最小布尔不可满足子式的求解算法

张建民, 沈胜宇, 李思昆
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 解释布尔公式不可满足的原因在众多领域都具有非常重要的理论与应用价值, 而最小不可满足子公式能够为公式不可满足的原因提供精确的解释, 帮助自动化工具迅速定位错误, 诊断问题失败的缘由. 针对最小不可满足子式的求解问题, 提出并证明了布尔公式最小不可满足性与极大可满足性之间的关系. 基于二者的关系, 提出了求解最小布尔不可满足子式的贪心遗传算法与蚁群算法, 并且通过实验与当前最好的方法分支-限界算法进行了对比, 结果表明: 两种算法在运算效率以及单位时间内剔除的短句数上都显著优于分支-限界算法, 而贪心遗传算法优于蚁群算法.

关键词: 形式化验证; 最小不可满足子式; 极大可满足子式; 贪心遗传算法; 蚁群算法
中图分类号: TP391 文献标识码: A 文章编号: 0372-2112 (2009) 05-0993-07

Algorithms for Deriving Minimum Unsatisfiable Boolean Subformulae

ZHANG Jian-min, SHEN Sheng-yu, LI Si-kun
(*School of Computer Science, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract: Explaining the causes of infeasibility of Boolean formulae has practical applications in various fields. A smallest-cardinality unsatisfiable subformula can provide a succinct explanation of infeasibility, and help automatic tools to rapidly locate the errors, and determine the underlying reasons for the failure. We present the relationship between maximal satisfiability and minimum unsatisfiability. Based on the relationship, a compounded greedy genetic algorithm and an ant colony algorithm are proposed to derive a minimum unsatisfiable subformula. We report experimental results on practical benchmarks, as compared with the best known branch-and-bound algorithm. The results show that two algorithms strongly outperform the branch-and-bound algorithm, and the compounded greedy genetic algorithm outperforms the ant colony algorithm on both efficiency and size of unsatisfiable subformulae.

Key words: formal verification; minimum unsatisfiable subformula; maximal satisfiable subformula; compounded greedy genetic algorithm; ant colony algorithm

1 引言

众多领域的实际问题, 例如硬件的形式化验证、等价性检查、自动测试向量生成等, 都可以等价为约束可满足问题来解决, 即这些问题能够规约为合取范式 (Conjunctive Normal Form, CNF) 格式的布尔公式. 布尔可满足 (Boolean Satisfiability, SAT) 求解器, 由于实现了增强 DPLL (Davis-Putnam-Logeman-Loveland) 回溯搜索算法, 能够高效地求解 CNF 公式的可满足性. 如果公式不可满足, 需要查找不满足的原因, 这就要求剔除与不可满足无关的短句, 保留能够反映其原因的一部分短句, 即提取不可满足子式. 不可满足子式的应用领域包括 FPGA 的布线策略[1]、错误定位[2]、RTL 级 Verilog 的谓词抽象[3]等. 文献[4]指出, 在大多数情况下, 最小不可满足子式远小于极小不可满足子式, 这是由于极小不可满足子式往往包含简单的消解规则所不能剔除的冗余短句, 因此最小不可满足子式能够给出关于公式不可满足更加精确的解释, 帮助自动化工具迅速定位错误, 在实际应用中具有更重要的价值.

近年来 SAT 求解器得到了飞速地发展, 因此基于 SAT 求解器的不可满足子式提取方法逐渐成为了研究的主流方向. zCore[5] 是一种基于 SAT 求解器产生的消解悖论来提取不可满足子式的算法. 自适应核搜索方法[6]采用短句复杂度的概念来构造不可满足子式. AMUSE[7]通过增强 DPLL 过程在增加选择变量的公式上搜索不可满足子式. 局部预先赋值方法[8]在对剩余公式进行可满足性测试时将部分变量预先赋值. CoreTimmer[9]通过证明短句蕴含图中的结点与公式短句的等价

收稿日期: 2007-11-19; 修回日期: 2008-12-12
基金项目: 国家自然科学基金 (No. 60603088)

性来构造不可满足子式. Dershowitz 等[10] 提出了一种利用 SAT 求解器产生的消解悖论来提取极小不可满足子式的可扩展算法. HYCAM 算法[11] 采用局部搜索算法来求解公式的全部极小不可满足子式. CAMUS 算法[12] 提出了基于公式极大可满足性与极小不可满足性之间的关系来求解所有极小不可满足子式的方法. Maaren[13] 等提出了基于 Brouwer 不动点定理来求解极小不可满足子式的算法.

求解布尔不可满足子式的方法可以分为两类: 一类是小的或极小不可满足子式的提取算法, 一类是最小不可满足子式的提取算法. 前面提到的算法都是属于第一类. 相对而言, 最小不可满足子式的求解难度更大, 算法复杂度也更高. 但是近几年已有一些研究工作开始涉及到如何求解最小不可满足子式. Lynce 等[4] 提出了一种使用 SAT 求解器穷尽搜索公式的全部空间来提取最小不可满足子式的算法. 由于其搜索空间太大, 能够处理问题的规模十分有限. Mneimneh 等[14] 提出了一种分支-限界算法来求解最小布尔不可满足子式. 它利用极大可满足性反复产生不可满足子式的上界与下界, 并在特定的子公式上分支从而得到最小不可满足子式. 它是目前已公开发表的求解最小布尔不可满足子式最好的方法.

本文针对最小布尔不可满足子式的高效求解问题, 提出并证明了最小不可满足性与极大可满足性之间的关系: 最小布尔不可满足子式是所有极大可满足子式的补集所构成的集合簇的最小碰集. 并且基于该关系, 提出一种贪心遗传算法(Compounded Greedy Genetic Algorithm, CGGA) 与蚁群算法(Ant Colony Algorithm, ACA) 来解决从 CNF 公式中提取最小不可满足子式的问题. 算法的基本思路是: 首先利用 SAT 求解器提取布尔公式的所有极大可满足子式, 而后通过计算极大可满足子式补集的最小碰集, 从而得到不可满足子式. 算法要求运算时间与结果大小兼顾, 因此单位时间内剔除的短句数就是衡量算法优劣的重要标准. 实验结果表明, 贪心遗传算法与蚁群算法能够快速地求解出最小或近似最小不可满足子式, 并且二者的运算效率以及单位时间内剔除的短句数都显著高于分支-限界算法[14], 通常在一个数量级之上; 而贪心遗传算法无论在运行时间还是不可满足子式的大小上都优于蚁群算法.

2 不可满足子式的定义

合取范式公式的构造规则是: 文字是变量本身或变量的非, 而若干个文字的析取构成短句, 若干个短句的合取组成公式.

定义 **1**(可满足问题) 给出一个 CNF 公式 φ: $\varphi=\bigwedge_{i=1}^{n} C_i$, 其中短句 $C_i=\bigvee_j x_j \vee \bigvee_k \neg x_k$, x 表示变量. 可满足问题是指给定如下形式的布尔函数: $B^N \rightarrow B$, 其中 $B=\{0,1\}$, 是否存在公式中所有变量 x_i 的赋值 $X \in B^N$ 使得 $\varphi=1$. 如果存在这样的赋值 X, 则公式 φ 是可满足的; 否则公式 φ 是不可满足的.

定义 **2**(不可满足子式) 给出一个公式 φ, ϕ 是公式 φ 的一个不可满足子式当且仅当 ϕ 是不可满足的, 并且 $\phi \subseteq \varphi$.

定义 **3**(极小不可满足子式) 给出公式 φ 的一个不可满足子式 ϕ, ϕ 是极小不可满足子式当且仅当从 ϕ 中删除任意一个短句 $\omega \in \phi$, 都使得 $\phi-\{\omega\}$ 是可满足的.

对于一个不可满足子式来说, 若移除其任何一个短句都会导致其可满足, 即它的所有真子集都是可满足的, 那么它是极小不可满足子式.

定义 **4**(最小不可满足子式) 给出一个公式 φ, 以及 φ 的所有不可满足子式构成的集合: $\{\phi_1, \phi_2, \cdots, \phi_j\}$. 那么 $\phi_k \in \{\phi_1, \phi_2, \cdots, \phi_j\}$ 是最小不可满足子式, 当且仅当 $\forall \phi_i \in \{\phi_1, \phi_2, \cdots, \phi_j\}$, $1 \leqslant i \leqslant j$, 使得 $|\phi_k| \leqslant |\phi_i|$.

根据定义 4, 最小不可满足子式是一个公式所有不可满足子式中最小的, 那么能够得出下面的结论: 所有不可满足的布尔公式都至少包含一个最小不可满足子式.

3 最小不可满足性与极大可满足性之间的关系

首先分析布尔公式极大可满足性与极小不可满足性之间的关系. 下面给出可满足子式与极大可满足子式的定义:

定义 **5**(可满足子式) 给出一个公式 φ, ϕ 是公式 φ 的一个可满足子式当且仅当 ϕ 是可满足的, 并且 $\phi \subseteq \varphi$.

定义 **6**(极大可满足子式) 给出公式 φ 的一个可满足子式 ϕ, ϕ 是极大可满足子式当且仅当在 ϕ 中增加任意一个短句 $\omega \in \{\varphi-\phi\}$, 都使得 $\phi \cup \{\omega\}$ 是不可满足的.

给出一个不可满足的布尔公式 φ, 以及 φ 的一个极大可满足子式 ϕ, 那么 ϕ 的补集可以表示为 $\eta=\varphi-\phi$, 即公式中不属于 ϕ 的短句构成的集合. 那么从 φ 中删除 η 中的短句能够改变公式的不可满足性. 而 ϕ 包含了公式不可满足的原因, 因此要想改变公式的不可满足性, 必须从每个极小不可满足子式中至少移除一个短句, 才能使得极小不可满足子式变为可满足的. 而 η 是由那些从公式 φ 中删除就可以改变 φ 的不可满足

性的短句构成, 因此它包含了每个极小不可满足子式的至少一个短句.

定义 **7**(极小碰集) 假设 $\Sigma=\{S_1, S_2, \cdots, S_n\}$ 是集合簇, 那么 H 是 Σ 的碰集当且仅当 $H \subseteq \bigcup_{1\leqslant i\leqslant n} S_i$, 并且 $H \cap S_i \neq \emptyset\ 1 \leqslant i \leqslant n$. H 是极小碰集当且仅当 $\forall H' \subset H, H'$ 为 H 的真子集, 使得 H' 不是碰集.

根据前面的分析得出, 一个公式的极大可满足子式的补集是该公式所有极小不可满足子式组成的集合簇的碰集. 根据二者之间的对等关系, 可以得到极小不可满足子式是极大可满足子式的补集所构成集合簇的碰集. 而后根据定义 3, 能够得出引理 1 中的结论[12].

引理 **1** 公式 φ 的极小不可满足子式是该公式所有极大可满足子式的补集所构成集合簇的极小碰集.

引理 1 给出了布尔公式极大可满足性与极小不可满足性的关系. 但是, 最小不可满足性与极大可满足性之间的关系是什么? 首先要证明最小不可满足子式与极小不可满足子式之间的关系.

引理 **2** 公式 φ 的最小不可满足子式一定是极小不可满足子式. (采用反证法易得, 此处略.)

定义 **8**(最小碰集) 给出一个集合簇 Σ, 以及 Σ 的所有碰集组成的集合簇: $\{H_1, H_2, \cdots, H_j\}$, 那么 $H_k \in \{H_1, H_2, \cdots, H_j\}$ 是最小碰集, 当且仅当 $\forall H_i \in \{H_1, H_2, \cdots, H_j\}, 1 \leqslant i \leqslant j$, 使得 $|H_k| \leqslant |H_i|$.

引理 **3** 一个集合簇 C 的最小碰集一定是极小碰集. (采用反证法易得, 此处略).

综合引理 1、引理 2 与引理 3 的结论, 就可以推导出下面的定理:

定理 **1** 公式 φ 的最小不可满足子式是 φ 的所有极大可满足子式的补集所构成集合簇的最小碰集.

证明: 给出一个 CNF 公式 φ, ψ 是公式 φ 的不可满足子式, $\Pi=\{H_1, H_2, \cdots, H_n\}$ 是由 φ 的所有极大可满足子式的补集集合簇的极小碰集组成, H 是极大可满足子式的补集的最小碰集.

采用反证法. 假设 ψ 是最小不可满足子式, 但不是公式 φ 的所有极大可满足子式的补集所构成集合簇的最小碰集.

$\because H$ 是公式 φ 的极大可满足子式的补集的最小碰集; $\therefore$ 根据引理 3, $\exists H_k \in \Pi, 1 \leqslant k \leqslant n$, 使得 $H_k = H$.

$\because \psi$ 是最小不可满足子式; $\therefore$ 根据引理 2, ψ 是极小不可满足子式. $\because$ 根据引理 1 可得到: $\Pi=\{H_1, H_2, \cdots, H_n\}$ 是公式 φ 的所有极小不可满足子式构成的集合簇; $\therefore \psi \in \Pi$, 即 $\exists H_i \in \{H_1, H_2, \cdots, H_n\}, 1 \leqslant i \leqslant n$, 使得 $\psi = H_i$, 并且根据假设条件, H_i 仅是极小而不是最小碰集, 即 $i \neq k$.

$\because H_k \in \Pi$ 是最小碰集, 根据定义 7, H_k 是所有碰集中最小的. $\therefore |H_k| < |H_i|$, 并且根据引理 1, H_k 为公式 φ 的极小不可满足子式.

那么得到一个不可满足子式 H_k 使得 $|H_k| < |\psi|$, 与假设矛盾. 因此假设错误. 证毕.

下面通过一个例子说明不可满足子式与可满足子式之间的关系. 给出一个 CNF 公式 φ 为:

$$\varphi = (x_1) \wedge (\neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3) \quad (1)$$

采用 DIMACS CNF 格式, 即每个短句以其在公式中的位置编码. 表 1 中列出了公式 φ 的所有可满足与不可满足子式. 该公式包含 4 个极大可满足子式及其补集. 从表 1 可以看出, 2 个极小不可满足子式是公式 φ 的极大可满足子式的补集所构成集合簇的极小碰集, 而最小不可满足子式是极大可满足子式的补集集合簇的最小碰集.

表 **1** 公式 φ 的可满足与不可满足子式

极大可满足子式	补集	极小不可满足子式	最小不可满足子式
{1, 2, 4, 5}	{3}		
{2, 3, 4, 5}	{1}	{1, 2, 3}	
{1, 3, 4}	{2, 5}	{1, 3, 4, 5}	{1, 2, 3}
{1, 3, 5}	{2, 4}		

4 最小布尔不可满足子式的求解算法

基于定理 1 的结论, 提出了一种贪心遗传算法以及蚁群算法, 解决最小布尔不可满足子式的高效求解问题.

4.1 贪心遗传算法

遗传算法是一种启发式的最优化方法, 它采用了优胜劣汰的达尔文进化机制. 其核心思想是: 一组潜在的解种群通过自然进化与自然选择策略进行不断的优化. 遗传算法自从诞生以来, 由于其具有较强的通用性与鲁棒性, 不受函数约束条件的限制等优点, 因此广泛应用于诸如图像处理[15]与自适应控制[16]等众多领域.

通常来讲, 贪心算法具有较快的收敛速度, 但是往往会收敛于局部最优解; 而遗传算法是全局进化方法, 搜索的全局性强, 不易陷入局部最优, 但是在初期缺乏有效启发信息的情况下局部收敛速度较慢, 对初始种群很敏感, 其初始种群的选择直接影响到解的质量和算法效率. 所以将贪心算法与遗传算法有效地结合起来, 相互弥补对方的不足之处. 贪心遗传算法的基本策略是: 首先采用贪心算法快速计算不可满足子式的近似最优解, 为遗传算法构造一个较优的初始种群, 减小其搜索空间, 而后利用遗传算法的全局性反复精化近似解, 从而得到更小的不可满足子式. 首先介绍算法中出现的概念和遗传算子的定义.

遗传编码: 采用二进制向量来表示公式中短句的

染色体编码, 其形式为$\{x_1x_2\cdots x_n\}$, 其中 $x_i=0$ 或 1, $1\leqslant i\leqslant n$, n 表示公式的变量数. 短句编码后的二进制向量称为染色体, 其中每一位叫做一个基因, 而所有的染色体构成种群. 例如, 一个包含 8 个变量的公式的短句 $S=\{1,2,4,7\}$, 那么对应的染色体编码 $C_S=\{11010010\}$.

适应度函数: 适应度函数定义为$f(S)=H_S/|S|$, 其中 H_S 表示集合簇中被S 的元素命中的集合的个数, 而$|S|$ 表示集合 S 包含元素的数目.

杂交算子: 算法使用单点杂交策略. 假设两个染色体 $S_1=\{x_1x_2\cdots x_n\}$ 与 $S_2=\{y_1y_2\cdots y_n\}$, 以及一个随机整数 k, $1\leqslant k\leqslant n$, 那么 $S_3=\{s_i|$ 若 $i\leqslant k$, 则 $s_i\in S_1$, 否则 $s_i\in S_2\}$, 与 $S_4=\{s_i|$ 若 $i\leqslant k$, 则 $s_i\in S_2$, 否则 $s_i\in S_1\}$ 称为 S_1 与 S_2 单点杂交的子代.

变异算子: 假设一个染色体 $S_1=\{x_1x_2\cdots x_n\}$ 与一个随机整数 k, $1\leqslant k\leqslant n$, 那么 $S_2=\{s_i|$ 若 $i=k$, 则 $s_i=\neg x_i$, 否则 $s_i=x_i\}$ 称为 S_1 的变异子代.

倒位算子: 假设一个染色体 $S_1=\{x_1\cdots x_kx_{k+1}\cdots x_{l-1}x_l\cdots x_n\}$, 其中 k 与 l 是 2 个随机整数, $1\leqslant k<l\leqslant n$, 那么 $S_2=\{x_1\cdots x_lx_{l-1}\cdots x_{k+1}x_k\cdots x_n\}$ 是 S_1 的倒位子代.

选择算子: 算法采用轮盘赌选择策略. 使用适应度函数评价所有的染色体, 能够命中所有集合并且较小的不可满足子式以较大的概率被选择为下一次繁殖的父代.

求解最小布尔不可满足子式的贪心遗传算法的步骤如下:

步骤 **1** 利用 MiniSAT 求解器[17] 的 DPLL 过程在求解公式可满足性过程中产生的搜索信息, 计算极大可满足子式;

步骤 **2** 求解所有极大可满足子式的补集, 构成集合簇;

步骤 **3** 将集合簇中的所有短句重新编码, 以压缩其大小. 这主要是由于集合簇所包含的短句仅是原始公式的子集, 重新将这些短句编码, 从而使得算法后面的步骤更加高效;

步骤 **4** 采用增量式贪心算法求解两个最小不可满足子式的近似最优解, 作为初始的父代; 该过程反复迭代, 将当前解所命中的集合删除, 直到最终极大可满足子式的补集集合簇为空, 此时的解即为一个近似最小不可满足子式;

步骤 **5** 将初始父代与集合簇编码为二进制向量, 得到其染色体;

步骤 **6** 对所选择的父代采用遗传算子按照前面定义中的方式进行繁殖, 包括杂交算子、变异算子与倒位算子, 从而得到下一代染色体;

步骤 **7** 将当前的子代与其父代组成一个新的种群, 使用选择算子以及轮盘赌策略筛选出下一次循环的父代;

步骤 **8** 如果达到循环次数的上限, 或者最小不可满足子式连续 10 代不再减小, 则终止程序, 并报告当前最优解; 否则返回到第 6 步继续执行.

4.2 蚁群算法

为了更加全面地评价贪心遗传算法的效率, 实现了蚁群算法与之对比. 近年来, 蚁群算法在诸如系统芯片设计[18] 与网络路由[19] 等众多领域得到了广泛地应用. 蚁群算法是利用蚂蚁之间的行为学习进行搜索的启发式仿生优化算法, 融入人工智能的蚂蚁采用正反馈机制, 个体之间不断进行信息交流和传递, 通过目标上信息素的积累与更新, 高效收敛到最优解. 基于定理 1, 提取最小布尔不可满足子式的蚁群算法的过程如下:

步骤 **1** 利用 MiniSAT[17] 在求解公式可满足过程中产生的搜索信息提取极大可满足子式, 并计算所有极大可满足子式的补集, 组成集合簇;

步骤 **2** 将极大可满足子式的补集集合簇中的所有短句重新编码, 以减小集合簇;

步骤 **3** 将集合簇中的每个短句 i 都关联一个信息素变量 τ_i, 它是一个全局的启发式信息, 蚂蚁在运动过程中根据该变量选择下一个元素; 每个短句的信息素变量都初始化为 τ_0;

步骤 **4** m 个蚂蚁随机地选择第一个短句, 而后在集合簇中不断地选择下一个短句, 直到产生一个不可满足子式, 表示为 M_k, 其中 $k=1,2,\cdots,m$;

步骤 **5** 评价每个蚂蚁所生成的不可满足子式 M_k, 记录最优解; 其评价标准是不可满足子式的大小 $|M_k|$, 即所包含的短句数;

步骤 **6** 更新每个短句上的信息素, 其公式为:

$$\tau_i(t+1)=(1-\rho)\tau_i(t)+\sum_{j=1}^{m}\Delta\tau_i^k(t) \qquad (2)$$

其中 $0<\rho<1$ 是信息素挥发系数. 而 $\Delta\tau_i^k(t)$ 表示第 k 只蚂蚁在本次循环中留在短句 i 上的信息素总量, 采用下面的公式进行调整:

$$\Delta\tau_i^k(t)=\begin{cases}C-|M_k|, & \text{若 } M_k \text{ 是不可满足子式;}\\ 0, & \text{否则;}\end{cases} \qquad (3)$$

其中 C 表示布尔公式所包含短句的总数.

步骤 **7** 如果达到循环次数的上限, 则终止程序, 并报告当前最优解; 否则, 返回到步骤 4 继续执行.

步骤 4 在整个算法中至关重要. 在搜索解的过程中, 每个蚂蚁依据短句上的信息素与局部启发式信息来决定其下一个目标, 那么蚂蚁选择短句 i 的概率为:

$$p_i(t)=\begin{cases}\dfrac{[\tau_i(t)]^{\alpha}\cdot[\eta_i(t)]^{\beta}}{\sum_{n\in N}[\tau_i(t)]^{\alpha}\cdot[\eta_i(t)]^{\beta}}, & \text{若 } i\in N;\\ 0, & \text{否则;}\end{cases} \tag{4}$$

其中 N 表示满足下列条件的短句集合: 至少命中当前极大可满足子式的补集集合簇中一个集合, 并且仍不属于不可满足子式; η_i 是局部启发式信息, 表示短句 i 命中集合簇中集合的数量. 参数 $\alpha \geqslant 0$ 为信息启发式因子, 表示短句上所积累的信息素对蚂蚁选择该元素的相对重要性; 而 $\beta \geqslant 0$ 为期望启发式因子, 表示短句的局部启发信息在蚂蚁运动过程中所受的重视程度. 经过反复实验, 当 $\alpha=1$、$\beta=3$、$\rho=0.5$ 时, 在本算法中效果最好, 下一节的实验将采用这组参数值.

5　实验结果与分析

为了验证算法的有效性, 采用业界公认的 DaimlerChrysler 不可满足公式集[20] 作为基准测试向量. DaimlerChrysler 测试集来源于实际生产应用, 用于验证 DaimlerChrysler 公司的 Mercedes 汽车生产线上产品配置数据的正确性. 基于 DaimlerChrysler 测试集中的 20 个公式, 将本文提出的贪心遗传算法、蚁群算法与分支-限界算法[14] 进行了对比与分析. 贪心遗传算法与蚁群算法采用 C++ 与 STL 实现. 实验环境是 1.6GHz 的 Athlon CPU, 内存 1GB, 操作系统为 Linux 的机器. 所有算法的输入都是 DIMACS CNF 格式的公式, 运行的时限设置为 600 秒.

三种算法基于 DaimlerChrysler 测试集的实验结果如表 2 所示. 表中 MUSes 列数据表示每个公式所包含的全部极小不可满足子式的数目, 采用 CAMUS 算法[12] 求解全部极小不可满足子式; min 列是最小不可满足子式所包含的短句数; max 列是最大的极小不可满足子式所包含的短句数; ave 列是所有极小不可满足子式的平均大小. BaBA time 列是分支-限界算法的运行时间, 由于该算法能够得到精确的最小不可满足子式, 其结果与第 3 列相同. ACA time 列与 size 列分别是蚁群算法的运行时间与所提取不可满足子式的短句数. 最后两列分别是贪心遗传算法的运行时间与不可满足子式的短句数. 表中所有算法的运算时间都以秒为单位. 有 3 个公式在设定的时限内没有计算出所有极小不可满足子式, 因此它们极小不可满足子式的平均大小无法得到, 在表中以“*TO*”表示.

表 2　三种算法在 DaimlerChrysler 测试集上的实验结果

Benchmarks	MUSes	min	max	ave	BaBA	ACA		CGGA	
					time	time	size	time	size
C220_FV_SZ_65	103442	23	40	35.1	20.40	2.83	23	1.21	23
C220_FV_SZ_46	> 566210	17	⩾64	*TO*	80.75	16.45	17	12.20	17
C220_FV_RZ_14	80	11	18	14.5	35.21	1.51	11	0.15	11
C220_FV_RZ_13	6772	10	27	21.5	34.98	1.20	10	0.43	10
C220_FV_RZ_12	80272	11	35	26.5	52.38	2.75	11	0.60	11
C210_FW_RZ_59	15	140	173	156.8	58.77	7.92	140	0.89	140
C210_FS_RZ_40	15	140	173	156.8	38.20	7.79	140	0.69	140
C208_FA_UT_3255	52736	40	74	60.3	96.69	16.12	41	1.02	40
C208_FA_UT_3254	17408	40	74	59.7	97.70	13.80	41	0.90	40
C208_FA_SZ_87	12884	18	27	22.9	16.95	3.81	19	0.70	19
C208_FA_SZ_120	2	34	34	34.0	4.12	1.08	34	0.15	34
C202_FW_SZ_123	4	36	38	37.0	15.84	1.14	36	0.31	36
C202_FW_RZ_57	1	213	213	213.0	60.50	3.09	213	1.08	213
C202_FS_SZ_122	1	33	33	33.0	4.01	1.72	33	0.16	33
C202_FS_SZ_121	4	22	24	23.0	2.94	1.58	22	0.19	22
C208_FA_RZ_43	> 20940	8	⩾28	*TO*	78.56	59.30	8	22.30	8
C170_FR_SZ_96	> 99155	53	⩾80	*TO*	328.10	20.15	55	14.00	54
C170_FR_SZ_92	1	131	131	131.0	16.81	1.37	131	0.39	131
C170_FR_RZ_32	32768	227	228	227.9	124.50	6.77	228	0.67	227
C168_FW_UT_851	102	8	16	14.1	62.98	1.75	8	0.57	8

从表 2 可以看出, 贪心遗传算法在运行时间上大大优于分支-限界算法, 通常要快 1 个数量级. 对于绝大多数公式, 贪心遗传算法都能求解出最小不可满足子式, 仅有 2 个公式得到近似最小不可满足子式, 比精确最小子式多 1 个短句. 蚁群算法在运算效率上也显著高于分支-限界算法, 并且对于大多数公式, 能够得到最小不可满足子式, 有 5 个公式比精确解多 1~2 个短句; 但是蚁群算法无论在运行时间还是不可满足子式的大小方面都明显差于贪心遗传算法.

不可满足子式作为一种诊断与定位错误的方式, 其目标是在尽可能短的时间内得到尽可能小的不可满足子式, 即兼顾运算时间与结果大小, 因此单位时间内剔除的短句数是衡量算法优劣的重要标准. 表 3 给出了三种算法的每秒剔除短句数 *NPS* 的实验对比结果, 其计算公式为: $NPS=(N-L)/T$, 其中 N 表示公式所包含的短句数, L 表示不可满足子式的短句数, T 表示算法的运行时间(单位为秒). 表中 vars 列与 clas 列分别给出了每个公式所包含的变元数与短句数; BaBA 列、ACA 列与 CGGA 列分别表示分支-限界算法、蚁群算法与贪心遗传算法每秒删除的短句数.

根据表 3 中的结果可以看出, 对于绝大多数公式, 蚁群算法与贪心遗传算法的 *NPS* 值都要比分支-限界算法高出 1~2 个数量级, 并且贪心遗传算法在单位时间内剔除的短句数上显著优于蚁群算法. 实验结果表明, 虽然贪心遗传算法与蚁群算法在不可满足子式的大小方面牺牲了一点精度, 但在运行时间上取得了很大的优势, 同时兼顾算法效率与结果质量. 因此在单位

时间内剔除的短句数方面要远远高于分支-限界算法.

表 3 三种算法的单位时间移除短句数

Benchmarks	vars	clas	BaBA	ACA	CGGA
C220_FV_SZ_65	1728	4496	219.3	1580.6	3696.7
C220_FV_SZ_46	1728	4498	55.5	272.4	367.3
C220_FV_RZ_14	1728	4508	127.7	2978.1	29980.0
C220_FV_RZ_13	1728	4509	128.6	3749.2	10462.8
C220_FV_RZ_12	1728	4512	85.9	1636.7	7501.7
C210_FW_RZ_59	1789	7394	123.4	915.9	8150.6
C210_FS_RZ_40	1755	5752	146.9	720.4	8133.3
C208_FA_UT_3255	1876	7337	75.5	452.7	7153.9
C208_FA_UT_3254	1876	7334	74.7	528.6	8104.4
C208_FA_SZ_87	1608	5299	311.6	1386.4	7545.7
C208_FA_SZ_120	1608	5278	1272.8	4855.6	34960.0
C202_FW_SZ_123	1799	8686	546.1	7587.7	27903.2
C202_FW_RZ_57	1799	8685	140.0	2741.7	7844.4
C202_FS_SZ_122	1750	6179	1532.7	3573.3	38412.5
C202_FS_SZ_121	1750	6181	2094.9	3898.1	32415.8
C208_FA_RZ_43	1608	5297	67.3	89.2	237.2
C170_FR_SZ_96	1659	4955	14.9	243.4	350.2
C170_FR_SZ_92	1659	5082	294.5	3613.9	12694.9
C170_FR_RZ_32	1659	4956	38.0	698.7	7058.2
C168_FW_UT_851	1909	7491	118.8	4276.0	13128.1

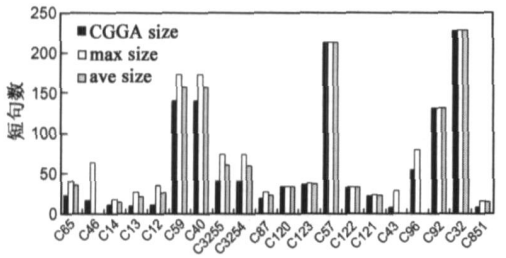

图1 最小不可满足子式与全部极小可满足子式的最大值与平均值的对比

对于测试集中的每个公式,图 1 给出了 CGGA 算法求解的最小不可满足子式包含的短句数与最大的极小不可满足子式以及所有极小不可满足子式的平均短句数的对比. 由于空间限制,因此图中所有公式的命名都以“首字母+ 最后的数字”的方式. 对于测试集中的 3 个公式: C220_FV_SZ_46、C208_FA_RZ_43 与 C170_FR_SZ_96, 在设定的时限内没有求得全部极小不可满足子式,因此其最大值只给出了当前已得到的最大的极小不可满足子式的短句数,而平均值则在图中没有给出.

图中有三个公式只包含 1 个极小不可满足子式,分别为 C202_FW_RZ_57、C202_FS_SZ_122、C170_FR_SZ_92, 因此其最小值、最大值与平均值都相同. 然而对于大多数公式而言,最小不可满足子式显著小于最大的极小不可满足子式, 以及全部极小不可满足子式的平均短句数, 甚至很多公式的最小不可满足子式所包含的短句数只有最大的极小不可满足子式的 1/2 甚至 1/3. 所以, 相对于极小不可满足子式, 最小不可满足子式能够给出关于公式不可满足更加精炼的解释, 更有利于准确地诊断与定位错误.

6 结束语

本文提出了布尔公式的最小不可满足子式与极大可满足子式之间的关系, 并给出证明. 基于二者的关系, 提出了贪心遗传算法与蚁群算法来求解最小布尔不可满足子式. 实验结果表明, 贪心遗传算法与蚁群算法在运算效率以及单位时间内剔除的短句数方面, 显著高于分支-限界算法, 而贪心遗传算法无论在运行时间还是不可满足子式的大小上都优于蚁群算法; 对于绝大多数布尔公式, 两种算法都能求解出最小不可满足子式.

参考文献:

[1] Nam GJ, Aloul F, Sakallah K, et al. A comparative study of two Boolean formulations of FPGA detailed routing constraints[A]. In Proc. of the 2001 International Symposium on Physical Design[C]. Sonoma County: ACM Press, 2001. 222-227.

[2] Suelflow A, Fey G, Bloem R, et al. Using unsatisfiable cores to debug multiple design errors[A]. In Proc. of the 18th ACM Great Lakes symposium on VLSI[C]. Orlando, 2008. 77-82.

[3] Jain H, Kroening D. Word level predicate abstraction and refinement for verifying RTL Verilog[A]. In Proc of the 42nd Design Automation Conference[C]. Anaheim, 2005. 445-450.

[4] Lynce I, Marques-Silva J. On computing minimum unsatisfiable cores[A]. In Proc of the 7th International Conf. on Theory and Applications of Satisfiability Testing[C]. LNCS 3542, Heidelberg: Springer-Verlag 2004. 305-310.

[5] Zhang L, Malik S. Extracting small unsatisfiable cores from unsatisfiable Boolean formula[A]. In Proc. of the 6th International Conf. on Theory and Applications of Satisfiability Testing[C]. 2003.

[6] Bruni R. Approximating minimal unsatisfiable subformulae by means of adaptive core search[J]. Discrete Applied Mathematics, 2003, 130(2): 85-100.

[7] Oh Y, Mneimneh MN, Z. Andraus S, et al. AMUSE: a minimally-unsatisfiable subformula extractor[A]. In Proc. of the 41st Design Automation Conf. [C]. San Diego: ACM Press, 2004. 518-523.

[8] 邵明, 李光辉, 李晓维. 极小布尔不可满足子式的提取算法[J]. 计算机辅助设计与图形学报, 2004, 16(11): 1542-1546.
Shao Ming, Li Guanghui, Li Xiaowei. Algorithms for extracting minimal unsatisfiable Boolean sub-formula[J]. Journal of Computer Aided Design and Computer Graphics, 2004, 16(11): 1542-1546. (in Chinese)

[9] Gershman R, Koifman M, Strichman O. Deriving small unsatisfiable cores with dominator[A]. Proc. of the 18th International. Conf. on Computer Aided Verification[C]. LNCS 4144, Heidelberg: Springer-Verlag, 2006. 109-122.

[10] Dershowitz N, Hanna Z, Nadel A. A scalable algorithm for minimal unsatisfiable core extraction[A]. In Proc. of the 9th International Conf. on Theory and Applications of Satisfiability Testing[C]. LNCS 4121, Heidelberg: Springer-Verlag, 2006. 36-41.

[11] Gregoire E, Mazure B, Piette C. Boosting a complete technique to find MSS and MUS thanks to a local search oracle [A]. In Proc. of 2007 International Joint Conf. of Artificial Intelligence[C]. Hyderabad, India, 2007. 2300-2305.

[12] Liffiton MH, Sakallah KA. Algorithms for computing minimal unsatisfiable subsets of constraints[J]. Journal of Automated Reasoning. 2008, 40: 1-30.

[13] Maaren H, Wieringa S. Finding guaranteed MUSes fast[A]. In Proc. of 11th International Conf. on Theory and Applications of Satisfiability Testing[C]. 2008: 291-304.

[14] Mneimneh MN, Lynce I, Andraus ZS, et al. A branch and bound algorithm for extracting smallest minimal unsatisfiable formulas[A]. Proc. of the 8th International Conf. on Theory and Applications of Satisfiability Testing[C]. LNCS 3569, Heidelberg: Springer-Verlag, 2005. 393-399.

[15] 郑伟, 刘文耀, 王涌天. 一种结合遗传算法和钻石搜索的多模式快速运动估计方法[J]. 电子学报, 2006, 34(10): 1911-1916
Zheng Wei, Liu Wenyao, Wang Yongtian. A fast multi-mode search algorithm combining genetic algorithm with diamond search in video coding[J]. Acta Electronica Sinica, 2006, 34(10): 1911-1916. (in Chinese)

[16] 周兰凤, 洪炳熔. 用基于知识的遗传算法实现移动机器人路径规划[J]. 电子学报, 2006, 34(5): 911-914.
Zhou Lanfeng, Hong Binrong. A knowledge based genetic algorithm for path planning of a mobile robot [J]. Acta Electronica Sinica, 2006, 34(5): 911-914. (in Chinese)

[17] Een N, Sorensson N. An extensible SAT-solver[A]. In Proc. of the 6th International Conf. on Theory and Applications of Satisfiability Testing[C]. LNCS 2919, Heidelberg: Springer-Verlag, 2003. 502-518.

[18] 熊志辉, 李思昆, 陈吉华. 具有初始信息素的蚂蚁寻优软硬件划分算法[J]. 计算机研究与发展, 2005, 42(12): 2176-2183.
Xiong Zhihui, Li Sikun, Chen Jihua. Hardware/software partitioning based on ant optimization with initial pheromone[J]. Journal of Computer Research and Development, 2005, 42(12): 2176-2183. (in Chinese)

[19] 孙岩, 马华东, 刘亮. 一种基于蚁群优化的多媒体传感器网络服务感知路由算法[J]. 电子学报, 2007, 35(4): 705-711.
Sun Yan, Ma Huadong, Liu Liang. An ant colony optimization based service aware routing algorithm for multimedia sensor networks[J]. Acta Electronica Sinica, 2007, 35(4): 705-711. (in Chinese)

[20] Sinz C. SAT benchmarks from Automotive Product Configuration [oL], http://www-sr.informatik.uni-tuebingen.de/~sinz/DC/.

作者简介:

张建民　男, 1979 年生于山西晋中. 分别于 2001 年与 2003 年在国防科技大学获工学学士与工学硕士学位, 现为博士研究生, 从事 VLSI 形式化验证方面的有关研究.
E-mail: jmzhang@nudt.edu.cn

沈胜宇　男, 1975 年生于广西南宁. 分别于 1997 年、2000 年、2005 年在国防科技大学获工学学士、工学硕士和工学博士学位, 现为副研究员, 从事 VLSI 形式化验证方面的有关研究.

李思昆　男, 教授, 博士生导师. 1941 年生于山东青岛. 主要从事电子设计自动化与 SoC 设计方法学等方面的研究工作.