算法工程的思想、方法与工具
——以SAT算法为研究实例

焦加麟　韩晓峰　徐良贤
（上海交通大学计算机科学与工程系　上海200030）

The Theory, Methodology and Tools of Algorithm Engineering: A Case Study with SAT Algorithms

JIAO Jia-Lin　HAN Xiao-Feng　XU Liang-Xian
(Dept. of Computer Science and Engineering Shanghai Jiaotong University, Shanghai 200030)

Abstract The recently emerging idea of Algorithm Engineering in the community of Computer Science is about how to engineer computer algorithms by integrating the design, analysis, implementation, experimental testing, refinement, etc. of them, aiming at improving their runtime performance.

In this paper, we try to develop the main idea of the theory and methodology of algorithm engineering, and give an introduction to some useful tools. In order to concretize our discussion and illustrate the effectiveness of algorithm engineering, we will base our study on the engineering of SAT algorithms. According to the rules and methodologies of algorithm engineering we have learned, we implement a simple but efficient SAT solver, QuickSAT, as a testbed. Although we haven't utilized most of the advanced techniques used in modern SAT solvers, the result of experimental comparison shows that QuickSAT is close in performance to zChaff, one of the leading SAT solvers in the world today.

Keywords Computer algorithm, Algorithm engineering, SAT

1. 什么是"算法工程"?

20世纪90年代中期,"算法工程"这个词开始频频出现。A. V. Aho等计算机科学家曾在1996年ACM计算理论研讨会(STOC'96)上呼吁研究者们要重视算法工程的研究和实践[1]。从那一年开始,算法工程越来越受到人们的关注,国际上每年都要举办一次算法工程研讨会。但作为一种新的思想,目前"算法工程"这个名词仍然缺乏精确、统一的定义,也难以找到相关的系统化论著。经过综合大量相关研究与实践成果,加上我们自己在算法研究中的实际经验,我们总结出了算法工程的几点基本内涵:

1.1 算法工程的出发点和目的

算法工程的出现可以说是为了解决算法研究中存在的"危机"。这种"危机",笼统地说,就是算法的理论研究与实验工作之间的距离。这种距离真的存在吗?让我们看一些统计数据。Cohen曾对1990年发表的150篇AAAI文章进行了统计[2],结果(见图1、2)表明这种距离是确实存在的。这种"危机"的具体表现包括:

第一,算法分析结果与实际性能之间的距离。特别是对于复杂算法,传统理论分析的结果与实际性能有较大的出入。其中的原因包括:①传统算法复杂度分析基于过于简化的计算模型,主要是RAM(随机存取机模型)极少考虑现代计算机广泛使用的多级存储(虚拟内存、高速缓存)、并行处理(处理器级、指令级)、数据通信等技术因素……而这些因素却影响着算法的运行效率。②传统渐近分析忽略掉所有常数项和低阶项,则当算法具有很大的渐近常量或低阶项系数时,算法实际运行效率会比预计的要差很多。

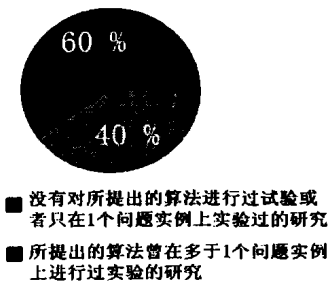

图1　理论研究:缺乏足够的实验支持

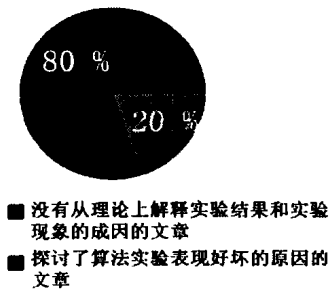

图2　实验工作:没能足够的理论分析

第二,算法设计与算法实现的脱节。算法研究中存在"纯理论性"研究的现象,由此产生的算法只是在理论上行得通,而并没有经过实验验证,到了人们真正要实现的时候,可能会遇到难以克服的困难,或者实际效率跟理论分析的结果有很大出入。反过来,不少算法的实现者的理论基础又可能相对薄弱,所以,依靠他们去改进算法也是比较困难的。

焦加麟　硕士研究生。韩晓峰　硕士研究生。徐良贤　教授、博士生导师。

第三，不同的算法实现之间的差距。同一个算法一般可以有不同的实现。当算法比较复杂的时候，一个好的实现和一个差的实现之间的运行效率可能会相差好几倍。

因此，算法工程思想的出现，是现代算法研究与开发的要求，其目的是为了更好地实现算法，提高其运行性能。

1.2 算法工程的研究内容

算法工程思想要求把算法开发中的各过程，包括算法设计、复杂性分析、实现、测试、实验结果分析、性能调节、算法改进等，看成相互作用、相互关联的整体，而不是各自孤立的。在进行前面阶段的工作时，要把后继工作的因素考虑在内；而后继工作的结果又可以引起前面阶段的工作的重新进行，反过来影响前面阶段的工作。这就是算法开发过程的一体化。算法工程的研究内容和任务就是要为算法开发的各个过程提供指导原则、方法与相关工具。

我们将在第3节中给出一些我们总结的算法工程的方法和规则。至于与算法工程的方法配套的工具，主要包括：①算法库：算法工程提倡建立高效、通用的算法程序库。这方面的例子有 C++ 的 STL(Standard Template Library)、LEDA(Library of Efficient Data Types and Algorithms)、Stony Brook Algorithm Repository 等等。②算法开发和分析工具：包括算法开发环境、性能跟踪器(profiler)、算法可视化演示系统(animation system)、算法环境模拟系统(simulation system)等等。③基准测试集和测试工具：其中后者包括测试实例自动生成程序和智能自动测试代理等等。

1.3 算法工程的核心思想

算法工程强调实验，包括算法的基于实验的性能分析、测试、比较、认证等。实验是研究算法的一种有效的手段，它们能够评估算法实际性能、验证理论结果、指导算法的修改等等。

当然，强调实验绝不是说要放弃传统的理论分析方法，而是要把二者好好地结合起来。算法工程的核心思想，就是在传统算法研究方法的基础上，强调实验方法，以提高算法的实际运行效率。

需要注意的是，实验方法也存在很多"陷阱"，如果处理得不好，容易得到错误的结论。检验实验正确性的一个原则是实验的可再现性。任何算法实验，换了另外一个实验者、另外一个地点和时间，只要程序、测试数据、运行环境等决定性因素没有改变，其结果都应该是一样或非常相近的。如果不能保证这一点，原来的实验工作就一定有问题。

综上所述，所谓算法工程就是为了提高算法的实际运行性能，而将计算机算法的设计、分析、实现、实验测试、改进等过程一体化、工程化，并为这些过程提供方法和工具的研究与实践。

2. SAT 问题及其算法简介

我们使用 SAT 算法作为我们的研究实例，是因为：一方面，SAT 算法是一种复杂的算法，对时间(NP 完全问题)和空间(问题规模非常大)的需求比较高，实现的好坏，对其实际运行性能有很大的影响，因此，算法工程方法在 SAT 算法的开发中有着很大的应用；另一方面，不少算法工程中的思想和方法最初都是来自于 SAT 算法的研究与实践中的，因此，结合 SAT 算法的研究来讨论算法工程，是再合适不过了。

SAT 是英文 Propositional Satisfiability 的简写，SAT 问题就是命题逻辑公式的可满足性判定问题，即给定一个命题逻辑公式，判断是否存在一个赋值，使得该公式的值为真，如果存在，给出这样一个赋值。由于所有的命题逻辑公式都可以在线性时间内转化为合取范式，而合取范式能够简化可满足性的判定，因此，SAT 问题一般是指合取范式的可满足性问题。

解决 SAT 问题主要有两种思路，分别是完备(complete)算法和不完备(incomplete)算法。我们这里研究的主要是完备算法。完备算法的基本思路是：系统地搜索问题的整个解空间，以确定公式是否可满足。完备算法的优点是能够确定可满足性和不可满足性。完备算法的代表是 DPLL 算法[3,4]及其变种。DPLL 算法的基本框架是：深度优先遍历一棵二叉决策树，在树中的每个结点都有一个变量被赋值，左右子树分别对应该变量的不同赋值(0或1)。在搜索过程中，一旦在某个结点上出现了冲突(即到该结点为止的变量赋值使公式为假)，就不必再搜索该结点的子树了(剪枝)，回溯到冲突结点的父结点，如果父结点的另一子树没有遍历过，则遍历该子树，否则，继续回溯，如此下去……直到遇到一个满足公式的赋值或者遍历完整棵树为止(此时，公式是不可满足的)。DPLL 使用称为单子句原则的前瞻(lookahead)技术。反复使用单子句规则来对子句集进行化简的过程，称为布尔约束增殖(Boolean Constraint Propagation)，简称 BCP。在 BCP 过程中，冲突表现为空子句的出现，而公式被满足则表现为整个子句集变为空。DPLL 算法中另外一个重要的内容是在每个结点中选择赋值变量(称为分支变量)时使用的启发策略，该策略是否高效对整个搜索树的大小有很大的影响。改进的 DPLL 算法利用了许多提高效率的先进技术，包括智能回溯技术、冲突分析和学习、搜索的重启动和随机化技术等。GRASP[5]和 Chaff[6]算法都是改进的 DPLL 算法。zChaff 是 Chaff 算法的一种实现。在最近的 SAT2002竞赛中，zChaff 获得了第一名。

3. 算法工程的一些原则和方法

3.1 算法的理论分析

①在传统方法的基础上，提出新的计算模型，将更多的相关因素考虑在内。②使用更为准确的性能衡量尺度。在传统的算法分析中，人们使用与机器、操作系统、编程语言等无关的操作计数来衡量算法的性能。但是，现代算法研究的经验表明，在很多实际情况中，仅仅依靠估算操作计数，已经难以得到实用的结果了，特别是对于像 SAT 问题这样的 NP 完全问题。事实上，在比较完备的 SAT 算法的搜索代价时，使用得更多的尺度是访问搜索树的结点数和执行 BCP 的次数。③将渐进分析中的具有较大系数的低阶项和较大的常量因子考虑在内。④要重视算法的平均性能分析。尽管在理论上，最坏情况下的复杂度具有重要意义，但是，在实际生产生活中，算法的平均性能可能是更加实用的。⑤要把理论分析得到的结果，作为改进算法的依据，还要把它与实验测试的结果进行比较来验证算法本身和算法实现。

3.2 实现技术和数据结构的选择

除了要大力开发新技术外，也要注意利用前人的技术，可以直接借用或者改进后使用。例如，Chaff 算法中处理冲突的方案就是取自 GRASP 的，而其提出的新的数据结构 watched literals 也是对 GRASP 的 head-tail links 的改进。

而选择数据结构的原则包括：①要能够很好地配合算法的实现。②要高效。除了要在时间上(能够用较少的运算实现同样的目的)和空间上(占用较少的内存)高效以外，还应该能够更好地跟硬件等相关因素配合起来。例如，数据结构应该具

有较高的访存局部性以减少高速缓存的不命中和内存的缺页错误。③尽可能简单，易于实现和测试。④要具有好的可扩展性等等。

3.3 实现工具的选择

主要是程序设计语言、函数库或者类库等的选用。这里面的要求包括：①选择高效的，强大的开发工具。②该工具应该是特别适用于要实现的算法的具体情况的。③还要考虑开发者对该工具的熟悉程度，要使开发者能够较好地利用该工具提供的功能。

3.4 编码实现

实现的过程中，要善于利用实现工具所提供的便利，包括已经实现好的数据结构和库函数。此外，实现的过程中要注意发现在算法设计中没有考虑到的细节。例如，Lintao Zhang 等人在 zChaff 的设计时并没有认真考虑如何选择 watched literals 的策略，只是打算采用随机选择的方法，而在实现中，他们才确定了最佳的策略是选择最近最少被更新的变量所对应的文字作为 watched literals。

3.5 测试与实验结果分析

①测试实例的设计和选用。测试实例的选择非常关键，选不好的话，甚至可能会得到有偏差的结论。例如，有些在固定概率模型(constant probability model)随机生成的测试实例上运行效率很高的 SAT 算法，一旦遇到结构性的实例(指由现实问题编码而成的公式)，效率就变得很差。其中的原因是固定概率模型随机生成的测试实例本来就是比较简单的。只有找出了真正"难"的测试实例，才能更好地测试程序的正确性和效率。②正确性和健壮性测试。正确性要分别从算法本身的正确性和实现的正确性两个方面进行测试。至于健壮性，要在各种类型和各种规模的测试实例上进行测试。③对算法的性能进行实验评估和比较。对于各精心实现的算法，通过比较 CPU 时间，能对算法的真正性能差别给出非常贴近真实的估计。当然，CPU 时间上的差异仍然会因为实现细节和优化程度而引起，但是，实践经验表明，如果两个算法运行的 CPU 时间的差别很大，大于一个数量级以上时，大多数情况下是由于算法本身的特性所致。④算法参数设定。算法中一些可以调整的参数，例如 SAT 算法中的重启动策略中搜索层数限制值、随机性算法所用的概率值都要依靠实验的结果来决定一个最佳值。⑤评估算法策略，以决定对某种策略的取舍。这是因为，算法中的某些措施，在带来简化的同时，本身的实施也需要一定的开销，而到底是这个开销大，还是带来的效率大，很多时候必须通过测试来确定。⑥研究实验结果，改进算法。对实验结果进行研究，总结其统计规律，往往能反过来推动算法的改进。例如，对 SAT 算法回溯次数的概率分布中的 heavy-tailed 现象的研究导致了随机快速重启动(randomized rapid restart)技术的出现；又如，SAT 算法在解决随机测试实例中运行时间所呈现出来的"相变"(phase transition)现象的发现，也导致了基于局部搜索的不完备算法 GSAT 和 Walksat 等高效算法的诞生。

3.6 优化

虽然优化工作是在算法实现基本完成且可以上机运行的时候才开始的，但是应该从一开始就计划优化工作。同时，在开发的过程中遵循一定的原则有利于得到高效的程序。优化要借助于性能跟踪技术(profiling)，其做法是：用相关工具(profiler)查出程序中的哪一部分对运行时间起决定作用，然后用各种办法改进这一部分程序，努力使之运行得更快，然后再进行性能跟踪，再改进，直到程序的各个部分达到相对的平衡。除了利用 profiler 外，还可以利用系统的性能监视器来分析程序的性能。其中，进程的工作集(working set)是操作系统对进程性能的极佳的衡量尺度。

3.7 算法维护

要注意对算法不同版本的实现进行保存。通过对不同算法版本的比较，往往会有新的发现。

4. 实验

我们实现了一个基于 DPLL 算法的 SAT 判定器原型 QuickSAT。在开发的过程中，我们始终贯彻了算法工程的原则和方法：

(1)为了消除递归的开销，我们通过循环的方式实现 DPLL(原始的 DPLL 算法具有递归的形式)。

(2)基于平衡高效性和访存局部性的考虑，我们采用如下数据结构：①每个子句都由一个单向链表来表示，链表中的每个结点存放了子句中的一个文字；②每个变量 p_i 都有两个数组，一个是正文字指针数组 pos，一个是负文字指针数组 neg，分别存放指向各子句中相应文字的指针；③为了对赋值和回溯过程进行控制，构造一个数组 value 来存放对各变量的赋值，同时跟踪赋值的顺序，例如，对于公式{(p_1，p_2)，($\neg p_1$，p_2)，($\neg p_2$，$\neg p_2$)}，其搜索过程和 value 数组变化过程如图3(变量 CurVar 标记当前赋值的变量)；④每个子句对应一个栈。如果在化简中，消去了某子句中的一个文字，则将该文字从子句的链表中摘去，同时将指向该文字的指针压入栈中；如果消去整个子句，则将整个子句摘去，将指向子句头结点的指针压栈；如果没有修改该子句，则什么也不做。回溯时，按照要恢复的子句的栈顶元素记录的信息对子句进行恢复。上面例子中从开始到第一次回溯，各子句对应栈的变化情况如图4。

(3)分支变量的选择，采取 Chaff 算法中的 VSIDS 策略。

(4)进行冲突分析，并记录冲突子句。为了降低空间要求，采用了冗余子句删除的技术。但是，我们对冲突原因的学习只是为决策树未来的搜索中进行剪枝提供信息，并不使用智能回溯方法，因为我们认为智能回溯技术在带来效率的同时，也引进了不少的开销。

我们的程序使用 C++语言来实现，实现中充分利用了 STL 中提供的容器类和库函数。程序在 Linux 平台上开发，与 GRASP 和 zChaff 比较的实验是在处理器为 Intel® Pentium® III 800M(L1 Cache 32k，L2 Cache 512k)，内存是512M 的 SDRAM，操作系统为 Linux(内核2.4.18-3)的环境下进行的。我们在两类广泛使用的基准测试实例上进行了比较，测试数据(见附表)显示，QuickSAT 的性能超过了 GRASP，接近了 zChaff。

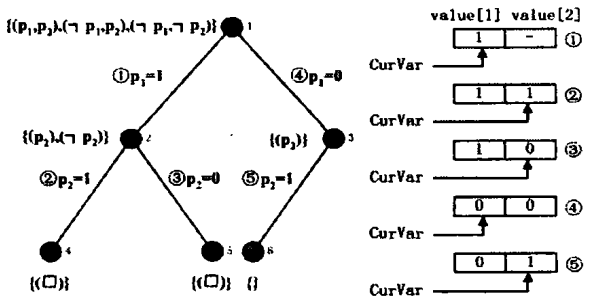

图3 搜索过程和 value 数组的变化

(下转第26页)

35：400～401
4 Jelinek F，Mercer R L. Interpolated estimation of Markov source parameters from sparse data. In：Proc. of the Workshop on Pattern Recognition in Practice，Amsterdam，The Netherlands：North-Holland，May 1980，381～397
5 Magerman D M. Natural Language Parrsing as Statistical Pattern Recognition：[PhD Thesis]. Stanford University，1994
6 Bahl L R，Brown P F，De Souza P V，Mercer R L. A tree-based statistical language model for natural language speech recognition. IEEE Transactions on Acoustics，Speech，and Signal Processing. 1989，37(7)：1001～1008
7 Rosenfeld R. Adaptive Statistical Language Modeling：A Maximum Entropy Approach：[PhD thesis]. Carnegie Mellon University，1994. CMU Technical Report CMU-CS-94-138
8 Darroch J，Ratcliff D. Generalized iterative scaling for log-linear models. The annals of Mathematical statistics 1972，43：1470～1480
9 Berger A L，Della Pietra S A，Della Pietra V J. A maximum entropy approach to natural language processing. Computational Linguistics 1996，22(1)：39～71
10 Rosenfeld R. Two decades of Statistical Language Modeling：Where Do We Go From Here? Proceedings of the IEEE，2000，88(8)

（上接第13页）

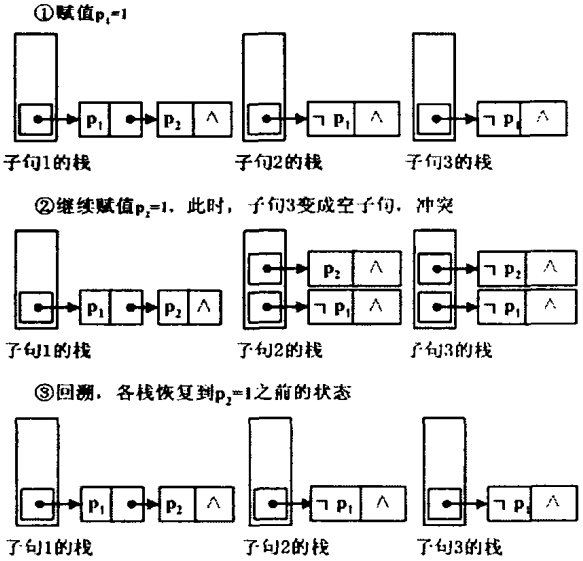

图4 子句栈的变化

总结 实验结果表明，我们开发的 QuickSAT 接近了国际先进水平。虽然 QuickSAT 还不是世界上最高效的，但我们相信这主要是因为我们有意地放弃了许多先进技术的使用，而这些技术已经被证明了能有效提高 SAT 算法的运行性能。我们的本意只是要通过这个程序的开发，来例证算法工程的原则和方法的有效性。这一点，我们相信已经做到了。

当然，算法工程的理论与方法的发展，还需要研究者们不断地努力，要提出更多行之有效的方法，要对各种方法进行进一步的论证，还要朝着系统化、形式化的方向发展。我们在这里的讨论，只是起到“抛砖引玉”的作用罢了。

参 考 文 献

1 Aho A V，Johnson D S，Karp R M，et al. Theory of Computation：Goals and Directions. STOC 1996，March 15，1996
2 Cohen P，Gent I P，Walsh T. Empirical Methods for CS and AI. given at AAAI，ECAI and Tableaux Conf. in 2000
3 Davis M，Putnam H. A Computing Procedure for Quantification Theory. Journal of the ACM，1960，7(3)：201～215
4 Davis M，Logemann G，Loveland D. A Machine Program for Theorem Proving. Communications of the ACM，1962，5(7)：394～397
5 Marques-Silva J P，Sakallah K A. GRASP：A Search Algorithm for Propositional Satisfiability. IEEE Transactions on Computers，1999，48(5)：506～521
6 Moskewicz M W，Madigan C F，Zhao Y，et al. Chaff：Engineering an Efficient SAT Solver. In：Proc of the 38th Design Automation Conference (DAC'01)，June 2001，530～535

附表：

基准测试集	测试实例组	组中实例数目	GRASP[1]		zChaff[2]		QuickSAT	
			超时次数	总运行时间	超时次数	总运行时间	超时次数	总运行时间
DIMAC 基准测试集[3]	ii8	14	0	1.4	0	0.1	0	0.1
	ii16	10	1	326.8[6]	0	7.6	0	12
	ii32	17	0	2.7[6]	0	0.6	0	0.6
	aim100	24	0	0.7[6]	0	0.1	0	0.1
	aim200	24	0	7.9	0	0.3	0	0.3
	pret	8	0	7.2[6]	0	0.8[5]	0	0.6
	par8	10	0	0.1[6]	0	0.1	0	0.1
	par16	10	7	1195.7[6]	0	54.9	0	91.8
	ssa	8	0	3.2[6]	0	0.3	0	1.1
	jnh	50	0	6.9[6]	0	0.7	0	0.7
	dubois	13	0	0.3	0	0.2	0	0.3
	hole	5	2	299.9[6]	0	128.7	0	134.4
CMU 基准测试集[4]	SSS 1.0	48	5	1084	0	62	0	85
	SSS 1.0a	8	6	9190	0	25	0	37
	SSS-SAT 1.0	100	100	-	0	632	7	8931
	PVP-UNSAT 1.0	4	2	2944	0	1033	0	1477
	VLIW-SAT 1.0	100	100	-	0	4665	4	9617

注：(1)GRASP 使用的是2000年2月的版本，其源程序和 Linux 二进制版本可以从 http://sat.inesc.pt/～jpms/grasp/下载；(2)zChaff 使用的是2001年的版本，源程序或 Linux 二进制代码可以从 http://ee.princeton.edu/～chaff/zchaff.php 下载；(3)超时时限为100s；(4)超时时限为1000s；(5)对于本组测试实例，将 zChaff 配置中的 maxLitsInClaseForConfDriven 由默认的10000改成10，除此之外，其他情况 zChaff 都是用 cherry.smj 中的默认配置；(6) GRASP 运行使用如下开关参数：+ T100 +B10000000 +C10000000 +S10000 +g20 +rt4 +dDLIS，其他情况下 GRASP 都使用以下开关参数：+T100 +B10000000 +C10000000 +S10000 +V0 +g40 +rt4 +dMSMH +dr5。