Randomized Algorithms and Cryptography

http://freemind.pluskid.org/misc/randomized-algorithms-and-cryptography

posted on Free Mind on May 20, 2013
generated with pandoc on December 3, 2015
category: MISC

tags: Inference, Randomized Algorithm, Fun, Cryptography

其实最初是在旁听的一门课的 problem set 里看到一个趣味附加题，描述是这样的：现在有两个密封的信封，里面各自装了一些钱，你除了知道两个信封里的钱数目不相等之外，其他一无所知。现在允许你选择其中一个信封打开看，然后做出是否保留那个信封或者是选择另一个信封的选择。你的目的是最大化拿到钱比较多的那个信封的概率。

如果我们知道信封里的钱数目服从的分布的话，倒是可以通过和 median value 进行比较的办法来简单地做决断，然而这里基本上什么信息都没有，打开了一个信封之后也不知道另外一个信封里面是什么，似乎没有什么可以做的，如果只是随机选择的话，只有 50% 的概率可以选到钱更多的那个信封。但是实际上却有一个可以保证能得到大于 50% 的概率的方法。

具体的做法是，先固定任意一个正实数 $\mathbb{R}^+$ 上的连续概率分布 $p_{\mathsf{z}}(\cdot)$，然后生成一个该分布的随机数 z，然后随便打开一个信封，如果信封里钱的数量小于 z 的话，就选择另一个信封，否则留下该信封。是不是怪怪的感觉？不知道哪里跑出来一个八杆子打不着关系的 z，就能帮上忙吗？`^_^bbb` 那下面我们来看看具体分析。

假设两个信封里的钱数目分别为 e_0 和 e_1，我们随机选了一个信封打开，用 y 表示我们打开的那个信封里的钱的数目。接下来不妨设 $e_0 < e_1$，因为反过来的 case 直接重复下面的分析可以得到同样的结论。

随机变量 z 是我们生成的服从分布 $p_{\mathsf{z}}(\cdot)$ 的参考数字，根据我们所使用的规则，我们拿到钱比较多的信封这件事主要会在两种情况下发生：

- $\mathsf{y} < \mathsf{z}$，并且此时 $\mathsf{y} = e_0$，此时我们由于是选择另一个信封，因此拿到 e_1
- $\mathsf{y} \geq \mathsf{z}$，并且此时 $\mathsf{y} = e_1$，此时我们由于是保留该信封，因此仍然拿到 e_1

下面计算总的概率

$$
\begin{aligned}
p &= \mathbb{P}(\mathsf{y} < \mathsf{z}, \mathsf{y} = e_0) + \mathbb{P}(\mathsf{y} \geq \mathsf{z}, \mathsf{y} = e_1) \\
&= \mathbb{P}(\mathsf{y} < \mathsf{z} | \mathsf{y} = e_0)\mathbb{P}(\mathsf{y} = e_0) + \mathbb{P}(\mathsf{y} \geq \mathsf{z} | \mathsf{y} = e_1)\mathbb{P}(\mathsf{y} = e_1) \\
&= \frac{1}{2}\left(\mathbb{P}(\mathsf{y} < \mathsf{z} | \mathsf{y} = e_0) + \mathbb{P}(\mathsf{y} \geq \mathsf{z} | \mathsf{y} = e_1)\right) \\
&= \frac{1}{2}\left(\mathbb{P}(e_0 < \mathsf{z}) + \mathbb{P}(e_1 \geq \mathsf{z})\right) \\
&= \frac{1}{2}\left(\int_{e_0}^{\infty} p_{\mathsf{z}}(z)dz + \int_{0}^{e_1} p_{\mathsf{z}}(z)dz\right) \\
&= \frac{1}{2}\left(1 + \int_{e_0}^{e_1} p_{\mathsf{z}}(z)dz\right)
\end{aligned}
$$

注意 $p_{\mathsf{z}}(\cdot)$ 是我们自己选的，这里可以看到我们只要保证它在 $\mathbb{R}^+$ 上总是正的，就能保证上面式子中最后那个积分总是正的，于是就有 $p > 1/2$ 了！是不是很神奇的样子？:D

当然从实际上来说，一方面这是一个随机算法，每次都需要生成随机数 z，并且 $p > 1/2$ 也只是一个概率而已，在重复了很多很多次之后才会看到效果[1]。而且，$p > 1/2$ 实际上也是非常微妙的一件事，因为即便有这个结果，不论对多小的 ϵ，我们却是无法保证做到 $p \geq 1/2 + \epsilon$。

[1] 当然，如果真的做这样的实验的话，反正每次都有钱拿，拿多拿少也都无所谓啦，哈哈！

总的来说，randomized algorithm 在实际中好像经常还是不太好直接拿来用的，但是在各种理论分析（比如博弈论什么的）中却是随处可见的。就计算理论的角度来看待 Randomized Algorithm 的话[2]，一个 Probabilistic Polynomial-time Turing Machine 就是在接受普通的图灵机的输入之外还接收一串随机串（比如说由 coin toss 产生的），注意这和 Non-deterministic Turing Machine 差别很大的，前者是可以实际实现出来的机器，而后者只是用于理论分析而构造出来的，要实际制造估计有点……ˆ ˆbb

[2] 从信息论或者说纯统计学的角度看和从计算理论的角度看很多问题会差别很大。

可以由 Probabilistic Polynomial-time Turing Machine 解决的问题类叫做 Bounded-error Probabilistic Polynomial-time (BPP)。随机的多项式时间图灵机是否会比普通的多项式时间图灵机要更加强大呢？或者说，P 是否等于 BPP 呢？虽然不像 P 和 NP 的问题那样家喻户晓，不过这个问题目前也还是没有搞清楚的。似乎大家比较倾向于认为 P=BPP。总感觉计算理论里似乎 “愿望” 比 “定理” 要更多的样子，ˆ_ˆbb 但是大家也都等不及所有基础的 foundation 被打扎实了再继续去发展后面的理论和实践，所以当你看到整个一套宏大的理论大厦建立在完全未经证明过的 “美好假设” 之上的时候，那种无比复杂的心情，就好像一个你曾经看过源代码知道里面各种乱七八糟各种 bug 的无比复杂的系统，却看着他居然成功跑起来了并且还被无数人使用并赞美着的感觉吧？当然就数学自己本身的历史而言其实许多地方都是后人再退回去修补之前的理论基础的。毕竟 hard-core 的基础工作也不是谁都做得了的。

就比如 Modern Cryptography，基于一个大前提就是 One-Way Function (OWF) 的存在性。而这个东西的存在性目前也还是没有被证明或者证伪

的。仅从理论发展的角度来说，似乎还好：如果哪天被证明 OWF 存在，当然一切皆大欢喜；即使哪天 OWF 被证明不存在了，所有的理论依据从逻辑的角度来说依然是没有问题的。只是就 Cryptography 这个东西来说，现在对称和非对称加密、证书、签名等等可以说已经在 cyberspace 里无处不在了……所以，让我们一起祈祷那个证明 OWF 存在的神人赶快出现吧！ ^ ^bb

不过既然提到了 Cryptography，就顺便多扯几句吧[3]。古时候的密码学，主要就是各种人斗智斗勇隐藏和揭露信息的方法，比如把消息刺到头顶然后再等头发长出来这种好像也是可以属于密码学的范畴的。后来香农大神在 1949 年的时候路过这边的时候看到这一片混乱，说这样搞不行啊，你再聪明的加密算法，如果没有严格的理论上的保证不能被破解的话，用起来也只能是提心吊胆的嘛，所以，我把信息论的奥义传授给你们吧。

[3] 旁白：大骗子，明明一开始就在不断地把话题往这个方向偏吧！

于是，一个密码系统被定义为一个三元组 (G, E, D)，其中 G 用于生成密钥，E 用于加密，D 用于解密。当然，必须要满足加密之后是能解密回来的，也就是说，对任意生成出来的密钥 k 和消息 m，满足 $D(k, E(k, m)) = m$。而为了保证密码系统的安全性呢，我们又要求任何人在看到密文 c 之后得不到任何有用的信息，用概率论的工具来描述就是

$$\mathbb{P}(M = m | C = c) = \mathbb{P}(M = m)$$

也就是说，密文 c 和明文 m 是独立的，观察到密文 c 之后对 m 的分布完全得不到进一步的改进。也就是说，一个想要破解明文的 adversary，在窃听到密文之后仍然和一个啥都没观察到过的废物没有两样，只能 random guess 了。这个性质就是香农的 perfect secrecy。

看上去是很强的一个要求，究竟有没有什么无敌加密系统能实现 perfect secrecy 呢？还确实是有的，而且也并不是超级复杂的那种，这个家伙叫做 One-Time Pad (OTP)，非常简单，比如说你要加密一段长度为 N 的明文 m，于是就生成一段相同长度的密钥 k，然后令 $c = m \oplus k$，就是 xor 一下，也就 OK 啦！可以证明，OTP 是满足 perfect secrecy 的。注意每次加密都需要重新生成新的密钥，否则就是不安全的。

Figure 1: 《幽游白书》里叼着奶嘴的小阎王，纯路过客串一下……

不幸的是，OTP 在实际中不太有用的样子：如果我都能把同样长度的密钥 k 安全地送到对方手里，我干嘛不直接就把明文交给对方？ -.-bb 另外一个不幸的事情就是，还可以证明，如果想要满足 perfect secrecy 的话，我们用的密钥空间的大小至少要跟明文空间一样大 ($|\mathcal{C}| \geq |\mathcal{M}|$) 才行。简单来说，就是基本上没什么可玩的，大家洗洗睡吧。此时 W. Diffie 这位大胡子老兄还没开始长胡子，而 M. Hellman 这位“来自地狱的男人”还叼着奶嘴。

一转眼过了二十七年……Diffie 和 Hellman 共同发表一一篇叫做 *New Directions in Cryptography* 的论文，宣布了 Modern Cryptography 的诞生。

当然，香农的难题还是铁板钉钉，但是我们其实并不需要这么强的 secrecy 要求：即使 Adversary 可以得到部分信息，如果他要从这个信息推断明文或者密钥这个过程的计算复杂度如果是高到不切实际的话，那么，我们也就心满意足了。

所谓 “高到不切实际” 的分隔线粗略来说就是多项式时间可计算了——很典型的 Computer Science 的套路。比如说，在 Cryptography 里有一种东西叫做 Pseudorandom Function Family (PRF)，它们可以通过 *Cryptographically Secure* Pseudorandom Number Generator 构造出来，而后者则可以通过 One-way Permutation 的 hard-core bit 构造出来。从直观上来讲，PRF 是这样一个函数类：从中随机选一个函数 f，你无法在多项式时间内把它和真正的随机函数区别开。

用接近于 machine learning 的语言来说，你可以对 f 进行多项式那么多次 query，得到一些 sample：$(x_1, f(x_1)), \ldots$，但是通过这些（多项式那么多个）sample，并在多项式时间内，你无法算出关于 f 的有用的信息，比如你要预测一个不在 query 过的 sample 里的 x_0 上的值 $f(x_0)$，能猜中的概率最多是 negligible 的。对比一下 Computational Learning Theory 里的 Probably Approximately Correct (PAC) Learning 模型，要求一个函数类是 learnable 的，说的是在多项式时间内通过多项式多个 sample 可以达到指定的学习误差范围。这样来说 Cryptography 和 Machine Learning 基本上就是在对着干了。`^_^bb` 虽然倒不是什么特别奇怪的事。

将 Security 的要求降低到 Computational 的层面上之后诞生的 Modern Cryptography，使得许多神奇的模型和协议都变得可能了。除了最基本的非对称加密、Hash、签名、证书等等应用之外，还有比较奇怪的比如允许 Alice 和 Bob 抛硬币的协议（突然想起了之前有一次和 hsys 在 Gtalk 上玩剪刀石头布的事）。另外一个比较好玩的是 Homomorphic Encryption，大致是虽然加密了之后的密文解不开，但是可以做运算。比如说我有两个数想让你帮我算一下它们的和，但是我又不想告诉你这两个数是什么，于是我用 Homomorphic Encryption 加密之后，你可以使用密文域里的加法计算出一个结果密文传给我，然后我可以通过解开那个密文来得到你的计算结果，而你完全不知道自己求的是哪两个数字的和。

另一个看起来比较诡异（但是是非常重要的 building block）的协议是 Oblivious Transfer (OT)：Alice 有两个值 x_0, x_1，Bob 通过这个 OT 协议向 Alice query 其中一个值，在协议结束之后 Bob 只拿到他要的那个值而对另一个值几乎一无所知，而 Alice 则搞不清楚自己到底是把两个值中的哪一个发给了 Bob。

还有一个比较好玩的是 Zero-Knowledge (ZK) Proof，传说古代的数学家们喜欢解方程比赛，比如如果 A 掌握了某些规律或者公式的话，就可以很容易解出那一类的方程来，但是 A 又不想把自己知道的公式公布出去，但是他为了向 B 证明他真的知道这样的公式，于是他允许 B 生成方程，A 把方程的根算出来告诉 B。B 虽然无法求根，但是他可以验证根的正确性。这样的协议虽然比直接告诉求根公式要隐晦一点，但是却说

不上 zero-knowledge，因为 B 完全可以把 A 当做一个黑盒子，换句话说，A 这个人就是 B 的 “求根公式”，当 B 要计算某个方程的根的时候，它就去向 A 做 query 就好了，而 A 为了证明自己能进行计算，又不得不回答正确的答案……

但是 ZK Proof 则可以避免这个问题。严格的定义和具体的例子由于牵涉到各种背景知识和准备工作，也没法在这里举了。简单地来说，一个 ZK Proof，就是当 A 在和 B 完成交互之后，B 会被 A 的证明给 convince，但是 B 得不到任何多余的信息，比如 B 没法通过把 A 当做一个黑盒子来完成他自己原来做不了的计算。说到这里，我前几天听到一个关于 ZK Proof 的笑话，因为 Shafi 讲课的内容很难，所以说 Shafi 上课的过程实际上是在对我们做了一个 ZK Proof：上完课之后我们被 Convince 了她说的都是对的，但是我们啥也没听懂…… `^_^bbb`

最后再回到 randomized algoritm 吧。本文最开始举了一个随机算法的例子，从某种意义上来说，随机算法是比 determinstic 算法要强大的。因为随机算法有随机性的存在，如果这个随机性和最后结果无关，那么这个随机算法其实就只是一个普通的 determistic 算法而已（比如抛硬币的时候永远返回正面即可），所以在考虑随机算法的时候我们需要允许它的结果依赖于算法运行时候的随机性，更具体地来说，我们会允许它 “有时候正确有时候错误”——只要，比如，正确的概率大于某个 threshold 即可。这样一来相当于把条件放宽了，所以 randomized algorithm 能比 determistic algorithm 做更多的事情也就并不是违反常理的了。

特别是对于 Cryptography 这样的领域，分析 randomized 算法也变得非常自然，比如，有些关于 security 的定义甚至很直接地要求算法必须是 randomized 的。例如公钥加密算法的 computational indistinguishability 的定义，大致是让 adversary 给出两个 message m_0, m_1，然后由 challenger 任选其中一个加密，把密文给 adversary，adversary 无法区分出是由哪个 message 加密的。在公钥加密中，由于公钥是公开的，adversary 自己就可以加密，所以如果加密算法是 deterministic 的话，adversary 要想知道哪个是哪个完全就是 trivial 的了：他只要自己加密一下 m_0 和 m_1 看和哪个密文对得上号就可以了。

另外一个很典型的 randomized 算法的例子是找一个 $\mathbb{Z}_p^*$ 里的 quadratic non-residue。一个数 $x \in \mathbb{Z}_p^*$ 是 quadratic residue (QR) 是说它是某个 $y \in \mathbb{Z}_p^*$ 的平方；否则它就是一个 quadratic non-residue (QNR)。在 p 是素数的情况下，$\mathbb{Z}_p^*$ 里有一半是 QR 另一半是 QNR，并且给定一个数之后很容易通过 Legendre Symbol 来判定它是否是 QR。但是要构造一个多项式时间的算法来找一个 NQR 却比较困难，如果使用 randomized algorithm 的话，只要返回一个随机数即可……正确的概率是 1/2。如果想要做得更好，可以重复许多次，直到找到 QNR 或者次数上限达到为止，基本上可以把正确的概率推到任意大，或者可以直接无限循环直到找到一个 QNR 为止，此时算法永远返回正确的答案，只是有一定的概率永远也不返回而已。不过这个时候需要分析的是 expected runing time，如果是多项式的，则仍然是可以接受的。找一个 QNR 来有什么用呢？它可以用来计算

$\mathbb{Z}_p^*$ 里的平方根。

后记：本文的思路已经相当不清晰了，最初的 draft 三个月前就有了，我已经不太记得当时想要写的是些什么了，拖了这么久一方面是没有时间，另一方面也许是觉得等 Cryptography 课上完之后应该能有一个更大局的认识所以可能会写得更清楚一些，但是结果似乎最后还是只知道一些比较细碎的东西而已啦。