分类号_____ 密级_____

UDC [注1]_____

学 位 论 文

基于 DPLL 的 SAT 算法的研究及应用

陈稳

指导教师姓名 闫炜

电子科技大学 成都

申请学位级别 硕士 专业名称 计算机软件与理论

论文提交日期 2011.3 论文答辩日期 2011.5

学位授予单位和日期 电子科技大学

答辩委员会主席_____

评阅人_____

年 月

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：_____ 日期：　　年　　月　　日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：_____　　导师签名：_____

日期：　　年　　月　　日

摘　要

命题逻辑公式的可满足性问题（SAT）是数理逻辑、计算机科学、集成电路设计与验证和人工智能等领域中的核心问题，并且是第一个被证明出来的 NP 问题。SAT 问题在计算复杂性理论中具有非常重要的地位，设计并实现解决该类问题的高效算法意义重大。但目前不存在一种求解算法在最坏情况下的时间复杂度是多项式级别，其求解速度仍是制约 SAT 算法发展的一大难题。因此，世界各国的学者都在努力研究新的求解算法，以寻求出一种高效的求解算法。

SAT 算法分为完备算法和局部搜索算法这两类。其中，完备算法能给出命题逻辑公式的解或证明公式是不可满足的，但是效率较低；而局部搜索算法求解速度相对较快，但是不一定能够找到对应 SAT 问题的解。本文在对比分析这两类算法的基础上，对 DPLL 算法进行了改进，并通过实验证明了改进后算法的优越性。

基于上述思想，本文的研究工作主要包括：

（1）深入分析了基于 DPLL 的完备性 SAT 算法的实现原理，给出了算法思想和详细的实现过程，并对该算法中所使用到的数据结构和一些关键技术（加速搜索的启发式策略、子句删除机制、随机重启动机制）给予总结和分析。

（2）结合完备算法能够进行完备求解和局部搜索算法能够以较快速度进行求解的优点，在 DPLL 这一基本算法框架之上提出了一种混合的 SAT 求解器，将局部搜索算法 WALKSAT 作为一个必要步骤融入到 DPLL 算法的求解过程中。一方面，WALKSAT 算法得出的近似解可以指导 DPLL 对决策变量进行赋值，从而减小对决策变量赋值的随机性和盲目性；另一方面，每当顶层变量的赋值确定后，都引入 WALKSAT 对算法进行加速，从而提高求解效率。

（3）改进 DPLL 算法的整体框架和实现，给出了改进后算法的框架和实现步骤，并选取一些难的随机 SAT 实例对改进前后的算法进行了测试，实验结果表明，改进后的算法对于求解可满足的随机 3-SAT 实例效果显著。

（4）将改进后的算法应用到图论中的实际问题--四色问题。问题通过编码转换为 CNF 公式的形式后，输入到改进后的 DPLL 求解器中进行求解，并最终转换为对应的着色方案。实验表明，算法在极短的时间内就得出了四色问题的解。

关键词：可满足性问题，可满足性算法，完备算法，局部搜索算法， DPLL

ABSTRACT

The propositional satisfiability problem (also know as SAT problem) is the kernel problem in the field of mathematical logic, computer science, design and verification of integrated circuit, and artificial intelligence. It is the first proved NP problem. The satisfiability problem occupies an important position in the computational complexity theory, so designing and implementing an efficient algorithm to solve this kind of problem plays an important role in the computer theory and many practical applications. However, up till the present moment, there isn't an algorithm which has polynomial time complexity in the worst case, so the speed of solving SAT problems is still a difficult problem for its development. Therefore, international scholars all work hard at studying new algorithms which can solve a certain kind of problem.

There are two kinds of SAT algorithms: the complete algorithm and the local search algorithm. The complete algorithm can find a solution of the given propositional formula or prove its incompleteness, but needs longer time. While the local search algorithm can find a solution in a relatively short time, yet it may not find a solution. This dissertation improves the DPLL algorithm based on the comparision and analysis of the two kinds of algorithms and proves the advantage of the new algorithm.

Based on the above idea, this dissertation mainly includes:

（1）First the implementation principle of complete SAT algorithm based on DPLL is deeply analyzed. Then it presents the basic idea and steps of implementation of the algorithm. In addition, data structures used in the algorithm and some key techniques such as heuristic stategies for improving search speed, clause deleting technique and rondom restaring technique are analyzed.

（2）Combing the advantages of both complete algorithms and local search algorithms, this dissertation proposes a hybrid SAT solver. Based on the summary and analysis of the complete algorithm and the local search algorithm, WALKSAT algorithm is integrated into the process of DPLL algorithm. On one hand, the result of WALKSAT algorithm will guide the assignment of decision variable in DPLL, eliminate the randomness and blindness of variable assignment and improve the efficiency of

algorithms. On the other hand, whenever the assignment of the top variable is decided, the WALKSAT module will be called to speed DPLL algorithm and improve the solver's efficiency.

（3）This paper improves the framework and implementation of the DPLL algorithm and presents the new algorithm framework and steps of implementation. Then it selects some random hard SAT instances to test and compare the two kinds of algorithms. Experiments show that the new algorithm performs better than the previous algorithm for satisfiable random 3-SAT instances.

（4）The improved algorithm is applied to a practical problem of the graph theory—the four color problem. The practical problem is first converted into conjunctive normal form, and then is input into the DPLL solver to find a solution. At last, the solution is transformed into a color scheme. Experiment shows that it finds a solution in a very short time.

Keywords: Propositional Satisfiability Problem, Satisfiability Algorithm, Complete Algorithm, Local Search Algorithm, DPLL

目 录

图目录

表目录

缩略语说明表

SAT	Satisfiablility Problem	可满足性问题
CNF	Conjunctive Normal Form	合取范式
NP	Non-Deterministic Polynomial	多项式复杂程度的非确定性问题
DPLL	Davis Putnam Logemann Loveland	DPLL 算法
BCP	Boolean Constraint Propogation	布尔约束传播
VSIDS	Variable State Independent Decaying Sum	独立变量状态衰减和
MOM	Maximum Occurences in clauses of Minimum	最短子句出现频率最大优先
IG	Implication Graph	蕴涵图
HT	Head/Tail list	头尾链表数据结构
WL	Watched Literals	观察文字数据结构
htLS	Head/Tail Lists with Literal Sifting	带文字过滤的 H/T 链表
WLS	Watched Literals with Literal Sifting	带文字过滤的观察文字结构

第一章 绪论

可满足性问题（Satisfiability Problem）即 SAT 问题，是对一个以合取范式（Conjunctive Normal Form，常简称 CNF）的形式给出的命题逻辑公式进行判断，以找出是否存在一个真值指派，使得该命题逻辑公式的值为真。SAT 问题看似简单，但它却是计算机领域和人工智能领域所要研究的中心问题，被称为理论计算机科学和数理逻辑中的第一问题，在硬件验证、人工智能、电子设计自动化、自动化推理、组合等式检测等领域具有非常重要的理论和实践意义。

1.1 课题研究背景及出发点

从 1960 年至今，SAT 问题一直备受人们的关注，世界各国的研究人员在这方面都做了大量的工作，提出了许多求解算法。每年可满足性理论和应用方面的国际会议都会组织一次 SAT 竞赛以求找到一组最快的 SAT 求解器，而且会详细展示一系列的高效求解器的性能。2003 年的 SAT 竞赛中，就有 30 多种解决方案针对从成千上万的基准问题中挑选出的一些 SAT 问题实例同台竞争。国内也经常会组织一些 SAT 竞赛及研讨会，这些都促进了 SAT 算法的飞速发展。尽管命题逻辑的可满足性问题理论研究已趋于成熟，但在 SAT 求解器被越来越多地应用到各种实际问题领域的今天，探寻解决 SAT 问题的高效算法仍然是一个吸引人并且极具挑战性的研究方向。

1.2 SAT 问题研究意义

可满足性问题是计算机科学领域和人工智能等领域中的重要研究对象[1]。但是其求解算法的时间开销和空间开销却异常惊人。对于一个含有 n 个命题逻辑变量的合取范式来说，如果使用穷举法来罗列所有真值指派进行求解，虽然在理论上是可行的，但算法的时间复杂度却是指数级规模，为 $O(2^n)$，计算机如果采用这种方式进行求解将负担不起如此大的开销。搜索空间如此庞大，使得计算机在可等待的时间里不能计算出结果，进而产生组合爆炸问题，所花费的计算时间是人们不

能容忍的。S.A.Cook 于 1971 年首次证明了布尔表达式的可满足性问题属于 NP 完全问题。NP 完全问题排在七大数学难题之首，在计算复杂性理论中具有非常重要的地位，一方面因为它有着极大的理论价值并且非常难解，另一方面是一旦被破解以后，在诸多的工程领域里还可以得到广泛的应用。但是在 $P \neq NP$ 的假设条件下，SAT 问题不可能有多项式时间的求解算法。SAT 问题已经成为了一类问题的难度标准，所以称其为 NP 完全问题的种子。

现实生产和生活中存在着大量的 NP 完全问题，寻找高效快速的算法来解决该类问题是计算机理论研究和实际应用领域中的重要工作。由于 SAT 问题是 NP 完全问题，它如果能够得到高效解决，那么一定可以高效地解决所有其它 NP 完全问题，这是因为所有的 NP 完全问题都能在多项式时间内进行相互转化，即所有的 NP 完全问题都能够在多项式时间内转换为可满足性问题。所以，如果能找到求解 SAT 问题的有效算法，那么所有其它的 NP 完全问题也都可以解决了，设计和实现更高效的求解算法意义重大[2]。

研究能够解决 SAT 问题的高效算法在理论上有着重要的意义。

SAT 问题可以归结为两类，一类是存在一个可满足的真值指派使得以 CNF 形式表示的布尔逻辑公式取值为真，另一类是在任何真值指派下公式都不能取值为真。人们已经进行了诸多研究以寻求更高效、适用性更广的 SAT 求解器。但是从上述两类 SAT 问题的实例解决情况来看，尽管许多的 SAT 实例能够在占用比较合理的计算机资源的条件下得以解决，但是由于 SAT 问题归根到底仍属于 NP 完全问题，并不存在一种完美的 SAT 求解方法能够适用于所有类型的 SAT 实例，解决许多的 SAT 实例还是非常困难。

SAT 问题的应用领域非常广泛，例如在数学研究和应用领域，它能用来解决旅行商（Traveling Salesman Problem，TSP）和逻辑算术问题；在计算机和人工智能（Artificial Intelligence）领域中，它能解决 CSP（约束满足问题）问题、语义信息的处理和逻辑编程等问题；在计算机辅助设计领域中，它能很好的解决任务规划与设计、三维物体识别等问题[2]。

许多的实际问题如人工智能、积木世界规划问题、数据库检索、Job shop 排工问题、超大规模集成电路设计和图着色都可转换为 SAT 问题进行求解[2]。例如在维护知识库时，当把知识表示成合取范式之后，知识库的一致性检查问题便转化为 CNF 公式上的可满足性问题。谓词逻辑公式的不可满足性问题经过一定的消减技术便可以转化为 CNF 公式的不可满足性问题。不仅如此，一些实际问题通常还具有一定的结构特性，可以据此来对具有某一类结构特性的实际应用问题进行求

解，进而得到解决该类问题的一种通用解法。

1.3 可满足性问题的研究现状及挑战

由于 SAT 问题在计算机理论研究和人工智能领域的重要作用，使得许多学者都在 SAT 问题求解领域做了大量的研究，可满足性问题进而也成为了国内外研究的热点问题，并在算法研究和技术实现上取得了较大的突破，这也推动了形式验证和人工智能等领域的发展。

Bart Selman 和 Henry Kautz 分别于 1997 年和 2003 年在人工智能第五届国际合作会议上提出了 SAT 问题面临的十大挑战性问题[1]，并在 2001 年和 2007 年先后对当时的可满足性问题现状进行了全面的阐述和总结。这十大挑战性问题的提出对于 SAT 基准问题的理论研究和算法改进都起到了强有力的推动作用。

SAT 解决器的实现是我们关心的主要问题，目前的 SAT 算法大致可以归结为两大类：完备算法（也称回溯搜索算法）和局部搜索算法。其中，完备算法大都是基于回溯搜索的，局部搜索算法是基于局部随机搜索的。完备算法基于穷举法思想，它的优点是能保证找到对应 SAT 问题的解或证明公式不可满足，但是效率极低，它的平均时间复杂度虽是多项式级的，但是最坏情况下的时间复杂度却是指数级的。一些完备算法采用了精巧的技术来减小搜索空间和问题规模，提高了算法的时间效率，这一类算法将是本文研究的重点。局部搜索算法相对于完备算法而言，由于采用了启发式策略来指导搜索，使得求解速度相对较快，但是在某些实例上可能得不到解，它不保证一定能够找到对应 SAT 问题的解，即它不能证明 SAT 问题的不可满足性。局部搜索算法的研究热潮是在最近几年才兴起的，本文没有重点研究这类算法，而只是将其作为一个必要步骤应用到了完备算法的求解过程中。

最经典的求解 SAT 问题的完备算法是 DPLL 算法，它是由 Davis 和 Putnam 等人在 1960 年提出[3]，其它的完备算法大都是在 DPLL 算法的基础上衍生出来的，是对 DPLL 算法的改进。由于 SAT 问题本身的特性使得其最坏情况下的时间复杂度是指数级别，最初这使得许多的研究者望而却步。而后，S.A.Cook 在 1971 年证明了 SAT 问题是 NP 完全问题，这更加削弱了许多学者研究 SAT 问题的兴趣，从而导致了 SAT 问题在很长的一段时间里都没有得到较好的重视，发展非常缓慢，研究成果较少。但是 1996 年以后，很多国家都相继举办了一些 SAT 竞赛和研讨会，

这使得越来越多的人开始关注并研究 SAT 问题，所以这段时间也涌现出了众多新的高效的 SAT 算法如 MINISAT[2]、SATO[4]、CHAFF[5]、POSIT[6]和 GRASP[7]等，SAT 算法的研究成果显著，求解算法也越来越多地应用到了实际问题领域。这些新兴的算法大都是基于 DPLL 算法的改进算法，改进的方面包括：采用新的数据结构、新的变量决策策略或者新的快速的算法实现方案。国内也涌现出了许多高效的求解算法，如 1998 年作者梁东敏提出了改进的子句加权 WSAT 算法[8]，2000 年金人超和黄文奇提出的并行 Solar 算法[9]，2002 年作者张德富在文献[10]中，提出模拟退火算法。表 1-1 就说明了过去的十几年中 SAT 算法的发展所取得的惊人的变化速度。

表1-1　过去十几年中 SAT 求解速度的进展

问题实例	Posit (1994)	Grasp (1996)	Sato (1998)	Chaff (2001)	Siege (2003)
ssa2670-136	40.66s	1.2s	0.95s	0.02s	0.01s
bf1335-638	1805.21s	0.11s	0.04s	0.01s	0.01s
pret150-25	>7200s	0.21s	0.09s	0.01s	0.01s
dubois100	>7200s	11.85s	0.08s	0.01s	0.01s
aim200-2_0-no-1	>7200s	0.01s	0s	0s	0.01s
2dlx_cc_mc_ex_bp_f2_bug005	>7200s	>7200s	>7200s	2.9s	0.22s
c6288	>7200s	>7200s	>7200s	>7200s	6676.17s

尽管 SAT 算法已经取得了举足轻重的改进，但是仍有一些问题没有得到高效的解决，已经解决的问题可能还存在更好的求解算法，因此研究并实现高效率的求解算法仍是当前要解决的中心问题之一。

在人们追求高效率的求解算法的时候，却忽视了算法的正确性和完备性，这推动着我们在探寻更高效的 SAT 求解算法的道路上不断前进，对 SAT 问题研究的脚步不会停止。与此同时，我们还能提出一些值得探讨的问题：

1) 能否找到一种更有效的预处理技术或者其它搜索技术并将其融入到基于 DPLL 的算法中？
2) 哪种算法可以解决绝大多数的 SAT 实例？
3) 是否能够将局部搜索同 DPLL 算法集成为一种兼备两种算法优点的高效算法？

4) 能否找到其它类型的更好的 SAT 解决方案？

1.4 主要研究内容

本文主要学习了命题逻辑可满足性问题的相关理论知识，对比分析了求解 SAT 问题的完备算法和局部搜索算法这两类算法，并对基于 DPLL 的求解器所采用的关键技术及其算法框架进行了研究。

结合完备算法能够进行完备求解和局部搜索算法能够以较快速度进行求解的优点，在分析 DPLL 算法框架的基础上，提出了一种将 WALKSAT 算法融入到 DPLL 求解过程并指导其对决策变量进行赋值的混合求解算法。实验结果表明，该算法求解较快，比之前单纯的 DPLL 算法优秀。引入该策略的出发点是：原始的 DPLL 算法在确定好决策变量时，对变量赋以何种极性的值往往是随机决定的，带有一定的盲目性，并且计算代价较高。对给定的 CNF 公式运用 WALKSAT 算法进行局部搜索，就可以先得到问题的一个近似解向量，由于近似解向量能使大部分子句都是可满足的，因此可以据此来指导 DPLL 算法对所选择的决策变量进行赋值，这样可以避免盲目的对决策变量进行赋值，并且可以减少算法的决策次数和冲突次数，从而加速搜索过程[11]。并且，每当有顶层变量的赋值确定以后，就可以引入 WALKSAT 对缩小了的 CNF 公式进行求解，从而提高求解效率。

具体来说，本文的主要研究内容如下：

(1) 对 SAT 问题的研究背景、意义及现状进行了简要总结，学习了命题逻辑可满足性问题的基本理论知识。

(2) 对完备算法和局部搜索算法这两类算法进行了对比分析，得出各自的优劣，通过对比说明本文将重点介绍的基于 DPLL 算法的研究价值和意义。

(3) 对基于 DPLL 的 SAT 算法进行了深入研究，并仔细分析了 DPLL 算法的实现，对该算法中所使用到的数据结构和一些关键技术给予总结和分析。

(4) 改进 DPLL 算法的整体框架和实现，给出了改进后算法的框架和实现步骤，并选取一些难的随机 SAT 实例对改进前后的算法进行了测试，实验结果表明，改进后的算法对于求解可满足的随机 3-SAT 实例效果显著。

(5) 将改进后的算法应用到图论中的实际问题--四色问题。挖掘四色问题中存在的一些潜在的约束条件，通过编码转换为 CNF 公式的形式后，输入到改进后的 DPLL 求解器中进行求解，并最终转换为对应的着色方案。实验表明，算法在极短

的时间内就得出了四色问题的解。

1.5 本文结构及内容安排

本文重点对基于DPLL的SAT算法的设计与实现过程进行了研究。首先对SAT相关的一些基本概念进行了介绍，然后对完备算法和局部搜索算法这两类算法进行了比较分析，而后阐述了基于 DPLL 的算法所采用的数据结构和关键技术，最后基于 DPLL 的算法框架设计并实现了一种混合高效的 SAT 求解器，并以随机3-SAT实例作为测试用例对算法进行了仿真，给出了运行结果，并将其应用到了一个实际问题领域中。本文共分为6章，具体内容如下：

第一章 绪论 主要介绍了命题逻辑可满足性问题的研究背景、现状和研究意义，阐述了SAT问题的主要研究内容，给出了全文的内容纲要。

第二章 SAT基本知识及相关算法 着重介绍了SAT问题的相关理论基础和公式的化简策略。并对完备算法和局部搜索这两类算法进行了对比分析，总结各自的性能优势和不足。

第三章 基于DPLL的完备性SAT算法研究 主要对完备算法所采用的数据结构和加速搜索的一些关键技术点进行了深入研究，包括BCP过程、变量决策策略、冲突分析和子句学习机制等，并给出了基于DPLL的算法发展史及其算法框架。

第四章 WM算法——改进后的DPLL算法 对改进的算法思想进行介绍，并通过仿真环境进行测试，同之前的DPLL算法进行性能对比。

第五章 WM 算法的具体应用 将改进后的算法应用到图论中的四色问题，主要是对四色问题进行编码求解得出一组着色方案。

第六章 总结和展望 对全文工作进行总结，并对后续研究工作进行了分析和展望。

第二章 SAT 基本知识及相关算法

本章将介绍一些 SAT 算法的研究所涉及的相关基本理论知识和相关求解算法。

2.1 算法的复杂度

在计算机领域中，算法复杂度这一概念非常重要。有两种类型的算法复杂度：时间复杂度（即运行算法所需的时间资源）和空间复杂度[12]（运行算法所需的空间资源，即常说的存储器资源）。

时间复杂度是指使用计算机来完成一个算法所需要消耗的时间，它是衡量一个算法优劣性的重要参数。时间复杂度越小，表明该算法的求解效率越高，那么该算法越具有实际应用价值。

空间复杂度是指使用计算机来完成一个算法所需要占用的存储空间，它一般是输入参数的函数。它是衡量算法优劣的重要指标，一般而言，算法的空间复杂度越小，说明算法性能越好。

算法的复杂度能较好的反应算法的求解性能，它是依赖于算法的输入、将要求解的问题规模和算法本身的函数。对于任意一个给定的问题，设计出具有较低时间复杂度和空间复杂度的算法是在进行算法设计时需要参考的一个重要准则。如果已经存在多种算法来求解给定的问题，我们应选择复杂度最低者，这一原则在我们选择算法时非常重要。因此，分析算法的时间复杂度对算法的设计或选用意义重大。

如果一个算法的时间复杂度是 $O(n^m)$，其中 m 是正整数，那么就称该算法是多项式时间有界的。一个多项式时间有效的算法也称是有效的算法，这个算法就是我们所期待的解决问题的高效算法。

2.2 P 类与 NP 类问题

绝大部分复杂度理论研究的基本都是判定问题。判定问题是指提出一个问题，

仅需要回答是/否的问题。

例如：求某一有向图中从节点 A 到节点 B 的最短路径，该问题可以转化为如下判定问题的形式：

从节点 A 到节点 B 是否存在长度为 1 的最短路径？　　否

从节点 A 到节点 B 是否存在长度为 2 的最短路径？　　否

从节点 A 到节点 B 是否存在长度为 m-1 的最短路径？　　否

从节点 A 到节点 B 是否存在长度为 m 的最短路径？　　是

如果询问到 m 的时候，回答为是，则停止询问。那么就可以说从节点 A 到节点 B 的最短路径长度为 m。

P 类问题和 NP 类问题都属于判定问题，只是其计算复杂度有所不同。

(1)P 类问题

P 类问题是指可以用一个确定性算法在多项式时间内得到判定或者解决的问题，它是所有复杂度为多项式时间的问题的集合。此类问题即使在最坏情况下也可以进行高效的求解。

(2)NP 类问题

NP 类问题是指可以用一个确定算法在多项式时间内检查或者验证出解的问题，即这些问题不能用一个确定性算法在多项式时间内得到确定性的解。它不要求给出一个求解问题的算法，而仅要求给出一个能在多项式时间内验证该问题的解的确定性算法即可。

(3)NP 完全问题

该问题是 NP 问题中判定问题的一个子类。如果一个 NP 类问题的所有可能答案都能够在多项式时间内进行正确与否验证的话，那么该问题就称为完全多项式非确定性问题，简称 NP 完全问题（或 NPC 问题）。

P 是否等于 NP 是当前计算机科学领域中最突出的问题之一，该问题在千禧年七大难题中排在首位，但是人们目前既不能证明它也不能推翻它，于是 SAT 问题成为了计算机科学领域中的一大开放性问题。

所有的 NP 完全问题在多项式时间内都可以进行相互转化，因此所有的 NP 完全问题在多项式时间内都能转化为 SAT 问题。如果能够找到求解 SAT 问题的有效解法，那么所有其它的 NP 问题都将得到较好的解决，因此研究 SAT 问题意义重大。

2.3 布尔表达式（布尔代数）

布尔表达式是 SAT 问题研究的基础，本节将介绍一些与本文工作相关的布尔表达式内容。

布尔代数在数理逻辑中属于命题逻辑范畴（或称布尔逻辑，或称 0 阶逻辑），布尔表达式是由 0，1 以及布尔变量、布尔运算符构成的符号串，与逻辑公式定义一致。布尔变量的取值只能是 0 或 1。基本的逻辑运算符包括：与（$\wedge$），或（$\vee$），非（$\neg$）。其中“$\neg$”是单目运算符，“$\wedge$”和“$\vee$”是双目运算符。

定义 2.1：设有布尔代数$(B;\wedge,\vee,\neg)$及其上的n个变元$x_1,x_2,\cdots,x_n$，则布尔表达式(Boolean Expression)可归纳地定义为：

（1）0，1 和$x_1,x_2,\cdots,x_n$均是布尔表达式；

（2）如果a_1和a_2是布尔表达式，则$a_1\wedge a_2$，$a_1\vee a_2$，$\neg a_1$也都是布尔表达式；

（3）除此之外再没有其它的字符串是布尔表达式

定义在布尔集合上的映射称为布尔函数。假设 A 是一个布尔集合 A={0，1}，那么一个由n个布尔变量构成的布尔函数就是一个映射：$A^n\to A$。

2.4 范式相关概念

范式是具有标准形式的公式。在命题逻辑中有两种范式：析取范式和合取范式。

定义 2.2 单式

称原子公式和原子公式的否定为单式。

定义 2.3 析取子式

称以单式作为析取项的析取式为析取子式。

定义 2.4 合取子式

称以单式作为合取项的合取式为合取子式。

定义 2.5 合取范式（Conjunctive Normal Form）

称以析取子式为合取项的合取式为合取范式（简称 CNF）。具体表述如下：

已知 B 是一个命题公式，B 中所出现的命题变元为$X_1,X_2,\ldots,X_n$，以Y_i来表示X_i或$\neg X_i$，$i=1,2,\ldots,n$，则$Y_1\vee Y_2\vee\cdots\vee Y_n$是$X_1,X_2,\ldots,X_n$的一个析取项，由一些不同的析取项的合取式组成的公式就称为合取范式。

表示为一般的形式如下：

$$(L_{i1} \vee L_{i2} \vee \cdots \vee L_{in_1}) \wedge \cdots \wedge (L_{k1} \vee L_{k2} \vee \cdots \vee L_{kn_k})$$

其中，L_{ij} 表示任意文字。

2.5 SAT 问题的基本定义和性质

由命题逻辑的基本知识可知，任何命题逻辑公式都逻辑等价于某一 CNF 公式，因此，本文仅考虑 CNF 公式的可满足问题。下面是一些描述 SAT 问题时将要用到的术语。

设$U=\{y_1, y_2, \cdots y_n\}$表示一个布尔变量集合，其中 y_i 表示命题变量，用 0 和 1 来表示命题的真值和假值。

定义 2.6：文字、子句、子句长度

若 y_i 是U 的一个变量，那么形式符号 y_i 和 $\overline{y_i}$ 是U 上的文字，$i=0,1,2,\cdots n$，其中 y_i 为正文字，$\overline{y_i}$ 为负文字。如果 L 是文字，那么 L 的反面记作 $\neg L$，定义如下：

$$\neg L = \begin{cases} \overline{y_i} & \text{若} L = y_i \\ y_i & \text{若} L = \overline{y_i} \end{cases}$$

子句由一个或多个文字通过逻辑或（$\vee$）连接起来组成，是U 上的一个或多个文字的析取式。

子句长度是指一个子句中所包含的所有文字数目。子句若在某一赋值 u 下取真值，只需使该子句包含的所有文字中至少有一个在赋值 u 下取值为真即可。若某一真值赋值使得该子句取值为真，那么称该子句是可满足的。

SAT 问题中的合取范式（CNF）由一个或多个子句通过逻辑与（$\wedge$）连接起来组成。所有的布尔公式都可以表示为合取范式的形式。将公式表示为 CNF 形式时，如果判断公式是否是可满足的，那么此时只需让每个子句都可满足即可。

定义 2.7：真值赋值（真值指派）

用函数 $t:U \to \{0,1\}$ 表示布尔变量集合U 上的真值指派（1：逻辑真值，0：逻辑假值）。那么，一个 n 元布尔向量就构成了布尔变量集合U 上的一组真值指派，U 上的不同真值指派总数为 2^n 个。若某一布尔变量 $u \in U$ 能使得函数 t 的值为 1（0），则称函数 t 在真值指派 u 下取值为真（假）。

定义 2.8：SAT 问题

SAT：设 S 为布尔变量集合 U 上的子句集，判断是否有一组真值指派，使得 S 中所有子句都是可满足的（即所有子句都取值为真）。若该指派能使得所有子句都是可满足的，那么称 S 是可满足的，否则是不可满足的。该问题看似简单，但求解难度很大，为方便研究实际应用中各种行之有效的算法，一般来说，在求解之前要先对 SAT 问题进行一定的约束，而后再用算法求解。

CNF SAT：设 B 为布尔变量集合 U 上的合取范式，判断是否有一组真值指派使得合取范式 B 中所有子句都是可满足的（即 B 中所有子句的取值都为真）。所有的命题逻辑公式都可以转化为 CNF 公式的形式，因此接下来本文所述的 SAT 问题均指该类问题。

k-CNF SAT：设 B 为布尔变量集合 U 上的合取范式， B 中所有子句包含的文字数都是 k，判断能否找到一组真值指派，使得合取范式 B 中的所有子句都是可满足的。

MAX-SAT（最大可满足问题）：设 B 为布尔变量集合 U 上的合取范式，判断能否找到一组真值指派使得 B 中可满足子句的数目达到最多。

以下是 SAT 问题中对 CNF 公式进行化简时常用的一些化简规则：

（1）单子句规则（单元子句规则）

若一个子句中仅有一个文字未赋值，所有其它的文字赋值都是 0（或者该子句本身只含有一个文字），那么此时可推断出该文字赋值为 1[13]。

（2）纯文字规则

在一个合取范式中，若一个命题变量要么以全是正文字的形式出现，要么以全是负文字的形式出现，那么该文字就称为纯文字。消除这种所有出现即为正或出现即为负的原子公式所在子句的规则就称为纯文字规则[14]。

例如：CNF 公式的子句集

$$\varphi=(\neg p\vee q\vee r\vee\neg t)\wedge(p\vee\neg q\vee\neg r\vee\neg t)\wedge(\neg p\vee\neg q)$$

其中 $\neg t$ 是纯文字，则公式 φ 等价于 $\varphi'=(\neg p\vee\neg q)$

（3）消除原子公式规则

假设给定的合取范式或一部分子式可以表示为以下这种形式：

$$(P\vee u)\wedge(Q\vee\bar{u})\wedge R$$

其中，P、Q、R都是析取子式，u和$\bar{u}$不在其它子句中出现，且P、Q、R都与原子公式u无关，那么该公式可以消减为不含原子公式u的形式：

$$(P \vee Q) \wedge R$$

（4）重言子句规则

如果一个子句是重言子句（一般是子句中存在互补的文字），那么就可以将它从合取范式中删除。

（5）包含规则

如果A_1，A_2是合取范式B的两个子句，且$A_1 \subseteq A_2$，则求合取范式B的可满足性问题时可以直接将子句A_1删去。

（6）分裂规则

设 CNF 公式的子句集为：

$$\varphi = (A_1 \vee r) \wedge \ldots \wedge (A_m \vee r) \wedge (B_1 \vee r) \wedge \ldots \wedge (B_n \vee \neg r) \wedge \ldots \wedge C_1 \wedge \ldots \wedge C_t$$

其中A_i，B_j，C_k均是不含文字r或者$\bar{r}$的子句。此时公式φ可以分裂成两个不含r或者$\bar{r}$的子句集：

$$\varphi_1 = A_1 \wedge \ldots \wedge A_m \wedge C_1 \wedge \ldots \wedge C_t$$
$$\varphi_2 = B_1 \wedge \ldots \wedge B_n \wedge C_1 \wedge \ldots \wedge C_t$$

2.6 SAT 问题相关算法

目前存在两类主流 SAT 算法，他们分别是完备算法（即回溯搜索算法）和局部搜索算法。这两类算法各有千秋，共同推动着高效的 SAT 算法的发展。

2.6.1 完备算法

完备算法基于穷举思想和回溯搜索机制，系统地搜索给定 SAT 问题的解空间，如果该问题是可满足的，则给出命题逻辑公式的所有解，如果公式是不可满足的，则给出完备性证明。事实上，在第二种情况下，给出完备性证明，这一点极其重要，具有实际意义，因为许多不可满足的实例都要求给出无解证明。它一个一个的确定命题逻辑变量的值，直到所有的变量都被赋值并且公式的值为真，或者所

有可能的赋值都试过但都不合适。

完备算法的优点是能保证找到对应 SAT 问题的解或证明公式不可满足，但是效率极低，它的平均时间复杂度虽是多项式级的，但是最坏情况下的时间复杂度却是指数级的。

事实上，完备算法就是对整个解空间进行深度优先搜索的过程，搜索空间非常庞大，这有可能使得计算机在可等待的时间里不能计算出结果，进而产生组合爆炸问题，所花费的计算时间是人们不能容忍的。

由 Davis 和 Putnam 等人提出的 DPLL 算法[3]是最具代表性的完备性算法，之后相继有很多学者都对 DPLL 算法进行了一些改进，采用了一些启发式的搜索策略，如智能回溯策略、搜索的重新启动策略和随机化技术等，提出了一些改进的二代 DPLL 算法，比如：SATO[4]、POSIT[6]、GRASP[7]等，但是这些算法还都没有引入冲突子句学习的技术。这几种算法之后，出现了一些 SAT 新技术的聚集，如冲突子句的分析和学习、回跳、监视字等。随着这些新技术的引入，又诞生了一批更高效的 SAT 算法，如 MINISAT[2]、CHAFF[5]和 BERKMIN[15]。ZCHFF 算法是 CHAFF 算法的一种高效实现方法，它是在 GRASP 算法基础上改进而来，主要优化了 BCP 过程、VSIDS 机制、变量的删除和随机重启动机制，在 2002 年举行的大规模 SAT 竞赛中，该算法取得了第一名的好成绩。

在各式各样的 SAT 求解算法中，完备算法是解决难的、结构化的、真实 SAT 实例最强壮的算法，许多的实验都证明了这一点[5][7] 。

2.6.2 局部搜索算法

局部搜索算法基于随机搜索策略，对于任意给定问题 P，它不一定能判定 P 的可满足性，但对 P 的某些特定子集，它能较快的给出结论，时间复杂度可以是多项式级的。它从最初的猜测给定问题的解，到最后按照一定的搜索策略逐渐地靠近能使问题可满足的最优解。局部搜索算法相对于完备算法而言，由于采用了启发式策略来指导搜索，求解速度比一些完备算法较快[16]。我国的李未、黄文奇等在局部搜索算法中的研究成果比较多，他们引入拟人拟物的思想，提出了一种高效的 Solar 算法[17]来求解 SAT 问题，在 1996 年的 SAT 竞赛中该算法取得了第一名的好成绩。

常用的局部搜索算法是基于贪婪搜索的 GSAT 和基于噪声模型的分析的 WALKSAT[18]。在 GSAT 算法中，变量的翻转很有可能会陷入一个局部圈子，无法

跳出，从而影响求解，而 WALKSAT 算法是在 GSAT 算法的基础上引入随机噪声参数改进而来的，主要目的就是减少其陷入局部圈子的概率。

算法的基本思想如下：

1) 首先为所有变量产生一个完全指派。由于每一变量均已被赋值，那么每个子句或者是可满足的或者是不可满足的（子句中所有的文字均为假）。

2) 减少不可满足子句的数量。例如，在 WALKSAT 算法中，算法的每一步都会在不可满足子句的集合中选择一个子句，然后翻转该子句中某个文字的赋值，再查看子句的可满足情况，不可满足时再选择一个变量翻转其赋值。

3) 重复步骤 2，进行若干次的迭代，不可满足子句的数目可能就会减少，在某些情况下就会减少至 0，即不存在不可满足的子句，那么此时就可以断定公式是可满足的。若仍有子句是不可满足的，那么重复步骤 1，重新为公式选择一个完全指派。

该算法的不完备性就在于它不一定能够找到求解给定合取范式的可满足解，尤其当公式是不可满足的时候，它更不能给出一个完备性的证明。因此，现在对 SAT 算法的研究大都是集中在完备性算法的设计和实现方面或者是如何将局部搜索算法集成到完备性算法中以提高求解效率。

2.6.3 完备算法和局部搜索算法的对比

完备算法和局部搜索算法是主流的两类 SAT 算法，这两类算法各有千秋，共同推动着 SAT 问题的发展。

完备算法的基本思想是：根据命题逻辑公式中的部分赋值向量，运用启发式搜索策略来给一些还没有赋值的变量进行赋值，直到所有的变量均已被赋值。

局部搜索算法的基本思想是：给定一个全局赋值向量，在此基础上来减少不可满足子句的数目，其操作过程所用的启发式信息相对较少，大部分都是随机选择的。两种算法的详细比较如下：

表2-1 完备算法和局部搜索算法的对比

	完备算法(回溯搜索算法)	局部搜索算法
随机性	弱	强
逻辑性	强	弱
全局性信息	少	多
求解速度	慢	有解问题速度较快 无解问题不停机
主要困难	决策变量的选择	局部极值点的跳出

分析表 2-1，不难看出，局部搜索算法可以把全局信息传送给完备算法，从而可以指导决策变量的选择；而完备算法的逻辑性较强，可以给局部搜索算法提供逻辑指导。即完备算法和局部搜索算法这两类算法之间有一定的互补性，可以将二者紧密结合起来，以同时获得这两种算法的求解优势。

2.7 小结

SAT 作为数学与计算机科学领域中的重要问题之一，涉及到的理论基础较多，本章首先对于一些主要的相关知识进行了介绍，例如 N 和 NP 问题、命题逻辑等，给出了 SAT 问题的基本性质和定义。而后立足于可满足性问题的求解算法，对完备算法和局部搜索算法这两类算法进行了单独介绍，阐述了其中的一些优秀的算法思想及其所使用的关键技术，然后对这两类算法进行了优劣性比较分析，得出二者性能差异。

第三章 基于 DPLL 的完备性 SAT 算法研究

过去的几年里，研究人员提出了许多 SAT 算法，从最初的 DP 算法到最近的回溯搜索算法，等等。我们从上文的介绍可以知道，SAT 算法存在局部搜索算法和完备算法两类。在给定的 CPU 时间内完备性算法可以证明公式的不可满足性；而局部搜索算法却不能证明。在各式各样的算法中，完备算法（即回溯搜索算法）是最强壮的 SAT 算法，它能够解决难的、结构化的和实际应用中的问题。大量的实验证据和文献资料[7][18][19]都证实了这一点。

单纯的 DPLL 算法的时间复杂度一定是指数级别的，但是由于许多的改进算法加入了启发式搜索策略，使得算法效率大大提高。但是真正能够将算法从理论水平推向实际问题的应用，能够对大规模的实例进行求解，算法的数据结构则起着至关重要的作用。

3.1 使用的数据结构

本节的主要目的是回顾已有的 SAT 数据结构，以便更好的利用现有的数据结构来解决实际问题或者构造更好的数据结构。另外，我们也介绍了胀缩数据结构在预测动态子句大小中所起的重要作用。（比如子句中未赋值文字的数目）

算法实现的数据结构表示 BCP 和冲突分析实现的方式，在 SAT 求解器的时间性能上起着决定性的作用。主要有两类求解 SAT 问题的数据结构：基于邻接表的数据结构和胀缩数据结构。

3.1.1 邻接表

这是一类传统的数据结构，在算法中应用较多，它能准确地了解子句中每个文字的值，旨在减少搜索树中每个节点上所占用的 CPU 时间。应用该数据结构的算法有：1996 年的 GRASP 算法（Marques-Silva&Sakallah），1997 年的 Relsat 算法（Bayardo Jr. & Schrag）和 1997 年的 Satz 算法（Li & Anbulagan）。

绝大部分完备算法都把子句表示为文字链表的形式，并为每一个文字关联一个子句链表，用来表示该文字在哪些子句中出现过。一般用邻接表来表示 SAT 数

据结构，其中的每一变量都包含一个邻接于该变量的子句的完全链表。

下面的子章节描述了几种邻接表的不同实现方法，但是无论采用哪一种方式都能准确高效地分辨出子句何时变成可满足的、何时变成不可满足的或者何时变为单元子句。

（1）已赋值文字消隐方法

识别可满足子句、不可满足子句和单元子句的方法是：从子句的文字链表中提取出对所有取真值的文字和取假值的文字的引用，然后将这些提取出的引用添加到与该子句对应的关联链表中[20]。那么，在已满足的子句链表中就会包含一个或多个文字引用；在不可满足的文字链表中，不可满足的子句将包含所有的文字引用。而单元子句包含一个未赋值的文字和不可满足文字链表中所有其它的文字引用[21]。

这一数据结构如图 3-1 所示。当某一文字被赋值时，它便会被移至已满足文字链表或者不可满足文字链表。给定的示例中，当仅有一个文字未赋值并且其它两个文字都不可满足时，那么在第三步时三元子句就会被识别为单元子句。

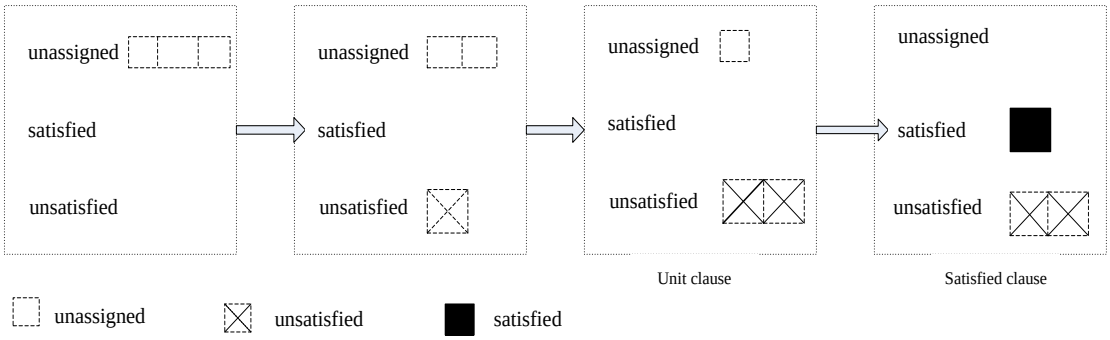

图3-1 已赋值文字消隐法的操作过程

经过实验证明，这种数据结构同其它的结构相比并无多大优势[21]。

（2）基于计数器的方法

追踪不可满足子句、已满足子句和单元子句的另外一种方法是：为每个子句关联一个文字计数器。通过这些文字计数器，可以看出有多少文字是已满足的，有多少是不可满足的，还有多少文字是没有赋值的。若一个子句中不可满足的文字计数等于该子句中的文字数目，那么该子句就是不可满足的。若已满足的文字数目大于等于 1，那么该子句就是可满足的。若取值为 0 的文字数目等于子句长度减 1，那么该子句就是单元子句。当子句变成单元子句的时候，可以遍历文字链表以找到哪一文字需要赋值。文献[7]中，作者提出 GRASP 算法，该算法利用基于计

数器的邻接表数据结构来求解 SAT 问题的很好的例子。

基于计数器的方法可以用图 3-2 来阐述。只要有一个文字被赋值了，那么不论是已满足文字还是未满足的文字计数器都要进行更新，其实这主要取决于该文字被赋值为 1 还是 0。值得注意的是，当某一子句变成单元子句时（即不可满足文字数=该子句中总文字数-1），需要遍历整个子句以找到最后一个未赋值的文字。除此之外，当搜索过程回溯的时候，计数器的值也要同步进行更新。

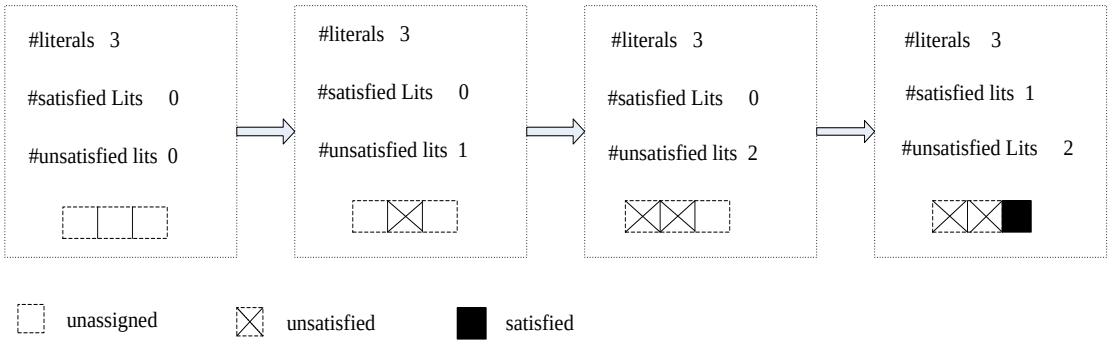

图3-2 基于计数器的求解过程

（3）带已满足子句消隐的基于计数器的方法

使用邻接表结构的一大缺陷是：同每个文字关联的子句链表的长度可能会很大，并且在搜索过程中添加新子句时，它也会随之不断增加。因此，每当有一个变量被赋值时，都需要遍历一个大规模的子句链表。可以使用各种方式来克服这一缺陷。针对上一节所讨论的基于计数器的方法，一种可行的办法是从每个变量的子句链表中删除所有已可满足的子句。因此，每当一个子句变为可满足子句的时候，那么所有包含该子句的文字链表都会将该子句隐藏起来。对子句进行隐藏的目的是为了减少文字赋值时的工作量，这样当文字赋值时，只需解析与该文字关联的尚未解析的子句。

例 1：已知一个合取范式φ有ω_1，ω_2和ω_3这三个子句，其中：

$$\omega_1=(x_1\vee x_2),\omega_2=(x_2\vee\neg x_3),\omega_3=(x_1\vee x_2\vee x_3)$$

执行搜索过程之前，变量x_2与子句ω_1，ω_2和ω_3相关。现假设变量x_1赋值为 1，那么子句ω_1和ω_3变为可满足的，因此这两个子句就会从与变量x_2关联的子句链表中隐藏起来，使得与变量x_2相关联的子句链表中仅剩下子句ω_2。

（4）可满足子句和已赋值文字消隐法

这是邻接表的最后一种组织结构。未满足的文字会从子句的文字链表中删除，

已满足的子句将从文字的子句链表中隐藏起来。使用子句和已赋值文字消隐法是为了减少跟变量赋值的工作量。实验表明，同其它一些简单的基于计数器的方法相比较而言，子句和文字消隐技术并不是特别的高效，下节将要介绍的胀缩数据结构会更高效。

从以上几种邻接表数据结构的分析中可以看出，邻接表数据结构的主要缺陷就是对子句的访问量很大[21]。特别是引入了学习过程之后，与每一文字相关联的子句数目增加也很快，这就降低了布尔约束传播过程的效率。为克服这一缺陷所引入的措施主要是对子句和文字进行消隐的方法，但是这些方法虽在一定程度上提高了布尔约束传播的效率，但却大大增加了回溯过程中的开销，所以总体表现不突出。

3.1.2 胀缩数据结构

分析上面几种数据结构可以知道，基于邻接表的数据结构都有一个共同的缺陷：每一变量对子句的引用规模都可能会很大，而且通常会随着搜索过程的继续不断增大，这就会增加与变量赋值相关联的工作量。另外通常情况下，当对变量 x 进行赋值时，不需要分析 x 所关联的大部分子句，这是由于它们并没有变成单元子句或是不可满足的子句。

为了克服这一弊端，引入了胀缩数据结构，目前在子句学习的回溯搜索算法中，它是最具竞争力的数据结构。当前许多先进的 SAT 求解器都采用该数据结构来求解，比如 1997 年的 SATO 算法就使用了 H/T 结构，2001 年的 Chaff 求解算法就使用了观察文字结构。

该算法中每一变量都有一个缩小的子句引用集合，该结构仅能了解一个子句是否是单元子句或不可满足子句。下面是几种胀缩数据结构的实现机制。图 3-3 详细地描述了这一结构的操作过程。

（1）H/T 链表（Head/Tail）

可满足性问题的算法中，第一个提出的胀缩数据结构就是 H/T 数据结构，它首先用在 SATO 求解器[22]中。这一数据结构为每一子句设置两个引用指针：头指针和尾指针。初始时，头指针指向子句中第一个文字，尾指针则指向子句中最后一个文字。每当这两个引用指针中有一个所指向的文字被赋值时，就会搜索一个新的未赋值文字。一旦找到一个未赋值的文字，那么它就会成为新的头引用指针或尾引用指针，此时会创建一个新的引用指针并与每个文字的变量关联起来。一

旦找到一个可满足文字，那么该子句就声明为已满足的。如果无法找到未赋值文字，并且已到达了另外一个引用，那么子句就会被声明为单元子句、不可满足子句或者已满足子句，这主要取决于指向另一个引用指针指向的文字的值。当搜索进行回溯时，就会丢弃与头指针或者尾指针关联的引用，而之前的头指针或尾指针就会再次激活（用图 3-3 中的虚线箭头表示）。需要注意的是，在最坏情况下，与每个子句关联的文字引用的数目正好等于子句中的文字数目。

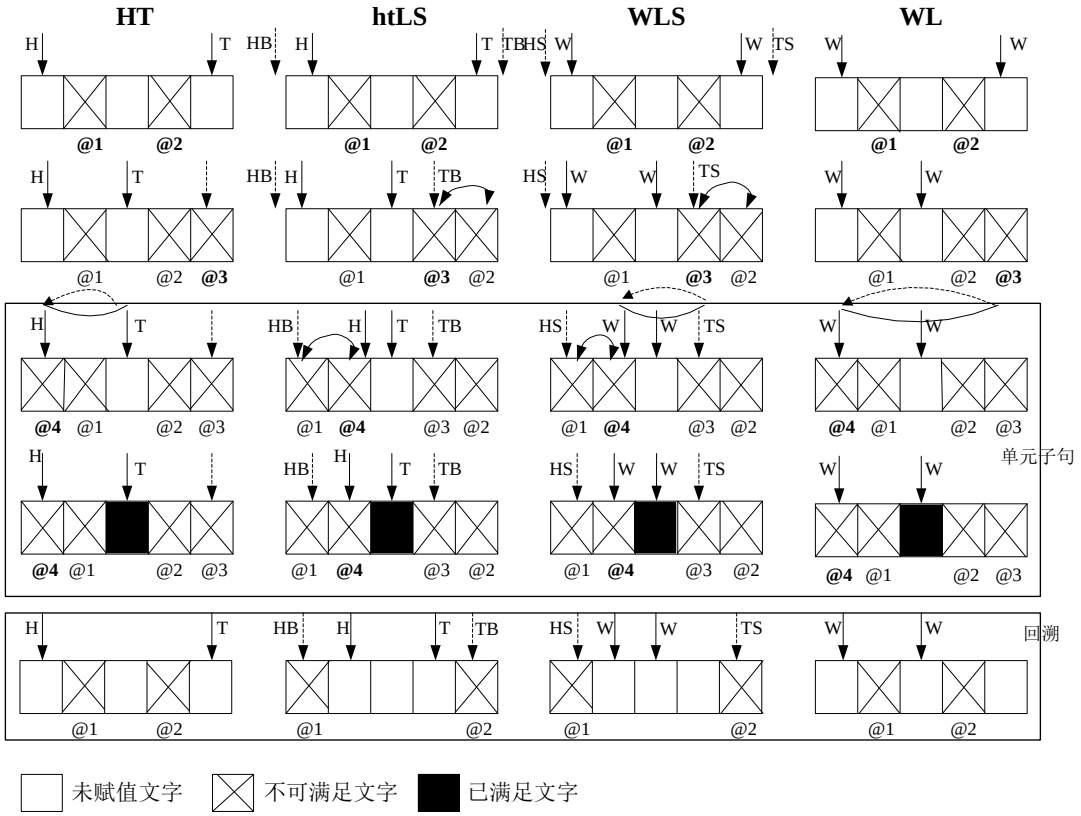

图3-3 各种胀缩数据结构的操作过程

（2）带观察文字的数据结构（Watched Literals，WL）

2001 年的 Chaff 求解算法，采用了一种新的数据结构----观察文字结构，它能解决 H/T 链表结构存在的一些问题。该算法类似 H/T 链表结构，也要为每一子句关联两个引用指针。但与 H/T 链表不同的是，这两个引用指针之间无先后次序关系。这样做的好处就是当算法回溯时，无需更新文字引用指针。但同时存在一个明显的缺点，即只有当遍历完所有子句中的文字后，才能识别出单元子句和未满足子句[23]。对于可满足子句的识别类似于 H/T 链表结构。带观察文字的数据结构操作过程如图 3-3 所示。

对比图 3-3 中的 H/T 和 WL 结构，可以看出 H/T 结构同观察文字结构最大的不同之处是：回溯搜索时，观察文字结构不用修改观察文字的引用指针。另外，针对每一子句而言，观察文字结构跟变量相关的文字引用的数目是一常量。

该数据结构主要缺陷存在于确认子句是单元子句时的操作次数太多。

（3）带文字过滤的 H/T 链表（htLS）

该结构可以解决 H/T 结构和 WL 结构存在的问题。它类似于 H/T 链表结构，但它能动态改变文字链表，通过提高决策级别对子句的已赋值文字进行排序[21]。基于文字过滤这一方案有诸多优点：

A）当子句变成单元子句或不可满足子句时，无需访问所有子句中的文字来对此进行确认。除此之外，识别可满足子句的方式类似于其他的胀缩数据结构。

B）正如图 3-3 所示，每一子句仅需要关联四个文字引用。而 H/T 链表结构在最坏情况下需要与子句中文字数目等量的引用指针。

C）文字引用永远不可能跳出已赋值文字范围，即使是当搜索前进或者回溯时也不可能。

（4）带文字过滤的观察文字结构（WLS）

这一数据结构采用了文字筛选策略，但是会监视未赋值文字的引用，即当回溯时不会更新文字引用。WLS 数据结构的优点是它拥有简化的回溯搜索过程；缺点是每当子句变成单元子句或者不可满足子句时，都需要遍历所有的文字。

3.2 SAT 问题预处理

所谓 SAT 问题预处理技术，就是根据逻辑蕴涵推理，结合布尔约束传播过程、冗余子句删除和添加等值子句等相关技术，对输入的合取范式实例进行一系列的化简，然后将化简后的 CNF 公式送至原来的 SAT 求解器进行求解[24]。目标是使得化简后的合取范式能够减轻后续的求解工作量，即预处理的时间加上 SAT 求解器真正用于求解的时间小于求解器直接对原 CNF 公式进行求解所用的时间。

SAT 预处理器存在化简程度和时间开销折衷的问题。根据化简程度的高低和处理时间的多少，可以将它分为轻量级方法和重量级方法这两大类。

轻量级预处理器对 CNF 公式的化简程度较低因而所需要的时间开销也较少。文献[25]中的 2Simplify、文献[26]中的 NiVER 算法以及文献[27]中的 SatELite 算法都属于轻量级的 SAT 预处理器。其中，NiVER 算法对子句的分布进行分析后再进

行变量和子句化简。而 2Simplify 预处理器则通过蕴涵图进行逻辑蕴涵推理，并结合等值化简技术等对原问题进行化简[28]。这类预处理器尽管也采用了各种各样的技术来化简 SAT 问题，但只是简单地对原问题进行化简，对于降低 SAT 问题的求解时间方面效果并不显著。

重量级预处理器在预处理器需要较长的时间开销，但它们对 SAT 实例的化简程度较深，因而能较大程度的减少原问题的求解规模。文献[29]中提出的 Hypre 方法是这一类型预处理器的典型代表，该方法类似于 2Simplify 方法，也运用蕴涵图来进行逻辑蕴涵推理，并结合了等价化简技术来进行子句和变量化简，不同的是，它采用了更为完整高效的分解技术，对原问题的化简程度更深。这类方法的缺点就是需要的时间开销较大，对于一些大规模的 SAT 问题求解来说不是非常有效。

3.3 加速搜索的一些启发式策略

经典的 DPLL 算法主要采用的是基于对二叉树进行深度优先搜索的思想，即可以将问题搜索空间表示为由各个问题变量所组成的搜索树，然后用深度优先搜索的策略来遍历所有的通路，进而找到使问题可满足的解。原始的 DPLL 算法在最坏情况下的时间复杂度肯定是指数级别的，但可喜的是，从 DPLL 算法产生到现在的五十年里，人们并没有放弃对 DPLL 算法的研究和改进，而是经过不断的探索和努力，提出了许多行之有效的启发式的算法和相关技术，这就使得 DPLL 算法从最初只能解决几十个变量规模的问题发展成为现在已能解决几十万个乃至上百万个变量规模的问题，SAT 求解器处理速度出现了飞跃式的发展，这也使得对 SAT 问题的研究从单纯的理论研究转到了各种实际应用领域，进而也促进了其它领域的发展。本节将侧重介绍在 SAT 发展过程中出现的一些具有里程碑意义的技术和启发式策略。

3.3.1 BCP（Boolean Constraint Propagation，布尔约束传播）

布尔约束传播过程是对 DPLL 算法的求解进行的一种优化过程，也可称为单元推导[30][3]。BCP 就是反复地运用单元子句规则对合取范式的子句集进行化简，直到子句集中不再存在单元子句的过程。通常在实际的 SAT 问题中，每个子句包含的文字数通常很小，因此会频繁的出现一元子句，所以 BCP 过程对于问题的求解非常重要[31]。实验结果表明，基于 DPLL 的 SAT 算法的整个求解过程中，要占

用 80%~90%的时间来进行布尔约束传播过程。因此，BCP 过程的加速对提高整个 SAT 算法的运行效率意义重大[4]。

BCP 过程的工作原理是：在搜索过程中，随着搜索的不断前进，一些变量被赋值，在当前赋值下，为使合取范式 f 可满足，必然要求每一子句都是可满足的，这就会导致一些必要的变量赋值。如果一个子句 φ 中除了一个未赋值的文字 x 外其余文字的赋值都是 0，那么为了使子句 φ 成为可满足的，文字 x 必需赋值为 1。这是利用了单元子句规则来进行子句化简。通过推导导致的必要的变量赋值（将一个未赋值的文字赋值为 1）常称之为蕴涵。通常，BCP 过程包含单元子句的识别和相关蕴涵的创建。例如，对于子句 $\omega=(x_1\vee\neg x_2\vee x_3\vee\neg x_4)$，当在 $x_1=0$，$x_2=1$，$x_3=0$ 的条件下，该子句就变为单元子句，很显然，为使该子句可满足，子句 ω 中的文字 x_4 必需取值为 0，这就是所谓的单元子句规则。

3.3.2 变量决策策略

从上文的介绍中我们已经知道，SAT 问题是一个 NP 完全问题，如果变量的赋值顺序没有建立在高效的决策策略基础上，那么算法对时间的消耗将会十分大。在基于 DPLL 算法中，能快速有效地决策出下一个将要赋值的变量从而使算法进入新的分支决策过程，对提高 SAT 求解器的运行效率起着决定性的作用。

变量决策策略主要用在回溯算法的搜索过程中，这一过程可看成是对一棵二叉树应用分裂规则，使得从未赋值变量中找出下一个分支节点。变量决策策略的选择将直接影响到二叉树的大小与深度，对计算效率起到决定性作用。好的决策策略能够快速找到下一个决策变量，从而加速 BCP 过程的执行；而差的变量决策策略可能会沿着二叉树的某一分支一直执行却无法找到解，会增加回溯次数[32]。但现今为止还没有一种确定的标准来衡量变量决策策略的优劣，一方面这是因为不同的变量决策策略可能适用于不同的问题规模；另一方面好的决策策略会因为算法的复杂度较高使得长时间不能找到决策变量，而差的变量决策由于算法简洁却可以快速确定决策变量，但不保证所找到的变量最有价值[4]。

下面是一个具体的实例，通过该实例可以对比下采用不同的决策策略对 DPLL 算法求解效率的影响。

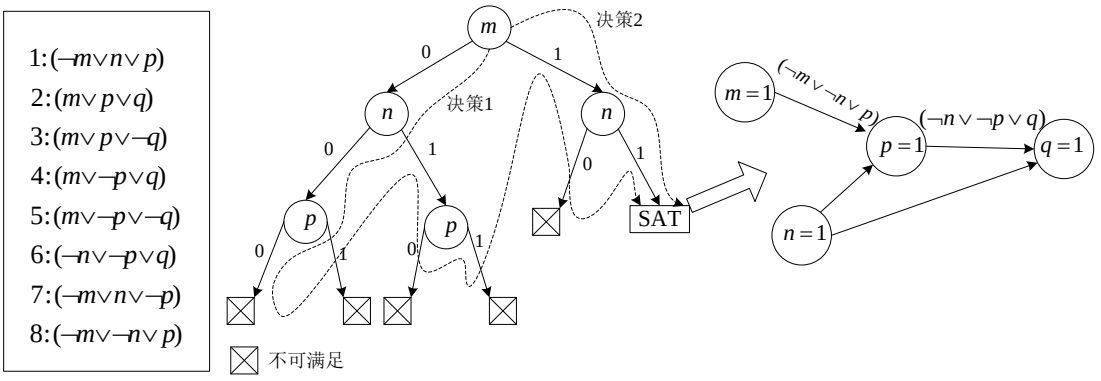

图3-4 不同变量决策策略的效果对比

从图 3-4 所示二叉搜索树的求解过程可知，上面的 SAT 问题是可满足的。决策 1 最初选择 $m=0$ 开始搜索二叉树，绕过许多弯路后最终找到解（如图 3-4 中决策 1 的虚线所示），显然不是一个好的决策策略。而决策 2 则选取了一条捷径，只经历了变量 m 和变量 n 这两个变量的赋值就得到了问题的解，确实是一个好决策。由于 DPLL 算法的求解本质上就是对一个由各决策变量构成的二叉树进行 DFS 的过程，因此不同的变量决策策略直接影响了二叉搜索树规模的大小，进而会直接影响算法求解效率。一个优秀的变量决策策略应能使算法尽快地找到问题的解或者可以尽早发现当前赋值存在的冲突。当前对于变量决策策略的研究主要是集中在提出各种启发式算法来进行决策变量的选择。本文仅对在众多的变量决策策略中最具影响力的三种方法做介绍。

（1）最短子句出现频率最大优先(MOM)

MOM 方法也就是常说的 JW 方法[33]，该算法最重要的是求解出和文字 l 相对应的 J 值。最终会选取使得 $J(l)$ 取值最大的文字 l 将其赋值为真，求解公式如下所示：

$$J(l)=\sum_{l\in C_i}2^{-n_i}\quad（i=1,2,\cdots,m，假设共有\ m\ 个子句）$$

其中，n_i 是包含文字 l 的子句 C_i 的子句长度（$i=1,2,\cdots,m$）。那么，C_i 的长度越长，相对而言，J 值便会越小。简单来说，只要子句中某一文字得到满足，那么该子句就是可满足的。所以，一个子句的长度越大（所含文字数多），那么使得该子句可满足的“候选”文字数目就越多，相对而言该子句就越容易得到满足；而如果子句长度越小（所含文字数少），则越不易得到满足[34]。因此在这些文字数较

少的子句中挑出出现频率较大的文字使其优先满足，这样做的目的就是让那些不易满足的子句得到优先满足，使搜索过程可以尽快找到可满足解。

这是一种高精度的计算方式，许多要求计算精度的方法都是以 JW 方法为基础的。这一方法曾被许多 SAT 求解器采用，但它也存在一定缺陷：一方面当所有子句的长度都相同时，这种所谓的在最短子句中进行选择决策变量便失去意义；另一方面由于计算精度越高，决策过程所需要的计算时间也就越长，所带来的计算代价也越大，因此这种方法是速度和精度的折中。尽管 JW 方法计算精度较高，但由于所花费的决策时间较长，现今的 SAT 求解器更倾向于采用计算时间较短却仍有一定精度的算法。

（2）独立变量状态衰减和策略（Variable State Independent Decaying Sum，简称 VSIDS 策略）

2001 年，zChaff 求解器[35]提出了一种全新的称作独立变量状态衰减和的变量决策策略。该策略较好地结合了 DPLL 算法的学习过程和布尔约束传播过程，使得 DPLL 算法的求解水平有了显著提高，并且与 MOM 方法相比较，该算法的计算代价比较低廉。VSIDS 策略的具体操作如下：

（A）为每一变量的正、负文字各设立一个计数器，初始值一般设置为该文字在所有的子句集中出现的次数。

（B）若文字在某一子句中出现（包含最初的子句，也包含学习子句），则将文字对应的计数器值增 1；有冲突发生的时候，就将冲突导出子句添加到子句集中并增加冲突导出子句中相应文字的计数器值。

（C）根据未赋值变量中的文字计数器的值的排序，从中选择一个使得计数器值最大的文字作为决策变量，若存在几个相同的最大值，那么随机选择其中一个文字对其赋值并执行 BCP 过程即可。

（D）为了防止一些变量长时间得不到赋值，经过一定的决策次数后，每一文字的计数器的值都会除以一个常数（通常为 2~4 左右的数），这样做也可以增加新近学习到的一些子句对搜索过程的贡献。

现在的 SAT 求解器的变量决策策略大都是从独立变量状态衰减和策略改进而来，这一策略对于求解一般化的 SAT 问题效果显著，但对于某些专用领域的 SAT 问题，比如电路等值转换问题，则需对该策略进行针对性的改动[36]才能使求解过程更高效。

（3）最短正子句优先策略

对子句集中所有文字都是正文字的子句（称之为正子句）按照子句长度进行

排序，从中选择长度最短的子句，从该子句中任选其中一个文字作为决策变量对其进行赋值，然后进行 BCP 过程，发生冲突时则回溯到当前选择的最短子句，重新选择其它剩余的文字为决策变量。

分析以上 3 种变量决策策略，可以发现它们存在一个共同的缺陷即算法实现过程均需要有变量计数器的辅助操作和大量的排序操作，并且每次单元子句赋值后，所有的活跃子句都需进行重新排序并修改文字计数器的值，这无疑增加了计算开销。最短正子句优先策略的优点是大多数情况下都能找到最具价值的决策变量。VSIDS 策略的优点是：文字计数器独立于变量状态，即不会随着变量状态的变化而发生改变，这就避免了变量赋值后因修改文字计数器值所带来的计算开销；此外，当发生冲突时，求解器会把冲突导出子句添加到它的子句集中，那么此时新近添加的子句的变量计数器值便会增加，最终会使得冲突变量拥有最高的变量决策优先级。VSIDS 策略虽可以避免大量的排序操作，但仍需经常修改文字计数器的值，并且它依赖于智能回溯和冲突分析。

3.3.3 冲突分析、子句学习、回溯机制

冲突分析是找出 BCP 过程冲突产生的原因并试图找到解决方案的过程。它能让 SAT 求解器了解到某些搜索空间不存在解，从而直接进入新的搜索空间进行求解。GRASP（1996）和 relsat(1997)是最早将冲突学习与智能回溯机制应用到 SAT 问题中的算法。

SAT 算法中引入冲突分析与学习机制的原因如下：

当分析出冲突产生的原因时，导致冲突产生的子句就会被记录下来，称为学习子句。学习子句可以防止在接下来的搜索过程中再次发生相同的冲突。

利用新近学习到的冲突子句可以计算出最优的回溯点。冲突分析过程中先找到产生冲突的子句，然后根据冲突子句，判定将要回溯到的目标层数。

回溯机制又可以分为时序回溯和非时序回溯[7]（也称为智能回溯）两种。

（1）时序回溯机制是指，算法的 BCP 过程中如果在某一分支节点处发生了冲突，那么算法只是简单地回溯到该节点的父节点，并将父节点处的变量取值翻转即可，这样做的结果可能还会导致冲突存在，这说明冲突节点的父节点并不是冲突产生的最终原因，从而浪费了一次计算时间。而且在最坏情况下，对某一分支中的一个决策变量赋值，该分支的子树区间可能会不断的发生冲突，进行回溯也跳不出该区间，这会耗费许多计算开销。

（2）鉴于以上原因，研究人员提出了基于冲突分析和学习的非时序回溯算法，经过冲突分析之后可以得知准确的回溯目标层次，可能需要在决策树中回跳多个决策层，并不是简单地回溯到冲突节点的父节点，并将冲突导出子句加入到子句集中。

为利于下面子句学习过程和智能回溯机制具体实现的展开，下面引入蕴涵图的概念。

定义：蕴涵图（Implication Graph，简称 IG 图），是一个由顶点和边组成的有向无环图，是对 SAT 问题中各变量间逻辑蕴涵关系的直观表示[35]。具体描述如下：

图中的每一顶点都对应给定决策层次上的一个变量赋值，用 $x=v(x)@\delta(x)$ 来表示，有向边表示变量间的逻辑蕴涵关系。

若某一顶点有入射边，则入射边上的标号对应某一单元子句，该单元子句蕴涵出该节点的赋值。前驱节点是对应子句中其它节点的赋值。若某一节点无前驱节点，表示该节点所处的变量是一个决策变量。

蕴涵图中有一个特殊节点称作冲突节点，表示有冲突发生。该节点的入射边上的标号表示所有文字取值都为 0（也就是使该子句不可满足的文字赋值）的子句，它的所有前驱节点的值就是该子句中的所有变量的赋值。

图 3-5 所示为一个蕴涵图的实例。

当前真值指派：

$$\{x_9=0@1, x_{10}=0@3, x_{11}=0@3, x_{12}=1@2, x_{13}=1@2,...\}$$

当前决策变量赋值：

$$\{x_1=1@6\}$$

子句数据库：

$$\{x_9=0@1, x_{10}=0@3, x_{11}=0@3, x_{12}=1@2, x_{13}=1@2,...\}$$

$$\omega_1=(\neg x_1\vee x_2);\qquad \omega_2=(\neg x_1\vee x_3\vee x_9)$$

$$\omega_3=(\neg x_2\vee\neg x_3\vee x_4);\quad \omega_4=(\neg x_4\vee x_5\vee x_{10})$$

$$\omega_5=(\neg x_4\vee x_6\vee x_{11});\quad \omega_6=(\neg x_5\vee\neg x_6)$$

$$\omega_7=(x_1\vee x_7\vee\neg x_{12});\ \omega_8=(x_1\vee x_8);\ \omega_9=(\neg x_7\vee\neg x_8\vee\neg x_{13})\ldots$$

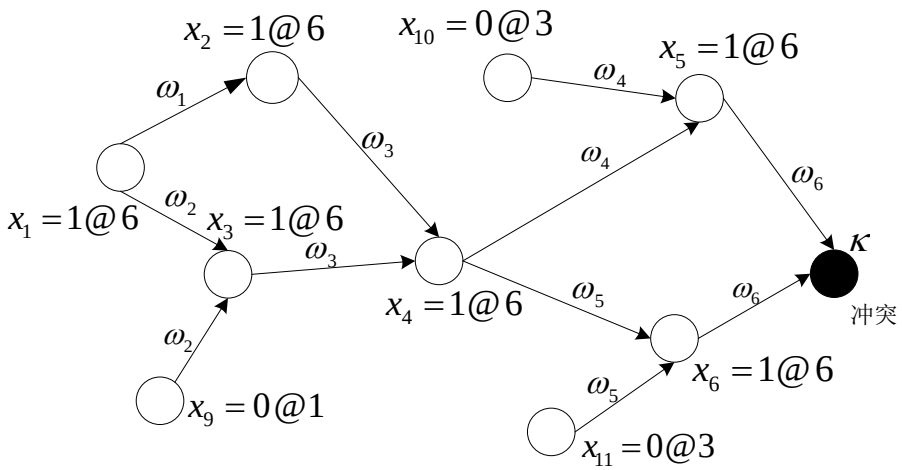

图3-5 蕴涵图

接下来通过图 3-6 来阐述下子句学习过程。上面已经给出了 CNF 公式的子集，假设当前决策层次是 6，决策变量 $x_1=1$。最终由于子句 ω_6 不可满足导致冲突的产生。通过观察蕴涵图，我们立即可以得出这一冲突产生的充分条件是：

$$(x_{10}=0)\wedge(x_{11}=0)\wedge(x_9=0)\wedge(x_1=1)$$

那么，通过创建并添加子句 $\omega_{10}=(x_{10}\vee x_{11}\vee x_9\vee\neg x_1)$，在接下来的搜索过程中就能预防相同的变量赋值出现。

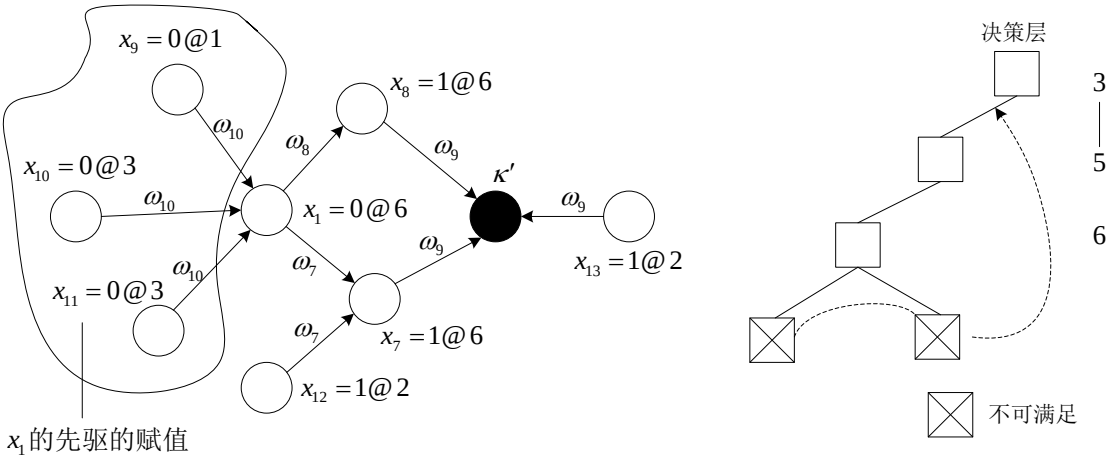

图3-6 非时序回溯过程

通过图 3-6 可以清楚的看到基于子句学习的非时序回溯过程。记录下子句 ω_{10} 之后，由于变量 x_9， x_{10}， x_{11} 均已赋值，子句 ω_{10} 在第 6 决策层时成为单元子句，

因此 BCP 过程蕴涵出赋值 $x_1 = 0$。另外，子句公式 ω_7 和 ω_8 也可以蕴涵出图中给定的赋值。最终由于子句 ω_9 不可满足导致产生冲突 κ'，此时添加子句 $\omega_{11}=(\neg x_1 \vee \neg x_3 \vee x_2 \vee x_{10})$ 就能预防再次发生相同的冲突。尽管当前决策层数是 6，但是与该冲突所关联的决策变量赋值分别处于第 1 层、第 2 层和第 3 层（均处于第 6 层之前），可以看出最高层是 3。因此可以直接回溯到第 3 层，而并不是直接回溯到当前决策层的上一层（即第 5 层）。

3.4 子句删除机制

无限制的子句学习在一些运行环境下是不实际的，学习到的子句会消耗内存，重复地记录子句最终会导致可用内存耗尽。考虑到学习子句的数目会随着冲突数目的增加而不断增大；在最坏情况下这种增长速度会是变量规模的指数级；此外，Marques-Silva 和 Sakallah 在 1996 年曾证明，大量的已学习子句对于减小搜索树的空间并不是特别有用，有时只会对搜索过程带来额外的开销。因此现今的许多 SAT 求解器都支持学习子句的删除功能，以避免出现内存爆炸问题。

鉴于以上问题，主要有 3 种方案来保证最坏情况下的学习子句数目是变量规模的多项式级别。

（1）采用 n 阶学习方法，仅记录有 n 个或者更少文字的子句。

（2）将能够蕴涵出其它变量赋值或者单元子句的子句临时添加到子句数据库中，一旦未赋值文字数目超过整数 m，就会删除这些子句。这种方法称为基于相关性的 m 边界学习法。

（3）在子树的空间搜索时记录长度小于 k 的子句，一旦未赋值文字的数目大于 1 就会删除这些子句。这一方法称之为 k -边界学习法。

Chaff 算法采用定期的“懒散”子句删除策略，类似上面的方法（2）。每当子句被添加到子句数据库中时，该方法会决定在未来的哪一时刻它将被删除。主要衡量标准就是相关性，当子句中第一次出现 N（一般 N 在 100~200 之间）个以上的文字未赋值时，子句就会被标记为需要“删除”的状态。

而 BerkMin 算法则采用更为贪心的删除策略，它不仅将学习子句的数目限制在一定范围内，而且还要考虑子句学习的先后顺序，这通常是通过栈来实现的。

最近人们又提出了一些启发式的子句删除策略。这一思想的本质是，是否需要删除某一子句不是取决于文字的数目，而主要看子句在制造冲突和创建以来所

产生的决策数目。

3.5 随机重启动机制

在 SAT 问题的求解过程中，由于最初的变量决策顺序可能不是最优的，这就会导致搜索陷入某些子空间中而白白耗费时间。重新启动机制，就是清除现在所有变量的赋值状态，重新选择一组决策变量进行赋值，然后进行正常的搜索过程。重启动机制主要是针对问题规模较大的实例，通常需要结合子句学习过程。

它主要包括暂停求解过程和利用之前学到的信息重新启动决策分析这两个阶段。由于重启之前添加的一些学习子句在重启过程之后仍然存在，因此求解器重启后并不是简单地进行重复分析，以前的搜索过程也不会因为重启过程而白费，反而有利于全局搜索顺序的调整。所谓随机重启动策略就是在新的搜索路径的选择过程中添加一些随机因素，比如重新选择一些决策变量就行赋值。

值得注意的是，重启可能会破坏算法的完备性。但是如果所有的子句都已经学习到了，那么算法仍然是完备的。因此算法的完备性应该靠不断增大上文中所提到的相关性系数 N 的值来维持。文献[7]中，作者提出的 GRASP 算法就使用了类似的策略来维持算法的完备性，主要通过延长每次重启周期来实现。文献[5]中，作者提出的 Chaff 算法默认情况下也采用了这一机制，通常情况下这对算法的性能是有改进作用的。

3.6 基于 DPLL 的算法发展历史综述

截止目前，可满足问题的研究仍是以完备算法为主要研究方向，特别是有一类称为 DPLL 的算法及其各种派生算法的发展速度尤其突出[37][38][39]，被称之为目前 SAT 问题的标准求解方案。基于 DPLL 的算法发展历史如表 3-1 所示。

表3-1 基于 DPLL 的 SAT 算法发展史

时间	算法名称/进展	采用的新技术/创新点
1960	DP	提出了最原始的 DPLL 算法框架
1962	DLL	基于深度优先搜索思想
1996	DPLL 有指数级下界	
1971	SAT 问题是 NP 完全问题	
1995	GRASP	引入非同步回溯和子句学习，有效减小搜索空间
1997	SATO	首次提出 H/T 胀缩数据结构，实现回溯搜索比较
2001	Chaff	首次提出基于观察文字的数据结构使得 BCP 得以高效实现；采用低开销的独立变量衰减和决策策略
2002	BerkMin	添加了子句删除策略
2004	zChaff	它是 Chaff 算法的一种高效实现
2005	Minisat	基于冲突向导学习、监视字和动态变量排序技术

最早的 DPLL 算法的研究可以追溯至 1960 年，由 M.Davis 和 H.Putnam 二人提出了 DP 算法[30]，这个算法是后来各种完备性 SAT 算法研究和发展的基础。紧接着 M.Davis, G.Logemann 和 D.Loveland 三人于 1962 年提出了 DLL 算法[3]，该算法是在深度优先搜索思想的基础上提出来的。DP 算法和 DLL 算法统称为 DPLL 算法，这两种算法的思想一直沿用至今，是以后的各种完备算法发展的基础。这两种算法虽然提出的时间比较早，但时间复杂度都是指数级的，效率比较低，它们至多能解决 10 个变量的问题规模，因此较难应用在实际问题中，这在一定程度上阻碍了 DPLL 算法的发展。

之后出现了许多新的启发式搜索的策略和新技术，如上面提到的智能回溯策略、搜索的重新启动策略、基于冲突子句的学习和随机化技术等。1995 年 J.P.Marques-Silva 和 K.A.Sakallah 提出了 Grasp 算法，该算法非常系统地总结了 DPLL 的算法流程，详细了介绍了子句学习过程，基本原理同 DP 算法差别不大，但它能有效地减小搜索空间，这使得 SAT 算法的发展出现了一次飞跃。2001 年出现的 Chaff 算法[5]通过采用新的胀缩数据结构改进 BCP 的实现过程、采用 VSIDS

决策策略、子句删除和重新启动策略，使得算法的求解速度相比其它的 DPLL 算法提高了一到两个数量级。2005 年出现的 Minisat 算法[2]是迄今为止各方面性能最好的 SAT 求解器，采用了冲突向导学习、监视字和动态变量排序等机制，算法设计精简高效，在诸多的 SAT 实际问题上都能获得较快的求解速度。经过验证，在十分钟的超时时间内，Minisat 能够求解 158 个 SAT 竞赛中的实例，而 zChaff 仅能求解 147 个。

上面提到的这些算法都是基于 DPLL 算法来实现的，它们用来求解问题的基本步骤是一致的，这些基本的求解步骤就是解决 SAT 问题的基本方案，接下来将会有章节专门介绍 DPLL 算法思路。

3.7 基于 DPLL 的 SAT 算法框架

3.7.1 算法的基本思想

对 CNF 公式中的文字进行真值赋值所得出的搜索空间可以用一棵二叉树来表示，树中的每个节点对应一个命题变量，取值只能为 1 和 0，左右子树表示该变量取真值和假值的分支，从二叉树中根节点到叶子节点的一条路径就表示 CNF 公式中的一组变量赋值序列。基于 DPLL 的算法都是对这棵二叉树从根节点开始进行深度优先搜索(DFS)遍历所有的通路，以找到使问题可满足的解。

设变量集合为U，该变量集合上的变量组成合取范式B，则有可满足性问题(B,U)和一个变量赋值顺序β，变量集合U上的所有可能的真值赋值在β下会构成一棵二叉树的搜索空间。

举例说明，设$U=\{x_1,x_2,x_3\}$，$\beta=\{\beta_1(U)=x_1,\beta_2(U)=x_2,\beta_3(U)=x_3\}$，那么真值赋值所构成的搜索空间如图 3-7 所示：

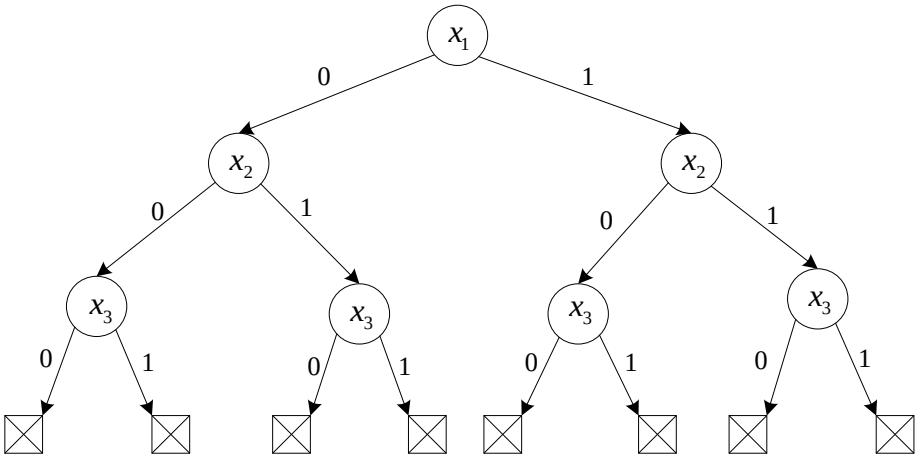

图3-7 变量的二叉搜索空间

容易知道，这课二叉树的叶子节点只能有两种：

1) 子句集取真值，该问题是可满足的。
2) 子句集中含有一个或多个空子句，该问题不可满足[40]。

绝大部分回溯搜索算法都是建立在原有的 DPLL 算法框架之上的。它们的求解过程基本上都是分为以下 3 个阶段：

1) 变量决策阶段（即 DECIDE 过程）：在搜索过程的每一分支阶段，选择未赋值的一个变量为其赋值为 0 或 1，该变量称为决策变量，决策变量在赋值时所处的二叉树中的高度称为它的决策层。
2) 推理阶段（也称为 BCP 过程）：每一次已选择变量赋值之后，识别该赋值所导致的必要的赋值或者根据已有的变量赋值对子句进行化简，即进行布尔约束传播过程。举例说明，比如子句$(x_1 \vee x_2 \vee \neg x_3)$，如果决策阶段选取$x_3$为决策变量并令它取值为 0，那么该子句已可满足，就可以将其从子句集中删除，原问题得到了化简。
3) 回溯阶段（BACKTRACK）：推理过程中发生冲突时，进行冲突分析和子句学习，并实现算法的回溯，使搜索过程从较深的变量决策层返回至较浅的决策层。冲突即指 BCP 过程中，至少出现了一个子句不可满足的情况。

3.7.2 实现过程

基本的 DPLL 算法执行流程可用图 3-8 表示：

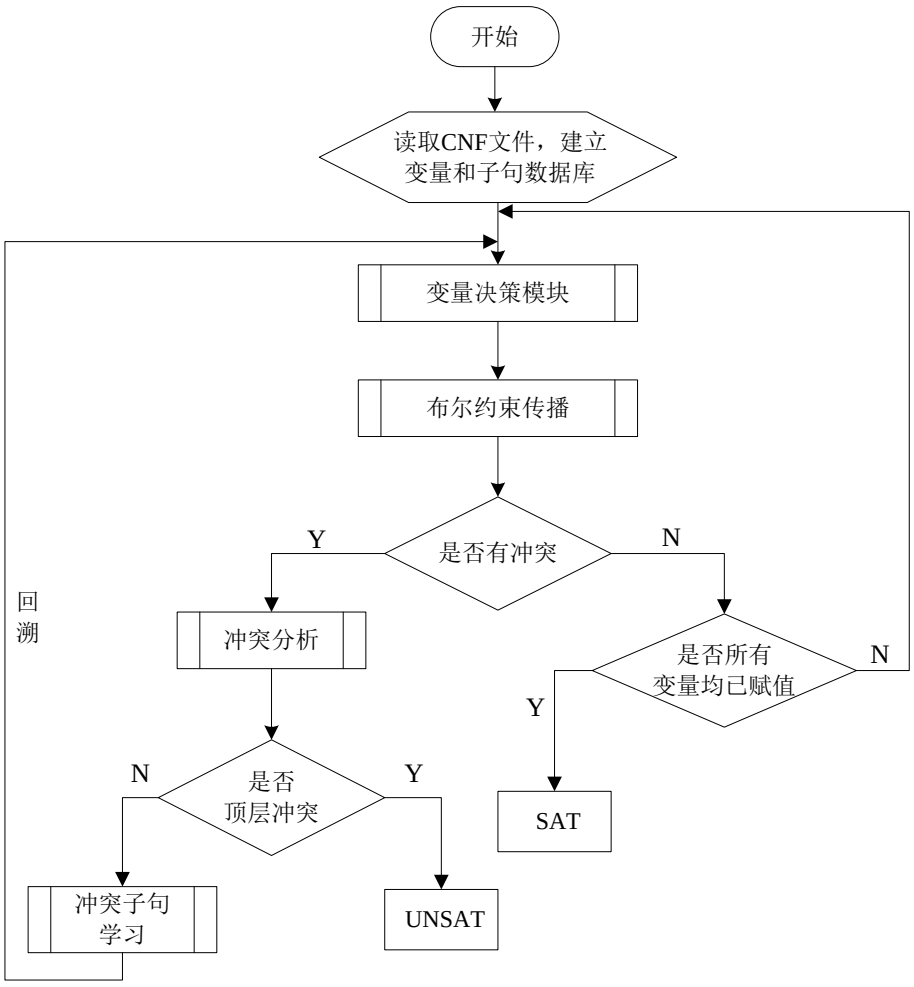

图3-8 基本的 DPLL 算法流程图

DPLL 算法就是循环使用上文中提到的单子句规则、纯文字规则、分裂规则、重言子句规则和包含规则等化简策略对不断地对 CNF 公式进行化简直到求得问题的解的过程。最原始的 DP 算法可用下述伪代码来描述：

```
bool DPLL(B)
{/*B 是所求 SAT 问题的子句集，当它是可满足时，返回 true；不可满足时返回 false*/
    while(1)
    {
        if(子句集 B 中存在单元子句 C)
        {
            利用单元子句规则，对该子句中的唯一文字 l 进行赋值，然后利用 l 化简子句集 B；
```

```
            if(B 为空)   /*子句集为空，说明公式可满足*/
            {
                return ture;
            }
        }
        /*存在非空子句，需要进行变量决策*/
        if(!variable_decide()) //从 B 中任选一个没有赋值的变量为其赋值，
    //赋值成功返回 1；若所有变量均已被赋值，则说明问题可满足
        {
            return true;
        }
        while(!bcp_deduce()) /*执行 BCP 推理过程，利用化简策略对子句集进
行化简；*/
        {
            if(Conflict())      /*存在冲突*/
            {
                选择最近的没有翻转的决策变量 v;
                if(v==NULL)
                {
                    return false;
                }
                else
                {
                    翻转变量 d 的赋值，并将其标记为已翻转变量状态；
                    取消上次 bcp 过程中导出的无效蕴涵；
                }
            }
        }
    }
```

下面是一个实际的 SAT 问题按照 DPLL 算法进行求解的过程，如图 3-9 所示。

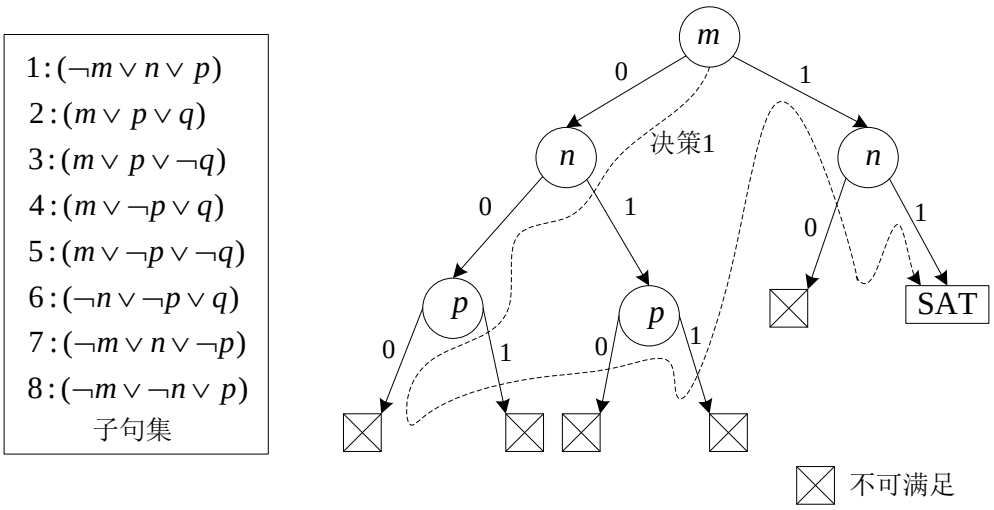

图3-9 一个 SAT 实例的求解过程

首先选取变量 m 作为决策变量进行赋值，图中虚线所示即为决策变量的选择和赋值过程，当执行到叶子节点时，若存在子句不可满足，那么便沿着图中最下边的虚线就行回溯搜索，直到问题可满足为止，或遍历完整个搜索空间问题仍不可满足，则可以断定该问题是不可满足的。

3.8 小结

本章立足于最原始的 DPLL 算法框架，深入分析基于 DPLL 的 SAT 算法的基本思想和实现过程，并通过一个实际 SAT 实例阐述了其求解的过程。尽管许多算法都是基于原始的 DPLL 算法来实现的，但是新近的一些求解算法都进行了必要的改进以提高求解效率，这些改进主要包括新的搜索策略、新的搜索技术和新的数据结构[32]。

第四章 WM 算法——改进后的 DPLL 算法

从前文的介绍可以知道，SAT 算法分为完备算法和局部搜索算法两大类。其中，完备算法在给定的 CNF 公式是可满足的情况下，能给出可满足的解，或者在公式不可满足的情况下给出完备性的证明，但求解速度相对较慢，最坏情况下的时间复杂度是指数级别的；而局部搜索算法在公式是可满足的情况下，对于许多难的 SAT 问题能较快的给出解并且很容易实现，但是对于不可满足的公式来说，它却无法给出一个完备性的证明。因此，结合完备算法能够进行完备求解和局部搜索算法能够以较快速度进行求解的优点，各取其长，可以构造一个混合的高效的 SAT 问题求解器。它能够利用局部搜索算法的快速性来加速完备算法的搜索，利用完备算法的完备性来弥补局部搜索算法不能给出完备性证明的缺点。

4.1 算法描述

我们知道，WALKSAT[18]和 DPLL[3]是两种不同类型的 SAT 求解器。2000 年提出的 WALKSAT 算法是一种局部搜索算法，而 DPLL 最早是在 1960 年提出的，它是一种完备的回溯搜索算法。

本文提出的改进算法是在 DPLL 这一基本算法框架下进行的，它结合完备算法能够进行完备求解和局部搜索算法能够以较快速度进行求解的优点，将 WALKSAT 算法作为一个必要步骤融入到 DPLL 的求解过程。

原始的 DPLL 算法在确定好决策变量时，对变量赋以何种极性的值往往是随机决定的，带有一定的盲目性，并且计算代价较高。对给定的 CNF 公式运用 WALKSAT 算法进行局部搜索，就可以先得到问题的一个近似解向量，由于近似解向量能使大部分子句都是可满足的，因此可以据此来指导 DPLL 算法对所选择的决策变量进行赋值，这样可以避免盲目的对决策变量进行赋值，并且可以减少算法的决策次数和冲突次数，从而加速搜索过程[11]。并且，每当有顶层变量的赋值确定以后，就可以引入 WALKSAT 对缩小了的 CNF 公式进行求解，从而提高求解效率。

为便于后续内容的展开，有必要先介绍下 WALKSAT 算法的基本框架。

1) 为 CNF 公式中的每一变量赋以一个随机值，并设置随机噪声参数 p（对于随机 SAT 实例来说，通常设为 0.5 时效果最佳[41]）；

2) 若当前赋值可以满足所有的子句，则算法结束并返回该赋值结果；

3) 否则，产生一个随机数 $r \in [0,1)$，如果 $r < p$，先从不满足子句集中随机选择一个子句，然后从该子句中随机选择一个变量并将其赋值翻转；如果 $r > p$，则从所有变量中选择一个能使翻转该变量前所有不满足的子句数目减去翻转之后不满足子句数目的差值最大的变量。如果该差值的最大值小于等于 0，则不翻转任何变量，否则翻转该变量的取值。

4.1.1 算法思想分析

事实上，相对于完备 SAT 算法，局部搜索 SAT 算法更符合人类的思维习惯。它从最初的猜测给定问题的解，到最后按照一定的搜索策略逐渐地靠近能使问题可满足的最优解，是一个不断迭代的过程。

从空间的角度分析，可以把一个 SAT 问题看成 n 维空间上的求解问题。比如一个仅包含两个布尔变量 x 和 y 的 SAT 问题 P，可以把问题 P 本身看成一个二维空间的正方形，它位于第一象限，其四个顶点的坐标分别为（0，0）、（0，1）、（1，1）和（1，0），求解问题 P 就是在该四方形找到一个使得 P 可满足的顶点。

同理，对于包含有三个布尔变量的 SAT 问题，就是一个三维空间的立方体，求解就是在该立方体上找到一个有效的顶点。包含有 n 个布尔变量的 SAT 问题，则是在 n 维空间的正方体上，寻找一个有效的顶点。

所谓局部搜索算法就是在这个 n 维空间正方体任意一个顶点出发，寻找那个可能使得问题有解的顶点。这种寻找过程就是在顶点之间的移动，当然，移动只可能在邻近的顶点之间发生，因为每个顶点的可视范围局限在周边的 n 个顶点内。这种思想符合人类的思维，人们在解决实际问题时，总是局限在自己所熟悉的范围之内，比如自己所熟悉的朋友、同事以及地点场所等。因而这种方法称做局部搜索，在邻近顶点之间的移动，在 SAT 求解算法中称之为 FLIP，中文一般译为翻转。很明显，在 n 维空间正方体上，任意两个邻近顶点之间的取值只在某一个维次上有所不同。

假如 SAT 问题 P 有解，并且该解位于顶点 A，很明显可以得出下面的结论：

结论：设包含有 n 个布尔变量的 SAT 问题 P 是可满足的，则从任意一个真值指派出发，至少存在一条最短路径，在该路径上，最多只需 n 次翻转，便可到达

目标指派。

该结论是显而易见的。如果将路径上的翻转次数定义为路径长度，通常情况下，最短路径的长度不会大于 n，当然在极端情况下，需要 n 次翻转，例如，从(0，0，……，0)到(1，1，……，1)。

虽然该结论看起来很诱人，然而事实上，这条路径并不易寻找，对于茫然漂浮于 n 维空间诸顶点之间的 SAT 求解器来说，它只能根据每一次移动有可能获得的眼前利益来判定下一次移动的方向，而这个眼前利益有可能与全局利益是相背离的，由此，它不知道自己所选择的每一步离目标顶点是近了一步还是又远了一层。甚至有时会遇到更不幸的情况，在这个茫然的空间上，根据眼前利益得到的结果可能会构成一个死循环，来来回回就局限在几个地方，却总是找不到突破点。

因而，局部搜索算法存在两个关键问题：

问题 1：如何确定一个好的翻转方案，使得求解器移动方向总是向目标顶点靠近？

问题 2：当翻转方案局限于一个无解的圈子内时，采用何种手段跳出这个圈子？

对于问题 2，可以采用平移的方式来解决。换句话讲，当原有的方案不能奏效时，可以采取重新选取起始顶点的方式，以打破固有圈子。有人认为该方法会带来效率上的浪费，然而在以前的实验中，这种方法被证明是有效的。因为，事实上在局部搜索算法中，除非显而易见的结论，没人知道下一次翻转会真正带来什么后果，所以每一次翻转都可以看成是重新开始。

遗憾的是，对于问题 1，目前并无有效答案。但是，从完备算法中，可以带来一些有价值的信息，比如冲突及学习子句。

完备算法中，冲突的解决较为麻烦，可能会带来较多的回溯，但是一旦冲突结点回到最低层（根结点，也可称为顶层结点），就产生了一个有效的信息，要么该问题是不可满足的，要么顶层结点变量获取了一个有效指派，此时的 n 维搜索空间就变成了 n-1 维空间。我们的改进算法，其思想便从此处入手，每产生一个有效指派，便确定离目标结点近了一步，因而，便可在每次有效指派产生之后，引入局部搜索算法。

理想情况下，引入该局部搜索算法之后，问题可直接在较短时间内就得到解决。即便不能得出解时，也能为后续 DPLL 算法的执行提供一个近似解向量[11]，以指导其对变量进行赋值，从而减小对决策变量赋值的随机性和盲目性，使搜索朝着更有利的方向前进。

引入该策略的出发点是：原始的 DPLL 算法在确定好决策变量时，对变量赋以何种极性的值往往是随机决定的，带有一定的盲目性，并且计算代价较高。对给定的 CNF 公式运用 WALKSAT 算法进行局部搜索，就可以先得到问题的一个近似解向量，由于近似解向量能使大部分子句都是可满足的，因此可以据此来指导 DPLL 算法对所选择的决策变量进行赋值，这样可以避免盲目的对决策变量进行赋值，并且可以减少算法的决策次数和冲突次数，从而加速搜索过程。并且，每当有顶层变量的赋值确定以后，就可以引入 WALKSAT 对缩小了的 CNF 公式进行求解，从而提高求解效率。

4.1.2 WM 算法执行流程

改进后的算法（本文简称为 WM 算法）首先对给定的 CNF 公式运用 WALKSAT 算法进行局部搜索，主要是对缩小规模的变量和子句数据库进行一次 WALKSAT 处理，但是相对于单纯的局部搜索算法来讲，总的迭代次数和 FLIP 次数不能过大，如果设置过大，便会影响全局搜索的性能。

理想情况下，引入该局部搜索算法之后，问题可直接在较短时间内就得到解决；不能得出解时，也能为后续 DPLL 算法的执行提供一个近似解向量（能使绝大多数子句都可满足的解），由于近似解向量能使大部分子句都是可满足的，因此可以据此来指导 DPLL 对所选择的决策变量进行赋值，从而减少变量赋值的随机性和盲目性[11]。然后对原公式运用 DPLL 算法求解，选择决策变量，并按照近似解向量中对该变量的赋值为决策变量赋值，这就保证了近似解向量所在的子空间能得到 DPLL 算法的优先搜索，从而可以加速搜索过程。

接下来仍按照 DPLL 算法的原思路求解，进行布尔约束传播过程，执行完 BCP 过程后若所有子句都是可满足的，那么原问题就是可满足的，否则表明发生了冲突，需要进行冲突分析和回溯过程。图 4-1 表示 WM 算法的实现流程图。

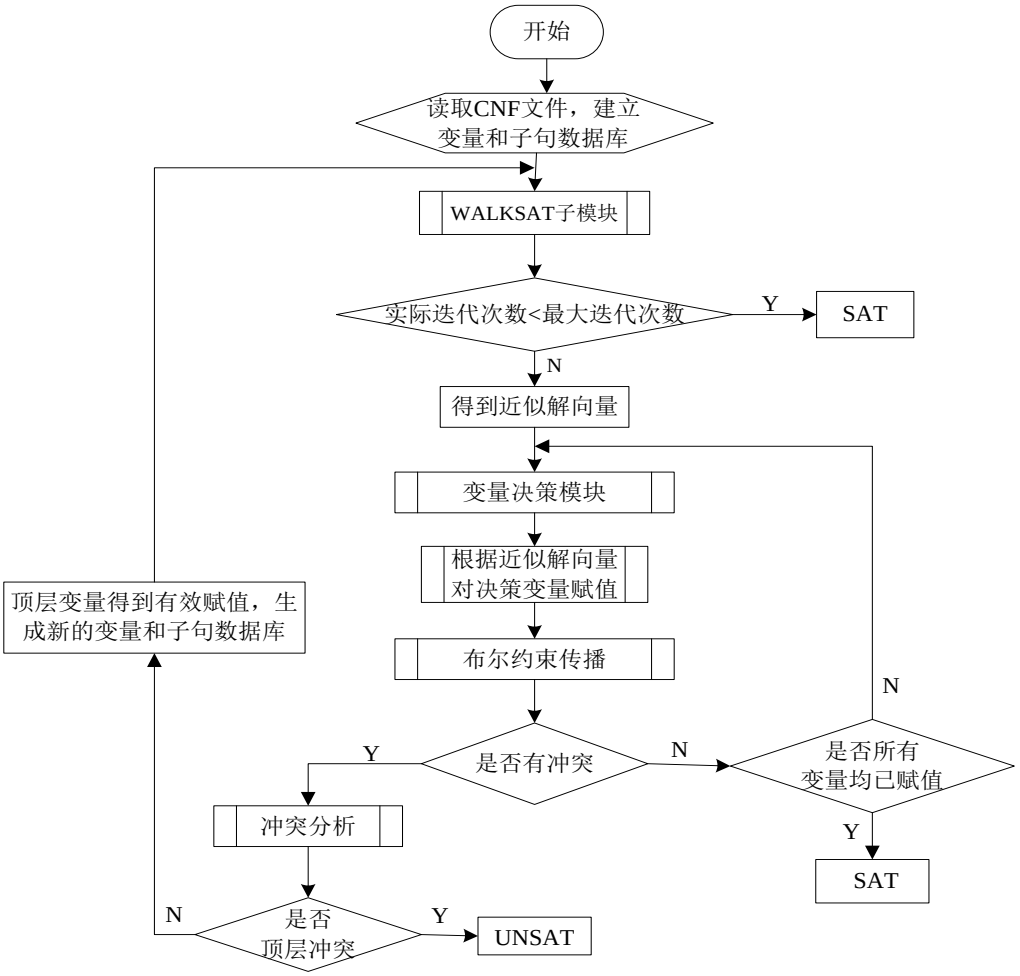

图4-1 WM 算法流程图

通过图 4-1 可以看出，WM 算法是在 DPLL 这一算法框架基础上建立起来的，WALKSAT 仅仅作为其中的一个必要步骤，但却起到指导决策变量赋值和加速搜索的作用。

其详细的执行过程如下：

（1） 读入 CNF 文件，进行一次 WALKSAT 求解。①设置最大迭代次数 N，最大翻转次数 M 和随机噪声参数 p（此处取 0.5），并统计每个文字在子句中出现的次数，按照正负文字次数大小生成一个待翻转变量队列；②为每个变量赋随机值；③若该赋值满足所有子句，则返回 SAT；否则进入第④步；④变量翻转模块：如果未达到最大翻转次数，产生一个随机数 r，如果 $r<p$，则从任一不可满足子句中任选一变量进行翻转，如果 $r>p$，则选择文字队列中的第一个变量进行翻转，如果该变量赋值与当前值相同，则选择队列下一项，将本结点移到最后。如果已经达

到最大翻转次数，则进入下一步。⑤如果此时未达到最大迭代次数，则进入第②步。否则，得到近似解向量 $X=[x_1',x_2',\cdots,x_n']$（假设有 n 个布尔变量）。

（2） 变量决策模块，选取决策变量 x_i（$1\le i\le n$）。

（3） 根据第（1）步中得到的近似解向量中的 x_i' 的值对第（2）步中的决策变量 x_i 进行赋值。

（4） 根据当前赋值，进行 BCP 推理。

（5） 推理过程中发生冲突时，进行冲突分析和子句学习，并实现算法的回溯。如果是顶层冲突，则直接返回 UNSAT，否则顶层变量得到有效赋值，此时总的变量数减少 1，并且会生成新的变量和子句数据库，然后返回第（1）步。如果没有发生冲突，则进入下一步。

（6） 如果所有变量均已赋值，则返回 SAT；否则返回至第（2）步。

图 4-1 中的 WALKSAT 子模块执行流程如图 4-2 所示：

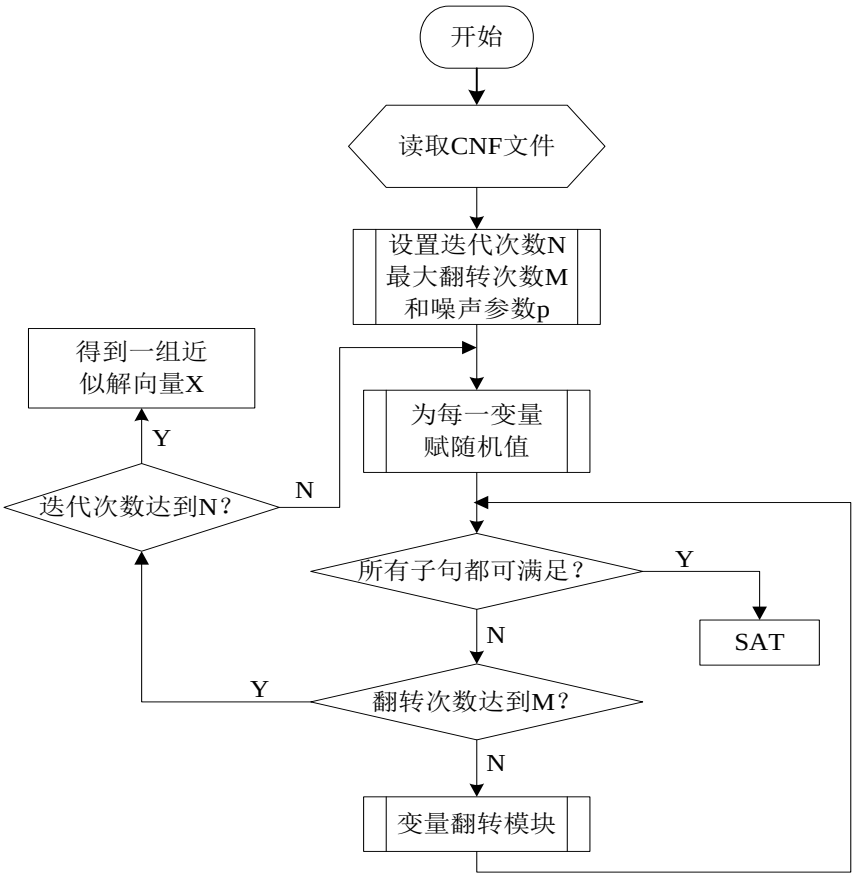

图4-2 WALKSAT 子模块流程图

例如，某一 CNF 公式通过 WALKSAT 求解后，得到近似解向量 $x=[x_1',x_2',\ldots,x_n']$，其中 x_i' $(1\le i\le n)$取值为 0 或 1，表示该变量赋值为 0 或 1。若经过 DPLL 算法决策后，确定的决策变量为 x_3，此时应该在近似解向量中查找第三个分量值 x_3'的值， x_3 的赋值要根据 x_3'的值而定。

4.2 算法改进前后性能对比

Minisat 算法[2]最早是在 2004 年提出的，它采用基于冲突的子句学习和智能回溯技术，变量决策采用独立变量状态衰减和策略（VSIDS），其求解速度远远领先于其它的一些高效 SAT 求解器，被认为是目前为止最简洁高效的 DPLL 算法，在 2010 年的 SAT 竞赛中，它取得了第一名的好成绩[42]。因此，本文选取 Minisat 算法作为基准算法来进行实验结果的对比。而 WALKSAT 算法在 GSAT 方法中加入随机噪声，来避免算法陷入局部最优的赋值，它在局部搜索算法中求解速度较快，比较有代表性。本文中的改进算法——WM 算法，以 Minisat 和 WALKSAT 这两种算法为基础进行改进，解决了一些难的随机 SAT 问题，求解效果显著。对测试用例进行测试后，WM 算法求解结果将和迄今为止性能最好的 Minisat 算法的运行结果做对比，以显示出性能优势。

4.2.1 仿真环境

硬件环境：PC 机，Intel Core2 双核 CPU 2.0GHz，1G 内存；

使用的操作系统：Microsoft Windows XP 系统；

算法运行平台：Microsoft Visual C++ 6.0；

4.2.2 测试用例

实际测试中，各种求解算法往往都需要有一些 SAT 实例来对算法性能进行对比和衡量。SAT 问题如何产生测试用例也是一个极具挑战性的问题，和 SAT 算法的研究一样也是计算机科学要研究的重要课题。主要有三类 SAT 实例：工业化 SAT 实例，随机 SAT 实例和人工 SAT 实例[43]。现阶段常用的是子句长度固定的难的随机 SAT 实例来对算法进行测试。

表 4-1 中的测试用例均为难的随机 3-SAT 问题实例，它们都是来源于文献[44]，是 SATLIB 中的基准问题，这些问题用一般的算法求解起来都比较困难，因此本文

选取这些实例来进行测试，以显示性能优势。这些实例有些是可满足的，有些是不可满足的，变量个数在 100 到 300 之间，子句个数在 500 到 1100 之间。本文从中选取了 34 组难的随机 3-SAT 实例，其中表 4-1 中的 17 组实例均是可满足的，表 4-2 中的 17 组实例是不可满足的。

随机 3-SAT 问题是一类非常难解的 SAT 问题，无论对于完备算法来说还是对于局部搜索算法而言都是一大挑战，因此能求解出该区域实例的算法意义重大。

4.2.3 仿真结果对比和分析

设置 Minisat 算法的超时时间为 1000s，超出 1000s 则不再继续求解。这里的运行时间指的是 CPU 时间，时间单位为秒。

相对于单纯的局部搜索算法来讲，WM 算法中 WALKSAT 模块总的迭代次数和 FLIP 次数不能过大，如果设置过大，便会影响全局搜索的性能。

WALKSAT 输入参数设置如下：

(1) 迭代次数：100

(2) flip 次数：10000

(3) 噪声参数：5/10

(4) 超时时间：1000s

在执行 WALKSAT 模块时，如果超时时间已经到了但还未达到最大迭代次数，此时也会退出该模块，继续执行 Minisat 中的其它模块。

为了更好的衡量算法的平均性能和稳定性，我们对每组实例都运行 20 次，取平均值作为最终结果。可满足实例的运行结果如表 4-1 所示。

表4-1 Minisat 算法和 WM 算法运行时间比较 1

测试用例	变量个数	子句个数	Minisat 运行时间(s)	WM 运行时间(s)	是否 可满足
uf125-01.cnf	125	538	0.057	0.023	SAT
uf150-01.cnf	150	645	0.013	0.020	SAT
uf175-01.cnf	175	753	0.090	0.054	SAT
uf200-01.cnf	200	860	0.157	0.114	SAT
uf200-021.cnf	200	860	0.130	0.046	SAT
uf225-01.cnf	225	960	0.363	0.023	SAT
uf225-013.cnf	225	960	5.280	0.062	SAT
uf250-01.cnf	250	1065	0.407	0.021	SAT
uf250-096.cnf	250	1065	32.090	0.078	SAT
uf225-022.cnf	225	960	0.594	0.068	SAT
uf225-023.cnf	225	960	0.160	0.046	SAT
uf225-024.cnf	225	960	3.420	0.037	SAT
uf225-025.cnf	225	960	16.294	0.034	SAT
uf250-087.cnf	250	1065	16.326	0.059	SAT
uf250-088.cnf	250	1065	0.121	0.081	SAT
uf250-089.cnf	250	1065	1.026	0.037	SAT
uf250-094.cnf	250	1065	20.448	0.225	SAT

分析表 4-1 中的统计数据可知，对于可满足的 SAT 实例来说：

几乎对于所有的实例（除实例 uf150-01.cnf 外），WM 算法的求解速度均有了一定的提高，尤其是对实例 uf250-096.cnf、uf225-025.cnf、uf250-087 和 uf250-094，求解速度有了明显的提高。

经过分析可知，这是由 WALKSAT 模块提供的近似解加速了 MINISAT 的搜索过程所导致的。当使用近似解向量来指导 MINISAT 对变量赋值时，MINISAT 算法会优先搜索近似解所在的子空间，从而保证了搜索过程中绝大部分子句是可满足的，从而加速了搜索速度。

因此，WM 算法适合于求解该类问题。

不可满足实例的运行结果如表 4-2 所示。

表4-2 Minisat 算法和 WM 算法运行时间比较 2

测试用例	变量个数	子句个数	Minisat 运行时间(s)	WM 运行时间(s)	是否可满足
uuf225-022.cnf	225	960	5.714	8.400	UNSAT
uuf225-023.cnf	225	960	5.826	8.408	UNSAT
uuf225-024.cnf	225	960	5.520	8.356	UNSAT
uuf225-025.cnf	225	960	8.086	10.916	UNSAT
uuf250-092.cnf	250	1065	28.350	32.510	UNSAT
uuf250-093.cnf	250	1065	17.258	20.398	UNSAT
uuf250-094.cnf	250	1065	21.360	24.466	UNSAT
uuf250-095.cnf	250	1065	11.960	14.940	UNSAT
uuf250-096.cnf	250	1065	25.522	28.300	UNSAT
uuf175-01.cnf	175	753	0.447	0.702	UNSAT
uuf200-01.cnf	200	860	1.483	4.807	UNSAT
uuf200-09.cnf	200	860	2.410	5.500	UNSAT
uuf225-01.cnf	225	960	7.392	12.082	UNSAT
uuf225-06.cnf	225	960	5.551	8.090	UNSAT
uuf250-01.cnf	250	1065	16.142	23.221	UNSAT
uuf250-010.cnf	250	1065	20.770	23.360	UNSAT
uuf250-023.cnf	250	1065	16.580	19.661	UNSAT

分析表 4-2 中的统计数据可知，对于不可满足的 SAT 实例来说：

WM 算法的求解速度较 Minisat 算法均减慢了。经过分析可以知道，这是由 WALKSAT 这一局部搜索算法的不完备性导致的。当给定的公式不可满足时，在给定的迭代次数和翻转次数之内，WALKSAT 模块肯定不能得到原问题的解，这势必会耗费一部分求解时间。之后得到近似解去指导 Minisat 求解便失去了固有的意义，这是由于该问题本身是不可满足的，即使有近似解去指导永远也不能得到一个有效解。

因此，可以得出如下结论：

WM 算法不适合于求解不可满足的随机 3-SAT 问题，但对于求解可满足的随机 3-SAT 问题效率比较高。它适合于求解可满足的随机 3-SAT 问题。

4.3 算法复杂度分析

对于一个随机生成的 CNF 公式，假设它的变量数为m，子句数为n，每个子句都有l个文字。

关于 WALKSAT 局部搜索算法的平均时间复杂度已有文献给出[45]。文献[45]给出了局部搜索算法的平均时间复杂度分析结果，即当 SAT 问题的约束较弱时，局部搜索算法具有多项式时间复杂度。具体说来，当$n<2^l m$时，求解随机 SAT 问题的算法复杂度为$O(n^{O(1)}l(1+n^{1+\varepsilon}/m))$，其中$\varepsilon$为充分小的正数。特别地，对于$n/m\le 1$的随机 3-SAT 问题，算法复杂度为$O(n^2(1+n^{1+\varepsilon}/m))$。当$n\square\ m$时，算法复杂度为$O(nl)$，这时局部搜索算法有多项式时间复杂度。应用到本文中的测试用例，l=3，所有实例均满足$n<2^3 m$，故本文中 WALKSAT 的时间复杂度为$O(3n^{O(1)}(1+n^{1+\varepsilon}/m))$。

人们已经证明 Minisat 算法的时间复杂度为$O(1.696^n)$，是指数级别的[2]。因此将 WALKSAT 融入 Minisat 的求解过程后，其最终在最坏情况下的时间复杂度仍是指数级别的。这是因为 WALKSAT 算法加入 Minisat 后，一方面为 Minisat 算法提供决策变量的赋值指导，另一方面在子句和变量规模缩小时可以用它来加速求解，但在最坏情况下仍是按照原有的 Minisat 流程来执行算法，上述两方面的作用都无法改变 Minisat 算法最坏情况下的时间复杂度是指数级别这一本质。虽然如此，但本文的算法针对那些可满足的随机 3-SAT 问题求解效果仍有一定的优越性。

4.4 小结

本章首先对改进后的算法---WM 算法思想的引入进行了详细的分析，然后结合完备算法能够进行完备求解和局部搜索算法求解快速的优点，各取其长，提出了一种基于 DPLL 的混合的 SAT 求解算法。该求解器首先利用 WALKSAT 算法预先求解出问题的近似解向量，从而指导 DPLL 算法对决策变量进行赋值。给出的 WM 算法的实现流程，并将其运用到具体的 SAT 实例求解中。本文选取了一些难的 SAT 实例进行对比测试，给出了改进前后算法的运行时间对比情况。最后，给出了 WM 算法的时间复杂度分析。

完备算法能够证明给定的 CNF 公式的不可满足性，但求解速度相对较慢；而局部搜索算法在公式是可满足的情况下，对于随机产生的许多难的 SAT 问题能较

快的给出解并且很容易实现。本章通过结合二者的优点，构造了一个效率更高的混合 SAT 求解器。通过对选取的 SAT 实例进行仿真，实验结果表明算法对于求解随机 3-SAT 问题，性能有了一定的改善和提高。

随机 3-SAT 问题是一类非常难解的 SAT 问题，无论对于完备算法来说还是对于局部搜索算法而言都是一大挑战，因此能求解出该区域实例的算法意义重大。本文将 WALKSAT 算法融合到 DPLL 算法中，这一举措大大减少了难的随机 3-SAT 实例求解过程中的决策次数和冲突次数，使得算法的运行时间减少，求解效率得以提高。

第五章 WM 算法的具体应用

SAT 问题是理论计算机科学要研究的重要问题，在许多的实际领域都有应用，比如图论、数理逻辑、模型检测[34]、EDA 领域等。这些问题都可以通过一些转换表示为 SAT 问题的形式进而运用 SAT 算法来进行求解。本文主要将其应用在图论中图染色问题的求解，通过算法求解得到一个正确的图染色方案。

5.1 四色问题简介

给定图$G=(V,E)$，称映射$\pi:V\to\{1,2,\cdots,k\}$为图G的一个k着色，$\{1,2,\cdots,k\}$为色集，其中，对G中任意两个相邻顶点u和v均满足$\pi(u)\neq\pi(v)$。k最小的一种着色方案称为图G的最小着色。找到图G的最小着色数k和对应的着色方案是我们的最终目标。

四色问题又称“四色猜想”，是图着色问题的一种特例（k值取 4），它是世界近代三大数学难题之一。四色问题的内容是：对地图上的每一国家涂上一种颜色，仅用四种颜色就能使所有相邻的国家染上不同的颜色。表示为数学语言，即将某一平面任意划分为许多不相重叠的区域，每一区域都能用[1,2,3,4]这四个数字中的一个来标记，而不能使任意相邻的两个区域的数字相同。如图 5-1 所示。

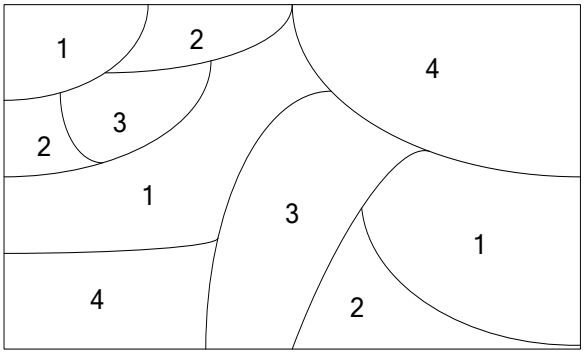

图5-1 四色图示例

四色猜想最初是在 1852 年由英国学者弗南西斯·格思里提出的，他在研究图染色工作时发现了一种有趣的现象：每幅地图都可以用四种颜色着色，使得有共同

边界的国家都被着上不同的颜色。之后，许多学者都开始考虑能否从数学的角度对这一猜想进行严格的证明。高速数字计算机的发明，促使更多的数学家对四色问题进行研究。最终，美国伊利诺大学哈肯与阿佩尔在 1976 年合作编制了一个很好的程序，在两台不同的电子计算机上用了 1200 小时，作了 100 亿次判断，完成了四色定理的证明，但其证明过程缺少了传统数学证明的简洁、精巧和智慧，对它的评价仍是众说纷纭。

可以使用四色问题来有效地解决学校排课表问题、航空班机日程表设计和计算机的编码程序设计等实际问题。比如需要制定一张理想的学校排课表，这个问题可以表示为：设有 m 位教师 $x_1, x_2, \cdots, x_m$ 和 n 个班级 $y_1, y_2, \cdots, y_n$，已知一周内教师 x_i 需要给班级 y_j 上 k_{ij} 节课。若将上一节课所用的时间称为一个课时。问如何制定一张课时数尽可能少的学校课表？（假定在同一课时内一位教师仅能为一个班上课，一个班只能有一个教师上课）转换为着色问题就是在满足任意两个相邻的顶点染不同颜色的条件下，找到颜色最少的着色方案。这样一来就将排课表问题转换为了图着色问题。

接下来，本章将重点介绍下如何将改进后的 SAT 求解算法 WM 应用到四色图的染色问题，主要包括 SAT 编码转换和应用 WM 算法对其求解来实现的。

5.2 四色问题到 SAT 问题的转换

为方便问题的研究，本文以中国地图（图 5-2）为例对四色问题进行研究。给每一个省涂上一种颜色，任意两个邻接的省不能着相同的颜色，问题转换为求对应的着色方案。

图5-2 中国地图—各省份划分

首先将该问题表示为 WM 算法可以接受的合取范式的形式。

由于中国地图中有 34 个省（标号为 0~33），而每个省有四种可能的颜色取值（设为红黄蓝绿），因此需要 136 个原子变量来构造合取范式，假设这些原子变量分别是 $x_1, x_2, \cdots, x_{13}$，每连续的四个变量表示某个省的四种着色取值，x_{4i+1}（$0 \le i \le 33$）可取值为 1 或 0，取值为 1 表示第 i 个省着 1 号色（红色），为 0 表示该省不着红色；x_{4i+2} 取值为 1 表示第 i 个省着 2 号色（黄色），为 0 表示该省不着黄色；x_{4i+3} 为 1 表示第 i 个省着 3 号色（蓝色），为 0 表示该省不着蓝色；x_{4i+4} 表示第 i 个省着 4 号色（绿色），为 0 表示该省不着绿色。

例如，$x_1x_2x_3x_4$ 表示第 0 个省对应的四个变量，x_1 取值为 1，表示第 0 号省着红色，取值为 1 表示第 0 号省不着红色。同理，x_2 取值为 1，表示第 0 号省着黄色，取值为 0 表示第 0 号省不着黄色。依次类推，x_3 为 1 表示第 0 号省着蓝色，为 0 表示第 0 号省不着蓝色。x_4 为 1 表示第 0 号省着绿色，为 0 表示第 0 号省不着绿色。

各省对应的编号如表 5-1 所示。

表5-1 各省份对应的编号

编号	省份	编号	省份	编号	省份
0	新疆	12	陕西	24	北京
1	西藏	13	湖北	25	天津
2	甘肃	14	湖南	26	山东
3	青海	15	广东	27	江苏
4	四川	16	香港	28	上海
5	重庆	17	澳门	29	浙江
6	云南	18	山西	30	台湾
7	内蒙古	19	河南	31	辽宁
8	宁夏	20	安徽	31	吉林
9	贵州	21	江西	33	黑龙江
10	广西	22	福建		
11	海南	23	河北		

对各省变量在 CNF 公式中所作的限制或约束主要有两部分组成，以毗邻别的省份较少的黑龙江省（第 33 个省）为例，$x_{133}x_{134}x_{135}x_{136}$是该省对应的四种着色变量，它首先应该满足不能同时染红、黄、蓝、绿这四种颜色，因此 CNF 公式应加入如下的子句：

$$(x_{133}\vee x_{134}\vee x_{135}\vee x_{136})\wedge(\neg x_{133}\vee\neg x_{134})\wedge(\neg x_{133}\vee\neg x_{135})$$
$$\wedge(\neg x_{133}\vee\neg x_{136})\wedge(\neg x_{134}\vee\neg x_{135})\wedge(\neg x_{134}\vee\neg x_{136})\wedge(\neg x_{135}\vee\neg x_{136})$$

其次，它毗邻吉林省（第 32 个省，$x_{129}x_{130}x_{131}x_{132}$是该省的四个着色变量）和内蒙古省（第 7 个省，$x_{29}x_{30}x_{31}x_{32}$是该省的四个着色变量），应满足该省和这两个省着色不同，因此应加入如下子句：

$$(\neg x_{133}\vee\neg x_{129})\wedge(\neg x_{134}\vee\neg x_{130})\wedge(\neg x_{135}\vee\neg x_{131})\wedge(\neg x_{136}\vee\neg x_{132})$$
$$\wedge(\neg x_{133}\vee\neg x_{29})\wedge(\neg x_{134}\vee\neg x_{30})\wedge(\neg x_{135}\vee\neg x_{31})\wedge(\neg x_{136}\vee\neg x_{32})$$

按照上述规则可以生成一个 CNF 文件，该公式中的变量数为 136，子句数为 522。

5.3 应用 WM 算法进行求解

生成 CNF 文件后，便可将其输入到 WM 求解器中进行求解，便可得出该问题的一组解，这个解是一个从 x_1 到 x_{136} 的解向量，然后对该解向量进行解析就可以映射到对应省份的着色值。求解结果如图 5-3 所示：

```
total elapsed seconds = 0.031000 sec
ASSIGNMENT FOUND
x1=0 x2=0 x3=0 x4=1 x5=1 x6=0 x7=0 x8=0 x9=0 x10=0 x11=1 x12=0 x13=0 x14=1 x15=0
 x16=0 x17=0 x18=0 x19=0 x20=1 x21=1 x22=0 x23=0 x24=0 x25=0 x26=0 x27=1 x28=0 x
29=1 x30=0 x31=0 x32=0 x33=0 x34=0 x35=0 x36=1 x37=0 x38=1 x39=0 x40=0 x41=1 x42
=0 x43=0 x44=0 x45=0 x46=0 x47=0 x48=1 x49=0 x50=1 x51=0 x52=0 x53=0 x54=0 x55=0
 x56=1 x57=0 x58=0 x59=1 x60=0 x61=0 x62=0 x63=0 x64=1 x65=1 x66=0 x67=0 x68=0 x
69=0 x70=1 x71=0 x72=0 x73=0 x74=0 x75=1 x76=0 x77=1 x78=0 x79=0 x80=0 x81=0 x82
=0 x83=1 x84=0 x85=1 x86=0 x87=0 x88=0 x89=0 x90=1 x91=0 x92=0 x93=0 x94=0 x95=0
 x96=1 x97=0 x98=1 x99=0 x100=0 x101=0 x102=0 x103=1 x104=0 x105=0 x106=1 x107=0
 x108=0 x109=1 x110=0 x111=0 x112=0 x113=0 x114=1 x115=0 x116=0 x117=0 x118=0 x1
19=0 x120=1 x121=0 x122=0 x123=1 x124=0 x125=0 x126=1 x127=0 x128=0 x129=0 x130=
0 x131=0 x132=1 x133=0 x134=0 x135=1 x136=0
```

图5-3　WM 算法求解四色问题结果

图 5-3 可以看出，用 WM 算法求解四色问题所需时间仅为 0.031s，此外，得到的取值为 1 的变量有 $x_4, x_5, x_{11}, x_{14}, x_{20}, x_{21}, x_{27}, x_{29}, x_{36}, x_{38}, x_{41}, x_{48}, x_{50}, x_{56}, x_{59}, x_{64}, x_{65}$ $x_{70}, x_{75}, x_{77}, x_{83}, x_{85}, x_{90}, x_{96}, x_{98}, x_{103}, x_{106}, x_{109}, x_{114}, x_{120}, x_{123}, x_{126}, x_{132}, x_{135}$。解析这些值为 1 的变量，就可以得到对应的着色方案。比如，x_{11} 取值为 1，即 $x_{2\times4+3}$ 为 1，表明编号为 2 的省对应的着色为 3 号色（黄色）。

那么按照上面描述的解析方法，对取值为 1 的各变量进行解析之后，各编号对应的染色方案为：

（1）着 1 号色（红）的省份编号有：1，5，7，10，16，19，21，27

（2）着 2 号色（黄）的省份编号有：3，9，12，17，22，24，26，28，31

（3）着 3 号色（蓝）的省份编号有：2，6，14，18，20，25，30，33

（4）着 4 号色（绿）的省份编号有：0，4，8，11，13，15，23，29，32

根据表 5-1 中的编号--省份对应关系，转换为对应的省份之后，确定的最终方案见表 5-2。

表5-2 各省份最终着色

省份	颜色	省份	颜色	省份	颜色
新疆	绿	陕西	黄	北京	黄
西藏	红	湖北	绿	天津	蓝
甘肃	蓝	湖南	蓝	山东	黄
青海	黄	广东	绿	江苏	红
四川	绿	香港	红	上海	黄
重庆	红	澳门	黄	浙江	绿
云南	蓝	山西	蓝	台湾	蓝
内蒙古	红	河南	红	辽宁	黄
宁夏	绿	安徽	蓝	吉林	绿
贵州	黄	江西	红	黑龙江	蓝
广西	红	福建	黄		
海南	绿	河北	绿		

5.4 小结

本章主要介绍了四色问题的基本概念和用 SAT 算法求解具体的四色问题的过程。首先挖掘四色问题中存在的一些潜在的约束条件，经过一系列的转换将这些约束条件表示为一个个析取子式的形式，最后再将所有的析取子式连接起来表示为合取范式的形式，这就将四色问题表示成了合取范式的形式，然后用 WM 算法对其进行求解得到一组解，完成了地图的着色。

容易看出，该问题的规模较小，求解过程所花费的时间也很少，我们的目标是展示如何将实际的问题编码为 CNF 形式并用 SAT 算法进行求解。今后的方向是将更大规模的实际问题进行编码转换并用 SAT 算法进行求解。

第六章 总结与展望

6.1 本文工作总结

本文在学习了 SAT 问题的基本理论知识的基础上，分析了求解 SAT 问题的完备算法和局部搜索算法这两类算法的优缺点，并对 DPLL 求解器所采用的关键技术及其算法框架进行了研究。提出了一种将随机搜索算法融入基于 DPLL 算法中的的混合 SAT 求解器。实验表明，该算法比单纯的 DPLL 算法求解速度要快，并且变量决策次数和冲突次数均有了大规模的减少。

对基于 DPLL 的算法研究过程中所作的主要工作如下：

(1) 对 SAT 问题的研究背景、意义及现状进行了简要总结，学习了命题逻辑可满足性问题的基本理论知识。

(2) 对完备算法和局部搜索算法这两类算法进行了对比分析，得出各自的优劣，通过对比说明本文将重点介绍的基于 DPLL 算法的研究价值和意义。

(3) 对基于 DPLL 的 SAT 算法进行了深入研究，并仔细分析了 DPLL 算法的实现，对该算法中所使用到的数据结构和一些关键技术给予总结和分析。

(4) 改进 DPLL 算法的整体框架和实现，给出了改进后算法的框架和实现步骤，并选取一些难的随机 SAT 实例对改进前后的算法进行了测试，实验结果表明，改进后的算法对于求解可满足的随机 3-SAT 实例效果显著。

(5) 将改进后的算法应用到图论中的实际问题--四色问题。首先挖掘四色问题中存在的一些潜在的约束条件，经过一系列的转换将这些约束条件最终表示为合取范式的形式，然后输入到改进后的 DPLL 求解器中进行求解，并最终转换为对应的着色方案。实验表明，算法在极短的时间内就得出了四色问题的解。

改进后的算法对求解随机 3-SAT 实例上虽效果显著，但是仍存在许多不足之处，如：对于求解超大规模的实例能否有好的效果呢？对于实际应用领域中的一些问题效果如何？这些都是后续研究中需要考虑的问题。

6.2 下一步工作的展望

基于 DPLL 的混合 SAT 求解器在求解性能上有了一定的改善，但是还有许多地方有待改进。现今的许多求解器在求解许多大规模的 SAT 实例时往往需要先对 CNF 公式进行预处理，经过预处理化简后导出一个简化的 CNF 公式，然后再用算法对简化后的公式进行求解，这样效率会更高一些，本文在最初的算法设计时忽略了这一点。另外，耦合性过高是本文算法实现上的不足。而且仅将算法应用到了图论中的四色问题中，并没有尝试去解决一些大规模的 SAT 实例。

下面列出了几个将来的工作重点和研究方向：

(1) 为算法加入合适的预处理器，如引入高级前向推理机制，对 CNF 公式进行化简，以便能更高效地进行求解。

(2) 可以尝试将本文中的算法应用到更大规模的实际问题中，比如硬件验证、自动化推理等领域。

(3) 算法采用的数据结构和具体实现有待改进，可以引入更巧妙的数据结构来表示文字和子句信息，也可以优化代码实现以增强程序的内聚性。

(4) 可以在 DPLL 的求解过程中寻求更高效的启发式变量决策策略，以减少算法的回溯次数和搜索空间。

(5) 可以通过对命题逻辑的可满足性问题的逻辑结构进行分析，采用一种基于解方程组的 SAT 问题求解思路。即先将合取范式转换为等价的方程组形式，然后再对方程组进行求解，进而得到对应 SAT 问题的解。

致　谢

感谢导师闫炜老师。工作上，闫老师工作认真、踏实肯干，在他的身上我懂得了一个成功人背后的艰辛；生活上，闫老师实时的了解我的状态，一次次的鼓励我、让我一次次的学会怎样在压力大的环境下快乐的生活；学习上，闫老师渊博的知识一直让我敬佩，平日里讨论问题的情景时时浮现眼前，一次次的帮我修改论文，给我讲解写硕士论文应该注意的事项，因为有了闫老师手把手的指导，我才顺利的完成了我的硕士论文。在此我向我的导师表示深深的谢意，给闫老师说声“谢谢”。

感谢推理与可信实验室的张景中老师、冯勇老师老师，对三年来我有机会聆听教诲的各位老师表示由衷的敬意。感谢鲁东大学的赵永生院长、宋丽华教授、张忠磊老师、刘婵娟老师。因为有了这么多的老师及学长的帮助与培养，才使得我的求学之路一路顺风，充满着欢声笑语。谢谢你们！

感谢我的家人。感谢爸爸妈妈，在爸爸妈妈的身上我学会了做人的道理、懂得了尊重别人、也学会了孝顺、勤劳、培养了乐观的心态；感谢我的好友高中杰。人生路漫漫，此生得此知己足矣！我会永远记住我们在一起的点点滴滴，研究生三年因为有此知己而深感荣幸！我的生活因为有了你的陪伴而显得充实、开心！现在我最想对她说的是“谢谢你，我的朋友”。

感谢我的室友张蓉蓉，学习上你帮助了我许多，谢谢你在生活中给予的关怀和帮助。感谢朱娜、高绪华、乔亚男、张培友等兄弟姐妹三年来对我的关心和照顾；我的生活因为有了你们的陪伴而显得精彩和充实，你们是我一生最宝贵的财富！在此谢谢您们！

参考文献

[1] Henry Kautz and Bart Selman. The State of SAT. In: Discrete Applied Mathematics. pp. 1514-1524, June 15, 2007

[2] Niklas Een and Niklas Sorensson. An Extensible SAT-solver. In SAT 2004. pp. 502-518. 2004.

[3] M. Davis, G. Logemann and D. Loveland. A Machine problem for theorem proving. Communications of the ACM5 (7), pp. 394-397, 1962.

[4] Hantao Zhang. SATO: an Efficient Propositional Prover. Proc. of International Conference on Automated Deduction (CADE-97), 1997.

[5] M. Moskewicz, C. Madigan, Y. Zhao, et al. Chaff: Engineering an Efficient SAT Solver [C]. Proc. of 38th Conference on Design Automation, pp. 530~535, 2001.

[6] J. W. Freeman. Improvements to Propositional Satisfiability Search Algorithms. PhD Dissertation, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[7] J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability, in: Proceedings of the ACM/IEEE International Conference on Computer-Aided Design pp. 220-227, 1996.

[8] 梁东敏，吴晔，马绍汉．一个求解结构 SAT 问题的高效局部搜索算法[J]计算机学报，1998, (S1)

[9] 金人超，黄文奇．并行计算：提高 SAT 问题求解效率的有效算法[J]．软件学报，2000,(03).

[10] 张德富，黄文奇，等．求解 SAT 问题的拟人退火算法[J]．计算机学报，2002，25（2）：148-152.

[11] 荆明娥，周电，唐璞山，周晓方．利用近似解加速求解 SAT 问题的启发式完全算法．计算机辅助设计与图形学学报. 2007, 19(9):1184~1189.

[12] 朱清新，杨凡，等，计算机算法设计与分析导论[M]．北京:人民邮电出版社, 2008.

[13] Daniel le Berre. Exploiting the real power of unit propagation lookahead. In LICS Workshop on Theory and Applications of Satisfiability Testing, June 2001.

[14] Joao P. Marques-Silva. Algebraic simplification techniques for propositional satisfiability. In Proceedings of the International Conference on Principles and Practice of Constraint

Programming, pp. 537-542, September 2000.

[15] Goldberg E, Novikov Y. BerMin: A fast and robust SAT-solver. In Proceedings of Design Automation and Test in Europe (DATE), pp. 142-149, 2002.

[16] Selman, B., Kautz, H., Cohen, B. Noise strategies for improving local search. In: Proc 12th National Conference on AL, American Association for Artificail Intelligence, pp.337-343, 1994.

[17] 黄文奇, 金人超. 求解 SAT 问题的拟人拟物方法---Solar[J]. pp 179-186. 1997.

[18] L. Baptista and J. P. Marques-Silva. Using randomization and learning to solve hard real-world instances of satisfiability, in: International Conference on Principles and Practice of Constraint Programming, ed. R.Dechter, Lecture Notes in Computer Science, pp.489-494, 2000.

[19] N. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik. Engineering and efficient SAT solver, in: Proceedings of the Design Automation Conferencepp, 530-535, 2001.

[20] 刘歆. SAT 数据结构与组合测试生成[J]. 微电子学与计算机. 2003. 2003(5): 40-42.

[21] Lynce and Ines. Propositional Satisfiability: Techniques, algorithms and applications [J]. AI Communications, pp. 187-189, 2006.

[22] H. Zhang and M. Stickel. Implementing the Davis-Putnam method, in: Proceedings of SAT 2000, pp. 309-326, 2000.

[23] L.Zhang, C. F. Madigan, M. W. Moskewicz, S. Malik. “Efficient Conflict Driven Learning in Boolean Satisfiability Solver” in Proc of the International Conference on Computer Aided Design (ICCAD), 2001.

[24] Drake, L., Frisch, A., Lynce, I., Marques-Silva, J. P. and Walsh, T. Comparing SAT preprocessing techniques [J]. In: Workshop on Automated Reasoning, April 2002.

[25] R.I.Brafman. A simplifier for propositional formulas with many binary clauses. In: Proceedings of the International Joint Conference on Artificial Intelligent (IJCAI), 2001.

[26] S. Subbarayan and D. Pradhan. NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT instances. In: Prel Proc. SAT04.

[27] Niklas Een and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In: SAT05, pp. 61-75.2005.

[28] Chu Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. In: Proceedings of AAAI-2000. pp. 291-296, July 2000.

[29] F. Bachhus and J. Winter. Effective preprocessing with Hyper-Resolution and Equality

Reduction. In: SAT03, pp. 341-355, 2003.

[30] M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of ACM7 (3), pp. 201-215, 1960.

[31] Ines Lynce and Joao P.Marques-Silva. The interaction between simplification and search in propositional satisfiability. In CP 2001 Workshop on Modelling and Problem Formulation (Formul'01), Paphos, Cyprus, December 2001.

[32] Jing Minge, Zhou Dian, Tang Pushan. Solving SAT problem by heuristic polarity decision-making algorithm [J]. Schi China SerF Inf, 2007, 50(6):915-925.

[33] R.Jeroslow, J.Wang. "Solving Propositional Satisfiability Problem", Annals of Mathematics and AI, pp. 167-187, 1990.

[34] J. P. Marques-Silva and K.A.Sakallah. Boolean satisfiability in electronic design automation. In Proceedings of the ACM/IEEE Design Automation Conference, June 2000.

[35] M. Moskewicz. Engineering an Efficient SAT Solver. [C]. Proceedings of the 3th ACM/IEEE conference on Design Automation Conference., pp. 530-535, 2001.

[36] L. Feng, L.-C Wang, K.-T Cheng, et al. A circuit SAT solver with signal correlation guided learning [C]. Proc. Of Design, Automation and Test in Europe Conference and Exhibition, pp. 892-897, 2003.

[37] K. Pipatstrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In SAT, pp. 294-299, 2007.

[38] Picosat sat solver. http://fmv.jku.at/picosat/.

[39] A. Biere. Picosat essentials. Journal on Boolean Satisfiability, Boolean Modeling and Computation (JSAT), pp. 75-98, 2005.

[40] 张健. 逻辑公式的可满足性判定[M]. 北京:科学出版社, 2000.

[41] Walksat as an informed heuristic to dpll in sat solving. http://www.cs.washington.edu/homes/bdferris/papers/WalkSAT-DPLL.pdf.

[42] Sat-Race 2010. Available from: http://baldur.iti.kit.edu/sat-race-2010/results.html.

[43] Zchaff. Available from: http://www.princeton.edu/~chaff/zchaff.

[44] Satlib-Benchmark Problems. http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

[45] 刘涛, 李国杰. 求解SAT问题的局部搜索算法及其平均时间复杂性分析[J]. 计算机学报. 1997. 20(1):18-26.

个人简历

陈稳(1985-)，女，山东省邹城市人。从 2004 年 9 月至 2008 年 6 月就读于鲁东大学网络工程专业，同期获得工学学士学位；2008 年 9 月起就读于电子科技大学计算机科学与工程学院，专业为计算机软件与理论，闫炜老师为导师，硕士生期间主要从事计算机可信推理和形式化验证方面的研究。