My code consists of three parts

1. Process the input to build CNF in set of set form
2. Class definition for Partial assignment, etc
3. DPLL algorithm related functions (plus some helper functions)

* <Part 1>

I think part 1 is not worth mentioning, so I omit it.

* <Part 2>

I think it looks weird. Because I’m not used to Python, class definitions and its method look weird. Also, I usually use Scala, which has case class feature, I used class similarly with case class in Scala. I think it is not appropriate way to use the class in python, but I’m used to coding with Scala.

* Case class like class
  + I make AssignmentType class to mark that the assignment type
  + AssignmentType is somewhat similar to trait in Scala, and Decision, Implied, Free is similar to case class in Scala
  + In the field of Implied class, it stores the clause which triggered assignment
  + Also I make State class to check SAT
  + Similarly, State corresponds to trait, SAT, UNSAT, NotDetermined corresoponds to case class
* PartialAssignment
  + For storing the partial assignment, I make a class with some methods
  + First, the datastructure for assignment is ordered dictionary
  + Key is the variable index and the element is a tuple (index, AssignmentType, value)
  + Value stands for true, false, and absent (represented as 1, -1, 0 respectively)
  + Other methods are related to add/remove the element,
* <Part 3>

The most tricky part of the DPLL algorithm for me is the way to maintain the CNF. I had two approaches.

* Make CNF independent of assignment
  + Though new assignment is made, CNF doesn’t change
  + Iterate through the clause of CNF and check/get the info with assignment (not mutating the CNF), when checking the SAT, get unit clause, etc.
  + CNF is only mutated when learned clause is added
* Change the CNF whenever assignment changes
  + Whenever new assignment is made, remove the assigned variables in the CNF

Actually, both of them has problem. As learning goes farther, the first approach takes much time in iterating the CNF. In the second approach, problems are in the unit propagation and backtracking. The variables assigned by unit propagation have to know the clause being unit in the original CNF (contrast to assigned CNF). Also, when back tracking, we have to retrieve the variables. But we don’t know where the variable should locate and whether it is negated or not.

I implemented both approach to solve the SAT problem. In the second approach, the way to handle the problems was somewhat inefficient in my implementation, so it was slower than the first approach. (I used small sized test cases uf50, uuf50 uf75 uuf75). But the first approach was still slow. So, I mixed both approaches, and the result seems slightly faster (because I only use test case with formula small, I’m not certain)

* Implementation of DPLL algorithm
* Unit propagation
  + While there is unit clause, set the literal in the unit clause true
* ~~Preprocess~~
  + ~~If there are unit clause before any decision is made, I changed the CNF according to assignment. Because that assignment never changes~~
  + ~~But I’m not sure it is worth, because most time no such variable exist and the function just consumes time without effect.~~
  + It seems to slow down the solver in my implementation, so I remove it
* Reduce CNF
  + It makes new CNF that assign value to the CNF
* Check SAT
  + Then it checks satisfiability using assigned CNF
  + It checks the CNF is empty set, it contains box, or the other
* 1 if SAT
  + Return SAT
* 2 if satisfiability is not determined
  + Do decision
  + Pick any clause, pick any literal in the clause, and set the literal true
* 3 if UNSAT
  + It gets a conflict clause and perform clause learning by that clause
    - It iterates through the assignment in reverse and perform resolution if needed
    - If the learned clause is box, return UNSAT
  + While learned clause become unit clause, pop the assignment

Actually, I mostly follow the step in the lecture note.

I perform unit propagation without assigning value to the CNF, because I think, implied variable should know the corresponding unit clause. However, by using cProfile command, this unit propagation takes most time. So, if I optimize this function, I can improve my implementation.

* Result
  + I solve 6\_SAT in 1 hour
  + 7\_UNSAT doesn’t terminate in 1 hour
  + I solve 8\_UNSAT in 1 minute
  + I solve 9\_SAT in 10 sec