文章编号:1671-4229(2023)05-0041-11

可满足性问题的精确算法和计算复杂性

陈建二,杨 伟
(广州大学 计算机科学与网络工程学院,广东 广州 510006)

摘 要:可满足性(SAT)问题是计算机科学中最重要的理论研究和实际应用问题之一。文章从标准计算复杂性理论的角度论述SAT问题的精确算法和计算复杂性,主要论述算法的发展,分析算法(最坏情况)的复杂度,并探讨SAT问题的复杂度上限。对一些具有意义的算法结果进行了解释和分析,并讨论了算法的重要性,同时还介绍了近几年的相关研究进展。

关键词:可满足性;SAT算法;NP完全性;精确算法;计算复杂性理论
中图分类号:TP301 **文献标志码**:A

Satisfiability problem: Exact algorithms and complexity

CHEN Jian-er, YANG Wei
(School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China)

Abstract: Satisfiability (SAT) problem is among the most important problems in computer science, for both theoretical research and practical applications. In this survey, we study exact algorithms and complexity of the SAT problem from the standard complexity theoretical perspective. We will focus on the research that develops algorithms, analyzes algorithm (worst-case) complexity, and studies the complexity bounds for the SAT problem. We will review fundamental algorithmic techniques and some of the most famous milestone algorithms for the SAT problem, give intuitive explanations and detailed formal analysis for these techniques and algorithms, and discuss their significance. We will also describe the most recent advances in the past couple of years, and discuss further research directions in the development of SAT algorithms and the study of the complexity issues of the SAT problem.

Key words: satisfiability; SAT algorithms; NP-completeness; exact algorithm; complexity theory

可满足性问题是计算机科学中最重要且十分著名的问题之一。在理论计算机科学的研究中,可满足性问题是第一个NP(Non-deterministic Polynomial, NP)完全问题[1],在计算困难问题的研究中起着重要作用[2]。实践中可满足性问题是解决很多应用领域问题的基础,包括人工智能[3]、模型检查[4]、自动推理[5]、计算机辅助设计[6]、数据库[7]、机器人学[8]、调度[9]、电路设计[10]、计算机体系结构设计[11]和计算机网络[12]等。

布尔变量是只能采用布尔值1(TRUE)或0(FALSE)的变量。布尔函数是布尔变量上输出布尔值的函数,仅使用逻辑运算∧(AND)、∨(OR)和¬(NOT)。对布尔函数F赋值是给F中的每个变量赋一个确定的0或1的布尔值,如果F在赋值集合π下输出1,则π满足F,称布尔函数F是可满足的;否则F是不可满足的。

文字是布尔变量x或x的否定$\bar{x}$,子句C是文字的

收稿日期:2022-03-30;**修回日期**:2023-02-21
基金项目:国家自然科学基金资助项目(61872097)
作者简介:陈建二(1954—),男,教授,博士生导师. E-mail:Jianer@gzhu.edu.cn
引文格式:陈建二,杨 伟. 可满足性部题的精确算法和计算复杂性[J]. 广州大学学报(自然科学版),2023,22(5):41-51.

析取,$C = x_1 \vee x_2 \vee \cdots \vee x_k$,其中,$x_k$ 表示文字。当增加新的文字 x 到子句 C 上时,可以得到一个新的子句 $C' = x \vee x_1 \vee \cdots \vee x_k$,也可以直接写成 $C' = x \vee C$。CNF 公式 F 是一个合取范式的布尔函数,$F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$,其中,$C_m$ 表示子句,同样也可以在 CNF 公式中添加或删除一个子句来得到新的 CNF 公式。每个布尔函数都可以写成合取范式[2]。可满足性问题定义如下:

可满足性(SAT):给定 CNF 公式 F,确定 F 是否可满足。

SAT 是 NP 完全问题,除非有 P = NP,否则 SAT 问题在多项式时间内不能有效求解,P = NP 在计算机科学和数学领域共识中是不成立的。

本文从标准计算复杂性理论的角度论述了 SAT 问题的精确算法和计算复杂性。精确算法指的是对每个输入实例输出正确解的算法,并且根据输入实例的参数对其(最坏情况)运行时间有一个已证明的上界。这与启发式算法是一致的,启发式算法的正确性可能不会被正式验证而且其运行时间上限也可能不会被指定。文章重点论述了 SAT 问题的算法发展,算法(最坏情况)复杂性分析和复杂度上界;对一些著名的有意义的算法结果进行解释和分析,并讨论它们的意义;介绍了最近几年相关领域的研究进展;最后简要分析了已经在许多领域取得成功应用的启发式算法在 SAT 问题中的应用情况。

1 基础概念和相关技术

设 CNF 公式 $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ 为 SAT 问题的一个实例,C_m 表示子句。以下情况可以进行预处理:

(1)如果一个子句 C_i 包含空文字,那么 C_i 不可能被满足。对于这样的实例 F,直接输出 NO;

(2)如果一个子句 C_i 只包含一个文字 x(单位子句),为了满足 F,必须令 $x = 1$ 来满足 C_i;

(3)如果公式 F 包含文字 x 且不包含 $\bar{x}$,那么可以令 $x = 1$ 来满足 F 中包含 x 的子句,而且不影响 F 中其他子句的满足性;

(4)如果一个子句 C_i 同时包含一个变量 x 和它的否定 $\bar{x}$,那么 C_i 总是可满足的。可以忽略子句 C_i(不需要确定 x 的值)。

编写一个简单的算法可以处理上述情况。预处理完成之后,CNF 公式就是明确的。下面的研究放在明确的 SAT 问题实例上。

SAT 算法的发展有两种基本技术:消解和分支搜索。这两种技术已经被证明在 SAT 算法分析、计算复杂性以及在 SAT 问题的算法开发中有着重要作用。

1.1 消解

消解技术是由 Davis 等[13]提出的,设 $C_1 = x \vee C_1'$、$C_2 = \bar{x} \vee C_2'$ 为两个子句,分别包含一个变量 x 和它的否定 $\bar{x}$,其中,C_1' 和 C_2' 是子句。变量 x 上 C_1 和 C_2 的消解子句为 $C_1' \vee C_2'$。其中,子句 C_1' 和 C_2' 可能有共同的文字,那么就需要删除 $C_1' \vee C_2'$ 中的重复文字。F 为 CNF 公式,F 在变量 x 上的消解也是 CNF 公式,该公式通过删除 F 中包含 x 和 $\bar{x}$ 的所有子句,并将 F 中所有子句的消解子句添加到变量 x 上获得。具体如下:

$$F = F_0 \wedge \bigwedge_{1 \leqslant i \leqslant s} (x \vee C_i) \wedge \bigwedge_{1 \leqslant j \leqslant t} (\bar{x} \vee C_j'), \quad (1)$$

其中,F_0 是子句既不包含 x 也不包含 $\bar{x}$ 的 CNF 公式。那么 F 在变量 x 上的消解式为

$$\mathrm{DP}_x(F) = F_0 \wedge \bigwedge_{1 \leqslant i \leqslant s, 1 \leqslant j \leqslant t} (C_i \vee C_j'). \quad (2)$$

下面给出定理,并对完整性进行证明。

定理 1(消解原理) 设 x 是一个确定的 CNF 公式 F 中的任意变量,F 是可满足的当且仅当 $\mathrm{DP}_x(F)$ 是可满足的。

证明 设 F 是变量集 $X = \{x, x_2, \cdots, x_n\}$ 上的 CNF 公式,如(1)所示,在变量 x 上的消解 $\mathrm{DP}_x(F)$ 如(2)所示。

假设 F 是可满足的。π 是 F 的一个可满足赋值集合,$\pi(x) = x^*$,$\pi(x_k) = x_k^*$ $(2 \leqslant k \leqslant n)$,其中,$x^*$ 和 x_k^* 是布尔常数。设 π' 为 CNF 公式 $\mathrm{DP}_x(F)$ 中变量 $\{x_2, \cdots, x_n\}$ 的赋值集合,$\pi'(x_k) = x_k^*$ $(2 \leqslant k \leqslant n)$。如果 $\pi(x) = x^* = 0$,那么由于 $\pi(F) = 1$,必须有 $\pi(x \vee C_i) = 1$,因此 $\pi'(C_i) = 1$ $(1 \leqslant i \leqslant s)$($F$ 是明确的,C_i 既不包含 x 也不包含 $\bar{x}$);如果 $\pi(x) = x^* = 1$,那么 $\pi(\bar{x}) = 0$,必须有 $\pi'(C_j') = 1$ $(1 \leqslant j \leqslant t)$。在这两种情况下,总有 $\pi'(C_i \vee C_j') = 1$ $(1 \leqslant i \leqslant s$ 和 $1 \leqslant j \leqslant t)$。此外,由于 F_0 既不包含 x 也不包含 $\bar{x}$,有 $\pi'(F_0) = \pi(F_0) = 1$。因此,$\pi'(\mathrm{DP}_x(F)) = 1$。$\mathrm{DP}_x(F)$ 是可满足的。

另一方面,假设 $\mathrm{DP}_x(F)$ 是可满足的,π' 是可满足 $\mathrm{DP}_x(F)$ 的变量 $\{x_2, \cdots, x_n\}$ 上的赋值集合。如果 $\pi'(C_i) = 1$ $(1 \leqslant i \leqslant s)$,那么除了 π' 之外,指定 $x = 0$,则 $\bar{x} = 1$,赋值满足 F 中的所有子句;如果对于某些 i_0,有 $\pi'(C_{i_0}) = 0$,由于 $\pi'(\mathrm{DP}_x(F)) = 1$,可得 $\pi'(C_{i_0} \vee C_j') = 1$ $(1 \leqslant j \leqslant t)$,即 $\pi'(C_j') = 1$ $(1 \leqslant j \leqslant t)$。这种情况下,在 π' 之外指

定 $x=1$,满足公式 F。因此,只要 $DP_x(F)$ 是可满足的,公式 F 就是可满足的。证明完成。

根据消解原理,要判定 SAT 的一个实例 F 是否可满足,只需判定 F 中任意变量 x 上 F 的消解 $DP_x(F)$ 是否可满足。消解原理消去了 SAT 实例 F 中变量本身和它的否定都出现的变量。不好之处在于 $DP_x(F)$ 是通过从 F 中删除 $s+t$ 个子句,然后向 F 中添加 $s\times t$(最大)个子句来获得。当 s 和 t 较大时,这会明显增加实例的大小,因此,消解原理不能保证总是有效地解决 SAT 问题,但是在子句数量较少的情况下,消解原理是非常有用的,它减少了 CNF 公式中的变量数量,不会显著增加实例大小。目前,消解原理仍然是大多数启发式算法的基础,这些算法可以有效地解决一组非常大的 SAT 实例。

1.2　分支搜索

算法 1 是基于分支和搜索技术的 SAT 算法,$F[x=0]$(或 $F[x=1]$)为变量 x 赋值 0(或 1)。$F[x=1]$是通过删除包含 x 的所有子句(由于 $x=1$ 而满足),并删除文字 $\bar{x}$(因为通过赋值 $x=1,\bar{x}=0$),获得的 CNF 公式。

算法 1:SAT-Alg (F)

INPUT:一个 SAT 实例 F

OUTPUT:判断 F 是否满足

1. IF ($n(F)\leqslant 1$), THEN 求解 F;RETURN (YES/NO);
2. 选取 F 中任意变量 x;
3. $F_0=F[x=0]$,$F_1=F[x=1]$;
4. IF(SAT-Alg(F_0)可满足),THEN RETURN (YES);
   ELSE IF (SAT-Alg (F_1) 可满足), THEN RETURN (YES);
   ELSE RETURN (NO)。

SAT-Alg(F)显然能够求解 SAT 问题:如果 F 是可满足的,那么对 F 的一个满意赋值必须给变量 x 赋值,这个值可以是 0 或 1。因此,$F_0=F[x=0]$和 $F_1=F[x=1]$中的至少一个是可满足的,SAT-Alg(F)步骤 4 输出 YES。反之,如果 F 不可满足,那么 $F[x=0]$和 $F[x=1]$都不可满足,则 SAT-Alg(F)第 4 步输出 NO。

将 n 个变量上输入实例 F 的 SAT-Alg(F)时间复杂度记为 $T(n)$。F_0 和 F_1 最多都有 $n-1$ 个变量,有以下递推关系,其中,$L(F)$是输入实例 F 的长度:

$$\text{IF } n\leqslant 1:\ T(n)=O(L(F)),$$
$$\text{IF } n>1:\ T(n)\leqslant 2T(n-1)+O(L(F))。\quad (3)$$

容易得到 $T(n)=O(2^nL(F))$,有如下定理:

定理 2　SAT-Alg 在 $O(2^nL)$时间内解决有 n 个变量、长度为 L 的 SAT 实例 F。

许多参数可以用来衡量 CNF 公式的“大小”,也可以衡量 SAT 算法的时间复杂度。子句 C_i 的大小$|C_i|$是 C_i 中的文字数。CNF 公式 $F=C_1\wedge C_2\wedge\cdots\wedge C_m$,用 $n(F)$表示 F 中变量的个数,$m(F)$表示 F 中子句个数,$L(F)=\sum_{i=1}^{m}|C_i|$ 表示 F 的长度。一般情况下用 n、m 和 L 来表示这些参数。在一个明确的 CNF 实例中,有如下关系:n, $m\leqslant L$, $L\leqslant nm$, $m\leqslant 2^n$。SAT 是 NP 完全的,不期望 SAT 算法在这些参数多项式时间内运行。无论使用哪种参数来分析时间复杂性,SAT 算法在最坏情况下的运行时间都是该参数的时间上界。

一种典型的基于分支搜索技术的 SAT 算法$\mathcal{A}$的工作原理如下:在 CNF 公式 F 上,如果 F 足够简单,则算法$\mathcal{A}$直接求解 F。否则算法$\mathcal{A}$生成 CNF 公式的集合$\{F_1,\cdots,F_c\}$,其中,$c>1$,使得 F 是可满足的当且仅当$\{F_1,\cdots,F_c\}$中至少一个公式是可满足的。然后算法$\mathcal{A}$递归地处理公式 $F_1,\cdots,F_c$。F 上的分支和搜索算法$\mathcal{A}$可以描述为一个搜索树 $T_{\mathcal{A}}$,其中,根用实例 F 标记,搜索树 $T_{\mathcal{A}}$ 中用 CNF 公式 F'标记的节点 $v_{F'}$,如果用算法$\mathcal{A}$直接求解 F',则节点 $v_{F'}$是 $T_{\mathcal{A}}$ 的叶子,对于公式 F',算法$\mathcal{A}$生成一组 CNF 公式集合$\{F'_1,\cdots,F'_c\}$,并递归地处理这些公式,节点 $v_{F'}$在搜索树 $T_{\mathcal{A}}$ 中有 c 个孩子,分别标记为 $F'_1,\cdots,F'_c$。算法$\mathcal{A}$的执行对应从根开始的搜索树 $T_{\mathcal{A}}$ 的深度搜索遍历。

使用一个参数 q(n、m 或 L)来度量算法$\mathcal{A}$的时间复杂度。将参数值 q 的实例 F 分支到 c 个实例 $F_1,\cdots,F_c$,实例 $F_1,\cdots,F_c$ 的参数值 $q-d_1,\cdots,q-d_c$ 分别减小 d_i,$d_i>0$。将#(q)记为参数值 q 实例上算法$\mathcal{A}$的搜索树 $T_{\mathcal{A}}$ 中的叶子数,那么可以得出如下递推关系(算法$\mathcal{A}$在 $q\leqslant 1$ 时不分支):

$$\text{IF } q\leqslant 1:\ \#(q)=1,$$
$$\text{IF } q>1:\ \#(q)\leqslant\sum_{i=1}^{c}\#(q-d_i)。\quad (4)$$

为了求解递推关系(4),用 $p\mathcal{A}(x)=x^q-\sum_{i=1}^{c}x^{q-d_i}$ 定义了(4)的分支多项式。设 x_0 为多项式 $p\mathcal{A}(x)$的最大根,可以证明 $x_0\geqslant 1$(文献[14])。从关系式(4)中通过对 q 的归纳,容易验证$\#(q)\leqslant x_0{}^q$。因此,基于参数值 q 的 CNF 公式的算法$\mathcal{A}$的搜索树 $T_{\mathcal{A}}$ 中节点数边界为 $2x_0^q$。当 $q\leqslant 1$ 时,如果能在多项式时间内求解 SAT 实例,而且在多项式时间内由实例 F 生成实例 $F_1,\cdots,F_c$,则算法$\mathcal{A}$在参数 q 实例上的运行时间为 $O(x_0^qL^{O(1)})$。

讨论 SAT 算法时间复杂度的多项式上界因子,对所

选参数 q 使用符号 $O^*(f(q))$ 来表示 $O(L^{O(1)}f(q))$,即 $O^*(f(q))$ 具有隐藏在 O^* - 符号中 L 的多项式因子。如前所述,测量分支的时间复杂度并通过参数 q 进行搜索,得到算法分支多项式的最大根等于 x_0,则算法运行时间为 $O^*(x_0^q)$。

根据给定的实例结构,上述讨论可以很容易地推广到使用多个分支规则的 SAT 算法,定理如下:

定理3 让$\mathcal{A}$是基于参数 q 分支和搜索技术的 SAT 算法。如果算法$\mathcal{A}$有多个分支规则,每个分支规则都有一个相应的分支多项式,那么算法 $\mathcal{A}$的运行时间是 $O^*(x_{\max}^q)$,其中,$x_{\max}$是所有这些分支多项式的最大根。

2 基于参数 n 的算法

如定理2 所示,SAT-Alg 算法在时间 $O^*(2^n)$ 内解决 SAT 问题。通过检查给定 CNF 公式的所有 2^n 赋值,SAT 问题可以在时间 $O^*(2^n)$ 内得到解决。那么是否能比这个基本的算法做得更好? 这是理论计算机科学中长期存在的开放性问题。文献[15]中提出了强指数时间猜想(Strong Exponential Time Hypothesis, SETH)。

SAT 问题不能在 $O^*(c^n)$ ($c<2, c$ 为任意常数)时间内求解。

在最近的研究中,SETH 被广泛用于著名的计算优化问题开发计算下限(如文献[16]的 14.5 节)。在 SAT 算法和复杂性研究中,任何一种算法如优于基本 SAT 算法都是一个重大的进步,也将是许多其他优化问题算法研究的一个重要进展。

如果使用参数 n 来度量算法的复杂度,那么 SETH 为更优的算法开发留下了很少的空间。SAT 算法的研究热点已经发展到具有各种限制条件的算法研究上。

k-可满足性(k-SAT):给定 CNF 公式 F,其中,子句大小以 k 为界,确定 F 是否可满足。

当 $k=2,3$ 时,k-SAT 问题特别有趣。2-SAT 问题在多项式时间内是可解的,属于 P 类[17]。3-SAT 问题是一个非常著名的 NP 完全问题,它在 NP 完全性理论、SAT 算法和计算复杂性的研究中起着关键的作用。接下来重点讨论 3-SAT 问题的算法,讨论如何将这些算法推广到一般 k-SAT 问题。

为了研究运行速度快于 $O^*(2^n)$ 的 3-SAT 算法,笔者根据 Monien 等[18]的文章从简单分支和搜索算法(算法2)开始,在不丧失一般性的情况下,输入实例 F 是明确的—可以通过多项式时间的预处理使 F 明确。

算法 2:$3SAT_{ms}(F)$

INPUT: 3-SAT 的一个明确实例 F

OUTPUT: 判断 F 是否可满足

1. IF (F 没有大小为 3 的子句),直接求解 F;返回(YES/NO);
2. 选择大小为 3 的子句($l_1 \vee l_2 \vee l_3$);
3. $F_1 = F[l_1 = 1]$;$F_{01} = F[l_1 = 0, l_2 = 1]$;$F_{001} = F[l_1 = 0, l_2 = 0, l_3 = 1]$
4. IF ($3SAT_{ms}(F_1)$可满足),RETURN (YES);
   ELSE IF ($3SAT_{ms}(F_{01})$可满足),RETURN (YES);
   ELSE IF ($3SAT_{ms}(F_{001})$可满足),RETURN (YES);
   ELSE RETURN (NO)。

首先,如果 F 没有大小为 3 的子句,如步骤 1,那么 F 是一个 2-SAT 实例,其可满足性可以在多项式时间内确定[17]。

$3SAT_{ms}$算法中为了满足公式 F,必须满足子句($l_1 \vee l_2 \vee l_3$),满足 F 的赋值必须将值 1 赋值给 3 个文字 l_1、l_2 和 l_3 中的至少一个。总共有 7 组对文字 l_1、l_2 和 l_3 的赋值可以满足子句($l_1 \vee l_2 \vee l_3$)(仅排除赋值 $l_1 = l_2 = l_3 = 0$),F_1 中的赋值 $l_1 = 1$ 涵盖了这 7 组赋值中的 4 个($l_1 = 1$,l_2 和 l_3 是 0 或 1),函数 F_{01} 中的赋值 $l_1 = 0, l_2 = 1$ 涵盖了 7 组赋值中的两个(l_3 是 0 或 1),该算法步骤 3 中的 3 个赋值覆盖了对文字 l_1、l_2、l_3 的所有可能赋值,它们可以对输入实例 F 进行可满足赋值。这证明了该算法的正确性,即当且仅当 F_1、F_{01}、F_{001} 中至少有一个可满足,输入实例 F 才是可满足的。

为了分析算法 $3SAT_{ms}$ 的复杂度,设输入实例 F 是明确的,3 个文字 l_1、l_2 和 l_3 来自 3 个不同的变量。如果 F 有 n 个变量,那么函数 F_1、F_{01} 和 F_{001} 分别最多有 $n-1$、$n-2$ 和 $n-3$ 个变量。因此,如果将 $\#(n)$ 设为 $3SAT_{ms}$ 在输入 n 个变量情况下搜索树中的叶数,那么就有如下递推关系(F 是明确的,如果 $n \leqslant 2$,那么 F 就不能有大小为 3 的子句):

$$\text{IF } n \leqslant 2: \#(n) = 1,$$

$$\text{IF } n > 2: \#(n) \leqslant \#(n-1) + \#(n-2) + \#(n-3)。(5)$$

递推关系(5)的分支多项式为 $p(x) = x^n - x^{n-1} - x^{n-2} - x^{n-3}$,其最大根为 1.83…,通过上述讨论,得到如下定理:

定理4 算法 $3SAT_{ms}$ 在时间 $O^*(1.84^n)$ 内求解 3-SAT 问题。

算法 $3SAT_{ms}$ 可以进一步改进,并且推广到求解一般 k 的 k-SAT 问题。对于整数 $k(k \geqslant 3)$,基于此方法的算法在时间 $O^*(c_k^n)$ 内求解 k-SAT 问题,其中,c_k 是方程 c_k

$=2-1/c_k^{k-1}$ 的最大根($c_k<2$),当 k 增加时,c_k 接近 2。例如,$c_3=1.619$,$c_4=1.839$,$c_5=1.928$,$c_6=1.966$[18]。

对 SAT-Alg 和 $3SAT_{ms}$ 算法的讨论表明,基于分支和搜索的 SAT 算法的效率取决于如何为给定 SAT 实例 F 生成集合 $C=\{F_1,\cdots,F_c\}$,使得 F 是可满足的当且仅当 C 中至少有一个实例是可满足的。一般来说,集合 C 中的实例越少,其参数值 n 的减少越多,算法的效率就越高。这一结论导致了一系列改进运行时间的 3-SAT 算法,$O^*(1.579^n)$、$O^*(1.505^n)$、$O^*(1.497^n)$ 和 $O^*(1.476^n)$(文献[19])。这种方法产生的算法需要检查 SAT 实例详细的结构,并且需要繁琐复杂的逐步分析。

相比之下,Schöning[19] 提出了一种随机 3-SAT 算法,该算法从随机选择的初始赋值开始,对所选赋值应用局部搜索。Schöning 的算法如算法 3 所示,其中,b 是一个数字可以用来控制算法正确的概率,"翻转文字 l_i 的值"是将文字 l_i 的变量值从 0 改为 1,反之亦然。

算法 3:$3SAT_{uwe}(F)$

INPUT:一个明确的 3-SAT 实例 F
OUTPUT:判断 F 是否可满足
1　重复 $b\cdot(4/3)^n$ 次
1.1　随机选取一个赋值集合 π 到 F 上;
1.2　重复 $3n$ 次
1.2.1　IF(赋值 π 满足 F),THEN RETURN (YES);
1.2.2　在 F 中选取 π 不满足的子句 $C=(l_1\vee l_2\vee l_3)$;
1.2.3　在 C 中随机选取一个文字 l_i 并翻转赋值 π 中 l_i 的值
2　RETURN (NO)。

Schöning 算法只有在第 1.2.1 步得到可满足赋值时才输出 YES。如果输入实例 F 是不可满足的,它将永远不会输出 YES。设 F 是有 n 个变量的可满足 CNF 公式,π^* 是满足 F 的赋值集合。由于步骤 1.1 随机选取了一个赋值集合 π 给 F,可以期望 π 与 π^* 在 $n/2$ 个变量上一致。因此,对大约 $n/2$ 个变量的赋值 π 进行"修正",以获得满意的赋值集合 π^*。为了找出 π 中"不正确"的变量,步骤 1.2.2 选取 π 不满足的子句 $C=(l_1\vee l_2\vee l_3)$。如果 F 是可满足的而 π 不满足 C,那么 C 中至少有一个文字在赋值 π 下得到一个错误的值,因此应该"翻转"。为了决定要翻转 C 中的哪个文字,步骤 1.2.3 随机选取 C 中的一个文字。该算法的步骤 1.2.2 ~ 1.2.3 对赋值进行(随机)局部搜索,可视为从随机选择的赋值 π 开始的随机进行以达到满意的赋值集合 π^*。与以前的算法相比,Schöning 算法的一个不同之处是算法最多运行 $3n$ 步(参见步骤 1.2)。在 $3n$ 步中,随机进行没有达到目标 π^*,那么通过步骤 1.1 用一个新的随机选择的分配重新开始局部搜索。

Schöning 证明了在输入 F 是可满足的情况下,步骤 1.1 ~ 1.2 将在步骤 1.2.1 得到一个满意的赋值,并输出 YES,概率至少为 $(3/4)^n$。步骤 1.1 ~ 1.2 在算法中重复 $b\cdot(4/3)^n$ 次,实例 F 在步骤 2 中输出 NO(即在步骤 1.2.1 中不输出 YES)的概率有界于

$$(1-(3/4)^n)^{b(4/3)^n}\leqslant\frac{1}{e^b}。$$

式中,$e=2.718\cdots$ 是自然对数的常数,不等式 $(1-1/p)^p\leqslant 1/e$。由于 Schöning 算法在不可满足的输入上从不输出 YES,因此得出结论,Schöning 算法在每个输入上输出正确答案的概率至少为 $1-1/e^b$,当 b 足够大时,其值可以无限接近 1。就时间复杂性而言,步骤 1.1 ~ 1.2 显然需要时间 $O(L)$。只要 b 以 n 的多项式为界,Schöning 算法的运行时间为 $O^*(b\cdot(4/3)^n)=O^*(1.334^n)$。

定理 5　3-SAT 问题在时间 $O^*(1.334^n)$ 内可用随机算法求解。

Schöning 算法可推广到一般的 k-SAT 问题,k-SAT($k\geqslant 3$)问题给出了运行时间为 $O^*((2-2/k)^n)$ 的随机算法。

Dantsin 等[20] 为 k-SAT 问题开发了确定性算法,可以看作是 Schöning 算法的去随机化。本文使用覆盖码代替随机初始赋值。3-SAT 确定性算法运行时间在 $O^*(1.481^n)$ 之内,当 $k\geqslant 4$ 时,确定性算法运行时间在 $O^*((2-2/(k+1))^n)$ 之内。

3-SAT 和 k-SAT 随机算法是近年来非常热门和活跃的研究课题。可以参考文献[21 - 24]及相关文献了解最新的研究状况以及内容更新。3-SAT 算法的运行时间上界 $O^*(c^n)$ 已经取得了新的研究进展,从 Schöning 算法的 $c=1.334$,逐步改进为 $c=1.324$,1.323,1.322,1.321,…,1.308,1.307。目前,最佳上界 $c=1.307$ 是由 Hansen 等[24] 取得的。$k>3$ 的 k-SAT 也有进展,研究了这些随机算法的去随机化。

3-SAT 算法运行时间的指数上界 $O^*(c^n)$ 在不断减小。不能期望 $c=1$,因为这意味着 NP 完全问题 3-SAT 在多项式时间内可解。那么 3-SAT 指数时间算法的 c 是否可以无限接近 1? 这与 $P\neq NP$ 并不矛盾。事实上有一些 NP 完全问题可以在时间 $O^*(c^n)$ 内求解,其中,常数 $c>1$ 无限接近 1[16]。从实际计算的角度来看这个问题也有意义。例如如果 3-SAT 问题能在时间 $O^*(1.000\,1^n)$

内得到解决,该算法可用于多达 100 000 个变量的3-SAT实例,这可能满足许多工程应用的需要。在精确算法的研究以及整个理论计算机科学中,指数时间猜想(Exponential Time Hypothesis, ETH)排除了这种可能性[25]。

3-SAT不能在时间$O^*(c_0^n)$内求解,其中,常数$c_0>1$。

不确定猜想ETH中常数c_0的值接近目前最好的上限1.307,还是可以更小?如1.000 1?这方面的研究具有意义。

如果问题在时间$O^*(2^{n/a(n)})$内求解,则可以在次指数时间内求解,其中,$a(n)$是n的一个递增无界函数。能够证明一个问题在次指数时间内求解[26],当且仅当它可以在时间$O^*(c^n)$ $(c>1)$内求解。在指数时间算法的文献中,ETH通常被描述为"3-SAT不能在次指数时间内求解"。

根据SETH猜想,对于任意常数$c<2$,SAT问题不能在时间$O^*(c^n)$内求解。可以看出,SETH包含ETH[16]。SETH的猜想比ETH的猜想强(ETH被认为更可信)。ETH是近年来证明计算复杂性下限的一个非常流行的猜想,同时,SETH在某种程度上被许多研究者视为一个可疑的猜想。

3 基于参数m的算法

在本节中用输入CNF公式F中子句数量m来衡量SAT算法时间复杂度。m可以大到2^n,也可以比n小得多,那么对有n个变量的3-SAT实例的CNF公式F,F中子句数量m的边界为$8\cdot\binom{n}{3}=O(n^3)$。由于3-SAT是NP完全的,除非有P=NP,否则不期望3-SAT算法(一般SAT问题的算法)的运行时间是m的多项式。

对于常数$c<2$的一般SAT问题,用实例中的变量个数n来度量SAT算法的复杂度,与SETH猜想相比得到一个$O^*(c^m)$时间算法并不困难。

设F是一个CNF公式,l是F中的一个文字。如果文字l在F中正好出现a次且其否定$\bar{l}$在F中正好出现b次,则称l为(a,b)-文字;如果文字l在F中至少出现a^+次且其否定$\bar{l}$在F中正好出现b次,则称l为(a^+,b)-文字。同样可以定义(a,b^+)-文字和(a^+,b^+)-文字。由于通过多项式时间的预处理,总可以使公式F明确,在下面的讨论中,公式F中的每个文字都是$(1^+,1^+)$-文字。对于(a,b)-文字l,CNF公式F可以写为

$$F=F_0\wedge\bigwedge_{1\leqslant i\leqslant a}(l\vee C_i)\ \wedge\bigwedge_{1\leqslant j\leqslant b}(\bar{l}\vee C'_j),$$

其中,F_0是CNF公式,C_i和C'_j是既不包含l也不包含$\bar{l}$的子句。F在文字l上的消解为

$$\mathrm{DP}_l(F)=F_0\wedge\bigwedge_{1\leqslant i\leqslant a,1\leqslant j\leqslant b}(C_i\vee C'_j)。$$

如果公式F有m个子句,那么消解式$\mathrm{DP}_l(F)$最多可以有$m+ab-(a+b)$个子句,($\mathrm{DP}_l(F)$中的一个子句$C_i\vee C'_j$可能包含变量x和$\bar{x}$,那么可以删除这种子句)。当$ab\leqslant a+b$时,对(a,b)-文字l的消解操作不会增加子句的数目。因此,可以直接对公式F应用消解,为了确保运算是安全的,还需要CNF公式的长度是可控的。由n个变量和m个子句组成的CNF公式,由于F中每个子句的长度不能大于n,公式的长度L以nm为界。由于消解运算减少了变量的数量,如果它不增加子句的数量,那么所得公式长度边界为$(n-1)m\leqslant nm$。如果对有n个变量、m个子句、长度为L的CNF公式F中$ab\leqslant a+b$的(a,b)-文字应用消解运算,那么在这个过程中得到所有公式的长度边界为$nm\leqslant L^2$。这确保了消解运算可以在公式F长度内多项式时间完成。

上述分析表明,当公式中存在$ab\leqslant a+b$的(a,b)-文字时,对文字l进行消解运算减少了变量的个数,最多可以进行n次。经过多项式时间的预处理,得到了一个CNF公式F,其中,每个文字都是$ab>a+b$的(a,b)-文字,这意味着$a,b\geqslant 2$且$a+b\geqslant 5$。对于任何这样的(a,b)-文字l,用$F[l=1]$和$F[l=0]$进行分支。$a,b\geqslant 2$,$a+b\geqslant 5$,$F[l=1]$和$F[l=0]$都必须将子句的数目m至少减少2,并且其中至少有一个将子句的数目m至少减少3。如果将$\#(m)$设为该算法在m个子句的CNF公式上的搜索树中的叶数,有如下递推关系(由于$a+b\geqslant 5$,有$m\geqslant 5$):

$$\begin{aligned}&\text{IF } m\leqslant 4:\#(m)=1,\\&\text{IF } m\geqslant 5:\#(m)\leqslant\#(m-2)+\#(m-3)。\end{aligned}\quad(6)$$

求解这个递推关系$\#(m)\leqslant 1.325^m$,得到下面的定理:

定理6 SAT问题在时间$O^*(1.325^m)$内能求解,其中,m是输入公式中的子句个数。

定理6给出的SAT算法时间复杂度的上界$O^*(1.325^m)$是通过$(2,3)$-文字的分支和搜索过程。如果想降低上界,需要更有效地处理$(2,3)$-文字。对一个SAT算法,它的时间复杂度不比递推关系$\#(m)\leqslant 2\#(m-3)$给出的算法差,$\#(m)\leqslant 1.260^m$。因此,在$(3^+,3^+)$-文字上进行分支。对$ab\leqslant a+b$的(a,b)-文字,可以在多项式时间内使用无分支的消解进行处理。为

了达到定理6 中的上界并实现一个 $O^*(1.260^m)$时间的SAT 算法,只需要更有效地处理(2,3) - 文字和(2,4) - 文字,不是简单地在这些文字上进行分支。有一种通用技术可以更有效地处理所有 $b\geqslant 3$ 的$(2, b)$ - 文字。将此技术介绍如下:

按照给定的顺序,考虑重复应用以下步骤的过程:

过程 - M

M1. 使 F 明确;

M2. 如果 F 有两个子句 C'和 C'',满足 $C'\subseteq C''$,则从 F 中删除 C'';

M3. 如果 F 有一个 $ab\leqslant a+b$ 的(a, b) - 文字 l,则对 l 应用消解原理;

M4. 如果 F 有一个$(3^+,3^+)$ - 文字,那么在该文字上进行分支。

过程 - M 的步骤 M2 显然是准确的:如果赋值满足子句 C',那么它肯定满足 C''。如上所述过程 - M 的步骤至少与(3,3) - 文字上的分支同样有用。通常如果过程 - M 的任何一个步骤都不能应用于 F,那么 CNF 公式 F 是可简化的。

因此,只需要查看简化实例,它包含$(2,3^+)$ - 文字和$(3^+,2)$ - 文字。考虑下面两种情形:

情形 1　F 中包含$(2,3^+)$ - 文字的每个子句也包含$(3^+,2)$ - 文字。

这种情况很容易通过以下推导证明:(A1)简化实例 F 中的每个文字都是$(2,3^+)$ - 文字或$(3^+,2)$ - 文字;(A2)对于文字 l,l 和 $\bar{l}$ 不可能同时都是$(2,3^+)$ - 文字。因此,可以直接将 1 赋给所有$(3^+, 2)$ - 文字,然后(S1)如果一个子句只包含$(3^+, 2)$ - 文字,则该子句可满足;(S2)如果一个子句包含一个$(2,3^+)$ - 文字,根据情形的条件该子句还包含$(3^+,2)$ - 文字,因此该子句也满足。在第一种情形下,公式 F 是可满足的。

情形 2　F 中有一个子句只包含$(2,3^+)$ - 文字。

可简化公式 F 必须包含$(2,3^+)$ - 文字。如果情形 1 的条件不成立,那么 F 中必须有一个只包含$(2,3^+)$ - 文字的子句 C。由于 F 是明确的(过程 - M 的步骤 M1),$|C|\geqslant 2$。不是 C 中所有文字都可以包含在 F 中的另一个子句中(因为过程 - M 的步骤 M2)。因此,C 中必须有两个$(2,3^+)$ - 文字 l_1 和 l_2,F 中必须有另外两个子句 C_1 和 C_2 使得 $l_1\in C_1, l_2\in C_2, C, C_1$ 和 C_2 两两不同。在分支 $F[l_1=0]$中,至少 F 中的 3 个子句是可满足的(因为 l_1 是$(2,3^+)$ - 文字)。另外,在分支 $F[l_1=1]$中,子句 C 和 C_1 是可满足的,从公式中删除。现在 l_2 只出现在 C_2 子句中,变成 $F[l_1=1]$中的一个$(1,0^+)$ - 文字,可以在 l_2 上进行消解运算,将 $F[l_1=1]$中的子句数至少减少 1。通过分支 $F[l_1=1]$,然后在 l_2 上进行消解,子句的数量至少减少了 3 个。这表明,在 $F[l_1=1]$中文字 l_1 上的分支与 l_2 上的消解相结合,至少与(3,3) - 文字上的分支一样好。算法 4 对上述讨论进行了总结。

算法 4:$\mathrm{SAT}_m(F)$

INPUT: 一个 CNF 公式 F

OUTPUT: 判断是否可满足

1　减小 F;

2　IF (F 中有$(3^+,3^+)$ - 文字 l)
　　THEN ($\mathrm{SAT}_m(F[l=0])\vee\mathrm{SAT}_m(F[l=1])$)

3　IF (每个包含$(2,3^+)$ - 文字的子句也包含$(3^+,2)$ - 文字)
　　THEN RETURN (YES);

4　选取两个$(2,3^+)$ - 文字 l_1, l_2 和 3 个不同的子句 C, C_1, C_2,使 $l_1, l_2\in C$, $l_1\in C_1$, $l_2\in C_2$;

5　IF ($\mathrm{SAT}_m(F[l_1=0])$可满足), RETURN (YES);
　　ELSE 在 l_2 上进行消解,令 $F_1=F[l_1=1]$;RETURN ($\mathrm{SAT}_m(F_1)$)。

上述讨论证明了 SAT_m 算法的正确性。特别通过对情形 1 的讨论,在步骤 3 的条件下公式 F 必须是可满足的。对情形 2 的讨论确保了如果步骤 3 中的条件不成立,那么算法的步骤 4 总能找到所需的文字和子句。情形 2 所讨论的步骤 5 中的实例 $F[l_1=0]$和 F_1 中的每一个都将子句的数目 m 至少减少 3。因此,步骤 5 中的分支至少与(3,3) - 文字上的分支一样好。如果在 m 个子句的公式上,让$\#(m)$为算法 SAT_m 的搜索树中的叶子数,则所有分支规则至少与$\#(m)\leqslant 2\#(m-3)$一样好,得出算法 SAT_m 的运行时间为 $O^*(1.260^m)$。

定理 7　SAT 问题能在时间 $O^*(1.260^m)$内求解,其中,m 是输入公式中的子句数。

对 SAT_m 算法的讨论提出了一个合理的方法来分析 SAT 问题的分支和搜索算法:复杂的分支步骤是算法效率的"瓶颈"(SAT_m 算法是(2,3) - 文字分支),总会导致一些特殊结构,能够在这些结构上应用更有效的运算(SAT_m 上是公式 $F[l_1=1]$的消解)。通过将复杂的分支步骤与更高效的操作相结合,从而得到了更简易的分支,这将突破"瓶颈",得出更高效的算法。

SAT_m 算法的瓶颈是(3,3) - 文字的分支。在对瓶颈分支操作后公式结构更细致分析的基础上,得到了一些改进的算法。Hirsch[27] 开发了一种运行时间为

$O^*(1.239^m)$的SAT算法,后来Yamamoto[28]用一种在时间$O^*(1.234^m)$中运行的算法对其进行了改进。Chu等[29]也报告了运行时间为$O^*(1.223^m)$的SAT算法。

从目前的研究现状来看,用m来衡量SAT算法可以比用(3,3)-文字分支算法做得更好。下一个目标是(3,4)-文字。在(3,4)-文字上实现分支的效率将得到时间为$O^*(1.221^m)$的SAT算法。目前还未有符合这个上限的SAT算法。

4 基于参数L的算法

下面考虑用输入实例长度L来衡量SAT算法时间复杂度。由于SAT问题是NP完全问题,不期望SAT算法的运行时间是L的多项式,要得到由参数L度量的SAT算法,可以观察得到$L \geqslant m$。任何运行时间为$O^*(c^m)$的SAT算法也是运行时间为$O^*(c^L)$的SAT算法。根据第3节讨论的结果,易知SAT问题可以在时间$O^*(1.223^L)$内求解。

分支和搜索SAT算法通常使用参数L比使用参数m做得更好。对于子句C中的文字l,赋值$l=1$使子句C满足,从而将参数m减少1,将参数L减少$|C|$。当处理参数L时,用于参数m的一些方法变得无效。设公式F是明确的,没有子句是F中另一个子句的子集(参见第3节过程-M的步骤M1和M2),过程-M的步骤M3不再满足:即使在$ab<a+b$的条件下,在(a,b)-文字上的消解也可能增加公式的长度,(1,1)-文字上的消解不会增加公式长度。考虑(1,2)-文字l,公式F可以写成

$$F=F_0\wedge(l\vee C_1)\wedge(\bar{l}\vee C_1')\wedge(\bar{l}\vee C_2')。$$

如果$|C_1|\leqslant 3$,则不难验证消解不会增加公式长度。

当使用L作参数时,有以下过程:

过程-L

L1. F是明确的;

L2. 如果F有两个子句C'和C'',满足$C'\subseteq C''$,则从F中删除C'';

L3. 如果F有一个(1,1)-文字l,或者有一个(1,2)-文字l,其中,l在一个大小不大于4的子句中,那么对l进行消解;

L4. 选择文字l和$\bar{l}$上的分支。

步骤L1-L3是多项式时间的,没有增加公式长度。只需要考虑步骤L4,假设步骤L1-L3都不适用于公式F,让l是L4中F的文字。

如果l是(1,2)-文字,那么$F=F_0\wedge(l\vee C_1)\wedge(\bar{l}\vee C_1')\wedge(\bar{l}\vee C_2')$。由于步骤L3中$|C_1|\geqslant 4$,并且由L1易知$|C_1'|,|C_2'|\geqslant 1$。赋值$l=1$将公式长度至少减少$5+1+1=7$(也可以通过此赋值从公式中删除$\bar{l}$),赋值$l=0$会将公式长度至少减少$1+2+2=5$。这样分支至少和(5,7)-分支一样好。

如果l是$(1,3^+)$-文字,那么$F=F_0\wedge(l\vee C_1)\wedge(\bar{l}\vee C_1')\wedge(\bar{l}\vee C_2')\wedge(\bar{l}\vee C_3')$(CNF公式$F_0$可能仍然包含文字$\bar{l}$)。步骤L3不适用于$l$,$|C_1|$可以小到1。赋值$l=1$将公式长度至少减少$2+1+1+1=5$,赋值$l=0$将公式长度至少减少$1+2+2+2=7$。同样,至少和(5,7)-分支一样好。

如果l是$(2^+,2^+)$-文字,那么使用与上述类似的分析可以得出分支至少与(6,6)-分支一样好。

(5,7)-分支的分支多项式为$x^L-x^{L-5}-x^{L-7}$,其最大根为$x=1.123\cdots$,(6,6)-分支的分支多项式为x^L-2x^{L-6},其最大根为$x=1.122\cdots$。通过定理3,得出定理8。

定理8 SAT问题能在时间$O^*(1.124^L)$内求解,其中,L是输入公式的长度。

同样地,通过更仔细地检查复杂分支操作的公式结构,识别更多不需要分支就可以处理的情况,定理8中给出的上界可以得到改进。进展包括Gelder[30]提出的运行时间为$O^*(1.093^L)$的SAT算法,Hirsch[27]提出的进一步改进的时间上界$O^*(1.074^L)$,以及Wahlström[31]提出的运行时间为$O^*(1.067^L)$的新的SAT算法。

Chen等[32]提出了一种基于度量与控制方法的新算法,这是一种用于分析一般NP难问题的分支与搜索算法新技术[33]。将CNF公式F中变量x的度数$\deg(x)$定义为文字x和$\bar{x}$在F中出现的次数。$L=\sum_{x\in F}\deg(x)$。在更大程度的变量上进行分支,会使公式长度减少得更明显,处理效率更高。对一个变量进行分支会改变其他变量的度数,这可能会影响实例的后续处理。

可以根据变量度数来分配不同权重的变量。分支搜索算法不仅考虑了公式长度的减少,而且还考虑了公式中变量度数的变化。

形式上w_i是分配给度数为i的变量的权重($i\geqslant 0$)。将CNF公式F的加权长度$L'(F)$定义为$L'(F)=\sum_{i\geqslant 1}w_i n_i$,其中,$n_i$是$F$中度数为$i$的变量的个数。Chen等[32]选择了以下可变权重:

$w_1=0.32$,$w_2=0.45$,$w_3=0.997$,$w_4=1.897$,当$i\geqslant 5$时,$w_i=i/2$,$0.32\cdot L(F)\leqslant L'(F)\leqslant L(F)$。分支和搜索过程尝试对导致权值长度$L'$最大减少的变量进

行分支。

从正则公式长度 L 到加权公式长度 L'的转换并非没有意义。多项式时间预处理中使用的许多技巧不再有效,它们可能会增加加权公式长度。在运算之后检查结果公式的加权长度也更加复杂,因为还需要考虑运算中涉及的变量度数。Chen 等[32]提出了一种基于加权公式长度的分支搜索 SAT 算法,并且能够证明该算法运行时间为 $O^*(1.134\,5^{L'})$。由于 $L'\leqslant L/2$,因此,给出了一个运行时间为 $O^*(1.065L)$的算法,该算法是目前基于参数 L 最快的 SAT 算法。

5　结　论

本文介绍 SAT 问题最前沿的精确算法,SAT 问题是计算机科学研究的核心,在自然科学、社会科学、医学、工程和商业等许多领域都有应用。本文的方法遵循了算法发展和计算复杂性理论中的主流,基于最坏情况对算法的性能进行了定量分析。在这一研究领域,仍有许多有趣而重要的未解问题。

第一个基本的方向是用参数 n、m 和 L 开发更快的 SAT 算法。文献中大多数已知算法都在很大程度上基于两种基本技术:消解和分支搜索。消解是一种有效的操作,可以减少变量的数量,缺点是它可能会增加公式的长度。这可能是困难的,而且需要繁琐的个案分析以了解公式通过特定变量消解而实际影响的长度。事实上大多数算法在很多情况下都不应用消解,这仅仅是因为检查所有可能的情况来验证解析操作的效率变得过于繁琐。另一方面分支和搜索是一个复杂的操作,这是使算法指数时间运行的来源,基于最坏情况分析考虑了应用分支和搜索最坏可能的公式结构,并且在很大程度上忽略了复杂的分支和搜索操作次数以及应用这些操作后的结果。开发 SAT 算法的经验表明分支和搜索操作可能会产生一些非常受欢迎的公式结构,在这些结构上可以应用更有效的操作,然而同样不好的是这些公式结构研究困难,而且这种方法很难避免繁琐的逐个检查。许多逐步改进的 SAT 算法在很大程度上是基于识别更多公式结构的技术,在这些公式结构上可以应用有效的运算。研究复杂的分支和搜索产生的可以更有效处理的公式结构能够“平衡”分支和搜索操作所导致的高计算复杂性。进一步的研究仅能取得小的改进,想要获得突破还需要新的技术、算法和分析。

在第 2 节中所讨论的 Schöning 的 3-SAT 问题随机算法是为 SAT 问题开发的新算法技术,不使用消解也不使用分支和搜索。它从一个精心选择的初始赋值开始(在随机版本中,它是一个随机选择的赋值,而在其去随机化版本[20]中,赋值是由精心设计的“覆盖码”构成的,覆盖了一些 Hamming 半径的球的所有可能赋值)。在初始分配之后,该算法使用局部搜索来“纠正”初始分配中的错误位。这种方法还是很自然的,不便之处在于分析该算法的有效性。

第 4 节中描述的度量和控制方法是 SAT 算法的新分析技术。该方法对变量赋予不同的权值,使得不同变量的分支和搜索操作的计算开销得到评估。通过计算分支和搜索操作的权重,用高效的操作“平衡”分支和搜索的高计算复杂性,从而使算法总的复杂度变得更好。

这些新的算法和分析技术仍处于非常初级的阶段,前景很好,对发现更快的 SAT 算法具有重要作用。

对 SAT 问题的计算上限也需要更新、更深入的研究。人们希望有更有力的证据来支持 ETH 和 SETH 假说,假说失败的可能性并没有被完全排除。许多与 ETH 和 SETH 有关或类似问题的答案都是未知的。例如如果使用参数 L 来测量 SAT 算法,SAT 算法是否可能在时间 $O^*(2^{o(L)})$内运行,这是以 L 表示的次指数时间,这并非完全不可能,用 L 度量的当前最佳 SAT 算法运行时间为$O^*(1.065^L)$,1.065 已经非常接近 1(SAT 问题可以在时间 $O^*(2^{o(L)})$内解决当且仅当它可以在时间 $O^*(c^L)$ $(c>1)$内解决)。此外,SAT 问题的 $O^*(2^{o(L)})$ 时间算法与 SAT 问题的 NP 完备性并不矛盾。有许多已知的 NP 完全问题(如平面图上的 NP 完全图问题)可以在其输入长度上以次时间指数求解。

除了基于最坏情况计算复杂性分析的 SAT 问题精确算法的研究外,SAT 问题启发式算法的发展历史悠久,是计算机科学与工程领域一个非常活跃和热门的研究领域。启发式算法以实例结构中的直觉和实践经验为基础,试图有效地解决 SAT 问题。在某些情况下启发式算法可能无法正确地解决问题(如在 YES 实例中可能输出 NO),而且可能无法保证某些输入实例的计算效率。一些 SAT 算法被称为“启发式”算法,即使它们能够在所有输入实例上正确有效地解决 SAT 问题,但是不能为这些性能提供正式的证明。许多文献中发表的启发式算法在大多数 SAT 实例上运行良好,并在实际中得到了成功的应用。关于 SAT 问题启发式算法的更多细节可参阅文献[34]。

这些求解 SAT 问题的启发式算法在文献中被称为 SAT 求解器。在本文中详细讨论的两种基本技术消解和分支与搜索一直是 SAT 求解器开发中的主要工具。

尽管在最坏的情况下,所有已知的SAT算法的运行时间复杂度都具有很强的指数因子,但现代SAT求解器能够求解涉及数万个变量和数百万个子句的SAT实例[35]。某种程度上快速SAT求解器的存在与基于最坏情况分析的SAT算法时间复杂度的最佳理论上界相矛盾,这表明该研究领域的理论研究与实际实现之间存在着巨大的差距。显然为了更好地求解SAT问题和更准确地分析SAT算法,还需要新的技术和更深入的研究。

参考文献:

[1] Cook S A. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971[C]. New York: ACM, 1971.

[2] Arora S, Barak B. Computational complexity: A modern approach[M]. Cambridge: Cambridge University Press, 2009.

[3] Biere A, Heule M. Handbook of satisfiability[M]. Amsterdam: IOS Press, 2009.

[4] McMillan K L. International Conference on Computer Aided Verification, July 8-12, 2003[C]. Berlin: Springer, 2003.

[5] Armando A, Castellini C, Giunchiglia E, et al. SAT-based decision procedures for automated reasoning: A unifying perspective[M]//Mechanizing Mathematical Reasoning. Berlin: Springer, 2005.

[6] Smith A, Veneris A, Ali M F, et al. Fault diagnosis and logic debugging using Boolean satisfiability[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, 24(10):1606-1621.

[7] Guo S, Sun W, Weiss M A. Solving satisfiability and implication problems in database systems[J]. ACM Transactions on Database Systems, 1996, 21(2): 270-293.

[8] Rintanen J, Heljanko K, Niemeliä I, et al. Planning as satisfiability: Parallel plans and algorithms for plan search[J]. Artificial Intelligence, 2006, 170(12/13): 1031-1080.

[9] Horbach A. A Boolean satisfiability approach to the resource-constrained project scheduling problem[J]. Annals of Operations Research, 2010, 181: 89-107.

[10] Ali M F, Veneris A, Smith A, et al. Proceedings of the 2004 IEEE/ACM International Conference on Computer Aided Design, November 7-11, 2004[C]. Piscataway: IEEE, 2004.

[11] Velev M N, Bryant R E. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors[J]. Journal of Symbolic Computation, 2003, 35(2): 73-106.

[12] Mishchenko A, Zhang J S, Sinha S, et al. Using simulation and satisfiability to compute flexibilities in Boolean networks [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2006, 25(5): 743-755.

[13] Davis M, Putnam H. A computing procedure for quantification theory[J]. Journal of the ACM, 1960, 7(3): 201-215.

[14] Chen J E, Kanj I A, Jia W J, et al. Vertex cover: Further observations and further improvements[J]. Journal of Algorithms, 2001, 41(2): 280-301.

[15] Impagliazzo R, Paturi R. 14th Annual IEEE Conference on Computational Complexity, May 6, 1999[C]. Piscataway: IEEE, 1999.

[16] Cygan M, Fomin F V, Fowalik L, et al. Parameterized algorithms[M]. Berlin: Springer, 2015.

[17] Garey M R, Johnson D S. Computers and intractability: A guide to the theory of NP-completeness[M]. San Francisco: W. H. Freeman, 1979.

[18] Monien B, Speckenmeyer E. Solving satisfiability in less than 2^n steps[J]. Discrete Applied Mathematics, 1985,10(3): 287-295.

[19] Schöning T. 40th Annual Symposium on Foundations of Computer Science, October 17-19, 1999[C]. Piscataway: IEEE, 1999.

[20] Dantsin E, Goerdt A, Hirsch E A, et al. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search[J]. Theoretical Computer Science, 2002, 289(1): 69-83.

[21] Paturi R, Pudlak P, Saks M E, et al. An improved exponential-time algorithm for k-SAT[J]. Journal of the ACM, 2005, 52(3): 337-364.

[22] Iwama K, Tamaki S. Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithm, January 11-14, 2004 [C]. New York: ACM, 2004.

[23] Hertli T. 3-SAT faster and simpler-unique-SAT bounds for PPSZ hold in general[J]. SIAM Journal on Computing, 2014, 43(2): 718-729.

[24] Hansen T, Kaplan H, Zamir O, et al. Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, June 23-26, 2019[C]. New York: ACM, 2019.
[25] Impagliazzo R, Paturi R, Zane F, et al. Which problems have strongly exponential complexity? [J]. Journal of Computer and System Sciences, 2001, 63(4): 512-530.
[26] Chen J, Kanj I A, Xia G, et al. On parameterized exponential time complexity[J]. Theoretical Computer Science, 2009, 410(27/28/29): 2641-2648.
[27] Hirsch E. Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, January 25-27, 1998[C]. New York: ACM, 1998.
[28] Yamamoto M. Proceedings of the 16th International Conference on Algorithms and Computation, December 19-21, 2005 [C]. Berlin: Springer, 2005.
[29] Chu H, Xiao M, Zhang Z, et al. An improved upper bound for SAT[J]. Theoretical Computer Science, 2021, 887: 51-62.
[30] Gelder A V. A satisfiability tester for non-clausal propositional calculus[J]. Information and Computation, 1988, 79(1): 1-21.
[31] Wahlström M. Proceedings of the 13th Annual European Symposium on Algorithms, October 3-6, 2005[C]. Berlin: Springer, 2005.
[32] Chen J, Liu Y. 11th International Symposium on Algorithms and Data Structures, August 21-23, 2009[C]. Berlin: Springer, 2009.
[33] Fomin F V, Grandoni F, Kratsch D, et al. A measure & conquer approach for the analysis of exact algorithms[J]. Journal of the ACM, 2009, 56(5): 1-32.
[34] Gomes G, Kautz H, Sabharwal A, et al. Chapter 2 satisfiability solvers[M]//Handbook of Knowledge Representation. Amsterdam: Elsevier, 2008.
[35] Ohrimenko O, Stuckey O, Codish M, et al. Proceedings of the 13th International Symposium Conference on Principles and Practice of Constraint Programming, September 23-27, 2007[C]. Berlin: Springer, 2007.

【责任编辑: 卓祯雨】

(上接第 40 页)

[19] Shah M M. Prediction of evaporation from occupied indoor swimming pools[J]. Energy & Buildings, 2003, 35(7):707-713.
[20] Shah M M. Improved method for calculating evaporation from indoor water pools[J]. Energy & Buildings, 2012, 49:306-309.
[21] Shah M M. Evaluation of available correlations for rate of evaporation from undisturbed water pools to quiet air[J]. Hvac & R Research, 2002, 8(1):125-131.
[22] Shah M M. Methods for calculation of evaporation from swimming pools and other water surfaces[J]. ASHRAE Transactions, 2014, 120:3-17.
[23] Shah M M. Calculation of evaporation from indoor swimming pools: Further development of formulas[J]. ASHRAE Transactions, 2012, 118:1-8.
[24] Shah M M. Improved model for calculation of evaporation from water pools[J]. Science & Technology for the Built Environment, 2018,24(10):1064-1074.
[25] Association of German Engineers-Directives. Building services in swimming pools[S]. Dusseldorf: Association of German Engineers,2020.
[26] 中华人民共和国建设部. 体育建筑设计规范:JGJ31—2003[S]. 北京:中国建筑工业出版社,2004.
[27] 中华人民共和国住房与城乡建设部. 游泳池给水排水工程技术规程:CJJ122—2017[S]. 北京:中国建筑工业出版社,2017.
[28] 于凌燕, 刘传聚, 林亚宏. 室内游泳池水分蒸发量的实测与分析[J]. 暖通空调, 2006, 36(9):68-71.
[29] 林亚宏. 游泳馆室内热湿环境控制问题的研究[D]. 上海:同济大学, 2005.

【责任编辑: 孙向荣】