Moser 的熵压缩证明法

陶哲轩*

2016 年 5 月 31 日

组合数学中经常会遇到这样的情形: 我们连续地运行某种迭代算法来提升某个对象 A; 在算法的每轮循环中将 A 替换为性质更好的 A', 直到获得某种我们想要的性质, 从而算法停止. 为了使这种证明方法产生有用的结论, 常常要求算法在有限时间内停止, 或 (更强地), 在有界时间[1]内停止.

一个保证算法停止的基本策略是利用*单调性*, 或者更准确地讲, 证明在算法的每轮循环中某个关键量保持增加 (或保持减少) 且有界. (或, 正如经济学家 Herbert Stein 喜欢讲的那样, 如果某个东西无法永远持续下去, 那么它必须停止.)

下面是四种常见的单调性策略的表现形式:

1. *质量增加证明法*. 这或许是确保算法停止的最为人熟知的方法: 令每个改进的对象 A' 比先前的 A 重量增加一个非平凡的量 (比如, 保证 A' 的势严格大于 A, 即 $|A'| \geq |A| + 1$). 或者, 我们可以试图令 A 外部的质量在迭代的每个阶段在某种意义下减少. 如果我们有一个在迭代过程中保持不变的 A 的质量的好的上界, 和一个每个阶段质量增量的下界, 那么则证毕. 许多贪心算法的证明正是这种类型的. Hahn 分解定理的证明也在此范畴中. 此处的一般策略是寻找 A 外部的有用质量片并将其加入 A 以构造 A', 从而利用质量的可加性. 最终没有可用质量可被添加 (也即, A 在某种L^1 *意义下是极大的*), 这就可以获得某种我们想要的 A 的性质.

2. *密度增加证明法*. 这是质量增加证明法的一个变体, 我们增加 A 的密度而非质量. 比如, A 可能包含于某周围的空间 P, 我们试图将 A 改进为 A'(同时将 P 改进为 P') 的方法是: 令新对象在新周围空间中的密度比先前的对象更好 (比如, $|A'|/|P'| \geq |A|/|P| + c$, 对某个 $c > 0$). 另一方面, A 的密度显然被 1 控制. 只要我们在每个阶段有一个足够好的密度增量的下界, 就可以得到算法迭代步数的上界. 该方法的一个典型例子是 Roth 对其定理每个具有正上密度的的整数集都包含长为 3 的算数级数的证明. 此处的一般策略是寻找 A 内部的有用密度波动, 并通过适当地减少 A 和 P 来聚焦到一个密度增加的区域. 最终没有可用密度波动 (也即, A *是均匀分布的*), 这就可以获得某种我们想要的 A 的性质.

3. *能量增加证明法*. 这是基于 L^1 的质量增加证明法 (或基于 L^∞ 的密度增加证明法) 的一种 L^2 类比, 我们试图增加 A 从某个参照对象 X 中捕获的能量, 或 (等价地) 减少仍与 A 正交的 X 的能

*张艺瀚译

[1]一般来讲, 在组合问题中我们不能使用无限迭代工具, 比如*超限归纳法*或 *Zorn 引理*, 因为用于提升某个目标对象 A 的迭代过程常常会降低其他某个有限的量 B, 而无限迭代会产生我们不想得到的结果: 使 B 无限.

量. 此处 A 和 X 可能与一个 Hilbert 空间有关, 而能量则与空间上的范数有关. 该证明方法的一个经典的例子是 Hilbert 空间到其闭子空间上的正交投影的存在性[1]; 这指导了测度论中条件期望的构造, 从而引出遍历论中的诸多证明. 另一个基本的例子是 *Szemerédi* 正则性引理 (此处能量常指指标) 的标准证明. 事实上, 这些例子是相关的. 此处的一般策略是寻找正交于 A 的有用能量片, 并将其加入 A 以构造 A', 从而利用能量的平方可加性, 比如 Pythagoras 定理. 最终 A 的外部没有可用能量可以被添加 (也即, A 在某种 L^2 意义下是极大的), 这就可以获得某种我们想要的 A 的性质.

4. 秩减少证明法. 此处, 我们试图令每个新对象 A' 与之前相比具有更低的秩, 维数, 或阶. 一个经典的例子是线性代数定理任给有限向量集, 存在线性无关子集张成相同的子空间的证明; 更一般的 *Steinitz* 交换引理具有同样的精神. 此处的一般策略是在 A 中寻找冲突或依赖, 并利用它们将 A 坍缩成具有更低秩的 A'. 最终, A 中没有可用冲突, 这就可以获得某种我们想要的 A 的性质.

我本人在加性组合中的许多工作都依赖于这些证明方法中的至少一种 (并且, 某些情况下, 依赖于两种或更多的组合). 许多非线性偏微分方程中的证明也具有类似的思想, 它们依赖于各种方程的解的单调性公式, 尽管 PDE 中的目标通常略有不同, 因为在趋向于奇点 (或当某时间或空间坐标趋向于无穷) 时我们想把解控制住, 而非保证一个算法的停止. (另一方面, 许多在 PDE 中有广泛应用的集中紧性理论中的证明确实具有与组合证明中类似的算法停止的思想.)

近期, Moser 给出了 *Lovász* 局部引理的一个优雅的新证明, 证明的主要工具中使用了一种新的单调性方法. 该证明方法可以被称为一种熵压缩证明法, 它仅用于概率算法, 这种算法要求某个由随机比特构成的集合 R 和其他随机选择作为输入的一部分, 算法的每轮循环利用一个对象 A(可能也是随机生成的) 和随机串 R 的某部分 (确定性地) 构造一个更好的对象 A'(并抛弃 R 中在循环中使用过的那些位从而形成一个更短的随机串 R'). 关键之处是设计一个部分可逆的算法, 即给定 A' 和 R' 以及一些记录算法至此的累计历史的附加信息 H', 我们可以重构 A 和 R 中不包含于 R' 的部分. 从而, 证明的每个阶段串 $A+R$ 的信息论内容被无损压缩为 $A'+R'+H'$. 然而, 像 $A+R$ 这样的随机变量无法被压缩为期望规模小于其 *Shannon* 熵的串. 从而, 如果我们有 $A+R$ 的一个好的下界, 并且 $A'+R'+H'$ 的长度确实小于 $A+R$ 的 (也即, 历史文件 H' 每轮迭代的长度净增量小于随机位每轮迭代的净使用量)[2], 那么由于随机数据文件在长度无法继续减小前被压缩的次数存在极限, 从而算法迭代次数存在极限.

我们有兴趣将该方法与前述方法相比较. 在前述方法中, 算法无法停止将导致新一轮迭代, 使 A 更重, 更密, 捕获更多能量, 或比先前的 A 秩更低. 而此处, 算法无法停止将导致存在新信息可被用于将 A(或更准确地讲, 全状态 $A+R$) 压缩到一个更小的空间中. 我尚不知道这种新的终止策略在我所工作的领域中有何应用, 但可以想象它最终是有用的 (或许用于证明具有足够随机性初始条件的 PDE 的解不会形成奇点?), 因此我想我还是会在此讨论它 (的一种特殊情况) 的.

我们不处理一般的 Lovász 局部定理, 而是某种与 k-可满足问题 *(*在合取范式下*)* 有关的特殊情况. 此处, 给定布尔变量集 $x_1,\cdots,x_n$ 和其否定 $\neg x_1,\cdots,\neg x_n$; 这 $2n$ 个变量及其否定均称为文字. 固定一个

[1](译者按) 正交分解定理: 若 H 为 Hilbert 空间, $M\subset H$ 是闭的, 那么 $\forall x\in H, \exists x_0\in M, z\in M^{\perp}$, 使得 $x=x_0+z$, 且该分解是唯一的.

[2]译者按: 也即, $|R\backslash R'|\geq|H'\backslash H|$.

整数 $k > 2$, 定义一个 (长为 k 的) 句子为 k 个文字的析取, 比如

$$x_3 \vee \neg x_5 \vee x_9$$

是一个长为三的句子, 当且仅当 x_3 为假, x_5 为真, x_9 为假时为假. 定义一个句子的支撑为该句子中的变量的集合, 从而例子 $x_3 \vee \neg x_5 \vee x_9$ 具有支撑 $\{x_3, x_5, x_9\}$. 为避免退化我们假设所有句子仅使用一个变量一次 (或等价地, 所有支撑都恰具有势 k), 也即我们认为诸如 $x_3 \vee x_3 \vee x_9$ 或 $x_3 \vee \neg x_3 \vee x_9$ 不构成句子.

注意到若句子为假则可以获得关于支撑中全部 k 个布尔变量的全部信息; 在之后的证明中这是一个重要的事实.

k-可满足问题指: 给定关于 n 个布尔变量 $x_1, \cdots, x_n$ 的长为 k 的句子的集合 S, 是否存在 $x_1, \cdots, x_n$ 的真值指派使得所有句子同时被满足?

对于一般的 S, 该问题对 $k = 2$ 是容易的 (事实上等价于图的 2-着色问题), 但对 $k \geq 3$ 是 NP-完全的 (这就是著名的 Cook-Levin 定理). 但如果我们对句子集合 S 做某种假设问题就变得容易些. 比如, 如果 S 中的句子具有不交支撑, 那么它们可以彼此独立地被满足, 从而在这种情况下我们可以很容易地对可满足问题给出肯定的回答. (事实上, 我们只需令 S 中的每个句子的支撑中有一个变量与其他所有支撑不交即可令此证明生效[1].)

现在假设句子集合 S 不是完全不交的, 但相交部分规模有界; 也即 S 中的大多数句子具有不交支撑, 但非全部. 显然, 如果相交过多我们倾向于相信集合无法被满足 (比如, S 是全部长为 k 的句子的集合). 但如果相交部分规模足够小, 我们仍有可满足性:

定理 1 (Lovász 局部定理, 特例) *设 S 为一个长为 k 的句子的集合, S 中的每个句子 s 的支撑与至多 2^{k-C} 个 S 中的句子的支撑 (包括自身) 相交, C 是一个足够大的常数. 那么 S 中的句子可同时被满足.*

该结果的强大之处之一在于此处的界是对变量数量 n 一致的. 除了 C 的损失, 该结果是很强的; 比如考虑具有支撑 $\{x_1, \cdots, x_n\}$ 的所有 2^k 个句子的集合 S, 显然不可满足.

该定理的标准证明是: 对 n 个布尔变量 $\{x_1, \cdots, x_n\}$ 独立随机指派真值 $a_1, \cdots, a_n \in \{\text{true}, \text{false}\}$(各真值以 1/2 等概率出现); 从而 S 中的每个句子有正概率为真 (事实上, 概率值为 $1 - 2^{-k}$). 更进一步地, 如果 E_s 表示事件句子 $s \in S$ 被满足, 那么大多数 E_s 是相互独立的; 事实上, 每个 E_s 至多与 2^{k-C} 个其他事件 $E_{s'}$ 不独立. 应用 *Lovász 局部引理*, 我们可以得到 E_s 以正概率同时发生 (如果 C 略大于 $\log_2 \mathrm{e}$), 从而结论成立.

Lovász 局部引理的教科书证明是简短的, 但非构造性的; 特别地, 它无法提供任何简单快捷的方法计算 $\{x_1, \cdots, x_n\}$ 的满足性指派, 仅仅指出这样的指派存在. 相反, Moser 的证明给出了一个简单而自然的算法来确定这样的指派 (从而证明了定理 1). (常数 C 变为 3 而非 $\log_2 \mathrm{e}$, 尽管此后 Moser 和 Tardos 在其文章中又恢复了 $\log_2 \mathrm{e}$ 的界.)

在通常的证明中, 我们首先为 $\{x_1, \cdots, x_n\}$ 随机指派真值 $a_1, \cdots, a_n \in \{\text{true}, \text{false}\}$; 记此随机指派为 $A = (a_1, \cdots, a_n)$. 若 A 使 S 中所有句子被满足, 则证毕. 但是, 有可能存在 S 的非空句子子集 T 无法被 A 满足.

[1] 译者按: 只需指派该变量为真即可.

现在我们将用如下方法修改 A 以减少不被满足的句子的数量 $|T|$. 如果我们总可以找到 A 的修改 A', 其不被满足的句子集合 T' 严格小于 T(当然要假设 T 非空), 那么我们就可以迭代从而证毕 (这基本上是一个质量减少证明法). 为此一个显然的方法是取 T 中一不被 A 满足的句子 s, 在 s 的支撑上修改 A 的值从而得到修改的集合 A', 它满足 s, 这很容易做到; 事实上, 任何 A 在支撑上的非平凡修改在此处均有效. 为了将系统的熵极大化 (这正是我们在熵压缩证明中要做的), 我们将随机选择修改 A'; 特别地, 我们将使用 k 个新的随机位来代替 A 中在 s 的支撑中的 k 位. (这样做, 存在 2^{-k} 的概率我们事实上并未改变 A, 但如果我们不考虑这种情况, 证明过程会 (非常) 略微的简单一些.)

如果所有句子具有不交支撑, 那么该策略可以毫无困难地实现. 但当支撑有交时, 我们遇到一个问题: 每次我们通过修改 s 的支撑上的变量来修改 A 以修复句子 s 时, 可能导致支撑与 s 相交的其他句子 s' 不被满足, 从而潜在地将 T 的规模增加了 $2^{k-C}-1$. 我们可以尝试修复所有被第一次修复破坏的句子, 但事实上在这个过程中需要修复的句子数量可能无限增长, 我们可能永远无法终止到所有句子同时被满足的状态.

如前文所示, Moser 的关键观察在于每个在指派 A 下值为假的句子 s 暴露了 A 的 k 位信息, 即 A 指派到 s 的支撑上的准确值. 我们打算使用每个值为假的句子作为压缩协议的一部分, 将 A(及一些其他信息) 无损压缩至一个更小的空间. 一个关键之处是在该过程的每个阶段, 我们试图修复的句子几乎总与我们先前修复过的句子相交. 因此给定先前的句子, 每个当前句子的总可能数量基本上是 2^{k-C}, 这仅需要 $k-C$ 位的空间存储, 相比之下, k 位的熵被抛弃. 这就使得算法在有限时间内 (以正概率) 停止.

让我们将细节细化. 我们需要如下的对象:

1. 一个含 n 个真值 $a_1,\cdots,a_n$ 的指派 A, 首先随机初始化, 然后依算法流程修改;
2. 一个长随机比特串 R, 我们将从其中做出随机选择, 每读一随机位就将其从 R 中移除.

同时我们还需要一个递归算法 Fix(s), 它修改串 A 使其满足 S 中的句子 s(并且额外地, 可能使 A 满足 S 中之前不被满足的某些句子). 其递归定义如下:

1. 步骤 1. 若 A 已满足 s, 则什么都不做 (也即, 令 A 不变).
2. 步骤 2. 否则, 从 R 中读取 k 个随机位 (从而 R 缩短 k 位), 依前述方法用它们代替 A 中 s 的支撑上的 k 位 (按某固定顺序将 s 的支撑排序, 对 $1\le j\le k$, 指派支撑中第 j 个变量的值为 R 中第 j 位).
3. 步骤 3. 然后, 找出 S 中所有支撑与 s 相交且不被 A 满足的句子 s'; 该集合至多含 2^{k-C} 个句子, 可能含 s 自身. 以任意方式对这些 s' 排序, 然后依次对每个这样的句子应用 Fix(s'). (所有子进程 Fix(s') 执行时, 原始算法 Fix(s) 将被挂起至某 CPU 栈上; 一旦所有子进程均完成, Fix(s) 也停止.)

通过简单的归纳可以证明如果 Fix(s) 停止, 那么 A 的最终修改将满足 s; 更进一步地, S 中任何其他在 Fix(s) 被调用前已被 A 满足的句子 s' 在 Fix(s) 被调用后仍被 A 满足. 从而, Fix(s) 只会减少 S 中未被满足的句子 T 的数量, 我们可以通过对 T 中每个句子调用 Fix(s) 来修复所有的句子 - 如果这些算法都能停止的话.

每次 Fix 算法的步骤 2 被调用, 指派 A 变为新指派 A', 随机串 R 变为更短的串 R'. 该过程是否可逆? 是 - 如果我们知道该算法正在修复的是哪个句子 s 的话. 事实上, 如果 s, A', R' 已知, 那么通过在 s 的支撑上改变指派 A' 为不满足 s 的唯一集合, 可以恢复 A, 通过在 R' 中附加 A 在 s 的支撑上的位, 可以恢复 R.

这种可逆性似乎对熵压缩证明法没什么用, 因为尽管 R' 比 R 短 k 位, 但它需要约 $\log|S|$ 位存储句子 s. 因此映射 $A+R\mapsto A'+R'+s$ 为压缩仅当 $\log|S|<k$, 而这并未出现在我们的假设中 (并且不论如何, 由统一的界知在 $\log|S|<k$ 的情况下 S 的可满足性是显然的[1]).

关键的技巧是, 尽管任给句子 s 确实需要 $\log|S|$ 位存储, 但存在更小的规模: 在 Fix 算法递归应用多次之后, 存储 s 所需的净位数降至 $k-C+O(1)$, C 足够大时它比 k 小, 从而使熵压缩证明法生效.

让我们看一下这是为什么. 观察到上述算法 Fix(s) 调用的句子 s 分为两类. 第一类, 来自最初的不被满足的句子集合 T 的句子 s. 其中的每个句子需要 $O(\log|S|)$ 位存储 - 但它们只有 $|T|$ 个. 由于 $|T|\le|S|$, 存储这些句子所需的净存储空间至多为 $O(|S|\log|S|)$. 事实上, 我们可以仅用 $|S|$ 位存储 S 的子集 T(对 S 中的每个元素, 记录它是否在 T 中.)

我们更感兴趣的是另一类句子 s, Fix(s) 在某个先前的调用 Fix(s') 中被递归调用. s 是 S 中与 s' 支撑相交的至多 2^{k-C} 个句子中的一元. 从而我们可以用 s' 和一个 1 到 2^{k-C} 之间的数来编码 s, 以表示 s 在 S 中所有支撑与 s' 相交的句子列表中 (关于 S 的某个任意选定的顺序) 的位置. 称此数为调用 Fix(s) 的指标.

现在假设当 Fix 程序被调用时, 我们维护一个程序运行的日志文件 (或历史)H,[2] 每次最初的 $|T|$ 个 Fix$(s), s\in T$ 之一被调用时, 它记录下 s, 在递归过程中它记录下其他被调用的 Fix(s) 的指标. 最后我们假设该日志文件记录一个表征 Fix 程序何时停止的终结符号. 通过堆栈跟踪[3], 我们看到每当 Fix 程序被调用, 可以通过查看截至此处的日志文件 H 推导出正被该程序修复的句子 s.

从而, 在所有这些 Fix 调用执行的任意中间阶段, 指派和随机比特串的初始状态 $A+R$ 可由这些对象的当前状态 $A'+R'$ 加截至此处的历史 H' 推导出来.

现在反设 S 未被满足; 那么 Fix 调用的栈永远无法完全终止. 我们在该栈中跟踪 M 步, M 是一个之后被取定的足够大的数. 这些步之后, 随机串 R 缩短 Mk 位; 如果我们将 R 的初始长度设为 Mk, 那么串现在为空, $R'=\emptyset$. 另一方面, 历史 H' 的规模至多为 $O(|S|)+M(k-C+O(1))$,[4] 因为它使用 $|S|$ 位来存储初始的 T 中的句子, 当步骤 1 开始时, 它用 $O(|S|)+O(M)$ 位记录所有调用实例, 且 Fix 的每个后续调用产生一个 $k-C$ 位的数, 加上规模为 $O(1)$ 的终结符号. 从而我们得到了由 $n+Mk$ 个完全随机比特到 $n+O(|S|)+M(k-C+O(1))$ 个比特的无损压缩算法 $A+R\mapsto A'+H'$(回忆 A 和 R 是随机取的, 彼此独立). 但由于 $n+Mk$ 个随机比特无法被无损压缩到任何更小的空间, 我们有熵界

$$n+O(|S|)+M(k-C+O(1))\ge n+Mk \tag{1}$$

当 M 足够大 (且若 C 比某常数大) 时, 导出矛盾. 从而证明了定理 1.

[1]译者按: 设支撑与某给定句子 s 相交的句子的集合为 U, 已知 $|U|\le 2^{k-C}$, 又 $U\subset S$, 也即 $|U|\le|S|$, 故取定合适的 C 即有 $|S|<2^k$, 也即 $\log|S|<k$.

[2]译者按: 日志文件的具体结构见附录.

[3]译者按: 堆栈跟踪的例子见附录.

[4]译者按: H' 规模分析的具体细节见附录.

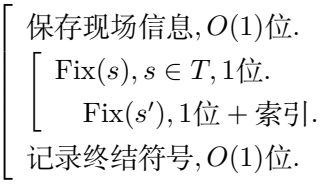
注 1 注意到上述证明事实上给出了一个 M 的明确的界, 且附加少许额外的比特即可转化为在关于 $|S|$ 和 n 的多项式时间内 *(以大概率)* 计算 S 的满足性指派的概率算法.

注 2 我们可以用 *Kolmogorov* 复杂性代替上述证明中的随机性与 *Shannon* 熵; 我们将 $A+R$ 设为无法被任何长为 $n+O(|S|\log|S|)+M(k-C+O(1))$ 的算法计算的长为 $n+Mk$ 的比特串, 其存在性由 *1*式的矛盾性保证; 现在证明变为确定性的了, 当然除了构造高复杂性串以外, 由其定义知它们只能用概率方法快速构造.

附录 (译者按)

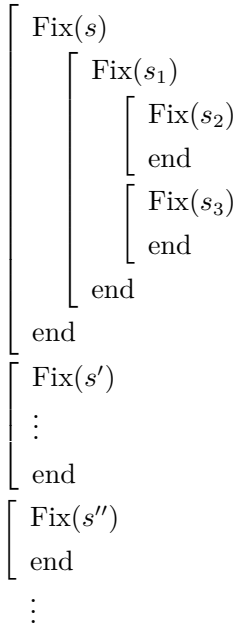
日志文件的结构

下图给出了日志文件 H 的结构, 包括在算法的各个阶段 H 记录的内容及花费的内存容量.



一个堆栈跟踪的例子

下图给出了一个堆栈跟踪的例子, 其中 $s, s', s'', \cdots \in T$. 任给栈中一节点, 容易知道当前正在修复的句子 s.



H' 规模的分析

根据日志文件的结构, 下表给出了在堆栈跟踪至第 M 步 H' 中所记录的各对象及对应花费的内存容量. 将各项相加即得 $|H'| = O(|S|) + M(k - C + O(1))$, 这与文中的分析结果相同.

对象	内存容量
现场信息	$MO(1)$
$s, s', s'', \cdots$	$O(\|S\|)$
s_i	$O(\|S\|)$
$\mathrm{Fix}(s_i)$ 的索引	$O(M)(k - C)$
终结符号	$MO(1)$