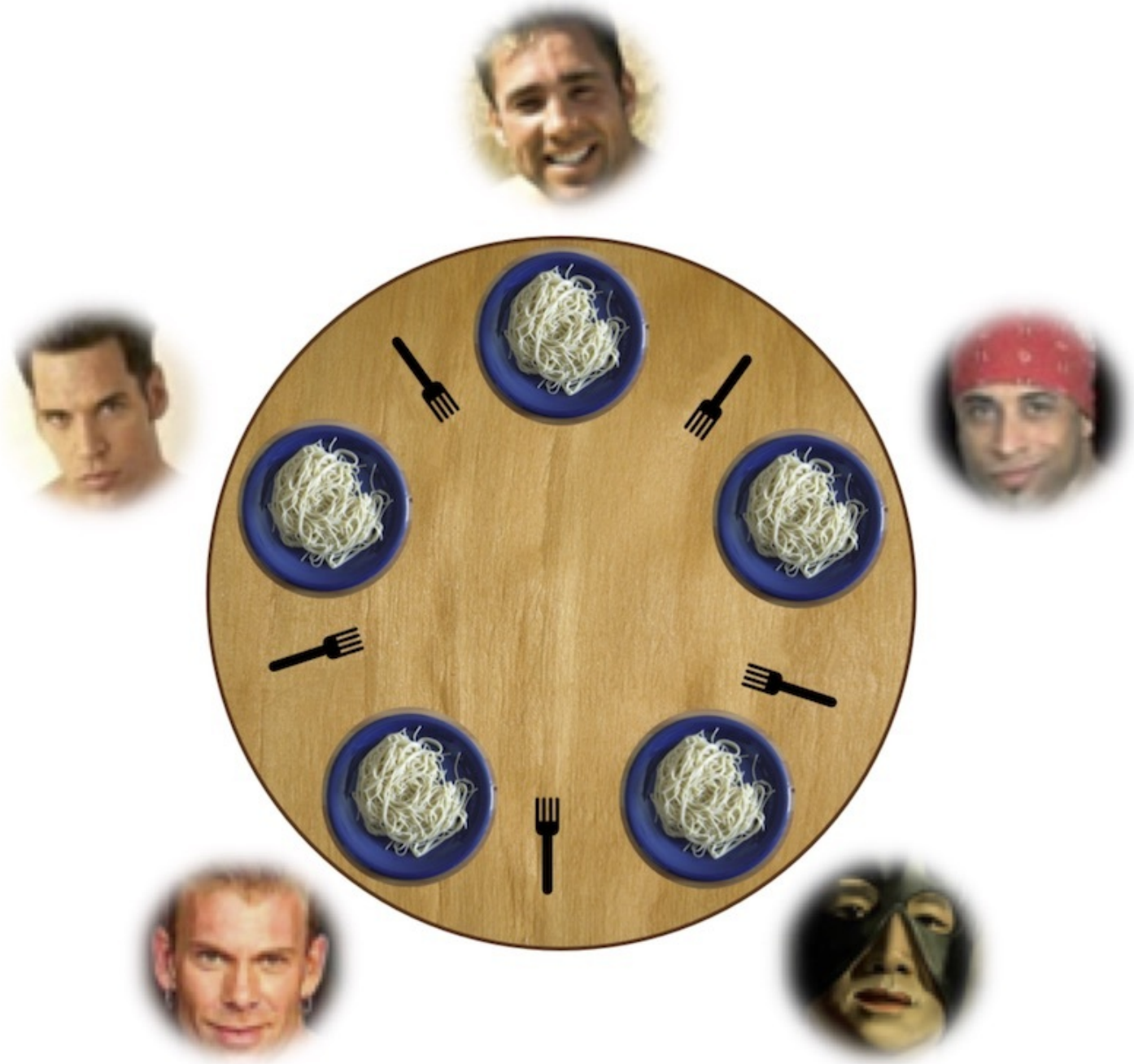
知乎 首发于 软件工程技术研究漫谈



# 读博那些事儿

蒋炎岩
计算机系助理研究员



> 本文正式版《读博士的难与易》发表在《中国计算机学会通讯》(CCCF) 2019年第8期 (CCF 优博专题)。在这里可以无拘无束，放飞自我了。这篇文章的结构和严肃版不太一样，技术的内容放到了最后，大家最关心的“怎么守住发际线”放在了前面。

## 0. 背景

发。

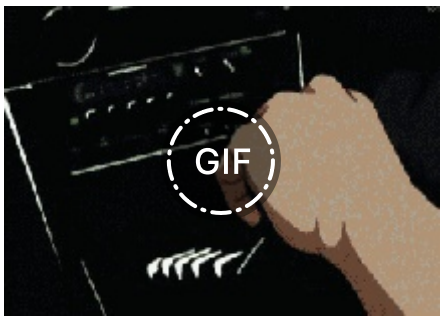
先交待一下为什么要读博。其实就是觉得想再浪几年做点什么有意义的事情 (真实原因是自己很懒没有考 G/T)，而且 Top 2 也不在南京，没办法就在南大读吧。做这个决定的时候完全不知道在国内读博意味着什么，就随便找了个据说很牛逼的组把自己给卖了。

那个时候人工智能已经很热了，为什么没选呢？是因为从小被数学好的人吓怕了……这里要提一下我小时候一起长大的两位好友，一个是 Richard Peng，另一个是 Zeyuan Allen-Zhu。尤其是 rpeng 那种过目不忘的神人，从幼儿园开始就相爱相杀 (哈哈哈)，后来就搞不过他了。再后来人家去了加拿大，眼看着他各种 STOC/FOCS/SODA，又后来当了美国队长，不跟咱玩了。


开玩笑。我觉得他给我树立了一个很好的榜样，大概是“知道厉害的人有多厉害”。既然玩不来这个游戏，就一定要找一个要积累足够的经验才能玩起来的领域：

> 永远不要跟一个傻X争论。他会把你拉到他的水平上，然后用他丰富的经验打败你。

对任何领域来说，经验都非常重要。不过数学的 training 有点从小就落下了；加上当时对系统软件还挺感兴趣，读博的时候南大基本没有人做 System 和 PL，所以选了一个最接近的软件方向，虽然我也不知道软件方向是做什么的，就这么上车了 (为此还让出了一个保送硕士的名额，嘿！) 这个路线和广大保研群众简直相似得不能再相似了。






## 1. 守住发际线

结起来就是：**脸皮要厚**、**药不能停**、**刻意练习**，然后在**发表论文的边缘疯狂试探**就好了。

## 脸皮要厚

我入坑的第一个研究问题简直就是四九年入国军，到今天这个方向的好论文已经非常……稀疏了。概括地说就是在基础的机制已经很难改进的前提下，设计策略去 “overfit” 应用场景，刷出更好的实验效果来。那段时间几乎每天要都拷问自己：做这东西有个卵用？试一试吧，实验设备又相当落后，只要方位角度差一点点，测出来的数据就飞到十万八千里之外了。但毕竟老板是 “软件方法学” 出身的 (虽然我也是做 “软件方法学” 的，但我大约的确不知道我到底是做什么的)，他们自带能说会道的光环，每次都能在方法学的制高点忽悠得你哑口无言，新手学生真是百口莫辩，项目也就一直这样莫名其妙地进行下去了。

后来我学会了一个经验，现在和老板们交流时，头脑里先预备好一个 SMT Solver。老板每说一句话，我就先把这句话在逻辑上取反，然后扔到 SMT Solver 里求解一下，看看他到底是怎么把这件事忽悠过去的。我惊讶地发现，求解的一般结果，要么是公理体系有些不同，要么是在一些非常基本的问题上没能达成一致。一个典型的例子是，老板们有时候会觉得 “再小的 contribution 也是 contribution” ，但如果你有一个预设的 significance 的 lower bound，这句话就不成立了。这一招屡试不爽，每次都说不过老板的同学们可以多多尝试 (许畅 ：有的时候说的话还看环境，对有不同抗打击能力的人要斟酌说不同的话，可以细细体会老板是鼓励你还是安慰你——不要把老板安慰你的好心硬戳破了)。

我觉得知乎上遇到自己不喜欢研究问题的博士生肯定相当多。这时候，脸皮厚的作用就发挥出来了：**我有腿，不喜欢这个问题可以跑路啊**！


我先后鸽了两个研究方向，科研进展为零 (但本人心态比较好，没有掉头发)。然后在选第三个研究方向的时候 (研一结束时，进组两年后)，非常非常非常感谢许畅：他非常谨慎地在 ICSE’ 12 的时候戳了 HKUST 的张成志教授，说我们这有个学生，给他点难整的问题整整。张老师就说，嘿嘿嘿不好做的问题那是大大的有，可以试试并发，给了两篇相关的paper。于是我就这样上车了——我们组没有人是这个领域的专家，甚至做程序 (静态或动态) 分析的人都没有。

PL、系统和硬件，可想而知在此过程中补充了多少基础知识。因此，我现在尽可能给新人推荐这种已经很多人研究过的领域，虽说在这种领域发论文可能相对更难一些，但能学到非常多的东西。相反 (也是另一种很常见的情况)，你如果进入一个特别小众的领域，只有几个人在玩，入门视野就窄了、格局就小了，就算能灌很多论文，我个人觉得不是很利于今后的职业生涯。

师傅领进门，修行看个人。这次终于算是入了豪门，老板们的支持功不可没。

### 药不能停

鸽人可以，学习姿势不能停 (废话)。

我觉得自己还算是个热爱学习的人，在那个时候读了很多与研究方向无关的教科书、论文和公开课，也听了很多学术报告。我记得 李沐 在PhD 回忆录里提到了“Parallel and Distributed Computer: Numerical Algorithms” 这本书给他很大启发，我也读过，写得非常好，那些经典的视角在今天都不过时。以及时候开始读杂志 Communications of the ACM (CACM)，到今天应该看完 10 年份了 (入坑之后往前补了不少，但后来也坚持不下去了)。虽然很多文章并不能完全读懂，但大部分时候都是在 “开眼界”，看看其他领域的人在用什么方法解决什么问题，算是构建计算机科学的世界观。

另一个成功的例子是 Tim Roughgarden 在算法课上一句话提到了一份 USENIX Security’03 的工作，然后就这一句话启发我们做了正则表达式复杂性攻击的工作 (当然做研究没有那么顺利，中间的波折就忽略了)，并且很意外地获得了 ACM SIGSOFT Distinguished Paper (真是意外，其实是 Conditional Accept，差点挂了)。这个论文的点睛之笔是非常规地使用了Pumping Lemma，已经不记得是什么时候在什么地方学过这个定理，但的确那一瞬间就想到了。

学习经历培养一个人的研究品味 (taste)，所以不必想着眼前立即得到的好处。具体来说，“学习”就是帮助博士生训练一个分类器，能判定“什么东西是好东西”，有真正的 contributio
好东西看多了，你看到
但无论如何靠自己都是

## 刻意练习

另一个经常遇到的问题是怎么阅读论文。网上也有很多建议、方法，我自己的理解是看论文有些类似神经网络的训练 (许畅：一定要明白人的学习能力远超过机器，其通过极少例子领悟到真相的泛化能力是极强的——想想悟道)：在看到研究问题以后，先试图给出自己的分析和方案，然后再看作者的做法，如果有自己没能想到的奇思妙想，再通过“反向传播”纠正自己思路的盲区，重点是把作者的方法用自己容易解释的逻辑解释出来 (这一步非常非常重要！)。老板有一句话说得非常在理：必须阅读 100 篇甚至更多的论文，才能对一个研究领域有一些基础的感觉。

其他方面也是可以练习的，比如写作和写代码 (因为我比较喜欢写代码，所以后者没有花去太多的时间也不痛苦，觉得 hacking 很好玩)。第一篇论文的写作被搞惨了 (有图为证，此处 @许畅 )。即便改到这个程度，因为底子太差还是被审稿人骂了，说这论文做的不错，写的也勉强能看懂，但行文真的是一坨屎，连标题都有语病。

收到审稿意见以后吓坏了，赶紧去学习了一些公开课、教程(有些老板给研究生的建议写得真的很棒，比如 Manuel Blum 给研究生的建议)，还有万能的 stackexchange。等到 camera ready 的时候已经有了非常显著的提高了；再后来自己也能在非常短的时间里写出读得过去的论文了。

## 在发表论文的边缘疯狂试探

在对相关工作有已经有一定了解的基础上，我有了 (若干) 个想法。其中一个我觉得“很有前途”的想法，经过跟老板讨论被否定了，觉得这东西不就是个 XXXXXX 嘛，没啥卵用。另一个我觉得“没前途”的想法，老板们一直在怂恿继续搞 (很可能是出于鼓励的心态，但我至今都不觉得这个 paper 能发得出来)。再后来一个我觉得“挺一般”的想法成了我的第一篇论文。

再之后我发现 Mike B



文，但实现得非常好，

受到了 “方法学” 的危险，有时候甚至觉得方法学就是逃避的借口——别人发了个什么论文，也就是把这么几个想法拼起来，提升了这么一点，不过如此嘛。但如果让大谈方法的人去实际做一个，很可能会因为缺少对其中各种坑的认识做不出来。

但突破天际的东西哪那么容易能做出来呢？不过也没事，这件事一方面让人有些不爽，另一方面还是让人挺爽的：毕竟自己的脑袋还是好使的，这次在 state-of-the-art 的边缘，也许下次就超过了呢。反正 99% 的 idea 都是别人已经想到或者没有用的，但小概率事件大量重复必然发生，只要每天坚持拍脑袋，脑袋一定会秃的，哦不，是一定会有靠谱的 idea 的。只要因为拍脑袋损失头发的速度不及头发生长的速度，发际线就能保住了。我曾经听说过 Yuanyuan Zhou 组里的轶事，每周组会的时候强行要求每个人出一个新的 idea，还会给好的idea 颁奖。这个机制压力超大，但的确成效很好。

我写第一篇论文的时候，就只是想做点什么好玩的东西，但之后就不满足于此了，总是想做更好的、有真正研究贡献的东西。从发第一篇顶会开始，保持自己研究工作的质量上升就已经是相当困难的了。所以就算从灌水开始，也不要有太大的负担，来日方长嘛。

读博大概就是这么简单，功夫到了该有的就有了，觉得痛苦的可能需要回去补基本功，或者是看看自己在大方向上有没有犯错误。

---

其实关于守住发际线，上面说的都是骗人的。真正的原因是我遗传了我爸优秀的基因。但关于做研究的故事，都是真的，有坚持就会有回报。

## 2. 奇闻趣事

### 拖延症

谁没有拖延症呢？我拖了两把超大的。第一次是因为论文得以发表，可以找机会出国转转，留学基金委掏钱，有个申请的流程。结果我把联系导师的事情拖延到了截止日期的前一天。对，你没看错，我们花一天时间联系了一位老板仅有一面之缘之人，在 24 小时内搞定了推荐信和材料。在材料交出去的那一瞬间，我只知道这个人做的东西好像和我有那么一丢丢关系。

有时候觉得运气来了挡也挡不住：

1. 我很惊讶地发现我读



性的论文，非常喜欢，

3. 实验室的大哥们 (按辈分算其实是师弟) 对我十分照顾，其乐融融大家庭，让美帝的科研工作也十分愉快。

我拖的第二把超大的是花两周时间写完了毕业论文。记得 2017 年 9 月初，我问老板是不是可以毕业了，老板说好啊，一般是 11 月交论文，嗯我们来看看……一看不得了，距离论文初稿提交还有两周多一丢丢，而那个时候我写完的部分大概只有 5 页。

这时候，是个懒人的好处就体现出来了：我会用工具啊！之前写论文的时候总觉得 LaTeX 无论用什么工具编辑，多多少少都会干扰写作的顺畅。我总结很大的原因是文字和代码混排在一起，比如用等宽字体吧，文字读起来怪；用 Serif 吧，代码没法看。所以之前的几篇论文，我就开发了自己的 LyX 模板。期间我修正了很多不满意已有学位毕业论文模板的地方，全文2632 个公式、40 个插图、12 个附表、144 篇参考文献，两周就搞定了，虽然提交以后到答辩前花了近一个月的时间玩命polish……

知乎 首发于 软件工程技术研究漫谈

这种极限操作的能力，可能来自于以前参加ACM-ICPC，三人一台机不太够用，所以练习在纸上写完整的代码。到后来可以做到几页纸的代码，proof read 一遍，敲进去编译过就能交，过了就是过了，不过稍微测点数据再读一遍代码也能搞定。总体来说，我在读博期间甚至到今天依然经常极限操作，但几乎没有错过任何自己预设好的 deadline，所以好像给老板 (以及其他共事者/合作者) 一个我很靠谱的假象 (实际上是因为总还有更不靠谱的人)。

## 神奇的美帝

在美国短短半年的访问让我感到非常舒适：所有的琐事都甩锅了，可以非常静心地体验一下不一样的生活。经常顺着 Olentangy River 步行到学校，从盛夏到秋冬，一路上的风景变化都还历历在目。

知乎 首发于 软件工程技术研究漫谈

在研究方面，我们做了一个很有趣的小工具：用 ptrace 拦截系统调用，用虚拟设备收集块设备的日志，然后用一些算法上的小技巧来自动检测应用程序中的崩溃一致性缺陷。因为之前玩过很多东西，所以做起来可以说是相当顺手，半年的访问期间就完成了选题、代码、实验，找到了包括 Coreutils、Gzip 等软件里的缺陷，论文也顺利发表。最近我收到了 Perl 社区的感谢，在最新的稳定版里修复了我们报告的 bug。回顾自己求学的经历，技术方面的积累都是互联网和开源软件培养的，自己一直在索取，如今能做一些微小的回报，欣喜甚至胜过博士毕业。

老秦 (forgive me! 实验室里的兄弟们都是这么称呼的哈哈！) 给了我很多指导，还教导我们两件事：“Get out of your comfort zone”，和 “Don’ t give up easy”。第二条我要特别提一下，我自己的很多想法在做的过程中很多次都觉得可能做不下去了，但之后总是找到办法又捞回来了。我记得很清楚，在美帝时论文已经基本成型，但缺一个 NP-Completeness 的证明(课后习题难度)，不过那时候可能脑子有点短路，很久都卡着。有时候想着不证明就不证明吧，反正不影响论文的结果，但忽然某一天我坐 Shuttle 回公寓的时候就想到了关键步骤，那时候正值俄亥俄州的冬天，但忽然就觉得一点也不寒冷了。

这一小段访问经历还很大程度地影响了我的价值观，我之后就爱上了“把软件放在地上摩擦”。我现在的风格是 “For fun and profits”，首先为了程序员的那种自我满足，其次是创造一些对社会有用的实际价值。我想出了一个评价自己研究工作的 metric：我如果做了个东西，我要想一下那些在 Google/FB 工作的码农同学们觉得怎么样，如果他们觉得无趣，那还是趁早放弃来得比较好。我觉得在美国的半年对我生涯产生了很大改变：从一开始只是为了 “做点什么不一样的东西”，到现在想真正做点 “有用的东西”。

最后，在美帝也没少出去玩，美国的历史不长，但每一个城市却都保留了很多旧时光的印记。

## 3. 劝退

所以你还想读博士吗？好像也没什么难的——找一个 (足够) 困难的方向，把相关的论文都读了，然后死命想还能做什么就完事了。当然要事情真那么好，读博士也就不至于知乎上说得那么惨了——没遇到对的人、如果脸皮不够厚……一步走错就可能半途夭折。现在中国的学术界在一个惊人的转折点上，随着下一代人从小受到的训练越来越好，我那点三脚猫的基本功迟早是保不住的。做研究可能并不是靠被动地“学习”就能搞定的，长江后浪推前浪，前浪立马就死在沙滩上。现在我觉得自己大概在活下去的 “及格线” 上——从小我自己就不是个 hardworking 的人，但 fully focused 的时候还是可以做出一些差强人意的东西。如果我读博士时候再努力一些，杂七杂八的事情别管那么多：

- 该死的系统实验课，哦没什么，也就是让本科生自己写个 CPU，上面跑个自己的操作系统以及应用程序。这也没啥技术含量 (没少人做过)，就是花时间；
- 带江苏省队的训练和省级竞赛的组织等，维护评测机、竞赛现场的各种脚本和临时背锅，当了好多年的救火队长；
- 带着 ACM-ICPC 集训队训练，再早些时候还打打 TC，在红名的边缘疯狂试探……

然后游戏再少打一点，不要谈恋爱结婚 (已屏蔽老婆)，publication list 也不会就这么孤零零的几篇了。我经常和学生们说，如果你们连导师都不能正面刚，还是谨慎考虑一下读博这件事吧。

知乎 首发于 软件工程技术研究漫谈

晒个娃！

因此在这里必须劝退一波想要 “通过读个博士过上安逸日子” 的人，这条路越来越行不通了。如果你还没有被劝退，恭喜你！读博的经历是独一无二的，如果当了一个社畜 (我曾经有过无数当社畜的机会)，就不会有那么多精彩的发现了。

## 简短的致谢

走上计算机科学的道路多多少少算是件 “命中注定” 的事，大概要从幼儿园起相爱相杀的基友彭泱 (Richard Peng) 开始吧。他是个非常神奇的人物：记忆力超强、解题速度超快、受到的训练超好。这个基本上是我能见到的 “天花板” ，虽然如他所说，做数学的人更可怕，不过好像那些人并不在我生活的世界里。然后一路都遇到很多有趣的人 (无法一一致谢)，包括大学时代一起参加 ACM-ICPC 的队友李昂和李珅，度过了非常愉快美好的本科时光，在美国重逢的时候还一起旅游了一场 Regional (说好的 AK 变成了各种 WA 到死)。到读博期间，我的导师们 (吕校长、马所长、许畅和秦锋老师) 都非常支持我，给了我超棒的研究环境，甚至有时候都不好意思差遣我当马仔，我自己是觉得不太好意思的 (我有一个做马仔的觉悟，但好像也许因为自己有同学多做了一些马仔活)。

一段精彩的生活告一段落，谨此纪念。

p.s. 感谢老板许畅在文中标注的 “老板内心独白” 。

知乎 首发于 软件工程技术研究漫谈

## 4. 附录：我到底做了个啥玩意?

### 从“哲♂学家吃饭”问题说起

从某种意义上来说，博士期间我是研究哲♂学的。“哲学家吃饭问题”是大学时代学习并发编程时的经典难题。现在哲学家们受到盛情的邀请来到舌尖上的中国，面对一桌丰盛的菜肴，他们不再受限于必须同时拿起手边的两根筷子才能用餐，因此可以更尽兴地享用美食。另一方面，吃中餐的哲学家们也带来了一个新的问题——聚餐是一个很复杂的过程，我们能否把吃饭时发生的事情像讲故事一样，完整地复述出来?

如果把哲学家看作是线程，每道菜看作是变量，那么每个哲学家在吃饭的过程中都可以多次执行以下两个操作：(1) 观察某一道菜，在瞬间记下这道菜的模样 (相当于读出变量的值)；(2) 吃某一道菜，在瞬间改变它的模样 (相当于为某个变量写入一个值)。哲学家们都是了不起的天才，在吃饭的过程中都已默默地记下自己所知的一切，即每个线程记录下自身发生的读/写操作的变量和它们的数值。

之后的一天，哲学家们重新聚在一起，试图解决“哲学家吃中餐问题”：他们各自整理了对那一天吃饭过程的记忆，并试图一起还原出吃饭的完整过程。更确切地说，我们希望在程序执行结束后，把线程本地的读/写操作日志合并成全局的变量读/写序列，相当于给所有读/写事件分配发生的先后次序，使得每个读事件读出的数值都等于最近一次对该变量写入事件写入的数值（即根据线程本地日志得到全局满足顺序一致性的事件排序），如图 1 所示。Gibbons 和 Korach 在1997年对于“哲学家吃中餐问题”给出了一个颇为悲观的答案[1]：如果 P ≠ NP，再聪明的哲学家也无法在多项式时间里恢复出满足顺序一致性的事件排序。甚至，即便哲学家吃的是火锅(只有一道菜，即一个变量的情形)，NP-完全性也是成立的。

图1： “哲学家吃中餐问题”示意图。R/W分别代表读/写事件，红色表示满足顺序一致性的线程调度

**哲学家和摄像机**

“哲学家吃中餐问题”为什么重要？想想我们身边的并发程序——操作系统内核、云计算/分布式系统、服务器应用，大家都有的体验是它们很难写、更难写对。因为不确定性的存在，即使我们已经知道程序里有缺陷 (bug)，要想复现缺陷触发的过程都很困难，更不用说测试了。哲学家吃中餐问题恰恰就是要解决从日志中恢复出程序执行过程的难题，这对并发程序调试技术的重要性是不言而喻的。不仅是调试，我们还可以针对观测到的一次执行预测程序中的缺陷，例如数据竞争或use-after-free；甚至在运行时就对程序的行为进行干预，避免并发缺陷被触发。这一大类基于程序执行轨迹实现的技术统一称为并发程序的动态分析技术。

“哲学家吃中餐问题”的 NP-完全性 (固有的困难) 并不意味着我们无法在实际中解决它。为了实现并发程序的动态分析，必须在运行时观测并发程序的执行，而观测却并不局限于“线程本地的记录”。我们希望有一台“摄像机”拍摄下哲♂学 (家吃饭) 的全过程，而修改程序的代码或运行时环境，恰好能实现这样的摄像机：为所有共享内存的读/写操作上锁，并在锁的保护下记录它们发生的顺序，就得到了共享内存访问的日志，如图 2 所示。从此意义上，观测并发程序执行又是容易的。

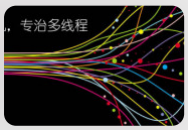
图2：修改程序代码对共享内存读操作进行观测 (标记“►”的为插入的代码)

难与易之间的矛盾是观测并发程序执行的开销。虽然锁可以实现有效的观测，但却降低了系统的并行度、拖慢了程序的执行。并发性存在于计算机系统栈的各个层次上，因此来自体系结构、计算机系统、程序设计语言和软件工程领域的研究者，全都对实现高效的并发程序动态分析有兴趣。在三十多年的研究历程中，我们看到了很多非常精彩的锁机制的实现：借助硬件的缓存一致性协议[2]，使用虚拟机、分支计数和用分页机制实现的CREW协议[3, 4]，以及在线程局部访问下“乐观”的锁机制[5]。另一方面，我们可以通过记录间接信息 (如每个线程执行的路径)，使用约束求解的方式去恢复共享内存访问的日志[6]。我们在这个问题上做了一些微小的

[7]。

在观测并发程序执行的基础上就可以实现各式各样的动态分析技术了。这里举一个我们测试并发程序的例子[8] (参见知乎专栏中的科普文)：






试想我们把线程中的读/写事件按顺序串在一根绳子上 (图 3)，处理器按照固定的速度按顺序执行绳子上的事件。改变绳子的长度 (例如拉绳子相当于把线程执行的速度变慢) 就得到了不同的线程调度——那些触发并发缺陷的调度，很可能隐藏在某些绳子长度的配置中。我们设计了聪明的策略生成具有多样性的线程调度，在先前研究者已经反复测试过的并发程序上找到了前所未知的并发缺陷。对于复杂性日渐增长的并发程序来说，动态分析是一项非常有前景的技术，对具体内容感兴趣的读者可以参考我们的中文综述[9]。

图3：基于线程调速的并发程序测试技术示意

**有没有免费的午餐?**

观测并发程序执行是非常基础且重要的问题，来自各个领域的研究者都取得了丰硕的成果，但唯独“完美”现在还做不到。现有观测并发程序执行的工作可分为两类：要么存在一个NP-难的最坏情况，要么不可避免会在某些情况下用锁 (类似图 2 的方式) 保证一次观测不被其他线程打断。

“哲学家吃中餐问题” 的 NP-完全性一定程度上反映了现有研究工作面临的困境——如果只允许程序进行少量的线程本地记录，则恢复满足顺序一致性的全局调度是困难的。另一方面，我们也已经知道共享内存上的互斥和可序列化并发对象必须借助读-写原子操作才能实现[10]。这启发我提出了一个猜想：观测并发程序的执行没有免费的午餐 (no-free-lunch)。具体的陈述是，在一个限定的计算模型下，即

顺序一致性的程序执行是NP-完全的。

如果这个猜想成立，就对这30年的研究成果给了一个“二分性”的总结——想要观测并发程序的执行，要么添加处理器之间的同步，要么付出 NP-完全的代价。这意味着观测并发程序执行是既困难又容易的。目前我们相信这个猜想是成立的。在试图证明它的过程中，我们对“哲学家吃中餐问题”给出了的一个新的（简化的）证明，并据此得出了一些有用的结论，例如在对程序的修改能写成只读前缀 + 只写后缀的情形下的 NP-完全性。受限于掌握的数学工具，我们还没能完全证明或否定这个猜想。如果猜想被推翻，我们就更惊奇了——说明 30 年来大家的努力都有本质上的不足，抑或 P = NP，也许我们就需要重新理解整个计算机科学了。

上面都是假话。这个猜想很早就想到了，但一直找不到合适的数学工具证明，加上研究方向的转变，猜想就只证明到这个程度 (也是个课后习题难度)，其实是博士论文烂尾了……

在试图解决这个猜想的过程中，其实给了我们很多启发——猜想必须在很多限定条件下才能成立。因此只要假设的条件被推翻，观测并发程序执行就变得不困难了。例如，我们的复杂性结论是在 “最坏情况” 下得出的，即存在一个 NP-完全的“极端”线程调度。但最坏情况也许像平滑分析[11]中指出的那样，在实际中很难存在。现实中的并发程序执行有它的特征 (例如各种局部性)，也被我们用来降低观测并发程序执行的开销。就在难与易之间，理论和实践得到相互的印证。然而在理论与实践中，我们都仍有很多未解决的难题，博士读了很多年，反而觉得研究才刚刚开始，而不像是告一段落了。

## 参考文献

[1] PB Gibbons, E Korach. Testing shared memory. *SIAM Journal on Computing*, 26(4), 1997: 1208-1244.

[2] M Xu, R Bodik, MD Hill. A “Flight data recorder” for enabling full-system multiprocessor deterministic replay. In *Proceedings of the International Symposium on Computer Architecture* (ISCA), 2003.

[3] GW Dunlap, ST King, S Cinar, et al. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the Symposium on Operating Systems Design and Implementation* (OSDI), 2002.

[4] GW Dunlap, DG Lucchetti, P Chen, et al. Execution replay for multiprocessor virtual machines. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (VEE), 2008.

*Object Oriented Programming Systems Languages & Applications* (OOPSLA), 2013.

[6] J Huang, C Zhang, J Dolby. CLAP: Recording local executions to reproduce concurrency failures. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), 2013.

[7] Y Jiang, C Xu, D Li, et al. Online shared memory dependence reduction via bisectional coordination. In *Proceedings of the International Symposium on the Foundations of Software Engineering* (FSE), 2016.

[8] D Chen, Y Jiang, C Xu, et al. Testing multithreaded programs via thread speed control. In *Proceedings of the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE), 2018.

[9] 蒋炎岩, 许畅, 马晓星, 吕建.获取访存依赖：并发程序动态分析基础技术综述[J].软件学报, 28(4):747-763, 2017.

[10] H Attiya, R Guerraoui, D Hendler, et al. Laws of Order: Expensive synchronization in concurrent algorithms cannot be eliminated. In *Proceedings of the ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages* (POPL), 2011.

[11] DA Spielman, SH Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3), 2004: 385-463.

编辑于昨天 21:50

读博 博士 计算机科学

**文章被以下专栏收录**

**软件工程技术研究漫谈**
不仅是职业码农，越来越多的人都学会了编写程序来解决日常生活中的各种问题，但...

进入专栏

**推荐阅读**

知乎 首发于 软件工程技术研究漫谈


**我们一起走过的六年直博生涯--2**

Lstone Woo

**判断自己适合读博与否，要从这几个方面考虑**

老杨叔聊志... 发表于老杨聊志愿...

**你到底适不适合读博士 一个博士的透彻分析！**

本文来源于中国科学人才[ 是适合继续读博士，这是[ 临读博的学生甚至开始读[ 的困惑。 基本概念 博士: A Doctor of Philosophy 简称Ph.D.，是最高学位...

Boom 发表于[

**14 条评论** ⇌ 切换为时间排序

写下你的评论...

知乎用户 10 小时前

大四的时候作为外专业的学生听了蒋老师的课，非常谢谢蒋老师让我发现了，南大有很多在发光的人

赞

林一二 10 小时前

这题图...哲♂学家问题

赞

X Giorgio 9 小时前

老哥，请抬一手。

赞

猫在乎 9 小时前

没想到蒋老师还是哲学家

赞

张凯成 9 小时前

当时我高二（2013）的时候还听过你的课 受益匪浅 哈哈哈

赞

杰杰杰

▲ 赞同 178 ▼ 14 条评论 分享 收藏 ···

赞

Ruler 7 小时前

一直都在等这篇

1

SODA 7 小时前

原来和彭大神还有朱大神是同学，跪了[笑哭]

赞

小小冰 7 小时前

蒋老师是一个优秀的科研工作者，也是一个非常称职的老师

赞

Shercklo 7 小时前

蒋老师

赞

赵夏 6 小时前

直观上想，用粗粒度的大虚拟变量sync会降低性能?

赞

陈越琦 6 小时前

jyy牛逼!

赞

没有银弹 5 小时前

蒋老师觉得给本科设计的那个课对自己没什么帮助吗[捂脸]

赞

Climber.pl 3 小时前

Dinner philosophers 其实展开讲挺有意思的，还有个工作证明了对于 quantum protoco

并没有dead lo

赞

知乎 首发于 软件工程技术研究漫谈