Knowledge Representation Lecture 4

Knowledge Representation (Vrije Universiteit Amsterdam)

KR 2021 – Lecture 4

CLAUSE LEARNING

Conflict Driven Learning and Non-chronological Backtracking

Chronological Backtracking

- Heuristic: at split with backtracking DP
- DP does this- you take a variable decide to make it T or F then you backtrack then you make another choice etc etc till you hit the end then backtrack- undo changes in the reverse order you made it
- It does well in artificial test sets (randomly generated) but we need to work with real test sets, they have much more structure- we can exploit that structure to do better than chronological backtracking
- Worst case complexity is NP-complete, is bad- unrepairable even when we are clever, instead of giving up hope we find heuristic approaches that work most of time
    - This worse case won't change
- CR you go back to the most recent choice, that is in the hope that it will fix the contradiction. However, the alternative does not solve the problem.
- So actually, you would prefer going to the top, taking nigger jumps back in the tree where you can address the cause of inconsistency
- Can we not be cleverer about this, and find where the clause is – higher in the tree?
    - Intelligent backtrack instead of chronological
        - You need to book keeping- track inconsistency and where it is

Example

- 14 minutes
- We set x1 to false
    - This makes x4 a unit clause, making it true
        - Dependent on x1 (dependency graph)
    - Then next random choice is x3, set it to true
        - Clause 5 has x3, not x3 is False
        - Now x8 has to be false
            - it is dependent is on x3 and x1 because the choices of the literals that cooccurred
            - We see this in formula 2
        - Now x12 must be true
            - It is dependent x8 and x1
    - → continue the game
        - Roots are yellow
        - Other dependent choices are blue/green
    - At x9 we have an inconsistency
        - Instead of backtracking x2, we will think hard of how far to backtrack
            - What is the real reason x9 we have a clash here? Is it x7 or another x? if we make x7 false we would still get the same contradiction
            - we make a cut between the clash and the root cause, we see that x2 is independent
                - all the link that connects clashing nodes with root nodes

- e.g. x8 is cut because x1 is

Exponential Complexity Growth: The challenge of Complex Domain

- Moorer Law & Search Space

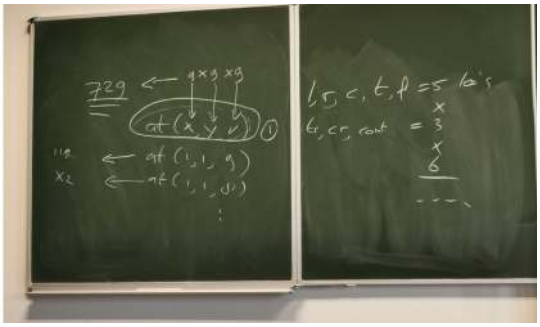
SAT Encoding

- Dimacs File
  - -1 7 0
    - Not x1, x7, 0
    - What do we do with x1?
      - Does it only appear negatively?

Chapter 7: Propositional Satisfiability Techniques

- Take a planning problem and show what it takes to do it
- Punch line: wow these problems really explode
- Motivation
  - Try translating classical planning problems into satisfiability
    - You do this for sudoku and planning
- State Transition Machine
  - You have an agent in the world (crane operator, the world is number of states)
    - S0, s1, s2, s3. S4. S5
      - Description of the world is the states
  - Daylight – solar panels are open, night – solar panels closed
  - You can perform number of actions (drive, open the thing, move action (between states))
    - Theres always a set of actions
    - State-transition function tells you that if you are in a certain state, and you perform a certain action you end in another state sx
      - All arrows in diagram
        - E.g. move 1 takes from s1 to s3, s2 to s0
  - In real world when you take an action, multiple things can happen- you are not in control
    - Not deterministic
  - But the STS world is deterministic (in this example)
    - S6s is a set of S (math speak)
- Planning problem
  - Has a state in which you find yourself and has a goal state where you want to be
    - E.g. you are in your bed but you want to be in class. Then you make a plan about how to get to the lecture by taking actions that change your states one by one
  - You can have one to one arrow or one to many arrow, this example is a simple system so the former
- Solution
  - a sequence of actions <get dressed, get breakfast, get on the bike>
- Bounded planning problem
  - Is a planning problem with maximum depth & maximum length
- Search Method

- If you have a search tree you are in trouble if some of those branches in your tree can be infinite
    - In our planning diagram if you went between s0-s1 would just go back and forth – its an infinite branch and you would never know you are doing it

- Iterative deepening
    - First we are going to explore all planning solutions of length 1, atleast we know there is no iterative processes. Then increment length by 1 (= 2), still no solution. Like this you restart.
        - You don't remember last solutions because it is exponential so you could not remember it anyway
    - It is super inefficient but it is a brute of ignoring the infinite loop
- Logic Notation
    - The planning problem you will need to add contrstaints
        - E.g. don't have the container on the truck because it is a stupid location
            - At(r1,loc1)- object 1, at location 1
        - Atomic symbol does not have T/F symbols, you need propositionalise () 9 can be at 1x1, 8 can be at 1x1
    - 
        - We made it into variable (propositional)
            - So then the only predicate we have is x2(I think)
        - Logix x by all possible combinations
- In planning we also do this at(r,loc) can also be written as propositional
- Particular state I → at(r1,location1, s3)
    - Then you can calculate how many fluents there
        - How many objects (truck, crane, container), how many locations, (left, right, crane, truck, floor), how many states (s0-s5; 6)
        - All the variables you have to count in the fluents
- You still need to encode the actions ; thus you still need to encode the rules
    - You define this by saying what is true before and after the action
        - Initially: Box on the floor & After: It is in the crane -> action is picking up a box
        - E.g. want to get to the lecture from your bed. Then the action of getting out of bed it: be in bed then be out of bed
- You cannot do two action at once
    - You cannot pick up the box and move the truck (exclusion is needed)
- Frame Action
    - You still need to describe what does not change
        - E.g. if you say pick up- it has as precondition that box is on the floor, postcondition that it is in the crane. But we also need to say that these are

the only things you use. So if you pick up the truck the box doesn't change, if you do something this doesn't change.
        - Hard to specify because we take for granted
        - E.g. when you get on the bike, your location does not change
            - Many frame actions
- Example
    - Times 0 & 1
    - You want to be at the other location
    - Operators are so many and look tiresome because we are propositionalising
- Now you want to make a plan, you take propositional coding and find a satisfying solution
    - Start with n=1 then keep on going and everything throw it away. The number of variables grows exponentially
    - Quite a silly approach so many people didn't try it in the beginning, but then in the 80s someone just did it and actually it beat the sat solver that existed