, S_2, \ldots, S_m)$ is that, for $i = 1, \ldots, m$, the jobs with indices from S_i have to be processed on the ith machine}.

Costs: For each $(S_1, S_2, \ldots, S_m) \in \mathcal{M}(p_1, \ldots, p_n, m)$,
$cost((S_1, \ldots, S_m), (p_1, \ldots, p_n, m)) = \max\{\sum_{l \in S_i} p_l \mid i = 1, \ldots, m\}$

Goal: *minimum*.

2. MakeSpan 哪些方面简化了调度问题？
将任务抽象成任务的执行时间，将机器抽象成包含任务的集合，机器间完全并行，略去任务切换，资源占用等内容，只考虑执行时间。

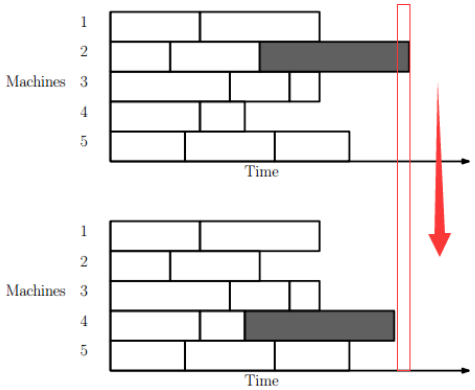
3. 总统选举，有可能平局吗？ => 两台机器的 makespan 问题？

Given a list $A = (a_1, a_2, \ldots, a_n)$ of n integers, can A be partitioned into A_1 and A_2 such that $\sum_{a_j \in A_1} a_j = \sum_{a_j \in A_2} a_j = \frac{1}{2}\sum a_j$?

4. 最优解的下限？

$$Opt_{\mathrm{MS}}(I) \geq \frac{\sum_{i=1}^{n} p_i}{m}$$

5. Local Search?



Local search algorithms are defined by a set of local changes or local moves that ***change one feasible solution to another***.

The simplest local search procedure for this scheduling problem works as follows: Start with any schedule; consider the job ℓ that finishes last; check whether or not there exists a machine to which it can be reassigned that would cause this job to finish earlier. If so, transfer job ℓ to this other machine.

会终止么？解的质量如何？

几个关键的问题：

a) 为什么每次移动任务后，C_{max} 不会增加？ 如果增加，不会移动。

b) 为什么可以假设每次移动的目标机器一定是当前结束运行时间最早的（此时间记为 C_{min}）？如果目标不是最小的，那么存在更小的，移动到该机器上会使得 C_{max} 尽可能小。

c) 为什么算法执行过程中 C_{min} 不会减小？ Cmax 减少，

d) [**最关键的问题**] 为什么任何任务不可能被移动两次？

Theorem 2.5: *The local search procedure for scheduling jobs on identical parallel machines is a 2-approximation algorithm.*

机器都忙的时间 $S_l \leq$ Cmax，最后一个任务执行时间 $p_l \leq$ Cmax，所以总的时间<=2C_{max}。

6. list scheduling 算法：有空闲机器就放上去。

Theorem 2.6: *The list scheduling algorithm for the problem of minimizing the makespan on m identical parallel machines is a 2-approximation algorithm.*

7. LPT(*longest processing time*)算法：处理时间最长的优先放。

Theorem 2.7: *The longest processing time rule is a 4/3-approximation algorithm for scheduling jobs to minimize the makespan on identical parallel machines.*

Proof. Suppose that the theorem is false, and consider an input that provides a counterexample to the theorem. For ease of notation, assume that $p_1 \geq \cdots \geq p_n$. First, we can assume that the last job to complete is indeed the last (and smallest) job in the list. This follows without loss of generality: any counterexample for which the last job ℓ to complete is not the smallest can yield a smaller counterexample, simply by omitting all of the jobs $\ell+1, \ldots, n$; the length of the schedule produced is the same, and the optimal value of the reduced input can be no larger. Hence the reduced input is also a counterexample.

So we know that the last job to complete in the schedule is job n. If this is a counterexample, what do we know about $p_n (= p_\ell)$? If $p_\ell \leq C^*_{\max}/3$, then the analysis of Theorem 2.6 implies that the schedule length is at most $(4/3)C^*_{\max}$, and so this is not a counterexample. Hence, we know that in this purported counterexample, job n (and therefore all of the jobs) has a processing requirement strictly greater than $C^*_{\max}/3$. This has the following simple corollary. In the optimal schedule, each machine may process at most two jobs (since otherwise the total processing assigned to that machine is more than $C^*_{\max}$).

However, we have now reduced our assumed counterexample to the point where it simply cannot exist. For inputs of this structure, we have the following lemma.

<= 证明 LPT 算法是 4/3-approximation 的。

8. 有 deadline 的 makespan：n 个任务，1 台机器，任务有最早开始时间和 deadline，最小化总延迟。

EDD schedule：选择预期完成时间最早，且开始时间不小于 time 值的任务。*(2-approximation)*

S denote a subset of jobs, and let $r(S) = \min_{j \in S} r_j$, $p(S) = \sum_{j \in S} p_j$, and $d(S) = \max_{j \in S} d_j$.

Lemma 2.1: *For each subset S of jobs,*

$$L^*_{\max} \geq r(S) + p(S) - d(S).$$

最晚的任务结束时间大于等于 r(S)+p(S)，其 deadline 等于或早于 d(S)。

为什么只考虑特定子集 S 的情况？ => 为证明任意调度都满足此式？

9. 局部搜索与贪心算法的异同？

局部搜索先求出一个可行解，然后慢慢搜索，达到局部最优解。

贪心策略一开始就制定策略/规则，并安排好输入的格式，然后遵循策略一遍得出最优解。

局部搜索中搜索一个最优邻居也包含着贪心的思想。（局部贪心）

10. 对于 Makespan 问题，能否从一个较优的而不是任意的可行解开始逐步改进？GMS!

Greedy Makespan Schedule

Input: $I = (p_1, \ldots, p_n, m)$, n, m, $p_1, \ldots, p_n$ positive integers and $m \geq 2$.
Step 1: Sort $p_1, \ldots, p_n$.
排序
To simplify the notation we assume $p_1 \geq p_2 \geq \cdots \geq p_n$ in the rest of the algorithm.
Step 2: **for** $i = 1$ **to** m **do**
先把 m 个"大"任务安排了
begin $T_i := \{i\}$;
$Time(T_i) := p_i$
end
{In the initialization step the m largest jobs are distributed to the m machines. At the end, T_i should contain the indices of all jobs assigned to the ith machine for $i = 1, \ldots, m$.}
Step 3: **for** $i = m + 1$ **to** n **do**
先分配当前负担最小的机器
begin compute an l such that
$Time(T_l) := \min\{Time(T_j) | 1 \leq j \leq m\}$;
$T_l := T_l \cup \{i\}$;
$Time(T_l) := Time(T_l) + p_i$
end
Output: $(T_1, T_2, \ldots, T_m)$.

"双重贪心"
为什么阅读材料中说如果用这个算法的结果为 local search 的起点，local search 立即可以结束了？

把后面 Local Search 需要做的移动都在前面做了。

11. 作业

2.2 Prove Lemma 2.8: show that for any input to the problem of minimizing the makespan on identical parallel machines for which the processing requirement of each job is more than one-third the optimal makespan, the longest processing time rule computes an optimal schedule.

见作业⑦

2.3 We consider scheduling jobs on identical machines as in Section 2.3, but jobs are now subject to *precedence constraints*. We say $i \prec j$ if in any feasible schedule, job i must be completely processed before job j begins processing. A natural variant on the list scheduling algorithm is one in which whenever a machine becomes idle, then any remaining job that is *available* is assigned to start processing on that machine. A job j is available if all jobs i such that $i \prec j$ have already been completely processed. Show that this list scheduling algorithm is a 2-approximation algorithm for the problem with precedence constraints.

见作业⑦，关键在于，从$t_i + P_i$到t_{i+1}这段时间内，所有机器都是忙的，这些忙时总处理工作量为：$m(t_1 + \sum_{i=1}^{l-1}[t_{i+1} - (t_i + P_i)] \leq \sum P_i - m\sum_{i=1}^{l-1} P_i \leq mC_{max}^{OPT} - m\sum_{i=1}^{l-1} P_i$
从而$t_l = t_1 + \sum_{i=1}^{l-1}[t_{i+1} - t_i] \leq C_{max}^{OPT}$，从而$C_{max} = t_l + P_l \leq 2C_{max}^{OPT}$

第 8 周：近似算法的基本概念与基本分析方法

教学目的：在前面两周通过一些具体技术与示例获得感性认识的基础上，系统理解近似算法的基本分析方法，并理解可能提高近似率的基本原理。

阅读材料：*G.Ausiello, etc.: Complexity and Approximation, Springer-Verlag 1999*，第 3 章

1. 近似算法？

Given an optimization problem $\mathcal{P} = (I, \mathrm{SOL}, m, \mathrm{goal})$, *an algorithm* $\mathcal{A}$ *is an* approximation algorithm *for* $\mathcal{P}$ *if, for any given instance* $x \in I$, *it returns an* approximate solution, *that is a feasible solution* $\mathcal{A}(x) \in \mathrm{SOL}(x)$.

近似算法又分为绝对近似 $D(x, \mathcal{A}(x)) \le k.$ 和相对近似 $E(x, \mathcal{A}(x)) \le \varepsilon.$ (*ε-approximation*)。

相对误差：$E(x,y) = \dfrac{|m^*(x) - m(x,y)|}{\max\{m^*(x), m(x,y)\}}.$ ，Performance Ratio：$R(x,y) = \max\left(\dfrac{m(x,y)}{m^*(x)}, \dfrac{m^*(x)}{m(x,y)}\right)$

Given an optimization problem $\mathcal{P}$ *and an approximation algorithm* $\mathcal{A}$ *for* $\mathcal{P}$, *we say that* $\mathcal{A}$ *is an* r-approximate algorithm *for* $\mathcal{P}$ *if, given any input instance* x *of* $\mathcal{P}$, *the performance ratio of the approximate solution* $\mathcal{A}(x)$ *is bounded by* r, *that is:*

$$R(x, \mathcal{A}(x)) \le r.$$

阅读材料中主要使用的评价标准是 performance ratio，满足条件的称为：*r-approximation* algos.

$$R(x,y) = \frac{1}{1 - E(x,y)}$$

2. 优化问题

An optimization problem $\mathcal{P}$ *is characterized by the following quadruple of objects* $(I_{\mathcal{P}}, \mathrm{SOL}_{\mathcal{P}}, m_{\mathcal{P}}, \mathrm{goal}_{\mathcal{P}})$, *where:*

1. $I_{\mathcal{P}}$ *is the set of instances of* $\mathcal{P}$;
2. $\mathrm{SOL}_{\mathcal{P}}$ *is a function that associates to any input instance* $x \in I_{\mathcal{P}}$ *the set of* feasible solutions *of* x;
3. $m_{\mathcal{P}}$ *is the* measure function, *defined for pairs* (x,y) *such that* $x \in I_{\mathcal{P}}$ *and* $y \in \mathrm{SOL}_{\mathcal{P}}(x)$. *For every such pair* (x,y), $m_{\mathcal{P}}(x,y)$ *provides a positive integer which is the value of the feasible solution* y;
4. $\mathrm{goal}_{\mathcal{P}} \in \{\mathrm{MIN}, \mathrm{MAX}\}$ *specifies whether* $\mathcal{P}$ *is a maximization or a minimization problem.*

最优解 $\mathrm{SOL}^*_{\mathcal{P}}(x)$ ：for every $y^*(x)$ such that $y^*(x) \in \mathrm{SOL}^*_{\mathcal{P}}(x)$:

$$m_{\mathcal{P}}(x, y^*(x)) = \mathrm{goal}_{\mathcal{P}}\{v \mid v = m_{\mathcal{P}}(x,z) \wedge z \in \mathrm{SOL}_{\mathcal{P}}(x)\}$$

3. 从三个侧面看优化问题？

Constructive Problem ($\mathcal{P}_C$) – Given an instance $x \in I$, derive an optimal solution $y^*(x) \in \mathrm{SOL}^*(x)$ and its measure $m^*(x)$.

Evaluation Problem ($\mathcal{P}_E$) – Given an instance $x \in I$, derive the value $m^*(x)$.

Decision Problem ($\mathcal{P}_D$) – Given an instance $x \in I$ and a positive integer $K \in \mathbb{Z}^+$, decide whether $m^*(x) \ge K$ (if goal = MAX) or whether $m^*(x) \le K$ (if goal = MIN). If goal = MAX, the set $\{(x,K) \mid x \in I \wedge m^*(x) \ge K\}$ (or $\{(x,K) \mid x \in I \wedge m^*(x) \le K\}$ if goal = MIN) is called the *underlying language* of $\mathcal{P}$.

什么叫 underlying language?

4. 为什么说判定问题不可能比相应的优化问题更“难”？
判定问题可解，优化问题就可解。

5. 从 NP 到 NPO

An optimization problem $\mathcal{P} = (I, \mathrm{SOL}, m, \mathrm{goal})$ *belongs to the class* NPO *if the following holds:*

1. *the set of instances* I *is recognizable in polynomial time;*
2. *there exists a polynomial* q *such that, given an instance* $x \in I$*, for any* $y \in \mathrm{SOL}(x)$*,* $|y| \leq q(|x|)$ *and, besides, for any* y *such that* $|y| \leq q(|x|)$*, it is decidable in polynomial time whether* $y \in \mathrm{SOL}(x)$*;*
3. *the measure function* m *is computable in polynomial time.*

如何体现非确定性“N (non deterministic)”？

为什么背包问题是 NPO 问题？

6. For any ontimization problem P in NPO, the corresponding decision problem P_D belongs to NP.
原来的意义下，不能说一个优化问题是 NP 问题，只能说讨论其对应的判断问题是不是 NP 问题，或者 NPC，NP-hard 问题。

7. P in NPO, if the *underlying language* of P is NPC, then P is NP-hard. Why?

8. 解背包问题的贪心算法：按性价比从大到小排序。

9. APX 问题类

APX *is the class of all* NPO *problems* $\mathcal{P}$ *such that, for some* $r \geq 1$*, there exists a polynomial-time* r*-approximate algorithm for* $\mathcal{P}$*.*

◄ Definition 3.9 *Class APX*

APX 是一类 NPO 问题，这些问题存在一个多项式时间的 r 逼近算法(r>=1)。
还有一类非 APX 的 NPO 问题，不存在任何 r>=1 的多项式 r 逼近算法。

10. PO 问题如下，NPO vs. PO related to NP vs. P. if P!=NP then PO!=NPO.

An optimization problem $\mathcal{P}$ *belongs to the class* PO *if it is in* NPO *and there exists a polynomial-time computable algorithm* $\mathcal{A}$ *that, for any instance* $x \in I$*, returns an optimal solution* $y \in \mathrm{SOL}^*(x)$*, together with its value* $m^*(x)$*.*

◄ Definition 1.18 *Class PO*

11. PTAS（Polynomial-Time Algorithm Scheme）. Note that *APX contains PTAS* if P!=NP.

Definition 3.10 *Polynomial-time approximation scheme* ► *Let* $\mathcal{P}$ *be an* NPO *problem. An algorithm* $\mathcal{A}$ *is said to be a* polynomial-time approximation scheme (PTAS) *for* $\mathcal{P}$ *if, for any instance* x *of* $\mathcal{P}$ *and any rational value* $r > 1$*,* $\mathcal{A}$ *when applied to input* (x, r) *returns an* r*-approximate solution of* x *in time polynomial in* $|x|$*.*

单纯找|x|的多项式方案可能计算代价巨大，或可达到 n 的$1/(r-1)$次方级别，heavy! => FPTAS

12. FPTAS（Fully Polynomial-Time Algorithm Scheme）. FPTAS *belongs to* PTAS if P!=NP.

Let $\mathcal{P}$ *be an* NPO *problem. An algorithm* $\mathcal{A}$ *is said to be a* fully polynomial-time approximation scheme (FPTAS) *for* $\mathcal{P}$ *if, for any instance* x *of* $\mathcal{P}$ *and for any rational value* $r > 1$*,* $\mathcal{A}$ *with input* (x, r) *returns an* r*-approximate solution of* x *in time polynomial both in* $|x|$ *and in* $1/(r-1)$*.*

◄ Definition 3.12 *Fully polynomial-time approximation scheme*

13. Gap Technique

Let $\mathcal{P}'$ be an NP*-complete decision problem and let $\mathcal{P}$ be an* NPO *minimization problem. Let us suppose that there exist two polynomial-time computable functions $f : I_{\mathcal{P}'} \mapsto I_{\mathcal{P}}$ and $c : I_{\mathcal{P}'} \mapsto \mathbf{N}$ and a constant $\mathrm{gap} > 0$, such that, for any instance x of $\mathcal{P}'$,*

$$m^*(f(x)) = \begin{cases} c(x) & \text{if } x \text{ is a positive instance,} \\ c(x)(1+\mathrm{gap}) & \text{otherwise.} \end{cases}$$

Then no polynomial-time r-approximate algorithm for $\mathcal{P}$ with $r < 1+\mathrm{gap}$ can exist, unless P = NP.

f 函数究竟是什么？

将P'问题的输入转化成 P 问题的输入？

14. 一个显然属于 APX 的问题。

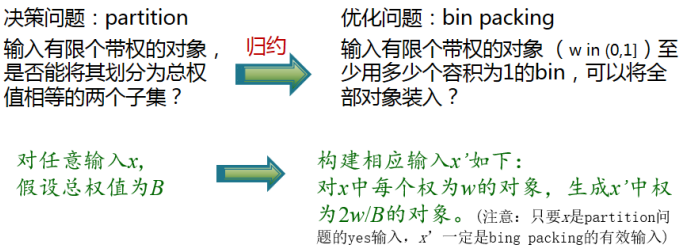
Problem 2.4: Minimum Bin Packing

INSTANCE: Finite multiset I of rational numbers $\{a_1, a_2, \ldots, a_n\}$ with $a_i \in (0, 1]$ for $i = 1, \ldots, n$ (in a multiset the same number may appear more than once).

SOLUTION: A partition $\{B_1, B_2, \ldots, B_k\}$ of I such that $\sum_{a_i \in B_j} a_i \leq 1$ for $j = 1, \ldots, k$.

MEASURE: The cardinality of the partition, i.e., k.

15. 利用 Gap 技术可将 Partition 问题归约到 Bin Packing 问题



利用输入的变换，将问题归约，2w/B 的 2 是为了使得 bin packing 输入集合至少占据 2 个 bin。

显然：partition输出*yes*或*no*，对应于bin packing输出 2或3。即gap为1/2，因此bin packing问题近似率不可能小于3/2，除非...

16. 背包问题 MAXIMUM KNAPSACK 全新 Approximation 框架：

Program 2.9: Knapsack Approximation Scheme

input Set X of n items, for each $x_i \in X$, values p_i, a_i, positive integer b, rational number $r > 1$;
output Subset $Y \subseteq X$ such that $\sum_{x_i \in Y} a_i \leq b$;
begin
  p_{max} := maximum among values p_i;
  $t := \lfloor \log(\frac{r-1}{r} \frac{p_{max}}{n}) \rfloor$;
  x' := instance with profits $p'_i = \lfloor p_i / 2^t \rfloor$;
  remove from x' items with zero profit;
  Y := solution returned by Program 2.8 with input x';
  return Y
end.

Dynamic Programming

分析其近似率与计算复杂度？

(Hint) the largest additive error introduced by pruning the profit is at most 2^t.

$$\frac{m^*(x) - m_{AS}(x,r)}{m^*(x)} \leq \frac{n2^t}{p_{max}}$$

$$O(n\sum_{i=1}^{n} p'_i) = O(n\sum_{i=1}^{n} p_i/2^t)$$

17. 作业

Exercise 3.8 Prove that, for any NPO minimization problem $\mathcal{P}$, if there exists a constant k such that it is NP-hard to decide whether, given an instance x, $m^*(x) \leq k$, then no polynomial-time r-approximate algorithm for $\mathcal{P}$ with $r < (k+1)/k$ can exist, unless P = NP.

见作业⑧。若存在 Poly 算法 A(x) < (1+1/k)m*(x)，分 case：m*(x) >= k 和 m*(x) < k。得知

A(x)可解决 NP-hard 问题，P=NP，矛盾。

Exercise 3.10 By making use of a technique similar to the one used for MINIMUM PARTITION, show that, for any integer k, there is a $k/(k+1)$-approximate algorithm for MAXIMUM KNAPSACK.

Program 3.3: Partition PTAS

```
input Set of items X with integer weights a_i, rational r > 1;
output Partition of X into two sets Y_1 and Y_2;
begin
  if r ≥ 2 then return X, ∅
  else
  begin
    Sort items in non-increasing order with respect to their weight;
    (*Let (x_1, ..., x_n) be the obtained sequence*)
    k(r) := ⌈(2 − r)/(r − 1)⌉;
    (* First phase *)
    Find an optimal partition Y_1, Y_2 of x_1, ..., x_k(r);
    (* Second phase *)
    for j := k(r) + 1 to n do
      if Σ_{x_i∈Y_1} a_i ≤ Σ_{x_i∈Y_2} a_i then
        Y_1 := Y_1 ∪ {x_j}
      else
        Y_2 := Y_2 ∪ {x_j};
    return Y_1, Y_2
  end;
end.
```

Exercise 3.13 An NPO problem $\mathcal{P}$ is *simple* if, for every positive integer k, the problem of deciding whether an instance x of $\mathcal{P}$ has optimal measure at most k is in P. Prove that MAXIMUM CLIQUE is simple and that MINIMUM GRAPH COLORING is not simple (unless P = NP).

对于 MAXIMUM CLIQUE 问题，判断解是否‘at most k’是容易的，只需C_n^k取出 k 个点判断是否形成 CLIQUE，对 n>=r>k，选取 r 个点判断是否不是 CLIQUE，所以是 simple 的。而对于 MINIMUM GRAPH COLORING 问题，设 m*(x)为最优解，由于 k>=3 时，判断 m*(x)<=k 是 NPC 问题，所以对于 MINIMUM GRAPH COLORING 问题，判断 m*(x)<=k 是 NP-hard 的，所以 not simple.

第 9 周：设计近似算法的典型方法与应用

教学目的：通过典型示例从近似解质量的角度理解问题的难度，并理解稳定性对提高近似算法适用性的影响。同时了解相关的分析技术。

阅读材料：*Juraq Hromkovic: Algorithmics for Hard Problems, Springer-Verlag 2004*，第 4.3.5 节

1. 我们是如何根据可能找到的近似解的情况来给 NPO 分类的？
APX：能找到 r > 1 的多项式 r 近似算法
PTAS：算法在|x|的多项式时间内能得 r 近似解
FPTAS：算法在|x|和$1/(r-1)$的多项式时间内能得 r 近似解。
2. 前面我们是如何用 gap 技术证明 TSP 不是 PTAS 的？

3. 某些子问题未必有那么难。

Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^+$ satisfying the triangle inequality

$$c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$$

for all three different $u, v, w \in V$ {i.e., $(G, c) \in L_\Delta$}.
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: Choose an arbitrary vertex $v \in V$. Perform depth-first-search of T from v, and order the vertices in the order that they are visited. Let H be the resulting sequence.
Output: The Hamiltonian tour $\overline{H} = H, v$.

子问题Δ-TSP

<= *Algorithm 4.3.5.1*
为什么是 2-近似的？
从欧拉回路的角度还可以做哪些改进？
可以看到，算法执行对于顶点或边的处理顺序对结果产生影响(dfs)。

4. CHRISTOFIDES ALGORITHM

Algorithm 4.3.5.4. CHRISTOFIDES ALGORITHM

Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^+$ satisfying the triangle inequality.
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: $S := \{v \in V \mid deg_T(v) \text{ is odd}\}$.
Step 3: Compute a minimum-weight[21] perfect[22] matching M on S in G.
Step 4: Create the multigraph $G' = (V, E(T) \cup M)$ and construct an Eulerian tour ω in G'.
Step 5: Construct a Hamiltonian tour H of G by shortening ω (i.e., by removing all repetitions of the occurrences of every vertex in ω in one run via ω from the left to the right).
Output: H.

Theorem 4.3.5.5. *The* CHRISTOFIDES ALGORITHM *is a polynomial-time* 1.5*-approximation algorithm for* △-TSP.

能否在多项式时间完成每一步？

（最小权完美匹配可用 KM 算法解决，存在 $O(n^3)$实现）

5. 前面说的 2-approximation 和这里的 1.5-approximation 都是 tight，什么意思？
在最坏情况下也不会超过 r，只会随着问题规模的增大而渐进 r？

6. 有时并不遵循 triangle inequality，但是也不会偏离太远，那么如何有效得出解？近似率随着偏离 Δ 条件的多少而如何增加？

7. 关于近似算法稳定性的定义，stable，p-stable。

Let h be a distance function for $\overline{U}$ according to L_I. We define, for any $r \in \mathbb{R}^+$,

$$\boldsymbol{Ball_{r,h}(L_I)} = \{w \in L \mid h(w) \leq r\}.^6$$

Let A be a consistent algorithm for $\overline{U}$, and let A be an ε-approximation algorithm for U for some $\varepsilon \in \mathbb{R}^{>1}$. Let p be a positive real. We say that A is ***p*-stable according to *h*** *if, for every real $0 < r \leq p$, there exists a $\delta_{r,\varepsilon} \in \mathbb{R}^{>1}$ such that A is a $\delta_{r,\varepsilon}$-approximation algorithm for $U_r = (\Sigma_I, \Sigma_O, L, Ball_{r,h}(L_I), \mathcal{M}, cost, goal)$.*

A is **stable according to h** *if A is p-stable according to h for every $p \in \mathbb{R}^+$. We say that A is* **unstable according to *h*** *if A is not p-stable for any $p \in \mathbb{R}^+$.*

8. (1+r)-relaxed Δ-TSP，离Δ-TSP 有多远？

We define for every $x = (G, c) \in L$,

$$dist(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{c(\{u,p\}) + c(\{p,v\})} - 1 \,\middle|\, u,v,p \in V(G)\right\}\right\},$$

$(1+r)$-relaxed triangle inequality

$$c(\{u,v\}) \leq (1+r)[c(\{u,w\}) + c(\{w,v\})]$$

$$distance(x) = \max\left\{0, \max\left\{\frac{c(\{u,v\})}{\sum_{i=1}^{m} c(\{p_i, p_{i+1}\})} - 1 \,\middle|\, u,v \in V(G), \text{ and } u = p_1, p_2, \ldots, p_{m+1} = v \text{ is a simple path between } u \text{ and } v \text{ in } G\right\}\right\}.$$

哪种更强？distance.

the CHRISTOFIDES ALGORITHM according to *dist*, the main problem is that shortening a path $u_1, u_2, \ldots, u_{m+1}$ to the edge u_1, u_{m+1} can lead to

$$cost(\{u_1, u_{m+1}\}) = (1+r)^{\lceil \log_2 m \rceil} \cdot cost(u_1, u_2, \ldots, u_{m+1}).$$

This can increase the cost of the constructed Hamiltonian path by the multiplicative factor $(1+r)^{\lceil \log_2 n \rceil}$ in the comparison with the cost of the Eulerian tour. The rough idea, then, is to construct a Hamiltonian tour by shortening only short paths of the minimal spanning tree constructed in Step 1 of both algorithms.

Why Algorithm 4.3.5.1 and the CHRISTOFIDES ALGORITHM are unstable for dist. ? This is the main problem.

9. 从 T 到 T^3，最关键的性质？

for every tree $T = (V, E)$, the graph $T^3 = (V, \{\{x,y\} \mid x,y \in V,$ there is a path x, P, y in T of a length at most $3\})$

Lemma 4.3.5.17. *Let T be a tree with $n \geq 3$ vertices, and let $\{p,q\}$ be an edge of T. Then, T^3 contains a Hamiltonian path $U = v_1, v_2, \ldots, v_n$, $p = v_1$, $v_n = q$, such that every edge of $E(T)$ occurs exactly twice in $P_T(H)$, where $H = U, p$ is a Hamiltonian tour in T^3.*

Why this property is so important? -- 三个要素：

a) 确实含有 Hamiltonian Tour（关键在于这也是原来输入的 H-tour）

b) 收缩代价不会快速增加

c) 在最小生成树中长度有上限

10. 改进算法：SEKANINA'S ALGORITHM

Algorithm 4.3.5.18. SEKANINA'S ALGORITHM

Input: A complete graph $G = (V, E)$, and a cost function $c : E \to \mathbb{N}^+$.
Step 1: Construct a minimal spanning tree T of G according to c.
Step 2: Construct T^3.
Step 3: Find a Hamiltonian tour H in T^3 such that $P_T(H)$ contains every edge of T exactly twice.
Output: H.

Step1 and 2 can be performed $O(n^2)$, using Lemma 4.3.5.17, step3 can be implemented in O(n), so, total is $O(n^2)$。

算法是对Δ-TSP 是 2-近似的。

Theorem 4.3.5.20. *For every positive real number r,* SEKANINA'S ALGORITHM *is a polynomial-time $2(1+r)^2$-approximation algorithm for* $\triangle$-TSP_r.

11. 最后一段话，如何理解？

Thus, we have reached our final aim to divide the set of all instances of TSP into an infinite spectrum in such a way that the sets of this spectrum have upper bounds on the polynomial-time approximability of their input instances. The above analysis of TSP shows that it is reasonable to measure the hardness of the TSP instances by the distance function *dist*, i.e., by the degree of violation of the triangle inequality.

12. 作业

a) 设计一个找最小完美匹配的算法

【Some Notes for Matching】:

定理：在二分图中，最大匹配的边数等于最小覆盖的顶点数。

定理(Hall,1935)：设 G 为具有二分类(X, Y)的二分图，则 G 包含饱和 X 的每个顶点的匹配当且仅当： $|N(S)| \geq |S|$，对所有$S \subseteq X$成立。（匈牙利算法的基础）

匈牙利算法找最大匹配：不断找增广路增广，直到不能增广。

最小完美匹配可用 KM 算法或者最小费用最大流建模解决。

Exercise 4.3.5.6. Consider the following modification of $\triangle$-TSP. A problem instance I is a weighted graph $((V, E), c)$ with two special vertices q and s from V, where c satisfies the triangle inequality. A solution for I is any path between q and s that contains each vertex of V exactly once (i.e., a path between q and s of length $n-1$, that does obtain any vertex twice). The aim is to find a solution with the minimal cost.

(a) Design a 2-approximation algorithm for this minimization problem.
(b) Try to modify the CHRISTOFIDES ALGORITHM in order to get a 5/3-approximation algorithm for this problem.

见作业⑨

Exercise 4.3.5.11. Prove that Algorithm 4.3.5.1 is a polynomial-time $2 \cdot (1 + r)$-approximation algorithm for $(\Sigma_I, \Sigma_O, L, Ball_{r,distance}(L_\triangle), \mathcal{M}, cost, minimum)$. For which input instances from L is Algorithm 4.3.5.1 better than CHRISTOFIDES ALGORITHM? □

a) 见作业⑨

b)

Exercise 4.3.5.13. Analyze the quasistability of Algorithm 4.3.5.1 according to *dist*. □

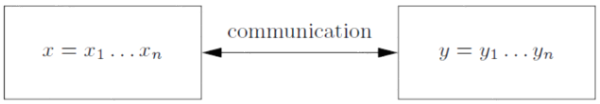
见作业⑨。

第 10 周：概率分析与随机算法的基本分析方法

教学目的：通过示例理解随机方法在问题求解中的意义，并系统理解以概率分析为基础的随机算法出错率分析方法，并理解什么条件下可以让出错率无限接近零。

阅读材料：*Juraq Hromkovic: Design and Analysis of Randomized Algorithms, Springer-Verlag 2005*, 第 1 章第 1.2 节，第 2 章

1. 确保两地数据一致的判定。为什么说确定方法解这个问题复杂度很高？



bit 数可能很大(10^6)!

2. 随机协议方法来做相等性判断。

$R = (R_{\mathrm{I}}, R_{\mathrm{II}})$ (Randomized Protocol for Equality)

Initial situation: R_{I} has a sequence x of n bits, $x = x_1 \dots x_n$, and R_{II} has a sequence y of n bits $y = y_1 \dots y_n$.
Phase 1: R_{I} chooses uniformly[9] a prime p from the interval $[2, n^2]$ at random.
Phase 2: R_{I} computes the integer

$$s = \text{Number}(x) \mod p$$

and sends the binary representations of s and p to R_{II}.
{Observe that $s \leq p < n^2$ and so each of these integers can be represented by $\lceil \log_2 n^2 \rceil$ bits.}
Phase 3: After reading s and p, R_{II} computes the number

$$q = \text{Number}(y) \mod p.$$

If $q \neq s$, then R_{II} outputs "$x \neq y$".
If $q = s$, then R_{II} outputs "$x = y$".

什么情况下会出错？什么情况下不会出错？

对于特定的 x, y，会不会出错取决于碰巧选到了那个质数(称为 bad prime)。
那么，碰到 bad prime 的几率有多大？
这决定了此算法多大程度上能用。

error probability: $\dfrac{\text{the number of bad primes for } (x,y)}{Prim\,(n^2)}$，素数定理：$\lim_{m\to\infty} \dfrac{Prim\,(m)}{m/\ln m} = 1$，有 $Prim\,(n^2) > \dfrac{n^2}{2\ln n}$ for all n >= 9，而 $\boxed{\begin{array}{c} a \text{ prime } p \text{ is bad for } (x,y) \text{ iff} \\ p \text{ divides the number } w = |\text{Number}(x) - \text{Number}(y)|. \end{array}}$ that w has at most $n-1$ different prime factors. 所以有：

$$\boxed{\begin{array}{l} \text{choosing a bad prime } p \text{ dividing } w \text{ is at most} \\ \qquad \dfrac{n-1}{Prim\,(n^2)} \leq \dfrac{n-1}{n^2/\ln n^2} \leq \dfrac{\ln n^2}{n} \\ \text{for all } n \geq 9. \end{array}}$$

3. “随机”的概念

确定：世界上任何事情都是有一系列互为因果的时间组成的，如果我们搞清楚了这些因果关系，我们就能够预测未来。

非确定：有些“因”可能有多种果，每次实际发生的是其中的某一种，但在实际发生之前没有任何可能知道究竟会是那一个“果”发生。

4. 是否存在真正的随机？

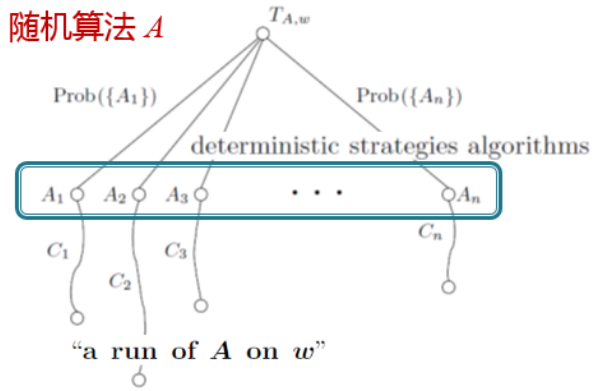
5. For us as computer scientists, the main reason to believe in randomness is that randomness can be ***a source of efficiency***. Randomness enables us to reach aims incomparably faster, and it would be very surprising for us if Nature left this great possibility unnoticed.

6. Stochastic 与 Randomized 在阅读材料中的差别？

7. 能否解释下面这段话？

For efficiently computing the correct result with a reasonable probability for each input, ***one has to investigate the behavior of a randomized algorithm on any feasible input***. It doesn't make any sense to consider probability distributions on input sets. The only randomness under consideration is the randomized control of the algorithm itself.

8. 基于确定算法的随机算法模型



We model the experiment of the work of A on an input w as the probability space

$$(S_{A,w}, \text{Prob})$$

9. 三个指标

Let Time(C_i) denote the length of the computation C_i. Can be seen as a random variable.

a) expected time complexity of A on w

$$\begin{aligned}\textbf{Exp-Time}_A(w) = \mathrm{E}[Z] &= \sum_{i=1}^{n} \text{Prob}(\{C_i\}) \cdot Z(C_i) \\ &= \sum_{i=1}^{n} \text{Prob}(\{C_i\}) \cdot \text{Time}(C_i)\,.\end{aligned}$$

The **expected time complexity of a randomized algorithm** A is the function Exp-Time$_A$: $\mathbb{N} \to \mathbb{N}$, defined by <= b)

$$\textbf{Exp-Time}_A(n) = \max\{\text{Exp-Time}_A(w) \mid \text{the length}^{17}\text{of } w \text{ is } n\}$$

for all $n \in \mathbb{N}$.

~~run of a randomized algorithm~~, one considers the **time complexity of a randomized algorithm** A as the function Time$_A(n)$: $\mathbb{N} \to \mathbb{N}$, defined by

$$\textbf{Time}_A(n) = \max\{\text{Time}(C) \mid C \text{ is a run of } A \text{ on an input of length } n\}.$$ <= c)

10. indicator variable X(C)

$$X(C_i) = \begin{cases} 1 \text{ if } C_i \text{ computes the correct result on } w_i \\ 0 \text{ if } C_i \text{ computes a wrong result on } w_i \end{cases}$$

$$\begin{aligned}\mathrm{E}[X] &= \sum_{i=1}^{n} X(C_i) \cdot \text{Prob}(\{C_i\})\,. \\ &= \sum_{X(C_i)=1} 1 \cdot \text{Prob}(\{C_i\}) + \sum_{X(C_i)=0} 0 \cdot \text{Prob}(\{C_i\}) \\ &= \text{Prob}(\text{Event}(X = 1)) \\ &= \text{the probability that } A \text{ computes the right result.}\end{aligned}$$

算法的错误率 $\textbf{Error}_A(n) = \max\{\text{Error}_A(w) \mid \text{the length of } w \text{ is } n\}$

11. Las Vegas Algorithm: guarantee that every computed output is correct.

Definition 2.4.41. *A randomized algorithm A is called a* **Las Vegas algorithm** *computing a function F if, for any input x (any argument x of F),*

$$\text{Prob}(A(x) = F(x)) = 1.$$

Las Vegas that allow "?"

Definition 2.4.42. *Let A be a randomized algorithm that allows the answer "?". We say that A is a* **Las Vegas algorithm** *for a function F if, for every input x,*

(i) $\text{Prob}(A(x) = F(x)) \geq 1/2$, *and*
(ii) $\text{Prob}(A(x) = \text{"?"}) = 1 - \text{Prob}(A(x) = F(x)) \leq 1/2$.

其实两种定义等价，为什么？

12. 阅读材料上说，如果一个 Las Vegas 算法对所有的输入执行时间差不多，那（几乎）就是说 P=NP 了，为什么？

13. Monte Carlo 算法

Let A be a randomized algorithm and let (Σ, L) be a decision problem. We say that A is an *one-sided-error Monte Carlo algorithm* for L, 1MC algorithm for short, if

(i) for every $x \in L$, $\text{Prob}(A(x) = 1) \geq \frac{1}{2}$, *and*
(ii) for every $x \notin L$, $\text{Prob}(A(x) = 0) = 1$.

14. 能否解释前面关于通信的算法是 1MC? 甚至是 1MC*?

15. One-sided-error 意义何在？

保证反例不会判断错，正例以大于等于 1/2 的概率判断出来。

16. Bounded Error Monte Carlo Algorithm

Definition 2.4.52. *Let F be a function. We say that a randomized algorithm A is a* **bounded-error Monte Carlo algorithm for F, 2MC algorithm**[50] *for short, if*

there exists a real number ε, $0 < \varepsilon \leq 1/2$ such that, for every input x of F,

$$\text{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon. \tag{2.12}$$

这个ε的价值何在？

17. 作业

Exercise 2.3.36. Let us modify the algorithm RSAM as follows.

Input: A formula $\Phi = F_1 \wedge F_2 \wedge \ldots \wedge F_m$ over $\{x_1, x_2, \ldots, x_n\}$ in CNF.

Step 1: Choose uniformly an assignment $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ to $x_1, x_2, \ldots, x_n$ at random.

Step 2: Compute the number $r(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of clauses that are satisfied by $(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

Step 3: If $r(\alpha_1, \alpha_2, \ldots, \alpha_n) \geq \frac{m}{2}$, then output $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, else repeat Step 1.

If this algorithm halts, then we have the assurance that it outputs an assignment satisfying at least half the clauses. Estimate the expected value of the number of executions of Step 1 of the algorithm (i.e., the expected running time of the algorithm).

Exercise 2.4.45. Modify the Las Vegas protocol LV_{10} in such a way that instead of choosing ten primes $p_1, \ldots, p_{10}$ at random one chooses only one prime $p \in \mathrm{PRIM}\left(n^2\right)$ at random. Then, one computes the remainders $s_1, \ldots, s_{10}$, $q_1, \ldots, q_{10}$ as

$$s_i = \mathrm{Number}(x_i) \mod p \text{ and } q_i = \mathrm{Number}(y_i) \mod p.$$

In this way, the protocol saves $18 \cdot \lceil \log_2 n \rceil$ communication bits. Explain why our analysis of the success probability of LV_{10} works without any change also for the modified protocol, despite the exchange of $p_1, \ldots, p_{10}$ for an only one prime p.

Exercise 2.4.46. Use the method described above to modify the algorithm LV_{10} from Example 2.4.44 in such a way[40] that the output "?" does not occur. Then, analyze the expected communication complexity of your Las Vegas protocol with forbidden "?".

Exercise 2.4.53. Modify the 2MC algorithm A_t to a randomized algorithm A'_t in such a way that A'_t takes the most frequent result as the output. What is the error probability of this modified algorithm A'_t?

第 11 周：利用随机方法设计好的近似算法

教学目的：通过典型例子理解随机方法在近似算法设计中的应用，特别是理解松弛方法如何与随机思想相结合用以提高近似率。

阅读材料： ***Teofilo Gonzalez (Ed.): Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC 2007*****，第 7 章**

Dorit Hochbaum Ed.: Approximation Algorithms for NP-Hard Problems, PWS Publishing Company 1997**，第 11 章**

第 12 周：新应用下经典问题的拓展

教学目的：以经典的 Bin Packing 问题为例，理解如何根据应用需要将其拓展为二维甚至多维的 Packing 问题，并讨论相应的算法。

阅读材料： ***Teofilo Gonzalez (Ed.): Handbook of Approximation Algorithms and Metaheuristics, Chapman and Hall/CRC 2007*****，第 32，35 章**

Juraq Hromkovic: Algorithmics for Hard Problems, Springer-Verlag 2004**，第 4.3 节**

1. Bin Packing 问题

Let $a_1, \ldots, a_n$ be a given collection of items with sizes $s(a_i) > 0$, $1 \le i \le n$. In mathematical terms, bin packing is a problem of partitioning the set $\{a_i\}$ under a sum constraint:

Divide $\{a_i\}$ into a minimum number of blocks, called bins, such that the sum of the sizes of the items in each bin is at most a given capacity $C > 0$.

2. Online: 启发式的 Next Fit 算法，asymptotic 2-approximation.

The strategy: Put a new item in the ***last bin*** if possible, or use a new bin. Never look back!

2'. Online: First Fit 算法：放到第一个合适的 bin 中，asymptotic 17/10-approximation

3. 假如每个 item 都不大，效果更好

假设每个item $a_i \le \gamma$ ，$0 < \gamma < 1$

$$NF(I) \le \left\lceil \frac{\mathrm{SUM}(I)}{1-\gamma} \right\rceil$$

$\sum_{i:f(i)=j} a_i > 1-\gamma$ for $j = 1, \ldots, NF(I)-1$

为什么？这对算法设计有什么启发？

4. First Fit Decreasing - FFD

The strategy: packing the largest as possible

可以证明：假设最优解使用 m 个 bin。a) FFD 放到额外的 bin 中的 item 最多 m-1 个；b) 放到额外的 bin 中的 item 最大不超过 1/3

4’. Bin Packing 问题定义：BIN-P

BIN-PACKING PROBLEM.

The bin-packing problem (Bin-P) is similar to the knapsack problem. One has n objects of rational weights $w_1, \ldots, w_n \in [0,1]$. The goal is to distribute them among the knapsacks (bins) of unit size 1 in such a way that a minimal number of knapsacks (bins) is used.

Bin-Packing Problem (BIN-P)

Input: n rational numbers $w_1, w_2, \ldots, w_n \in [0,1]$ for some positive integer n.

Constraints: $\mathcal{M}(w_1, w_2, \ldots, w_n) = \{S \subseteq \{0,1\}^n \mid$ for every $s \in S$, $s^T \cdot (w_1, w_2, \ldots, w_n) \leq 1$, and $\sum_{s \in S} s = (1, 1, \ldots, 1)\}$.
{If $S = \{s_1, s_2, \ldots, s_m\}$, then $s_i = (s_{i1}, s_{i2}, \ldots, s_{in})$ determines the set of objects packed in the ith bin. The jth object is packed into the ith bin if and only if $s_{ij} = 1$. The constraint

$$s_i^T \cdot (w_1, \ldots, w_n) \leq 1$$

assures that the ith bin is not overfilled. The constraint

$$\sum_{s \in S} s = (1, 1, \ldots, 1)$$

assures that every object is packed in exactly one bin.}

Cost: For every $S \in \mathcal{M}(w_1, w_2, \ldots, w_n)$,

$$cost(S, (w_1, \ldots, w_n)) = |S|.$$

Goal: *minimum*.

5. Online Algorithm

Online algorithms may be the only ones that can be used in certain situations, where the items arrive in sequence according to some physical process and must be assigned to a bin at arrival time. In many cases an algorithm is offline only in that it performs an initial ordering of the items before applying an online rule. An algorithm is *bounded-space* if the bins available for packing (called “open” bins) are limited in number.

Online 问题的挑战：

即使可以使用无限时间，某些问题可能也不会有“最优”算法

- 输入1：n object with the value 0.5- ε each, following by n object with the value 0.5+ε each; (0<ε<0.01)
- 输入2：only n object with the value 0.5-ε each

6. 能否想象我们是否可以放宽对“可行解”的要求？

假设“背包”或者 bin 有一点“弹性”？

7. 希望能够在“稍稍”放宽“可行”的要求的情况下，找到解 bin packing 问题的 PTAS。

考虑一个更容易的子问题：输入中不同的值的个数是一个常量。对这个问题我们可以找到一个确定的多项式时间算法。

7. constraint distance function

Definition 4.2.4.1. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem. A* **constraint distance function** *for U is any function $h : L_I \times \Sigma_O^* \to \mathbb{R}^{\geq 0}$ such that*

(i) $h(x, S) = 0$ for every $S \in \mathcal{M}(x)$,
(ii) $h(x, S) > 0$ for every $S \notin \mathcal{M}(x)$, and
(iii) h is polynomial-time computable.

For every $\varepsilon \in \mathbb{R}^+$, and every $x \in L_I$, $\mathcal{M}_\varepsilon^h(x) = \{S \in \Sigma_O^* \mid h(x, S) \leq \varepsilon\}$ is the **ε-ball of $\mathcal{M}(x)$ according to h.**

相对于可行解的限制条件

8. Dual Approximation Algorithm

Definition 4.2.4.2. *Let $U = (\Sigma_I, \Sigma_O, L, L_I, \mathcal{M}, cost, goal)$ be an optimization problem, and let h be a constraint distance function for U.*

An optimization algorithm A for U is called an **h-dual ε-approximation algorithm for U**, *if for every $x \in L_I$,*

(i) $A(x) \in \mathcal{M}^h_\varepsilon(x)$, and
(ii) $cost(A(x)) \geq Opt_U(x)$ if $goal = maximum$, and
$cost(A(x)) \leq Opt_U(x)$ if $goal = minimum$.

9. 先考虑简单一点--- s-Bin-P 问题

最多只有s个不同的值，每个值出现的频度是确定的

To simplify the matter, we consider an input instance of an s-BIN-P as $I = (q_1, \ldots, q_s, n_1, \ldots, n_s)$, where $q_i \in (0,1]$ for $i = 1, \ldots, s$, $n = \sum_{i=1}^{s} n_i$, $n_i \in \mathbb{N} - \{0\}$ for $i = 1, \ldots, n$.

动态规划！

$$\text{Bin-P}(m_1, \ldots, m_s) = 1 + \min_{x_1, \ldots, x_s} \left\{ \text{Bin-P}(m_1 - x_1, \ldots, m_s - x_s) \,\middle|\, \sum_{i=1}^{s} x_i q_i \leq 1 \right\}$$

如何解释这个递归？

Algorithm 4.3.6.1 (DPB-P_s).

Input: $q_1, \ldots, q_s$, $n_1, \ldots, n_s$, where $q_i \in (0,1]$ for $i = 1, \ldots, s$, and $n_1, \ldots, n_s$ are positive integers.
Step 1: BIN-P$(0, \ldots, 0) := 0$;
Bin-P$(h_1, \ldots, h_s) := 1$ for all $(h_1, \ldots, h_s) \in \{0, \ldots, n_1\} \times \cdots \times \{0, \ldots, n_s\}$ such that $\sum_{i=1}^{s} h_i q_i \leq 1$ and $\sum_{i=1}^{s} h_i \geq 1$.
Step 2: Compute Bin-P$(m_1, \ldots, m_s)$ with the corresponding optimal solution $T(m_1, \ldots, m_s)$ by the recurrence (4.61) for all $(m_1, \ldots, m_s) \in \{0, \ldots, n_1\} \times \cdots \times \{0, \ldots, n_s\}$.
Output: BIN-P$(n_1, \ldots, n_s)$, $T(m_1, \ldots, m_s)$.

different sub-problems

$$n_1 \cdot n_2 \cdot \cdots \cdot n_s \leq \left(\frac{\sum_{i=1}^{s} n_i}{s} \right)^s = \left(\frac{n}{s} \right)^s$$

复杂度：

$$O\left(\left(\frac{n}{s}\right)^{2s}\right)$$

10. 能将不确定数目的输入值转变为不超过常数个值，可以牺牲一点精确性？
Rounding：将输入值划分为常数个区间，每个区间内的值都近似为同样的值。

Algorithm 4.3.6.2 (BP-PTA$_\varepsilon$).

Input: $(q_1, q_2, \ldots, q_n)$, where $\varepsilon < q_1 \leq \cdots \leq q_n \leq 1$.
Step 1: Set $s := \lceil \log_2(1/\varepsilon)/\varepsilon \rceil$;
$l_1 := \varepsilon$, and
$l_j := l_{j-1} \cdot (1 + \varepsilon)$ for $j = 2, 3, \ldots, s$;
$l_{s+1} = 1$.
{This corresponds to the partitioning of the interval $(\varepsilon, 1]$ into s subintervals $(l_1, l_2]$, $(l_2, l_3]$, $\ldots$, $(l_s, l_{s+1}]$.}
Step 2: **for** $i = 1$ **to** s **do**
do begin $L_i := \{q_1, \ldots, q_n\} \cap (l_i, l_{i+1}]$;
$n_i := |L_i|$
end
{We consider that every value of L_i is rounded to the value l_i in what follows.}
Step 3: Apply DPB-P_s on the input $(l_1, l_2, \ldots, l_s, n_1, n_2, \ldots, n_s)$.

为什么需要输入值有个正的下界？

Step 3: Apply DPB-P_s 为什么是dual？

it is an h-dual ε-approximation algorithm for Bin-P_ε

11. 为什么上述算法对 round(I)计算的结果一定不大于原 bin-packing 问题的最优解？

12. 丢掉小尾巴ε

Algorithm 4.3.6.4 (Bin-PTAS).

Input: (I,ε), where $I=(q_1,q_2,\ldots,q_n)$, $0\le q_1\le q_2\le\cdots\le q_n\le 1$, $\varepsilon\in(0,1)$.

Step 1: Find i such that $q_1\le q_2\le\ldots\le q_i\le\varepsilon\le q_{i+1}\le q_{i+2}\le\cdots\le q_n$.

Step 2: Apply BP-PTA$_\varepsilon$ on the input $(q_{i+1},\ldots,q_n)$. Let $T=(T_1,\ldots,T_m)$ be the output BP-PTA$_\varepsilon(q_{i+1},\ldots,q_n)$.

Step 3: For every i such that $\sum_{j\in T_i} q_j\le 1$ pack one of the small pieces from $\{q_1,\ldots,q_i\}$ into T_i until $\sum_{j\in T_i} q_j>1$ for all $j\in\{1,2,\ldots,n\}$. If there are still some small pieces to be assigned, take a new bin and pack the pieces there until this bin is overfilled. Repeat this last step several times, if necessary.

Theorem 4.3.6.5. Bin-PTAS *is an h-dual polynomial-time approximation scheme for* BIN-P.

13. 从 dual 算法到 dual 问题："可行" 的条件 与 "优化"目标 之间的 dual

Bin-PTAS can be used to design a polynomial-time approximation scheme for the makespan scheduling problem (MS). Let Bin-PTAS$_\varepsilon$ denote the version of Bin-PTAS for a fixed $\varepsilon>0$. The idea is that Bin-P and MS are dual in the sense that the constraints of BIN-P may be viewed as the objective of MS. More precisely, for any input instance (I,m), $I=(p_1,\ldots,p_n)$, of MS,

$$Opt_{\text{Bin-P}}\left(\frac{p_1}{d},\frac{p_2}{d},\ldots,\frac{p_n}{d}\right)\le m \Leftrightarrow Opt_{\text{MS}}(I,m)\le d.$$

14. 作业

1. Let k be fixed. Describe a pseudopolynomial algorithm which – given an instance I of the BIN-PACKING PROBLEM – finds a solution for this instance using no more than k bins or decides that no such solution exists.

2. Design a dual PTAS for the knapsack problem.

3. 除了阅读材料中举的例子，结合你熟悉的应用领域，你能否举出什么问题，可以用 bin packing 的某种"variation"建立模型？变了后你还能解吗？

第 13 周：网络计算问题的图模型及其算法 (Lec12)

第 13 讲：最短路 – 新问题，新算法

教学目的：通过将网络中几个典型问题抽象为一个图中不相交通路问题并讨论其近似算法，理解传统算法的改进如何用于解决应用中的新问题。

阅读材料：*Krishnaiyan Thulasiraman, etc. (Ed.): Handbook of Graph Theory, Combinatorial Optimization, and Algorithm, Chapman and Hall/CRC 2016*，第 39 章

1. 受限最短路问题 RSP

Problem ***RSP*** (Restricted Shortest Path) : Given a source node s, a destination node t and a delay constraint D. find an (s, t)-path P such that:

1) $D(P) \le D$, and
2) $C(P) \le C(P')$ for any other (s, t)-path P' that satisfies $D(P') \le D$.

2. 根据应用问题扩展：2DP 问题

Problem ***2DP*** (2-Restricted Link Disjoint Paths) : Given a source node s, a destination node t and a QoS requirement D, find two link-disjoint (s,t)-paths P_1 and P_2 such that:

1) $D(P_1) \le D$ and $D(P_2) \le D$;
2) $C(P_1)+C(P_2) \le C(P_1')+C(P_2')$ for every other pair of link-disjoint (s,t)-paths P_1' and P_2' that satisfy $D(P_1') \le D$ and $D(P_2') \le D$.

这个像你熟悉的网络流模型吗？

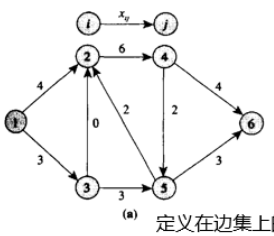
什么是 capacity？什么是 flow？

5. 流分解定理

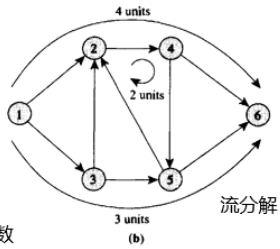
(**Flow Decomposition Theorem**). Every path and cycle flow has a unique representation as nonnegative arc flows. Conversely, every nonnegative arc flow x can be represented as a path and cycle flow (though not necessarily uniquely) with the following two properties:

(a) Every directed path with positive flow connects a deficit node to an excess node.
(b) At most $n+m$ paths and cycles have nonzero flow, out of these, at most m cycles have nonzero flow.

6. 网络流的两种表示？



定义在边集上的流函数



流分解：通路与回路

7. 能否设想一下，如果我们能知道最优解的上下界，我们怎么能利用这一点设计近似解？我们又如何能“知道”上下界？

8. 双重近似，(alpha, beta)-approximation

Definition ((α, β) –approximation):
Given an instance (G, s, t, D) of Problem ***2DP***, an (α, β)-approximate solution (P_1, P_2) to Problem ***2DP*** is a solution for which:

1) $D(P_1) + D(P_2) \le 2\alpha D$;
2) The total cost of two paths is at most β times more than that of the optimal solution. i.e.. $C(Pl) + C(P_2) \le \beta$ Opt.

是如何放宽可行条件的？

9. 先解值为 2 的流问题，MCF

Minimum Constrained Flow Problem (MCF):

Given a graph G, a source node s, a destination node t and a delay requirement D. Find an (s, t)-flow f such that:

1) $|f| = 2$;
2) $D(f) \le 2D$;
3) $C(f) \le C(f')$ for any other f' that satisfies $|f'| = 2$ and $D(f) \le 2$D

MCF 与 2DP 的可行解是什么关系？最优解呢？

10. 阅读材料后面的内容都是在前述算法的基础上追求特定方面的改进。你能梳理一下总的思路以及体现的基本想法吗？

11. RDP2

Algorithm RDP2 $(G(V, E), s, t, D, \varepsilon)$
1: Invoke Algorithm RDP1 with parameter ε to find two disjoint (s,t)-paths P_1 and P_2, define $\hat{E} = E(P_1) \cup E(P_2)$;
2: Construct an auxiliary graph G' from G by reversing the edges of $\hat{E}$, i.e., for each edge $e(u, v) \in \hat{E}$, add an edge $\hat{e}(v, u)$ to G' with cost $c(\hat{e}) = 0$ and delay $d(\hat{e}) = -d(e)$
3: Find a negative delay cycle W in G' that minimizes $\frac{D(W)}{C(W)}$
4: Augment $\hat{E}$ along W, i.e., add edges in $E(W)$ to $\hat{E}$ and then remove all interlacing edges
5: **If** $\sum_{e \in \hat{E}} d(e) \le 2D(1 + \varepsilon)$ **then** proceed to the next step,
else repeat steps 2–5
6: Decompose $\hat{E}$ into two edge disjoint paths P_1' and P_2'

Algorithm RDP2 belongs to the class of pseudopolynomial algorithms, since its complexity depends on the values of the edge parameters (delays and costs).

We conclude that Algorithm RDP2 provides a $(2(1 + \varepsilon), 2.5 + 1.5\varepsilon + 2\log(1/2\varepsilon))$ solution to Problem RDP. We note that the delay of the minimum path is at most $D(1+\varepsilon)$ so it only slightly violates the delay constraint.

The parameter ε captures the trade-off between the accuracy of the algorithm, the cost of the paths, and its running time.

12. 阅读材料中是基于什么想法设法降低计算复杂性的?

13. 作业

a) 设计一个算法，将一个用流函数形式表示的流分解为用通路与回路方式表示的流，以证明前面提到的流分解定理。

b) 写一个算法，找出 G(f)中使得 D(W)/C(W)最小的回路 W。

第 14 周：计算几何的基本问题求解方法与数据结构 (Lec13)

第 14 讲 搜索 – 从二分查找到平面分割

教学目的：以计算几何中的基本问题为例，理解如何针对特殊问题统一考虑数据结构与算法，达到解题的目的。

阅读材料：*Dinesh Mehta, etc. (Ed.): Handbook of Data Structure and Application, Chapman and Hall/CRC 2005*，第 63 章

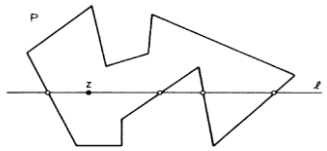
1. 从 Point Location Problem 说起

The *point location problem* is to preprocess a polygonal subdivision S in the plane into a data structure so that, given any query point q, the polygonal face of the subdivision containing q can be reported quickly. (In Figure 63.1(a), face 4 would be reported.) This

2. 最简单的问题

PROBLEM S.3 (CONVEX POLYGON INCLUSION). Given a convex polygon P and a point z, is z internal to P?

begin $L := 0$;
  for $i := 1$ **until** N **do if** edge (i) is not horizontal **then**
    if (edge (i) intersects l to the left of z at any of its points but its lower extreme) **then** $L := L + 1$;
  if (L is odd) **then** z is internal **else** z is external
end.



你能解释一下此算法么？

3. 二分搜索

Procedure CONVEX INCLUSION

1. Given a query point z, determine by binary search the wedge in which it lies. Point z lies between the rays defined by p_i and p_{i+1} if and only if angle (zqp_{i+1}) is a right turn and angle (zqp_i) is a left turn.[5]
2. Once p_i and p_{i+1} are found, then z is internal if and only if $(p_ip_{i+1}z)$ is a left turn.

4. Subdivision Search Problem

Given a subdivision S with n line segments and an arbitrary query point P, determine which region of S contains P.

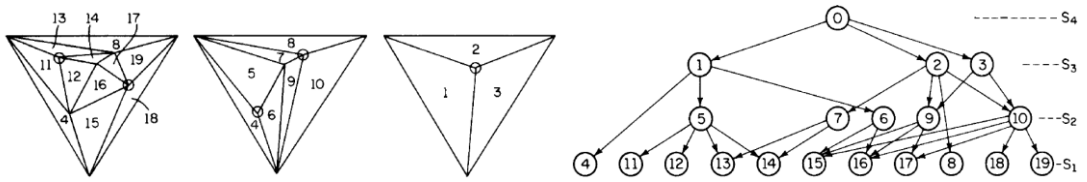
解此问题需要考虑哪些因素？ S 的表示，搜索结构，搜索结构上的 query point 定位。

5. Triangulation refinement technique

Let an N-vertex triangulation G be given and suppose we construct *a sequence of triangulations* $S_1, S_2, \ldots, S_{h(N)}$, where $S_1 = G$ and S_i is obtained from S_{i-1} as follows:

Step (i). Remove a set of independent (i.e., nonadjacent) nonboundary vertices of S_{i-1} and their incident edges (The choice of this set, to be specified later, is crucial to the performance of the algorithm.);

Step (ii). Retriangulate the polygons arising from the removal of vertices and edges.



有向边存在的条件是什么？

6. 结构定义好了，点定位就简单多了

```
    procedure POINT-LOCATION
begin if (z ∉ TRIANGLE(root)) then print "z belongs to unbounded region"
      else begin v := root;
                 while (Γ(v) ≠ ∅) do
                       for each u ∈ Γ(v) do if (z ∈ TRIANGLE(u)) then v := u;
                 print v
           end
end.
```

Point location in an N-vertex planar subdivision can be effected in O(log N) time using O(N) storage, given O(N log N) preprocessing time.

7. 你知道怎么计算平面上一个点集的 Convex Hull（CH）吗？

8. 最近邻问题

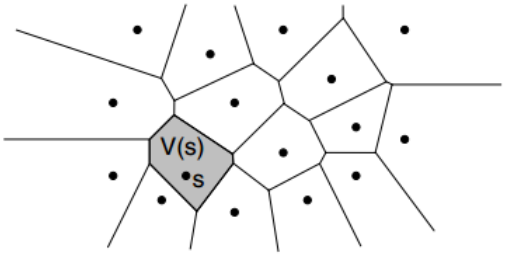
Given a set S of n planar points, find a nearest neighbor for each point in S.

除了计算 p 与其它所有点之间的距离以外有什么更好的办法吗？

9. Voronoi Diagram

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane. The locus of all points in the plane closer to a point p_i in S than to any other point in S defines a polygonal region $V(p_i)$ called the ***Voronoi region* of p_i**.

The collection of all n Voronoi regions, one for each point, constitute the *nearest-point Voronoi diagram*, or simply the ***Voronoi diagram*, of the point set S**, denoted by $V(S)$.



从图论的角度说说？

10. 为什么 Voronoi 图中每个顶点均为 3 次？

11. straight-line dual: Delaunay Triangulations

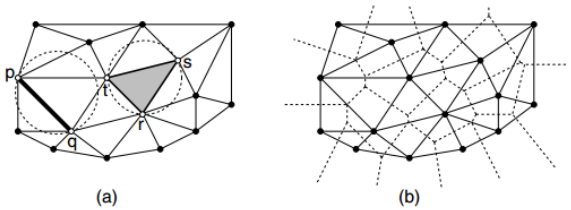


FIGURE 63.9: (a) The Delaunay triangulation of a set of points and (b) its overlay with the Voronoi diagram.

这里的 dual 与讨论地图着色问题时引入的对偶图有什么不同？

12. 如果|S|=n，保存 V(S)与 D(S)需要多少空间？

Let $\mathcal{V}(S)$ and $\mathcal{D}(S)$ be, respectively, the Voronoi diagram and Delaunay triangulation of a planar point set S, where $|S| = n \geq 3$. Then,

(1) *The number of vertices and edges in $\mathcal{V}(S)$ is at most $2n-5$ and $3n-6$, respectively.*
(2) *The number of edges in $\mathcal{D}(S)$ is at most $3n-6$.*

13. 再看最近邻居问题

Every nearest neighbor of p defines an edge of the Voronoi region V(p) !

为什么算一个点的最近邻居与算所有点的最近邻居 worst-case 代价是一样的？

14. Divide and Conquer 求 Voronoi 图

Algorithm 18.1 VORONOID

Input: A set S of n points in the plane.

Output: $\mathcal{V}(S)$, the Voronoi diagram of S.

1. Sort S by nondecreasing order of x-coordinates.
2. $\mathcal{V}(S) \leftarrow vd(S, 1, n)$

Procedure $vd(S, low, high)$

1. If $|S| \leq 3$, then compute $\mathcal{V}(S)$ by a straightforward method and **return** $\mathcal{V}(S)$; otherwise continue.
2. $mid \leftarrow \lfloor (low + high)/2 \rfloor$
3. $S_L \leftarrow S[low..mid]$; $S_R \leftarrow S[mid + 1..high]$
4. $\mathcal{V}(S_L) \leftarrow vd(S, low, mid)$
5. $\mathcal{V}(S_R) \leftarrow vd(S, mid + 1, high)$
6. Construct the dividing chain C.
7. Remove those rays of $\mathcal{V}(S_L)$ to the right of C and those rays of $\mathcal{V}(S_R)$ to the left of C.
8. **return** $\mathcal{V}(S)$

细节 1：找 lower common support

细节 2：分界线如何“转向”

15. 作业（见作业 14）

1，设计两个算法，分别经由 V(S)和 D(S)计算点集 S 的 convex hull。

2，证明： *Let S be a set of points in the plane, and p in S. Every nearest neighbor of p defines an edge of the Voronoi region $V(p)$.*

3. 假设 V(S)是点集 S 的 Voronoi 图。x∈S, y∉S。假设 y 在V(x)内，你能否找到一个方法可以有效地构造 V(S∪{y})。更进一步，你能否在此基础上给出一个计算点集的 Voronoi 图的算法。

第 15 周：大数据背景下的 property testing 算法

第 15 讲：Property Testing – 判定的“松弛”

教学目的：理解大数据背景下面临的一个典型问题，即如何在只能考察很小部分数据时得到关于对象性质的近似解，并探讨一些典型技术。

阅读材料：*Peter Buhlmann, etc. (Ed.)：Handbook of Big Data, CRC Press 2016*，第 10 章

1. What computational problems can we solve when we only have time to look at a tiny fraction of the data?..... Essentially, all of the research on this question starts with a simple observation: Except for a very small number of special cases, the only problems that can be solved in this very restrictive setting are those that admit approximate solutions.

2. Property Testing

Given the ability to perform (local) queries concerning a particular object the problem is to determine whether the object has a predetermined (global) property or differs significantly from any object that has the property. In the latter case we say it is far from (having) the property. The algorithm is allowed a small probability of failure, and typically it inspects only a small part of the whole object.

区分这两种情况，而不是区分yes/no。

“Property Testing”究竟是什么问题，它与传统的判定问题有什么不同？

上述意义下的 property testing 有什么意义？

a)为什么这可以看成是“近似”概念的推广？

b)为什么能在机器学习中发挥作用？

c)为什么可以导致效率非常高的算法？

d)为什么对大数据处理有价值？

3. Testing Graph Property

The testing algorithm should decide whether the graph G has property P, or whether it differs significantly from any graph having the property. In other words, the algorithm should accept every graph that has the property, and reject every graph for which many edge modifications should be performed so that the graph has the property.

To this end, the algorithm is given access to the graph in the form of being able to query on the incidence relationship between vertices. The testing algorithm is allowed a constant probability of error, and should perform its task by observing as few vertices and edges in the graph as possible.

解决图的 property testing,关键在哪里？

a) 如何表示图最适合算法进行 local “query”?

b) 对“property”满足的“距离”如何表示？

4. 计算机中最常使用的表示图的结构是什么？如果我们给一个百分比作为“距离”的度量，可能会产生什么问题？

5. 图的 Property Testing 算法

Let $\mathcal{A}$ be an algorithm which receives as input a size parameter $N \in \mathcal{N}$, a degree parameter $d \in \mathcal{N}$, and a distance parameter $0 < \epsilon \leq 1$. Fixing an arbitrary graph G with N vertices and degree bound d, the algorithm is also given oracle access to f_G. We say that $\mathcal{A}$ is a property testing algorithm (or simply a testing algorithm) *for graph-property Π, if for every N, d, and ϵ and for every graph G with N vertices and maximum degree d, the following holds:*

- *if G has property Π then with probability at least $\frac{2}{3}$, algorithm $\mathcal{A}$* accepts G*;*
- *if $\mathrm{dist}(G, \Pi_{N,d}) > \epsilon$ then with probability at least $\frac{2}{3}$, algorithm $\mathcal{A}$* rejects G*.*

ε–far

6. 图联通的测试

Let $d \geq 2$. If a graph G is ϵ-far from the class of connected graphs of degree bound d, then it has more that $\frac{\epsilon}{4}dN$ connected components.

If a graph G is ϵ-far from the class of connected graphs then it has at least $\frac{\epsilon dN}{8}$ connected components each containing less than $\frac{8}{\epsilon d}$ vertices.

能解释这两个关于非联通的特征吗？

7. 连通性测试算法

Connectivity Testing Algorithm

1. Uniformly choose a set of $m = \frac{16}{\epsilon d}$ vertices;
2. For each vertex s chosen perform a Breadth First Search (BFS)[4] starting from s until $\frac{8}{\epsilon d}$ vertices have been reached or no more new vertices can be reached (a small connected component has been found);
3. If any of the above searches found a small connected component then output REJECT, otherwise output ACCEPT.

有足够多的联通分支点数少于(8/εd)，所以选择 m 个点均碰不到这样的分支的概率足够小。

另外注意：联通图只有一个分支，不可能被 reject。

8. 是否还应该进一步提高模型的灵活性？

顶点次数的上限会决定图直径的下限。

9. 图直径测试问题

性质P_s: 图的直径最大为s

$\beta(s)$是s的线性函数，且$\beta(s) \geq s$

ϵ是常量，$0<\epsilon<1$

1. If G has property P_s, then the algorithm should output accept with probability at least 2/3;
2. If G is ϵ-far from having property $P_{\beta(s)}$, then the algorithm should output reject with probability at least 2/3.

9. 作业

(a) Describe an example of a graph where the diameter is more than three times as large as the average distance.

(b) Describe how you could extend your construction to produce graphs in which the diameter exceeds the average distance by as large a factor as you would like. (That is, for every number c, can you produce a graph in which the diameter is more than c times as large as the average distance?)

“头重尾轻”图

第 16 周：Local Search 与启发式算法

第 16 讲 局部搜索 – 从 Heuristics 到理论

教学目的: Local search 是许多启发式算法背后的基本思想，通过 Local Search 及其在模拟淬火方法中的体现，理解启发式算法的概念、意义与局限性。

阅读材料：*Emile Aarts, etc. (ed): Local Search in Combinatorial Optimization, John Wiley & Sons, 1997*, 第 1，4 章

1. 什么是 heuristics？

One often uses the metaphor of walking in a mountainous region to get an intuitive idea of local search. Consider an instance of a minimization problem. If we define the height of a solution as its cost, then walking through a neighborhood graph to find a global optimum can be seen as walking on a three-dimensional surface to find the lowest point. This looks like walking in a dense fog over the surface of the earth with its mountains and valleys, where we are looking for the lowest point and we cannot see farther than one step ahead. Furthermore, a local optimum corresponds to the lowest point in a valley and applying iterative improvement means that we only allow downhill moves.

你觉得哪些词是最能体现 “the idea”的？为什么是 heuristics?

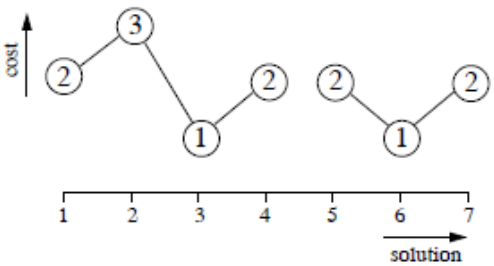
2. 什么是组合优化？

讨论 local search 算法时，总是在“优化问题”前加定语“combinatorial”。这是什么意思？这与 local search 的基本思想有什么关联？

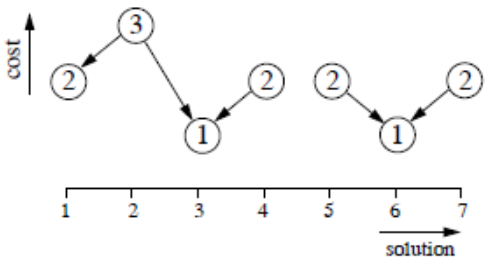
3. 什么是 Neighborhood？

For an instance (S, f) of a combinatorial optimization problem, a *neighborhood function* is a mapping $N : S \to 2^S$, where 2^S denotes the powerset $\{V|V \subseteq S\}$. The neighborhood function specifies for each solution $s \in S$ a set $N(s) \subseteq S$, which is called the *neighborhood* of s. The cardinality of $N(s)$ is called the *neighborhood size* of s. We say that solution s' is a *neighbor* of s if $s' \in N(s)$. The neighborhood function N is said to be *symmetric* in the case that we have $s' \in N(s)$ if and only if $s \in N(s')$. □

能否解释相应的 neighborhood 图以及 Transition 图是什么？



Neighborhood graph



Transition graph

如果每个局部最优也就是全局最优，这就是 exact neighborhood function。but 可遇而不可求。

4. Gradient Descent 解图的最小点覆盖问题

可能误入歧途，不可自拔！

5. 要使得 local search 真能有解题价值，我们必须解决什么问题？

6. Metropolis 算法

```
Start with an initial solution S0, and constants k and T
In one step:
   Let S be the current solution
   Let S' be chosen uniformly at random from the neighbors of S
   If c(S') ≤ c(S) then
      Update S ← S'
   Else
      With probability e^{-(c(S')-c(S))/(kT)}
         Update S ← S'
      Otherwise
         Leave S unchanged
   EndIf
```

能帮我们摆脱“陷阱”吗？

$e^{-E/(kT)}$ 的物理意义是什么？

7. 模拟退火：变化的 T

Select a temperature reduction function f as a function of two parameters T and time.

$I := 0$;
while $T > 0$ or T is not too close to 0) **do**
begin randomly select a $\beta \in Neigh_x(\alpha)$;
if $cost(\beta) \leq cost(\alpha)$ **then** $\alpha := \beta$
else begin generate a random number r uniformly in the range
$(0, 1)$;
if $r < e^{-\frac{cost(\beta)-cost(\alpha)}{T}}$
then $\alpha := \beta$
end;
$I := I + 1$;
$T := f(T, I)$
end
output(α).

这个方法为什么叫“模拟退火”，参数选择应该有什么考虑？

8. Hopfield 神经元网络的稳定状态

Does a Hopfield network always have a stable configuration, and if so, how can we find one?

状态反转算法

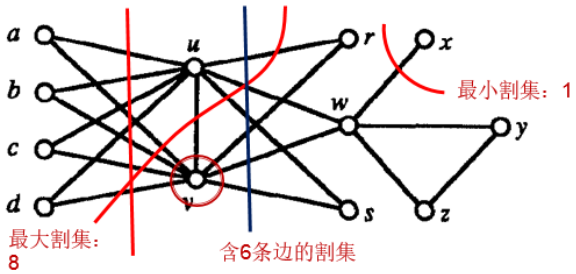
```
While the current configuration is not stable
  There must be an unsatisfied node
  Choose an unsatisfied node u
  Flip the state of u
Endwhile
```

注意：每翻转一次，被满足的点未必增加；也不能保证每个点只翻转 1 次。

这算法是“近似”还是“确定”？
算法一定会终止吗？

如何证明循环次数一定有限？
时间复杂度？

9. 最大边割集问题 – Guaranteed Solution



这个问题与前面的神经元网络问题有什么关系？

The State-Flipping Algorithm used for Hopfield networks provides a local search algorithm to approximate the Maximum Cut objective function φ (S) =w(A, B).

In terms of partitions, it says the following: If there exists a node u such that the total weight of edges from u to nodes in its own side of the partition exceeds the total weight of edges from u to nodes on the other side of the partition, then u itself should be moved to the other side of the partition.

上面第二段话是什么意思？为什么这算法用于最大边割集问题被称为“Single-flip”算法？

10. 作业

1. The set $A = \{1, 3, 6, 9\}$ of integers is given. We define the cost of a sequence τ of the numbers in A as $f(\tau) = \sum_{i=1}^{4} i\tau(i)$, where each number of A occurs exactly once in τ and where $\tau(i)$ denotes the ith integer in τ.
   We consider the problem of finding a sequence τ over A that minimizes $f(\tau)$. This problem is equivalent to finding a descending ordering of the numbers in A.
   a) Give the solution space of the described problem instance.
   b) Give the cost of each solution/sequence.

   Consider the swap neighborhood function in which sequence τ' is a neighbor of sequence τ if and only if τ' can be obtained from τ by changing the order of two adjacent numbers.
   c) Give the neighborhood graph for the swap neighborhood function.
   d) Give the transition graph for the swap neighborhood function.
   e) Verify that the swap neighborhood function is exact.
   f) Determine the depth of the local optima.

2，假设对 Hopfield 神经元网络问题输入边的权均为正值，则可以看作最大边割集问题的输入。假设输入本身就是二步图，那最优解应该是显然的。采用状态翻转算法是否一定能找到最优解对应的 configuration？

第 17 周：Bio-inspired 或 Nature-inspired 算法

第 17 讲 遗传算法 – 从自然智慧到计算思维

教学目的：通过典型的模仿自然进化结果的算法及其广泛应用的例子，理解启发式算法设计的背景，并领略“效法自然”思想在计算机科学中的体现。

阅读材料：*Xin-She Yang: Nature-Inspired Optimization Algorithms, Elsevier 2014*，第 4, 5 章

1. 达尔文？
2. 理解，比较，借鉴

为什么有启发意义

优化过程与优化问题求解

比较

a) 进化：物种还是个体
b) 信息：创造还是编辑
c) 方向：目标还是环境

借鉴：“模而不仿”

基于信息的选择与“变异”

3. 你了解“达尔文以后”人们对于“适者生存”更深入的理解吗？
4. 自然界如何寻找“某种”目标函数的“最优解”？最直观地方法就是搜索解空间。
5. 你听说过“种群”和“种质”吗？为什么采用遗传算法，首先必须确定要搜索的解的“表示”(encoding)?
6. 将“解”编码成标准形式

从计算机处理角度看，我们希望采用二进制向量作为解的“representation”。

对于前面的例子，假如我们希望精度为小数点后面6位，我们就至少要将定义区域平均分割为3,000,000个区间：

$$2097152 = 2^{21} < 3000000 \leq 2^{22} = 4194304.$$

因此，采用22位的二进向量，每个向量对应一个“解”的“chromosome”

于是，（实数）解x与（二进向量）“染色体”x'之间的换算关系很简单：

- convert the binary string $\langle b_{21}b_{20}\ldots b_0\rangle$ from the base 2 to base 10:

$$(\langle b_{21}b_{20}\ldots b_0\rangle)_2 = (\sum_{i=0}^{21} b_i \cdot 2^i)_{10} = x',$$

- find a corresponding real number x:

$$x = -1.0 + x' \cdot \frac{3}{2^{22}-1},$$

例如：0.637197编码为：
10 0010 1110 1101 0100 0111

where -1.0 is the left boundary of the domain and 3 is the length of the domain.

7. 适者生存，何为“适”？

对遗传算法而言，就是设计一个 chromosome 的评价函数。

8. 进化如何发生？

Crossover(染色体交换)：Swapping parts of the solution with another in chromosomes. The main role is to provide mixing of the solutions and convergence in a subspace.

Mutation(突变)：The change of parts of one solution randomly, which increases the diversity of the population and provides a mechanism for escaping from a local optimum.

Selection(选择)：The use of the solutions with high fitness to pass on to next generations, which is often carried out in terms of some form of selection of the best solutions.

9. 远没有那么简单

就算种群如何表示与评价函数如何设计确定了

a)究竟进化多少代算法终止？　　b)选择的“阈值”怎么定？

c)在多大的概率下让染色体交换发生？　　d)在多大的概率下让突变发生？

10. 这与随机算法有什么关系？

11. Genetic Algorithm

Genetic Algorithm

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_d)^T$
Encode the solutions into chromosomes (strings)
Define fitness F (eg, $F \propto f(\boldsymbol{x})$ for maximization)
Generate the initial population
Initialize the probabilities of crossover (p_c) and mutation (p_m)
  while (t <Max number of generations)
    Generate new solution by crossover and mutation
    Crossover with a crossover probability p_c
    Mutate with a mutation probability p_m
    Accept the new solutions if their fitness increase
    Select the current best for the next generation (elitism)
    Update $t = t + 1$
  end while
Decode the results and visualization

能详细解释这个过程吗？

为什么说这很适合“并行”？

12. 这里的 Optimization 含义有些什么新内涵吗？

The most important goal of optimization is improvement. Can we get to some good, "satisficing" level of performance quickly? Attainment of the optimum is much less important for complex systems.

13. 囚徒的困境 – 策略学习

是否能利用遗传算法得出对“己方”最有利的选择。

每一个player与同一或者不同的对手多次博弈。

假设每次player均根据前3次博弈的情况决定当前的选择，为了简单，假设决定的策略是固定的。希望能筛选出使payoff尽可能大的策略。

显然，population是参与的所有player(体现为策略）。

每一次博弈，可能的outcome有4种：DD（双方均defect, 结果Penalty); DC（己方defect，对方cooperate, 结果Temptation); CD（己方cooperate, 对方defect, 结果Sucker); CC(双方均cooperate, 结果Reward)

能说说如何编码吗？

Axelrod 算法

1. Choose an initial population. Each player is assigned a random string of seventy bits, representing a strategy as discussed above.
2. Test each player to determine its effectiveness. Each player uses the strategy defined by its chromosome to play the game with other players. The player's score is its average over all the games it plays.
3. Select players to breed. A player with an average score is given one mating; a player scoring one standard deviation above the average is given two matings; and a player scoring one standard deviation below the average is given no matings.
4. The successful players are randomly paired off to produce two offspring per mating. The strategy of each offspring is determined from the strategies of its parents. This is done by using two genetics operators: crossover and mutation.

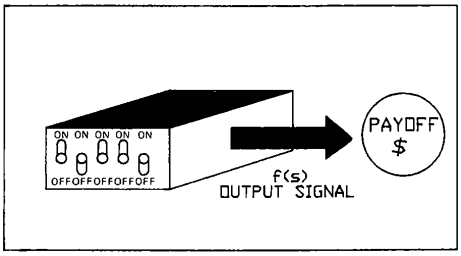
"种群"中被检查的样本相对不多，为什么"似乎"能得到有效的结果？

14. Schemata – Implicit Parallelism

假设我们将 Alphabet 从{0,1}扩大为{0,1,*}，则 11000 可以认为代表了多个 similarity template: 1****, 11***, 1*0**等等。

能说说 schemata 对算法研究的意义吗？

15. GA，有什么不同？



- Directed use of coding
- Search from a population
- Blindness to auxiliary information
- Randomized operators

黑盒子

谈谈与传统算法相比，上述特定带来的好处

16. 天下没有免费午餐

The problem of identifying fitness function

Definition of representation for the problem

Premature convergence occurs

The problem of choosing the various parameters like the size of the population, mutation rate, cross over rate, the selection method and its strength.

Cannot use gradients.

Cannot easily incorporate problem specific information

Not good at identifying local optima

No effective terminator.

Not effective for smooth unimodal functions

Needs to be coupled with a local search technique.

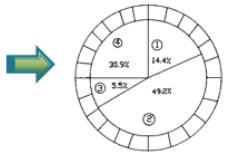
Have trouble finding the exact global optimum

Require large number of response (fitness) function evaluations

17. 作业

1，遗传算法中的选择可以用复制来实现，根据当前 generation中每个串的fitness值，按照相应的概率复制该串，放入下一generation的pool中。按照下图示意的方法实现复制：

No.	String	Fitness	% of Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0



2，对每个串 i 我们可以定义$ncount_i$, 它的值是第 i 个串的fitness与种群平均fitness的比值。我们用$ncount_i$值来确定复制并放入下一代pool中该串的数量：整数部分为确定复制的数量，小数部分为再复制一次的概率。如果用这中方法实现复制算法，与上述轮盘的方法有什么相同与不同之处？

3， Consider a binary string of length 11, and consider a schema, 1*********1. Under crossover with uniform crossover site selection, calculate a lower limit on the probability of this schema surviving crossover. Calculate survival probabilities under the same assumptions for the following schemata: ****10*****, 11*********, ***111*****, ****1*0****, **1***1**0*.

第 18 周：确定算法中的试探技术以及其它降低复杂度的可能性

第 18 讲 最大独立集问题 – 寻找有效的精确解

教学目的：通过典型问题，理解试探作为一种解题方法如何得到有效应用，并探讨利用分析技术针对一些难问题设计精确算法的可能性。

阅读材料：*Fedor Fomin: Exact Exponential Algorithms, Springer-Verlag 2010*，第 2，6 章

1. 最大独立集概念

Maximum Independent Set. In the MAXIMUM INDEPENDENT SET problem (MIS), we are given an undirected graph $G = (V,E)$. The task is to find an independent set $I \subseteq V$, i.e. any pair of vertices of I is non-adjacent, of maximum cardinality. For

Let us recall that an *independent set* $I \subseteq V$ of a graph $G = (V,E)$ is a subset of vertices such that every pair of vertices of I is non-adjacent in G. We denote by $\alpha(G)$ the maximum size of an independent set of G. The MAXIMUM INDEPENDENT SET problem (MIS), to find an independent set of maximum size, is a well-known NP-hard graph problem. A simple branching algorithm of running time $\mathcal{O}^*(3^{n/3}) = \mathcal{O}(1.4423^n)$ to solve the problem MIS exactly has been presented and analysed in Chap. 1.

If a vertex v is in an independent set I, then none of its neighbors can be in I. On the other hand, if I is a maximum (and thus maximal) independent set, and thus if v is not in I then at least one of its neighbors is in I. This is because otherwise $I \cup \{v\}$ would be an independent set, which contradicts the maximality of I. Thus for every vertex v and every maximal independent set I, there is a vertex y from the closed neighborhood $N[v]$ of v, which is the set consisting of v and vertices adjacent to v, such that y is in I, and no other vertex from $N[y]$ is in I.

如何解释上面这段话？它对设计求最大独立集的算法有什么启发？

2. reduce 和 branch

Algorithm `mis1`(G).
Input: Graph $G = (V,E)$.
Output: The maximum cardinality of an independent set of G.

```
if |V| = 0 then
    return 0
choose a vertex v of minimum degree in G
return 1 + max{mis1(G \ N[y]) : y ∈ N[v]}
```

A branching algorithm is recursively applied to a problem instance and uses two types of rules:

a) A reduction rule is used to simplify a problem instance or to halt the algorithm.

b) A branching rule is used to solve a problem instance by recursively solving smaller instances of the problem.

Branching is one of the basic algorithmic techniques for designing fast exponential time algorithms. It is safe to say that at least half of the published fast exponential time algorithms are branching algorithms.

In many branching algorithms any instance of a subproblem either contains a corresponding partial solution explicitly or such a partial solution can easily be attached to the instance. Thus a given algorithm computing (only) the optimal size of a solution can easily be modified such that it also provides a solution of optimal size.

3. Branch 算法 mis1 时间代价的上限取决于什么？

假设算法选择度数为 $d(v)$ 的顶点 v 做branching, 其相邻顶点为：

$$v_1, v_2, \ldots, v_{d(v)}$$

则算法依次解子问题：$G\setminus N[v]$, $G\setminus N[v_1]$, $\ldots$, $G\setminus N[v_{d(v)}]$

$$T(n) \le 1 + T(n-d(v)-1) + \sum_{i=1}^{d(v)} T(n-d(v_i)-1).$$

T 显然是单调递增的，所以：$T(n-d(v_i)-1) \le T(n-d(v)-1)$ v的次数最小

$$T(n) \le 1 + (d(v)+1)\cdot T(n-d(v)-1)$$

用s替换$d(v)$+1: $T(n) \le 1 + s\cdot T(n-s) \le 1 + s + s^2 + \cdots + s^{n/s}$

$$= \frac{1-s^{n/s+1}}{1-s} = \mathcal{O}^*(s^{n/s}).$$

$$T(n) = \mathcal{O}^*(3^{n/3})$$

$\mathcal{O}(1.4423^n)$

Standard branching:

$$\alpha(G) = \max(1 + \alpha(G\setminus N[v]), \alpha(G\setminus v))$$

标准 branching，要么 select v，要么 discard v。

4. Thinking in General

The main idea is that such an algorithm is recursive and that each execution of it can be seen as a search tree T, where a subproblem, here $G' = G\setminus V'$, is assigned to a node of T. Furthermore when branching from a subproblem assigned to a node of T then any subproblem obtained is assigned to a child of this node. Thus a solution in a node can be obtained from its descendant branches, and this is why we use the term branching for this type of algorithms and call the general approach *Branch & Reduce*.

5. Branch 算法计算代价的上限

Theorem 2.1. *Let b be a branching rule with branching vector $(t_1, t_2, \ldots t_r)$. Then the running time of the branching algorithm using only branching rule b is $\mathcal{O}^*(\alpha^n)$, where α is the unique positive real root of*

$$x^n - x^{n-t_1} - x^{n-t_2} - \cdots - x^{n-t_r} = 0.$$

6. 寻求更有效的 Rule - Domination

Lemma 2.5. *Let $G = (V,E)$ be a graph, let v and w be adjacent vertices of G such that $N[v] \subseteq N[w]$. Then*

$$\alpha(G) = \alpha(G\setminus w).$$

为什么说这是一个 Reduction Rule?

Proof. We have to prove that G has a maximum independent set not containing w. Let I be a maximum independent set of G such that $w \in I$. Since $w \in I$ no neighbor of v except w belongs to I. Hence $(I\setminus\{w\})\cup\{v\}$ is an independent set of G, and thus a maximum independent set of $G\setminus w$. □

Lemma 2.6. *Let $G = (V,E)$ be a graph and let v be a vertex of G. If no maximum independent set of G contains v then every maximum independent set of G contains at least two vertices of $N(v)$.*

7. 寻求更有效的 Rule - Simplicial

Let $G = (V,E)$ be a graph and v be a vertex of G such that $N[v]$ is a clique. Then

$$\alpha(G) = 1 + \alpha(G\setminus N[v]).$$

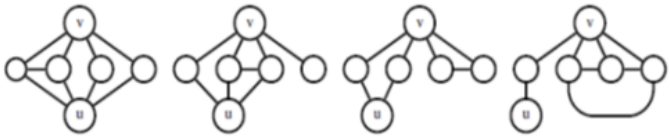
如果有最大独立集包含v，结果当然成立。否则任一最大独立集一定会包含$N(v)$中至少两个邻点，但那两个点必然相邻，与独立集的定义矛盾。

8. Mirror 及其对于最大独立集的意义

A vertex $u \in N^2(v)$ is called a **mirror** of v if $N(v) \backslash N(u)$ is a clique. Calling $M(v)$ the set of mirrors of v in G

如图 u 是 v 的 mirror。

这个概念与最大独立集的构成有什么关系



9. Mirror Branching

Let $G = (V, E)$ be a graph and v a vertex of G. Then

$$\alpha(G) = \max(1 + \alpha(G \backslash N[v]), \alpha(G \backslash (M(v) \cup \{v\}))$$

为什么这可以作为一个 Branch Rule?

If G has a maximum independent set containing v then $\alpha(G) = 1+\alpha(G \backslash N[v])$. Otherwise suppose that no maximum independent set of G contains v. Then, every maximum independent set of G contains at least two vertices of $N(v)$. Let w be any mirror of v. This implies that $N(v) \backslash N(w)$ is a clique, and thus at least one vertex of every maximum independent set of G belongs to $N(w)$. Consequently, no maximum independent set of G contains a $w \in M(v)$, and thus w can be safely discarded.

Clique中属于独立集的点最多1个。

10. 算法 MIS2（也见附录）

if $|V| = 0$ **then**
  return 0

α 的上限为1.1939

if $\exists v \in V$ *with* $d(v) \leq 1$ **then**
  return $1 + \texttt{mis2}(G \backslash N[v])$

$N[v]$是clique(孤立点或一条边)：适用simplical

if $\exists v \in V$ *with* $d(v) = 2$ **then**
  (let u_1 and u_2 be the neighbors of v)
  if $\{u_1, u_2\} \in E$ **then**
    return $1 + \texttt{mis2}(G \backslash N[v])$

$N[v]$是clique(三角形)：适用simplical

  if $\{u_1, u_2\} \notin E$ **then**
    if $|N^2(v)| = 1$ **then**
      (let $N^2(v) = \{w\}$)
      return $\max(2 + \texttt{mis2}(G \backslash (N^2[v] \cup N[w])), 2 + \texttt{mis2}(G \backslash N^2[v]))$

$\{w\}$是separator，$\{v,w\}$和$\{u_1,u_2\}$均为独立集

    if $|N^2(v)| > 1$ **then**
      return $\max(\texttt{mis2}(G \backslash N[v]), \texttt{mis2}(G \backslash (M(v) \cup \{v\}))$

适用mirror规则，弃v则必选u_1,u_2

if $\exists v \in V$ *with* $d(v) = 3$ **then**
  (let $u_1 u_2$ and u_3 be the neighbors of v)

如果任一u只与v相邻，则可用domination规则，所以可假设每个u均有邻点在$N^2(v)$中。

  if $G[N(v)]$ *has no edge* **then**
    if v *has a mirror* **then**
      return $\max(1 + \texttt{mis2}(G \backslash N[v]), \texttt{mis2}(G \backslash (M(v) \cup \{v\}))$
    if v *has no mirror* **then**

穷尽选v或不选v的一切可能

      return $\max(1 + \texttt{mis2}(G \backslash N[v]), 2 + \texttt{mis2}(G \backslash N[\{u_1, u_2\}]), 2 + \texttt{mis2}(G \backslash (N[\{u_1, u_3\}] \cup \{u_2\})), 2 + \texttt{mis2}(G \backslash (N[\{u_2, u_3\}] \cup \{u_1\})))$
  if $G[N(v)]$ *has one or two edges* **then**
    return $\max(1 + \texttt{mis2}(G \backslash N[v]), \texttt{mis2}(G \backslash (M(v) \cup \{v\}))$

1条边则选v，2条边则弃v而选$G(N(v))$中两个1次点

  if $G[N(v)]$ *has three edges* **then**
    return $1 + \texttt{mis2}(G \backslash N[v])$

Simplical规则

α 的上限为1.2721

if G *is disconnected* **then**
  (let $C \subseteq V$ be a component of G)
  return $\texttt{mis2}(G[C]) + \texttt{mis2}(G \setminus C)$

α 的上限为1.2852

if G *is* 4 *or* 5*-regular* **then**
  choose any vertex v of G
  return $\max(1 + \texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus (M(v) \cup \{v\}))$

在考虑复杂度时可忽略

if $\Delta(G) = 5$ *and* $\delta(G) = 4$ **then**
  choose adjacent vertices v and w with $d(v) = 5$ and $d(w) = 4$ in G
  return

这里选则v做mirror branching, 则在有mirror时分支向量至少为(2,6)，没有mirror时则为(1,6)

if $\Delta(G) = 5$ *and* $\delta(G) = 4$ **then**
  choose adjacent vertices v and w with $d(v) = 5$ and $d(w) = 4$ in G
  return
  $\max(1 + \texttt{mis2}(G \setminus N[v]), 1 + \texttt{mis2}(G \setminus (\{v\} \cup M(v) \cup N[w])), \texttt{mis2}(G \setminus (M(v) \cup \{v, w\})))$

11. 作业

1) 修改 mis1 和 mis2, 不仅输出最大独立集的大小，而且输出集合。

2) Improve upon the branching algorithm mis2 by a more careful analysis of the case $d(v) = 3$.

【附录】算法 MIS2

Algorithm `mis2(G)`.
Input: A graph $G=(V,E)$.
Output: The maximum cardinality of an independent set of G.

if $|V|=0$ **then**
  return 0
if $\exists v \in V$ *with* $d(v) \leq 1$ **then**
  return $1+\texttt{mis2}(G \setminus N[v])$
if $\exists v \in V$ *with* $d(v)=2$ **then**
  (let u_1 and u_2 be the neighbors of v)
  if $\{u_1,u_2\} \in E$ **then**
    return $1+\texttt{mis2}(G \setminus N[v])$
  if $\{u_1,u_2\} \notin E$ **then**
    if $|N^2(v)|=1$ **then**
      (let $N^2(v)=\{w\}$)
      return $\max(2+\texttt{mis2}(G \setminus (N^2[v] \cup N[w])), 2+\texttt{mis2}(G \setminus N^2[v]))$
    if $|N^2(v)|>1$ **then**
      return $\max(\texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus (M(v) \cup \{v\}))$
if $\exists v \in V$ *with* $d(v)=3$ **then**
  (let u_1u_2 and u_3 be the neighbors of v)
  if $G[N(v)]$ *has no edge* **then**
    if v *has a mirror* **then**
      return $\max(1+\texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus (M(v) \cup \{v\}))$
    if v *has no mirror* **then**
      return $\max(1+\texttt{mis2}(G \setminus N[v]), 2+\texttt{mis2}(G \setminus N[\{u_1,u_2\}]), 2+\texttt{mis2}(G \setminus (N[\{u_1,u_3\}] \cup \{u_2\})), 2+\texttt{mis2}(G \setminus (N[\{u_2,u_3\}] \cup \{u_1\})))$
  if $G[N(v)]$ *has one or two edges* **then**
    return $\max(1+\texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus (M(v) \cup \{v\}))$
  if $G[N(v)]$ *has three edges* **then**
    return $1+\texttt{mis2}(G \setminus N[v])$
if $\Delta(G) \geq 6$ **then**
  choose a vertex v of maximum degree in G
  return $\max(1+\texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus v))$
if G *is disconnected* **then**
  (let $C \subseteq V$ be a component of G)
  return $\texttt{mis2}(G[C])+\texttt{mis2}(G \setminus C)$
if G *is* 4 *or* 5*-regular* **then**
  choose any vertex v of G
  return $\max(1+\texttt{mis2}(G \setminus N[v]), \texttt{mis2}(G \setminus (M(v) \cup \{v\}))$
if $\Delta(G)=5$ *and* $\delta(G)=4$ **then**
  choose adjacent vertices v and w with $d(v)=5$ and $d(w)=4$ in G
  return
  $\max(1+\texttt{mis2}(G \setminus N[v]), 1+\texttt{mis2}(G \setminus (\{v\} \cup M(v) \cup N[w])), \texttt{mis2}(G \setminus (M(v) \cup \{v,w\})))$

Fig. 2.4 Algorithm `mis2` for MAXIMUM INDEPENDENT SET