Project of Algorithms on Condition Satisfiability

I. The "Condition Satisfiability Problem"

Let x_1, x_2, $\cdots$, x_n be n Boolean variables, where each variable x_i can take one of only two possible values: "True" or "False". Let $\wedge$ be the "AND" operation for Boolean variables, and let $\vee$ be the "OR" operation for Boolean variables. In addition, $\overline{x_i}$ means "NOT x_i", namely, if x_i is "True" then $\overline{x_i}$ is "False", and if x_i is "False" then $\overline{x_i}$ is "True". (We also call $\overline{x_i}$ the *negation* of x_i.)

Now let us define two types of conditions:

- A "*lead-to*" condition has the form
$$(x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k}) \Rightarrow x_j$$
and it means that "*if x_{i_1}, x_{i_2}, $\cdots$, x_{i_k} are all True, then x_j is also True*".
A degenerate type of lead-to condition has $k = 0$, namely, it has the form
$$\Rightarrow x_j.$$
Its meaning is simply "*x_j is True*".
- A "*False-must-exist*" condition has the form
$$(\overline{x_{i_1}} \vee \overline{x_{i_2}} \vee \cdots \vee \overline{x_{i_k}})$$
and it means that "*among the k Boolean variables x_{i_1}, x_{i_2}, $\cdots$, x_{i_k}, at least one of them must be False*". (This is because $(\overline{x_{i_1}} \vee \overline{x_{i_2}} \vee \cdots \vee \overline{x_{i_k}})$ is True if and only if at least one of those k Boolean variables is False.)

Notice that in a *lead-to* condition, no Boolean variable is negated. But in a *False-must-exist* condition, all Boolean variables are negated.

Given a set of "*lead-to*" conditions and a set of "*False-must-exist*" conditions, our goal is to determine whether there is a solution to the n Boolean variables x_1, x_2, $\cdots$, x_n that satisfies all those conditions. If yes, such a solution is called a *satisfying solution*, and we need to return such a *satisfying solution*; otherwise, we need to return "*no satisfying solution exists*".

Let us now formally define our problem, which we shall call the "**Condition Satisfiability Problem**":

Input: We have the following inputs:

1) n Boolean variables x_1, x_2, $\cdots$, x_n.
2) A set
$$L = \{T_1, T_2, \cdots, T_P\}$$
of P "*lead-to*" conditions. For $i = 1, 2, \cdots, P$, the i-th "*lead-to*" condition T_i has $k_i \geq 0$ Boolean variables to the left of its "$\Rightarrow$" symbol and of course, has 1 Boolean variable to the right of its "$\Rightarrow$" symbol. (These $k_i + 1$ variables are all different from each other.)
3) A set
$$F = \{M_1, M_2, \cdots, M_Q\}$$
of Q "*False-must-exist*" conditions. For $i = 1, 2, \cdots, Q$, the i-th "*False-must-exist*" condition M_i is the "OR" of $m_i > 0$ negated Boolean variables. (These m_i variables are all different from each other.)

Output: If there exists a solution to x_1, x_2, $\cdots$, x_n that satisfies all the $P + Q$ conditions in L and F, return such a *satisfying solution*; otherwise, return "*no satisfying solution exists*".

We show a couple of examples:

1) Suppose there are $n = 4$ Boolean variables x_1, x_2, x_3 and x_4. Suppose $L = \{T_1, T_2, T_3, T_4, T_5\}$ has $P = 5$ "lead-to" conditions:
$$T_1: \quad (x_1 \wedge x_3 \wedge x_4) \Rightarrow x_2$$
$$T_2: \quad (x_2 \wedge x_4) \Rightarrow x_1$$

$$T_3: \quad x_2 \Rightarrow x_3$$

$$T_4: \quad \Rightarrow x_2$$

$$T_5: \quad (x_2 \wedge x_3) \Rightarrow x_1$$

Suppose $F = \{M_1, M_2\}$ has $Q = 2$ "False-must-exist" conditions:

$$M_1: \quad (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

$$M_2: \quad (\overline{x_1} \vee \overline{x_4})$$

In the above **instance** of the "Condition Satisfiability Problem", we have $k_1 = 3$, $k_2 = 2$, $k_3 = 1$, $k_4 = 0$, $k_5 = 2$ and $m_1 = 3$, $m_2 = 2$.

The above *instance* has a "satisfying solution": $x_1 =$ True, $x_2 =$ True, $x_3 =$ True, $x_4 =$ False.

2) Suppose there are $n = 4$ Boolean variables x_1, x_2, x_3 and x_4. Suppose $L = \{T_1, T_2, T_3, T_4, T_5\}$ has $P = 5$ "lead-to" conditions:

$$T_1: \quad (x_1 \wedge x_3 \wedge x_4) \Rightarrow x_2$$

$$T_2: \quad (x_2 \wedge x_4) \Rightarrow x_1$$

$$T_3: \quad x_2 \Rightarrow x_3$$

$$T_4: \quad \Rightarrow x_2$$

$$T_5: \quad (x_2 \wedge x_3) \Rightarrow x_1$$

Suppose $F = \{M_1, M_2\}$ has $Q = 2$ "False-must-exist" conditions:

$$M_1: \quad (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

$$M_2: \quad (\overline{x_4})$$

In the above **instance** of the "Condition Satisfiability Problem", we have $k_1 = 3$, $k_2 = 2$, $k_3 = 1$, $k_4 = 0$, $k_5 = 2$ and $m_1 = 3$, $m_2 = 1$.

The above *instance* has no "satisfying solution".

II. What to Submit

In this project, you need to submit three items: (1) an algorithm report, (2) a program that implements your algorithm, (3) output of your program on test instances.

A. Submission Item One: An Algorithm Report

First, you are to design an efficient algorithm for the "Condition Satisfiability Problem", and submit it as a report. The report is just like what we usually do for algorithm design in a homework, and should have the four elements as usual:

1) The main idea of your algorithm.
2) The pseudo code of your algorithm.
3) The proof of correctness for your algorithm.
4) the analysis of time complexity for your algorithm.

B. Submission Item Two: A Program That Implements Your Algorithm

You need to implement your algorithm using a commonly used programming language, such as Python, C++, Java, etc. We encourage you to use Python if possible, but other languages are fine, too.

Your program will take a list of instances of the "Condition Satisfiability Problem" as input, and return a list of solutions (corresponding to those instances) as its output.

To test your program, we will provide you with input instances as a "dictionary" in Python (after you have submitted the program), and ask you to submit the output solutions as a "list" in Python. (Details on the two files, including their formats, will be explain in the next subsection for "Submission Item Three".)

When you submit your program, you need to explain clearly how to run your code. If your programming language is not Python, then you also need to include an explanation on how to turn the provided "dictionary" of "input instances" in Python to your programming language, and how to turn your "output solutions" in your programming language to a "list" in Python (following the formats specified in the next subsection).

C. Submission Item Three: Output of Your Program on Test Instances

To test your algorithm/program, we will provide you with three sets of "input instances", all in the Python language:

1) A set of instances of relatively small sizes.
2) A set of instances of medium sizes.
3) A set of instances of relatively large sizes.

Of course, if your algorithm/program is correct, then you should get the correct solutions for all three sets of input instances. However, an algorithm is not only about *correctness*, but also about *efficiency*. If your algorithm's time complexity is low, then computing should be easy. (When we tested the algorithm in Google Colab using Python, the set of small instances took less than 1 second, the set of medium-sized instances took about 6 seconds, and the set of large instances took about 1 minute 30 seconds.) However, if your algorithm's time complexity is high, you may experience much longer running times, or may not be able to get the solutions to the large instances.

To give you an idea what the three sets of input instances are like, we provide three "example" files (downloadable from our course webpage): "examples_of_small_instances", "examples_of_medium_instances", "examples_of_large_instances". They are all Python dictionaries, whose format will be explained below. You can download them by clicking on the links in our course webpage. After downloading each file, you can open it using *pickle.load(open(filePath, 'rb'))*, where "filePath" is the path of your downloaded file. (Of course, these three files are different from the three files we will send you for testing your program. But they are similar.)

After you run your program on the three sets of "input instances", save your results as three Python "lists" in three files:

1) File "small_solutions" corresponding to the input instances of small sizes.
2) File "medium_solutions" corresponding to the input instances of medium sizes.
3) File "large_solutions" corresponding to the input instances of large sizes.

Now let's explain the formats of the "input instances" and "output solutions".

The "input instances" is a "dictionary" (in Python) that contains 8 key-value pairs:

1) *Key* is "numInstances", *value* is the number of instances here. (In the following, let's use N to denote the number of instances.)
2) *Key* is "n_list", *value* is a list of N integers, which we shall call "n_list". For $i = 0, 1, ..., N-1$, "n_list[i]" is the number of Boolean variables in the i-th instance. (Note that here the instances are indexed by $0, 1, \cdots, N-1$ instead of $1, 2, \cdots, N$.)
3) *Key* is "P_list", *value* is a list of N integers, which we shall call "P_list". For $i = 0, 1, ..., N-1$, "P_list[i]" is the number of "*lead-to*" conditions in the i-th instance.
4) *Key* is "Q_list", *value* is a list of N integers, which we shall call "Q_list". For $i = 0, 1, ..., N-1$, Q_list[i] is the number of "*False-must-exist*" conditions in the i-th instance.
5) *Key* is "k_list", *value* is a list of N items, which we shall call "k_list". For $i = 0, 1, ..., N-1$, k_list[i] is a list of P_list[i] integers. For $j = 0, 1, ...,$P_list[i] $- 1$, k_list[i][j] is the number of Boolean variables to the left of the "$\Rightarrow$" symbol in the j-th "*lead-to*" condition in the i-th instance.
6) *Key* is "m_list", *value* is a list of N items, which we shall call "m_list". For $i = 0, 1, ..., N-1$, m_list[i] is a list of Q_list[i] integers. For $j = 0, 1, ..,$Q_list[i]-1, m_list[i][j] is the number of Boolean variables in the j-th "*False-must-exist*" condition in the i-th instance.
7) *Key* is "T_list", *value* is a list of N items, which we shall call "T_list". For $i = 0, 1, ..., N-1$, T_list[i] is a list of P_list[i] items. For $j = 0, 1, ...,$P_list[i]-1, T_list[i][j] is a list of k_list[i][j]+1 integers:
   - T_list[i][j][0], T_list[i][j][1], ..., T_list[i][j][k_list[i][j]-1] are the indexes of the Boolean variables to the left of the "$\Rightarrow$" symbol in the j-th "*lead-to*" condition in the i-th instance. (Note that here the Boolean variables in the i-th instance are indexed by 0, 1, $\cdots$, n_list[i]-1, instead of indexed by 1, 2, $\cdots$, n_list[i].)
   - T_list[i][j][k_list[i][j]] is the index of the Boolean variable to the right of "$\Rightarrow$" in the j-th "*lead-to*" condition in the i-th instance.
8) *Key* is "M_list", *value* is a list of N items, which we shall call "M_list". For $i = 0, 1, ..., N-1$, M_list[i] is a list of Q_list[i] items. For $j = 0, 1, ...,$Q_list[i]-1, M_list[i][j] is a list of m_list[i][j] integers, which are the indexes of the Boolean variables in the j-th "*False-must-exist*" condition in the i-th instance.

We can see that in the above dictionary, for $i = 0, 1, \cdots, N-1$, the seven items – n_list[i], P_list[i], Q_list[i], k_list[i], m_list[i], T_list[i], M_list[i] – describe all the information we need to know about the i-th instance.

The "output solutions" is a "list" (in Python) of the following format:

- Let's use "x_list" to denote the list of N "output solutions", which correspond to the N "input instances". x_list is a list of N items. For $i = 0, 1, \cdots, N-1$, x_list[i] is the solution to the i-th instance. For the i-th instance, if there exists a *satisfying solution*, then x_list[i] represents such a *satisfying solution*: x_list[i] is a list of n_list[i] integers (remember that the i-th instance has n_list[i] Boolean variables), and for $j = 0, 1, ...,$n_list[i]-1, x_list[i][j]=0 means that the j-th Boolean variable is "False", and x_list[i][j]=1 means that the j-th Boolean variable is "True". If the i-th instance does not have any *satisfying solution*, x_list[i] is an empty list.

We give two small "example" files for the "input instances" and "output solutions". Their filenames are "examples_of_instances" and "examples_of_solutions", and you can download them by clicking the links in the course webpage. (As before, after you have downloaded a file, you can use *pickle.load(open(filePath, 'rb'))* to open it.) The file "examples_of_instances" contains $N = 10$ sample instances, and the file 'examples_of_solutions" contains 10 corresponding solutions. You can run your program on the 10 sample instances, to see if your solutions match the 10 solutions given here. (However, note that if satisfying solutions exist for an instance, the solution may not be unique.) Here are two examples:

- Among the 10 instances in "examples_of_instances", Instance 1 has the following details:
  1) n_list[1] = 5. It means there are 5 variables: x_0, x_1, x_2, x_3, x_4.
  2) P_list[1] = 6. It means there are 6 *lead-to* conditions T_0, T_1, $\cdots$, T_5.
  3) Q_list[1] = 9. It means there are 9 *False-must-exist* conditions M_0, M_1, $\cdots$, M_8.
  4) k_list[1] = [0, 3, 1, 0, 3, 0]. It means the number of Boolean variables to the left of "$\Rightarrow$" is 0, 3, 1, 0, 3, 0, respectively, for the 6 *lead-to* conditions.
  5) m_list[1] = [3, 4, 3, 4, 3, 4, 3, 3, 3]. It means the 9 *False-must-exist* conditions involve 3, 4, 3, 4, 3, 4, 3, 3, 3 negated Boolean variables, respectively.
  6) T_list[1] = [[1], [4, 2, 3, 0], [0, 2], [3], [4, 3, 2, 0], [3]]. It means the 6 *lead-to* condition are:
     - $T_0:\ \Rightarrow x_1$.
     - $T_1:\ (x_4 \wedge x_2 \wedge x_3) \Rightarrow x_0$.
     - $T_2:\ x_0 \Rightarrow x_2$.
     - $T_3:\ \Rightarrow x_3$.
     - $T_4:\ (x_4 \wedge x_3 \wedge x_2) \Rightarrow x_0$
     - $T_5:\ \Rightarrow x_3$.
  7) M_list[1] = [[1, 4, 2], [1, 2, 4, 0], [3, 2, 4], [3, 4, 0, 1], [3, 4, 0], [0, 2, 4, 3], [3, 4, 1], [2, 1, 3], [2, 1, 0]]. It means the 9 *False-must-exist* conditions are:
     - $M_0:\ (\overline{x_1} \vee \overline{x_4} \vee \overline{x_2})$
     - $M_1:\ (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4} \vee \overline{x_0})$.
     - $M_2:\ (\overline{x_3} \vee \overline{x_2} \vee \overline{x_4})$.
     - $M_3:\ (\overline{x_3} \vee \overline{x_4} \vee \overline{x_0} \vee \overline{x_1})$.
     - $M_4:\ (\overline{x_3} \vee \overline{x_4} \vee \overline{x_0})$.
     - $M_5:\ (\overline{x_0} \vee \overline{x_2} \vee \overline{x_4} \vee \overline{x_3})$.
     - $M_6:\ (\overline{x_3} \vee \overline{x_4} \vee \overline{x_1})$.
     - $M_7:\ (\overline{x_2} \vee \overline{x_1} \vee \overline{x_3})$.
     - $M_8:\ (\overline{x_2} \vee \overline{x_1} \vee \overline{x_0})$.

  Among the 10 solutions in "examples_of_solutions", Solution 1 is the solution for the above Instance 1. Solution 1 is [0, 1, 0, 1, 0]. It means that in the solution, x_0 is "False", x_1 is "True", x_2 is "False", x_3 is "True", x_4 is "False". *You can plug the above solution into the 15 conditions above to verify that indeed all the conditions are satisfied.*

- Among the 10 solutions in "examples_of_solutions", Solution 2 is an empty list. It means that Instance 2 in "examples_of_instances" has no *satisfying solution*.

III. How to Grade Project

The project has a total of 100 points. If *any* of the three items below is not submitted on time, the whole project will receive no point.

Assuming that all the three items are submitted on time, the project will be graded as follows.

A. Submission Item One: An Algorithm Report

The algorithms report has 40 points:

- 20 points for the "main idea" and the "correctness proof". They should be rigorous and clear.

- 5 points for the "pseudo code". It should be correct and clear.
- 15 points for the "time-complexity analysis". It should not only be correct and clear, but should also show that your algorithm achieves **low time-complexity**. In other words, if your algorithm is correct but does not achieve the lowest time complexity that we require, some or most of these 15 points can be deducted.

B. Submission Item Two: A Program That Implements Your Algorithm

The program has 6 points.

If your program has bugs or does not have a clear interface – which means we have to revise your code or spend time trying to understand how to run your code – some points will be deducted.

C. Submission Item Three: Output of Your Program on Test Instances

After you have submitted your "algorithm report" (submission item one) and "program" (submission item two), we will post three files: test_set_small_instances, test_set_medium_instances, test_set_large_instances. They contains instances of small, medium and large sizes, respectively. You need to run your submitted program (the exact program that you have submitted as Submission Item Two) on the three sets of instances, save your results in three files "small_solutions", "medium_solutions", and "large_solutions" and submit them. Those three files are your "Submission Item Three", and they have 54 points:

1) If *all* the solutions in "small_solutions" are correct, you receive 18 points. If *any* of those solutions is incorrect, you receive 0 point.
2) If *all* the solutions in "medium_solutions" are correct, you receive 18 points. If *any* of those solutions is incorrect, you receive 0 point.
3) If *all* the solutions in "large_solutions" are correct, you receive 18 points. If *any* of those solutions is incorrect, you receive 0 point.

Note that for an algorithm to be *correct*, it needs to be correct for all instances. So we need to be very rigorous with our designed algorithm and its implementation. That is why in the above, if any solution is incorrect in a file, all the 18 points will be lost. (So there is no partial credit here.)

Please also note that the program you use to generate the solutions here should be exactly the same program that you have submitted as "Submission Item Two". If any modification is detected (e.g., if we find that some solution submitted here is not the same as what the already-submitted program would produce), the project will receive 0 point.

IV. Hint on the Algorithm

Here is a hint on how to solve the problem: *Consider a greedy algorithm. Start with an all-False solution. Then greedily change variables from "False" to "True" based on need.*