EECS 496: Sequential Decision Making

Soumya Ray

sray@case.edu

Office: Olin 516

Office hours: T 4-5:30 or by appointment

Recap

- What is sequential decision making about?
- What is the SDM loop?
- What are the different environment types?
- What is syntax? Semantics?
- How do we define propositional logic syntax?
- What is a model? What is a "model of a sentence"?
- How do we build knowledge bases with PL?
- What is entailment?
- What is inference by enumeration?
- What is a valid formula? A satisfiable formula?

Today

- Propositional Logic (Chapter 7, Russell and Norvig)
- Probability theory (Ch 13, Russell and Norvig)

Satisfiability and Inference

- Satisfiability and inference are linked by the proof-by-refutation theorem: $\alpha \models \beta$ iff $(\alpha \wedge \neg\beta)$ has no model (is unsatisfiable)

Conjunctive Normal Form (CNF)

- A formula is said to be in CNF if it is a conjunction of disjunctions

$$\underbrace{(l_{11} \vee l_{12} \ldots \vee l_{1k})}_{} \wedge \underbrace{(l_{21} \vee l_{22} \ldots \vee l_{2k})}_{\text{Clause}} \wedge \ldots$$

k-CNF

Literal: P or $\neg$ P

Clause

- Every sentence in propositional logic can be transformed into a 3CNF formula

Satisfiability as Goal-Directed Search

- Given a CNF formula, I want to find out if it is satisfiable by checking possible models

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

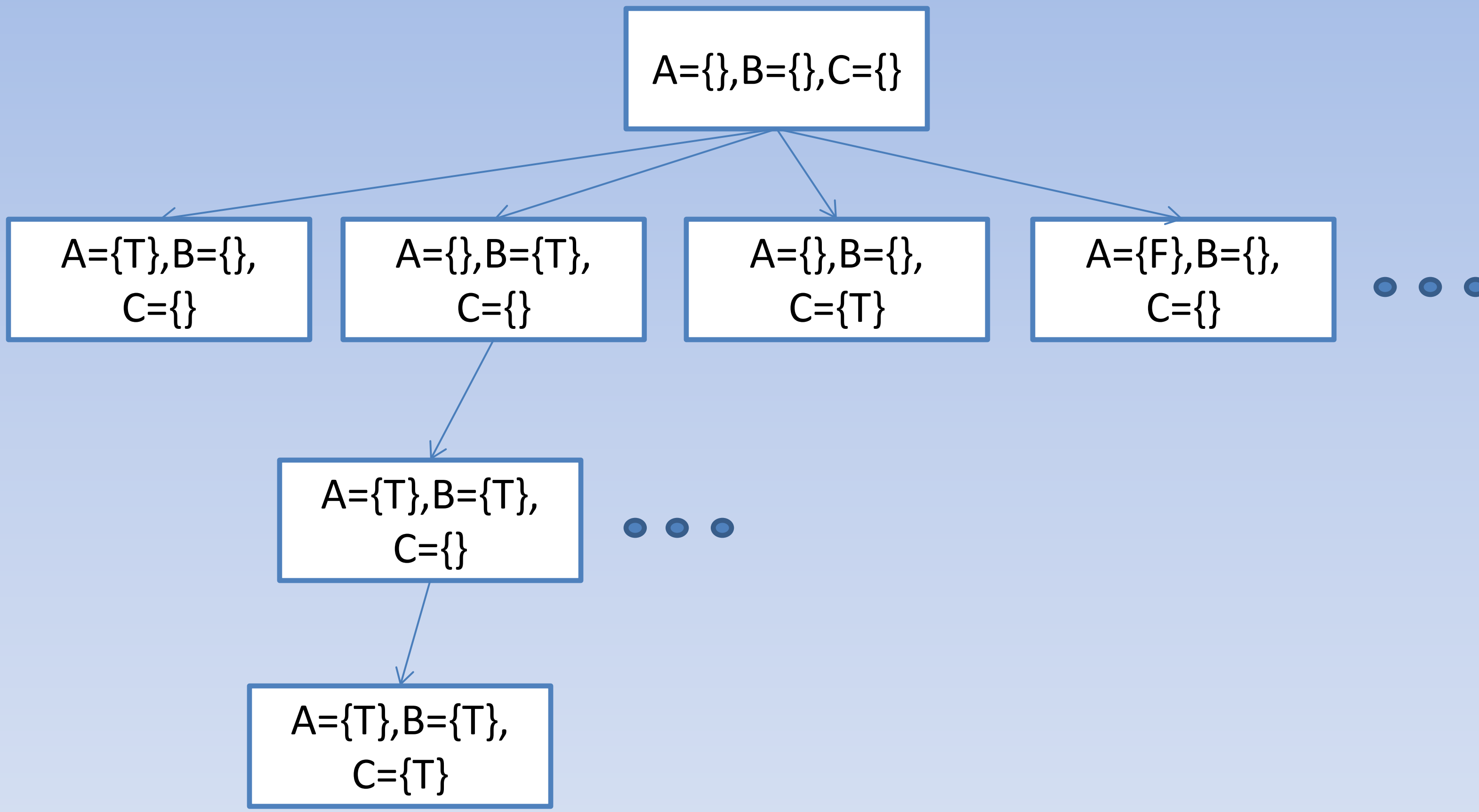
- How can we set up the problem?
  - States=partial value assignments to symbols
  - Initial state, goal state, operators, cost?

DPLL (Goal-Directed Search)

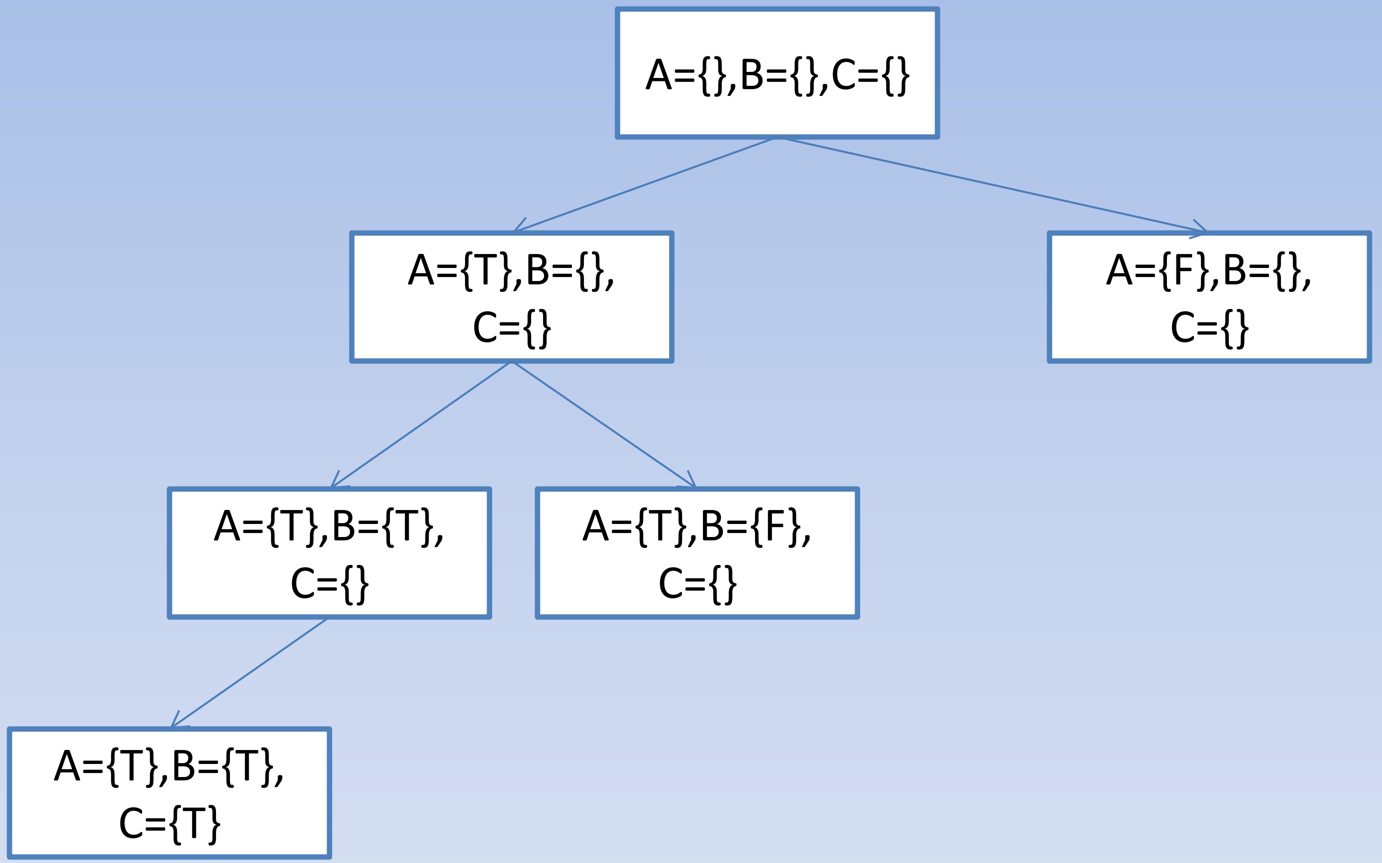
(Davis, Putnam, Logemann, Loveland 1960)

- Input: formula in CNF
- Output: satisfying assignment if any
- Algorithm: Depth first search of space of all models
  - Uses heuristics to guide search (prune search space)
  - Complete

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$



$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$



Heuristic 1: Pure literals

- Suppose a CNF formula has a symbol that appears only negated or only unnegated in all clauses
  - This is a "pure" literal
  - If the formula is satisfiable, it is satisfiable with the pure literal set to true
  - Example:

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

Heuristic 2: Unit Propagation

- Suppose a clause has only a single literal (because everything else is assigned *false*)
- Then that literal has to be set to *true*
  - This can cascade
  - Example

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

$$B = true$$

Heuristic 3: Early Termination

- Since the formula is in CNF, for any assignment to be satisfying, every clause has to be true
  - Any (partial) assignment that makes any clause false can be pruned
- Since a clause is a disjunction, once any literal is set to true, that clause does not have to be considered any more

DPLL

- Start with initial list of clauses and a current assignment (initially empty)
- If all clauses true return true
- If any clause false return false
- If :
  - There is an unassigned pure literal, set to true and call DPLL on result
  - Else if there is an unassigned unit clause, set to true and call DPLL on result
  - Else pick a random symbol, return DPLL with symbol set to true OR’ed with DPLL with symbol set to false

Satisfiability by Optimization (Local Search)

- How can we set up the problem?

$$(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$$

WalkSAT (Kautz and Selman, 1993)

- Start with initial complete assignment
- Repeat MAX_FLIPS times:
  – If all clauses true, return current assignment
  – Else, pick random unsatisfied clause
  – With probability 1-p, flip the assignment of the variable in clause that maximizes satisfied clauses
  – Else flip the assignment of a random variable in clause

Properties of WalkSAT

- Not Complete
  - If it returns no satisfying assignment, does not imply formula is unsatisfiable
  - But it turns out that depending on the formula, we can make a really good guess about this

Why is SAT hard?

- The great success of WalkSAT led to a huge amount of research on SAT
  - People knew theoretically that SAT was supposed to be a hard problem in the worst case
  - But here was a simple local search procedure which was able to solve extremely large SAT problems extremely fast
  - How to reconcile these facts? Could we isolate the "hard" SAT problems somehow? Are these an interesting subset of SAT problems?

Hard SAT Problems

- Suppose we generate a random CNF formula

- There are two key parameters
  - The number of clauses in the formula (M)
  - The number of propositional symbols (N)

- It turns out that the ratio M/N determines the hardness of a SAT problem

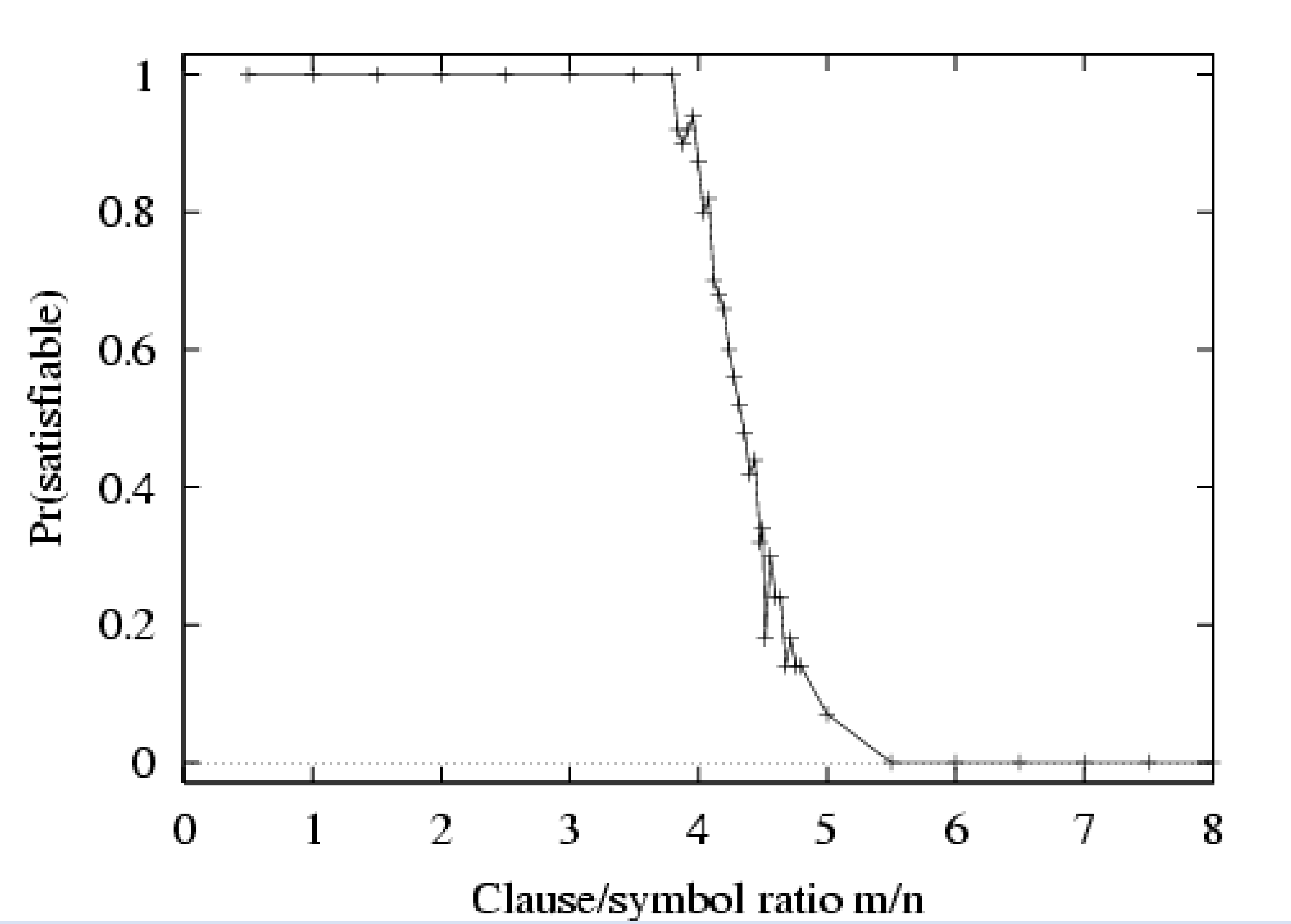
Intuition

- If there are many symbols and few clauses, the problem is *underconstrained*: the probability of a random assignment being satisfying is close to 1

- If there are few symbols and many clauses, the problem is *overconstrained*: the probability of a random assignment being satisfying is close to 0

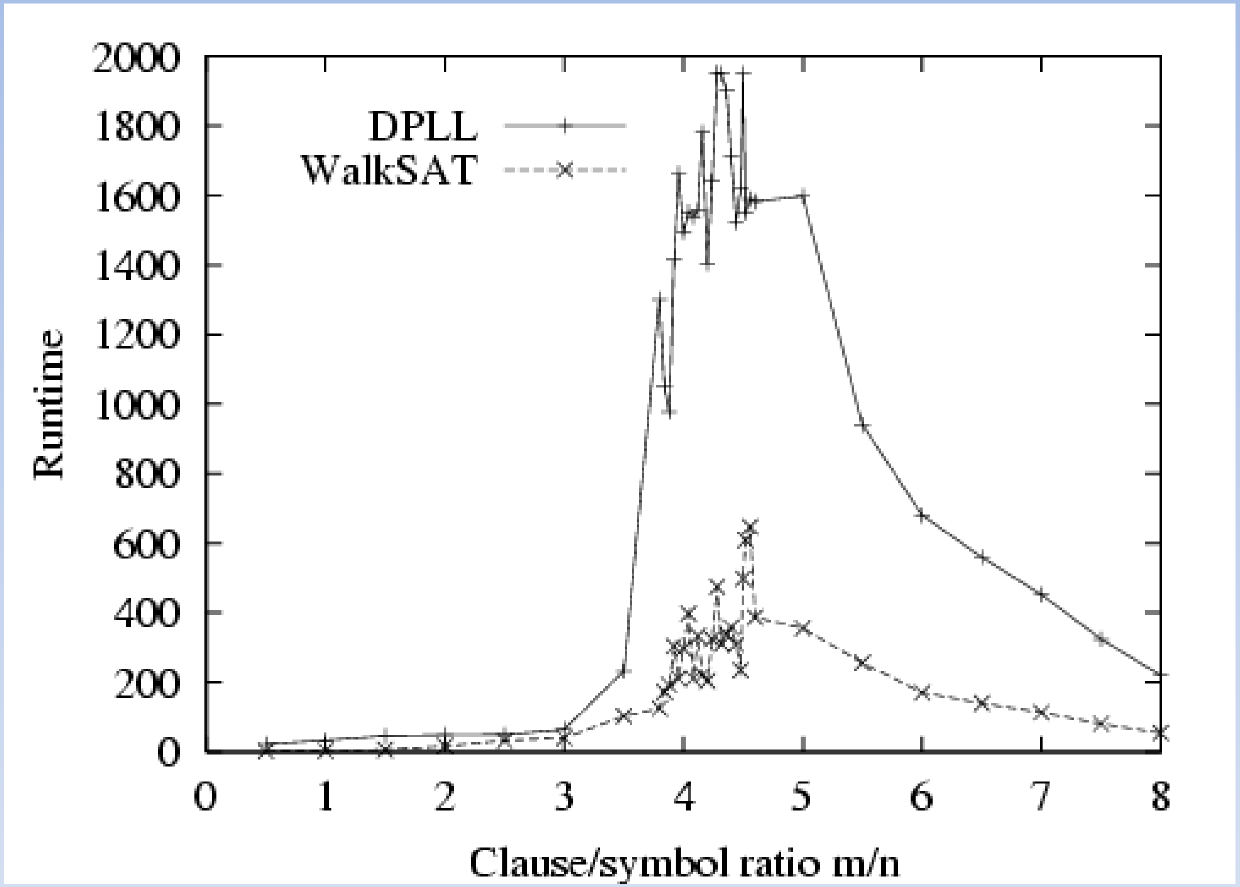
- For a critical value of M/N, the probability of a random assignment being satisfying is close to 0.5
  - These are the hardest SAT problems

Phase Transition



If a problem domain has this sort of characteristic, it is said to have a "phase transition." Typically in such cases, problems near the transition are hard to solve.

Runtime Characteristics



Median runtime for 100 satisfiable random 3CNF formulae, N = 50

Summary

- We learned about:
  - PL Syntax, semantics, models, entailment
  - Semantic Inference Algorithms
    - Systematic: DPLL
    - Local Search: WalkSAT
  - Phase Transition Characteristics of SAT