School of Computer Science, Sun Yat-sen University

E05 SAT Problem

19335016 HaoRan Chen

2021 年 11 月 2 日

目录

1 SAT

In logic and computer science, the Boolean satisfiability problem (SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula.

In other words, it asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.

If this is the case, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is unsatisfiable.

$$\text{Example: } (b \vee c) \wedge (\neg a \vee \neg d) \wedge (\neg b \vee d)$$

图 1: SAT Problem

2 Tasks

1. Please solve the problem in `data.txt`.
2. Write the related codes and take a screenshot of the running results in the file named E05_YourNumber.pdf, and send it to ai_course2021@163.com.

3 Codes

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

图 2: 爬山算法

在具体实现中, 我选择邻居的思路是贪心优先选择最优邻居, 进行递归.

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <algorithm>
#include <bitset>
#include <windows.h>
using namespace std;

int var_num, clause_num;
vector<vector<int> > var2clause;
vector<vector<int> > clause2var;
LARGE_INTEGER start_time, end_time, frequency;

int check_clause(vector<int>& c2v, vector<int>& var_value) {
    for (int num:c2v) {
        if ((num > 0 &&  var_value[abs(num)-1]) ||
        (num < 0 && !var_value[abs(num)-1])) {
            return 1;
        }
    }
    return 0;
}

struct State{
    vector<int> var_value;    // 变量值列表，找到一组变量赋值使得clause真值表全为1
    vector<int> clause_truth; // clause真值表
    State(){
        var_value.resize(var_num);
        clause_truth.resize(clause_num);
```

```
    }
};

bool cmp(State a, State b){
    int n1 = count(a.clause_truth.begin(), a.clause_truth.end(), 1);
    int n2 = count(b.clause_truth.begin(), b.clause_truth.end(), 1);
    return  n1 > n2;
}

vector<int> res_clause_truthfulness, res_var_value;
void serach(State *neiberState, State oriState){
    int true_num = count(oriState.clause_truth.begin(),
    oriState.clause_truth.end(), 1);
    if(true_num == clause_num){//找到全局最优解
        QueryPerformanceCounter(&end_time);
        printf("clause_truth: \n");
        for(int _=0; _<clause_num; _++){
            printf("%d ", oriState.clause_truth[_]);
        }
        printf("\nvar_value: \n");
        for(int _=0; _<var_num; _++){
            printf("%d ", oriState.var_value[_]);
        }
        long long dur_time = (end_time.QuadPart-start_time.QuadPart)
        *1000000 / frequency.QuadPart;
        printf("Total Time is %lld us\n", dur_time);
        exit(0);
    }
    for(int i = 0; i < var_num; i++){
        int n1 = count(neiberState[i].clause_truth.begin(),
        neiberState[i].clause_truth.end(), 1);
        int n2 = count(oriState.clause_truth.begin(),
        oriState.clause_truth.end(), 1);
```

```
67          if(n1 <= n2) break;
68          State nextState[var_num];
69          for(int i=0; i<var_num; i++){
70              nextState[i].var_value.assign(neiberState[i].var_value.begin(),
71              neiberState[i].var_value.end());
72              nextState[i].var_value[i] ^= 1;
73              for(int j=0; j<clause_num; j++)
74              nextState[i].clause_truth[j]=check_clause(clause2var[j],
75              nextState[i].var_value);
76          }
77          sort(nextState, nextState+var_num, cmp);
78          serach(nextState, neiberState[i]);
79      }
80  }
81
82  int main() {
83      QueryPerformanceFrequency(&frequency);
84      fstream fin; fin.open("data.txt");
85      // 读取文件头
86      string p, cnf; fin >> p >> cnf >> var_num >> clause_num;
87
88      //printf("var_num = %d  clause_num = %d\n", var_num, clause_num);
89      // 读取文件内容对两个map进行初始化
90      var2clause.resize(var_num);
91      clause2var.resize(clause_num);
92      for (int clause_id = 0; clause_id < clause_num; clause_id++) {
93          int var_id = -1;
94          fin >> var_id;
95          while (var_id != 0) {
96              clause2var[clause_id].push_back(var_id);
97              var2clause[abs(var_id)-1].push_back(clause_id);
98              fin >> var_id;
99          }
```

```
100     }
101
102     State oriState;
103     for(int i = 0; i < var_num; i++)
104     oriState.var_value[i] = (rand() % 2);
105
106     //for(int i=0; i < var_num; i++)
107     //    printf("%d ", oriState.var_value[i]);puts("");
108
109     for(int i = 0; i < clause_num; i++)
110     oriState.clause_truth[i] = check_clause(clause2var[i],
111     oriState.var_value);
112
113     //for(int i=0; i < clause_num; i++)
114     //    printf("%d ", oriState.clause_truth[i]);puts("");
115
116     int num = count(oriState.clause_truth.begin(),
117     oriState.clause_truth.end(), 1);
118     //printf("origin clause_truth count is %d\n", num);
119
120     State neiberState[var_num];
121     for(int i=0; i<var_num; i++){
122         neiberState[i].var_value.assign(oriState.var_value.begin(),
123         oriState.var_value.end());
124         neiberState[i].var_value[i] ^= 1;
125         for(int j=0; j<clause_num; j++)
126         neiberState[i].clause_truth[j]=check_clause(clause2var[j],
127         neiberState[i].var_value);
128     }
129     sort(neiberState, neiberState+var_num, cmp);
130     //贪心优先选择最好的邻居
131
132     puts("neiberState clause_truth:");
```

```
133     /*
134     for(int i=0; i<var_num; i++){
135         int _ = count(neiberState[i].clause_truth.begin(),
136         neiberState[i].clause_truth.end(), 1);
137         printf("%d ", _);
138     } puts("");
139     */
140     QueryPerformanceCounter(&start_time);
141     serach(neiberState, oriState);
142 
143     return 0;
144 }
```

4 Results

由于爬山算法解 SAT 问题性能较差, 且具有很大的随机性, 尤其是样例变量数、约束条件较多, 代码实验结果复现性较差, 在绝大多数情况下都只能得到局部最优解, 很难得到全局最优解, 在多次运行之后才终于得到相对满意的结果, 故这份代码仅作参考, 并不能保证一定得到好的结果.

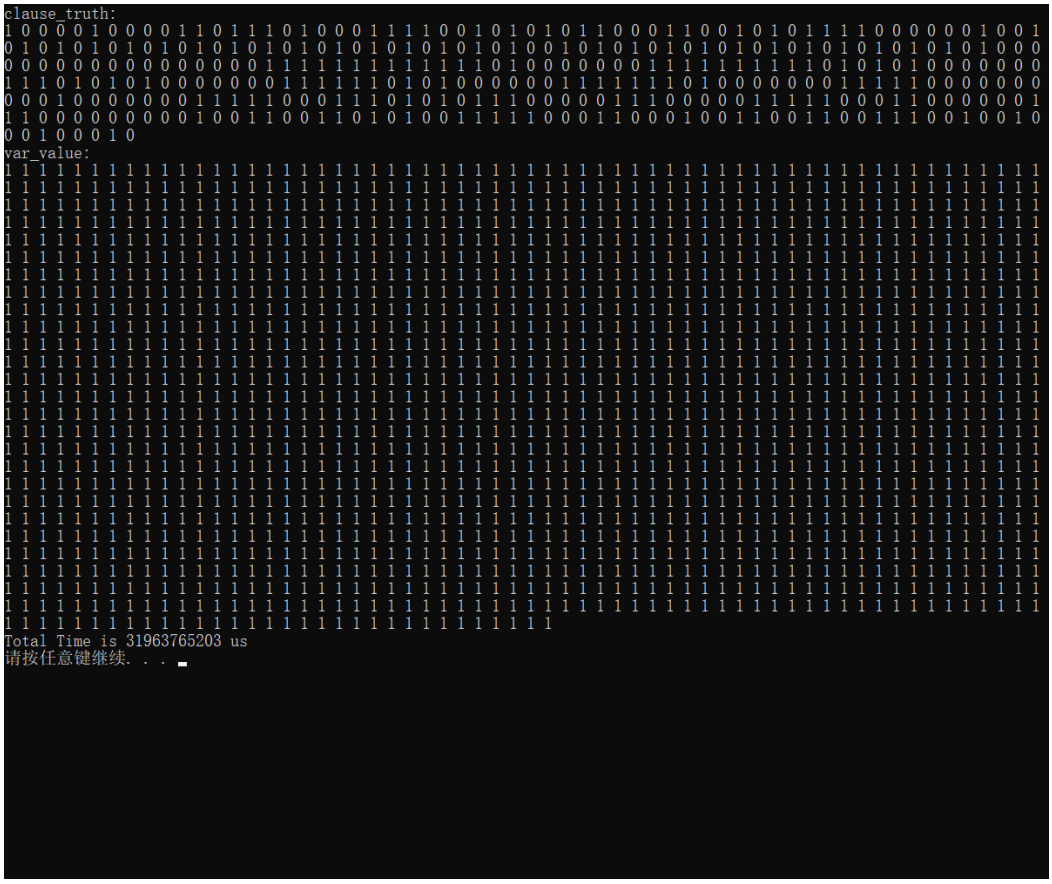
clause_truth:
var_value:
Total Time is 31963765203 us
请按任意键继续. . .

图 3: 实验结果