

面向 K-着色问题的非时序回溯 SAT 求解器设计

韩声虎

06222109

2025 年 11 月 5 日

Outline

① 引言

② K-着色问题的 SAT 编码

③ 冲突驱动子句学习 (CDCL)

④ 实现过程

⑤ 实现、优化与总结

K-着色问题 (K-Coloring Problem)

定义

给定一个无向图 $G = (V, E)$ 和一个正整数 K , 是否存在一个函数 $c : V \rightarrow \{1, 2, \dots, K\}$, 使得对于图中任意边 $\{u, v\} \in E$, 都有 $c(u) \neq c(v)$?

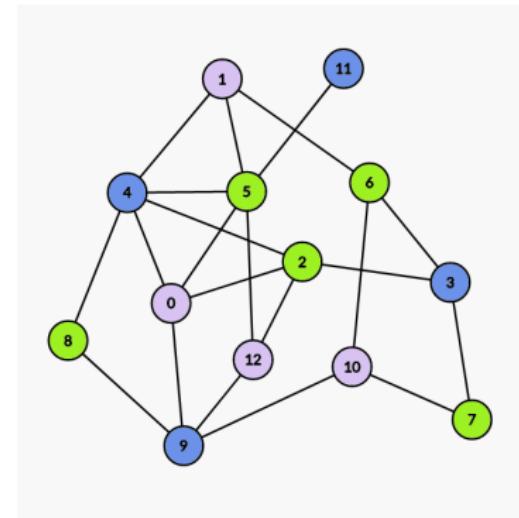


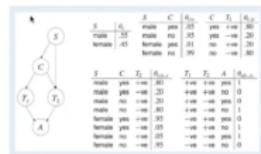
图 1: 图的 3-着色

布尔可满足性问题 (SAT)

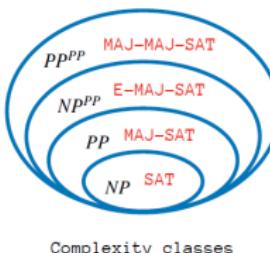
定义

给定一个布尔公式，是否存在一组对其变量的真/假赋值，使得整个公式的计算结果为真？

e.g. $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$



Combinatorial Opt. &
ML problems



Complexity classes



$((A \text{ or } B) \text{ and } (\text{not } C)) \text{ or } (\text{not } B \text{ and } D)$

Prototypical problems
(on Boolean/
Propositional formula)



Boolean Circuits

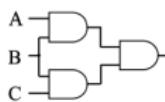


图 2: 利用 EDA 领域对布尔电路的知识高效解决 NP 问题

Outline

① 引言

② K-着色问题的 SAT 编码

③ 冲突驱动子句学习 (CDCL)

④ 实现过程

⑤ 实现、优化与总结

标准编码：变量与约束

对于一个有 n 个顶点和 k 种颜色的图，我们定义 $n \times k$ 个布尔变量：

- **命题变量：** $x_{v,c}$ 为真，当且仅当顶点 v 被赋予颜色 c 。

三类约束子句

- ① 每个顶点至少一种颜色 (At-Least-One-Color):

对每个顶点 $v \in V$, 添加子句: $(x_{v,c_1} \vee x_{v,c_2} \vee \cdots \vee x_{v,c_k})$

- ② 每个顶点至多一种颜色 (At-Most-One-Color):

对每个顶点 v 和每对不同颜色 c_1, c_2 , 添加子句: $\neg(x_{v,c_1} \wedge x_{v,c_2})$

- ③ 相邻顶点颜色不同 (Adjacency Constraint):

对每条边 $\{u, v\} \in E$ 和每种颜色 c , 添加子句: $\neg(x_{u,c} \wedge x_{v,c})$

标准 CNF 公式

$$(x_{v,c_1} \vee x_{v,c_2} \vee \cdots \vee x_{v,c_k}) \wedge \cdots \wedge (\neg x_{v,c_1} \vee \neg x_{v,c_2}) \wedge \cdots \wedge (\neg x_{v,c_{k-1}} \vee \neg x_{v,c_k}) \wedge \cdots \wedge (\neg x_{u,c} \vee \neg x_{v,c})$$

Outline

① 引言

② K-着色问题的 SAT 编码

③ 冲突驱动子句学习 (CDCL)

④ 实现过程

⑤ 实现、优化与总结

DPLL 时序回溯算法

```
while not all variables assigned in assignment do
    (status,  $C_{conflict}$ )  $\leftarrow$  BCP(assignment) ;
    if status == CONFLICT then
        |    $\beta \leftarrow 1$  ;
        |   if  $\beta < 0$  then
        |       |   return UNSAT ;
        |       end
        |       Backjump(assignment,  $\beta$ ) ;
        |   end
    else if all variables assigned in assignment then
        |   return SAT ;
    end
    else
        |   Decide(assignment) ;
    end
end
```

Algorithm 1: DPLL

冲突驱动子句学习实现非时序回溯

```
while not all variables assigned in assignment do
    (status,  $C_{conflict}$ )  $\leftarrow$  BCP(assignment) ;
    if status == CONFLICT then
        ( $\beta$ ,  $C_{learned}$ )  $\leftarrow$  AnalyzeConflict( $C_{conflict}$ ) ;
        AddClauseToDatabase( $C_{learned}$ ) ;
        if  $\beta < 0$  then
            | return UNSAT ;
        end
        Backjump(assignment,  $\beta$ ) ;
    end
    else if all variables assigned in assignment then
        | return SAT ;
    end
    else
        | Decide(assignment) ;
    end
end
```

Algorithm 2: Conflict-Driven Clause Learning

回溯策略对比

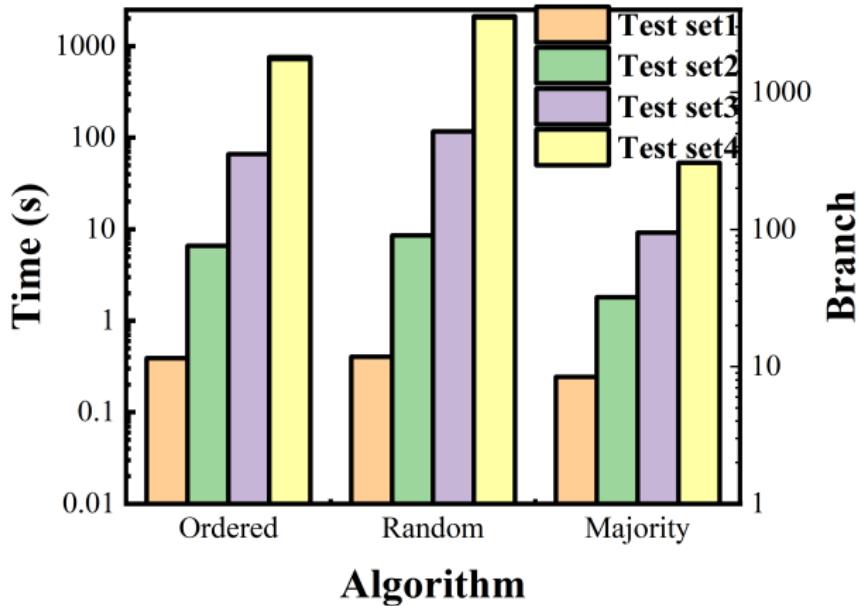
特性	时序回溯 (DPLL)[1]	非时序回溯 (CDCL)[2]
核心动作	撤销最近的决策	分析冲突以找到其根本原因
学习机制	无，冲突信息被遗忘	生成一个新的“学习子句”
跳转目标	上一个决策层级	可能跳过多个层级，直接回到相关的决策层
效率	可能重复探索失败的子树	通过学习到的子句剪除大片搜索空间

[1]Martin Davis and Hilary Putnam. 1960. A Computing Procedure for Quantification Theory. J. ACM 7, 3 (July 1960), 201–215.

<https://doi.org/10.1145/321033.321034>

[2]J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability," in IEEE Transactions on Computers, vol. 48, no. 5, pp. 506-521, May 1999, <https://doi.org/10.1109/12.769433>.

启发式决策算法对比



Test set

- Test set1:
20 变量, 91 子句
- Test set2:
50 变量, 218 子句
- Test set3:
75 变量, 325 子句
- Test set4:
100 变量, 430 子句

图 3: 三种启发式算法的求解时间(左 Y 轴)、平均分支(右 Y 轴)对比

Outline

① 引言

② K-着色问题的 SAT 编码

③ 冲突驱动子句学习 (CDCL)

④ 实现过程

⑤ 实现、优化与总结

SAT 求解器工作示例

$$(\neg x_2 \vee \neg x_{11} \vee \neg x_{12}) \wedge (\neg x_{10} \vee \neg x_{11} \vee x_{12})$$

```
LAPTOP-QE6HURH1 E:\proj\GCP_SAT_SOLVER base 3.11.5 main 97ms 10:14 PM
$ hsh python -m test.test
[_init_][INFO]: ===== create pysat from test/test5.cnf =====
[solve][INFO]: -----decision level: 1 -----
[solve][INFO]: picking 10 to be TRUE
[solve][DEBUG]: branching variables: {1: 10}
[solve][DEBUG]: propagate variables: {1: deque([])}
[solve][DEBUG]: learnts:
set()
[solve][INFO]: -----decision level: 2 -----
[solve][INFO]: picking 2 to be TRUE
[solve][DEBUG]: branching variables: {1: 10, 2: 2}
[solve][DEBUG]: propagate variables: {1: deque([]), 2: deque([])}
[solve][DEBUG]: learnts:
set()
[solve][INFO]: -----decision level: 3 -----
[solve][INFO]: picking 11 to be TRUE
[solve][DEBUG]: branching variables: {1: 10, 2: 2, 3: 11}
[solve][DEBUG]: propagate variables: {1: deque([]), 2: deque([]), 3: deque([])}
[solve][DEBUG]: learnts:
set()
[solve][INFO]: level reset to 2
[solve][DEBUG]: learnt: frozenset({-11, -2, -10})
[backtrack][DEBUG]: backtracking to 2
[solve][DEBUG]: propagate variables: {1: deque([]), 2: deque([])}
[solve][DEBUG]: learnts:
{frozenset({-11, -2, -10})}
[solve][INFO]: -----decision level: 3 -----
[solve][INFO]: picking 12 to be TRUE
[solve][DEBUG]: branching variables: {1: 10, 2: 2, 3: 12}
[solve][DEBUG]: propagate variables: {1: deque([]), 2: deque([-11]), 3: deque([])}
[solve][DEBUG]: learnts:
{frozenset({-11, -2, -10})}
[run][INFO]: Equation is SAT, resolved in 0.00 s
```

图 4: 求解过程

DIMACS 数据集解析

```
c Leighton graph
c data structure : sparse
c graph gen seed : 0
c number of vertices : 450
c max number of edges: 50000
c number of classes : 5
c a c m : 8401 6859 84035
c clique vector : clique sz num cliques
c
c      2    1890
c      3    877
c      4    540
c      5    175
c Leighton's proof: 5 coloring
c
c      Graph Stats
c number of vertices : 450
c nonisolated vertices: 450
c number of edges : 5714
c edge density : 0.056560
c max degree : 42
c avg degree : 25.40
c min degree : 13
p edge 450 5714
e 1 330
e 1 367
e 1 389
e 1 440
e 1 188
e 1 384
e 1 105
e 1 97
e 1 368
e 1 54
e 1 63
e 1 269
e 1 220
```



```
p cnf 1800 26006
1 2 3 4 0
5 6 7 8 0
9 10 11 12 0
13 14 15 16 0
17 18 19 20 0
21 22 23 24 0
25 26 27 28 0
29 30 31 32 0
33 34 35 36 0
37 38 39 40 0
41 42 43 44 0
45 46 47 48 0
49 50 51 52 0
53 54 55 56 0
57 58 59 60 0
61 62 63 64 0
65 66 67 68 0
69 70 71 72 0
73 74 75 76 0
77 78 79 80 0
81 82 83 84 0
85 86 87 88 0
89 90 91 92 0
93 94 95 96 0
97 98 99 100 0
101 102 103 104 0
105 106 107 108 0
```

```
p cnf 2250 33520
1 2 3 4 5 0
6 7 8 9 10 0
11 12 13 14 15 0
16 17 18 19 20 0
21 22 23 24 25 0
26 27 28 29 30 0
31 32 33 34 35 0
36 37 38 39 40 0
41 42 43 44 45 0
46 47 48 49 50 0
51 52 53 54 55 0
56 57 58 59 60 0
61 62 63 64 65 0
66 67 68 69 70 0
71 72 73 74 75 0
76 77 78 79 80 0
81 82 83 84 85 0
86 87 88 89 90 0
91 92 93 94 95 0
96 97 98 99 100 0
101 102 103 104 105 0
106 107 108 109 110 0
111 112 113 114 115 0
116 117 118 119 120 0
121 122 123 124 125 0
126 127 128 129 130 0
131 132 133 134 135 0
136 137 138 139 140 0
141 142 143 144 145 0
```

图 5: le450_5a.col 数据集

图 6: $k = 4$ CNF 公式

图 7: $k = 5$ CNF 公式

求解 K-着色问题

```
[solve][INFO]: level reset to -1  
[solve][DEBUG]: learnt: None  
[run][INFO]: Equation is UNSAT, resolved in 35.24 s
```

图 8: 使用我设计的求解器求解 k=4

```
c ---- [ statistics ] .....  
c  
c conflicts: 66 1775.44 per second  
c decisions: 56 1.25 per conflict  
c factored: 1895 100 % variables  
c learnt-clauses: 4 0 % variables  
c propagations: 33174 908400 per second  
c selected: 0 0 interval  
c  
c ---- [ glue usage ] .....  
c  
c focused glue 2 used 17 clauses 51.12% accumulated 57.58% tier1  
c focused glue 3 used 12 clauses 34.56% accumulated 93.84% tier2  
c  
c ---- [ resources ] .....  
c  
c maximum-resident-set-size: 28451320 bytes 27 MB  
c process-time: 0.83 seconds  
c  
c ---- [ shutting down ] .....  
c  
c exit 20
```

图 9: 使用 Kissat 求解 k=4

```
[run][INFO]: Equation is SAT, resolved in 141.97 s
```

图 10: 使用我设计的求解器求解 k=5

```
c ---- [ statistics ] .....  
c  
c conflicts: 5257 29700.99 per second  
c decisions: 7614 2.27 per conflict  
c factored: 2097 100 % variables  
c propagations: 1677120 1400000 per second  
c reductions: 2 3470 interval  
c selected: 2 3470 interval  
c learnt-clauses: 146 26 3176 interval  
c selected: 3 3119 interval  
c violations: 326 26 % check  
c solved: 1 1397 interval  
c  
c ---- [ glue usage ] .....  
c  
c focused glue 4 used 135 clauses 17.92% accumulated 52.00% tier1  
c focused glue 5 used 135 clauses 10.49% accumulated 72.53% tier2  
c focused glue 6 used 135 clauses 10.49% accumulated 78.53% tier3  
c focused glue 7 used 142 clauses 8.11% accumulated 84.62% tier4  
c focused glue 8 used 180 clauses 6.23% accumulated 91.05% tier5  
c  
c static glue 4 used 351 clauses 31.07% accumulated 69.38% tier1  
c static glue 5 used 351 clauses 12.03% accumulated 81.41% tier2  
c static glue 6-9 used 421 clauses 12.03% accumulated 83.76% tier3  
c static glue 10 used 399 clauses 5.74% accumulated 89.50% tier4  
c static glue 11 used 373 clauses 5.26% accumulated 91.76% tier5  
c  
c ---- [ resources ] .....  
c  
c maximum-resident-set-size: 20845520 bytes 27 MB  
c process-time: 0.11 seconds  
c  
c ---- [ shutting down ] .....  
c  
c exit 10
```

图 11: 使用 Kissat 求解 k=5

线性规划

从 $k = 2$ 开始递增 k 进行求解，直到 SAT，即可获得 K-着色问题答案

Outline

① 引言

② K-着色问题的 SAT 编码

③ 冲突驱动子句学习 (CDCL)

④ 实现过程

⑤ 实现、优化与总结

设计总结与结论

求解器完整架构

- ① 输入：图 G 的 K -着色问题。
- ② 编码：将 G 转换为 CNF 公式 F ，并加入对称性破除断言。
- ③ 求解器核心 (CDCL)：
 - 决策：启发式选择变量。
 - 传播：使用双观察文字方案执行 BCP。
 - 冲突时：学习新子句并加入 F ，执行非时序回溯。
- ④ 输出：SAT (存在 k -着色方案) 或 UNSAT (不存在 k -着色方案)。

结论

现代 SAT 求解器远非简单的回溯引擎。通过将 K -着色这样的 NP-hard 问题编码为 SAT，我们可以利用这种高度优化的通用技术，解决那些曾经被认为无法处理的难题。

Thanks for Listening.

Q&A

https://github.com/EasyMoneyTiger/GCP_SAT_SOLVER.git