Inf2D 07: Effective Propositional Inference

Valerio Restocchi

School of Informatics, University of Edinburgh

28/01/20

Slide Credits: Jacques Fleuriot, Michael Rovatsos, Michael Herrmann, Vaishak Belle

Outline

Two families of efficient algorithms for propositional inference:

- Complete backtracking search algorithms
  - DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - `WalkSAT` algorithm

Clausal Form: CNF

- DPLL and `WalkSAT` manipulate formulae in conjunctive normal form (CNF).
- Sentence is formula whose satisfiability is to be determined.
  - conjunction of clauses.
- Clause is disjunction of literals
- Literal is proposition or negated proposition
- Example: $(A, \neg B), (B, \neg C)$ representing $(A \vee \neg B) \wedge (B \vee \neg C)$

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

- Eliminate $\Leftrightarrow$ replacing $\alpha \Leftrightarrow \beta$ by $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ by $\neg\alpha \vee \beta$
  $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Move $\neg$ inwards using de Morgan's rules
  $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
  possibly also eliminating double-negation: replacing $\neg(\neg\alpha)$ by α
- Apply distributivity law ($\vee$ over $\wedge$) and flatten:
  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

- Early termination
- Pure symbol heuristic
- Unit clause heuristic

Early termination

- A clause is true if one of its literals is true,
    - e.g. if A is true then $(A \vee \neg B)$ is true.
- A sentence is false if any of its clauses is false,
    - e.g. if A is false and B is true then $(A \vee \neg B)$ is false, so sentence containing it is false.

Pure symbol heuristic

- Pure symbol: always appears with the same "sign" or polarity in all clauses.
  - e.g., in the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$ A and B are pure, C is impure.
- Make literal containing a pure symbol true.
  - e.g. (for satisfiability) Let A and $\neg B$ both be true

Unit clause heuristic

Unit clause: only one literal in the clause, e.g. (A)

- The only literal in a unit clause must be true.
  - e.g. A must be true.
- Also includes clauses where all but one literal is false,
  - e.g. (A, B, C) where B and C are false since it is equivalent to $(A,$ false, false$)$ i.e. (A).

The DPLL algorithm

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, [*P* = *value*|*model*])
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, [*P* = *true*|*model*]) **or**
      DPLL(*clauses*, *rest*, [*P* = *false*|*model*])

Tautology Deletion (Optional)

- Tautology: both a proposition and its negation in a clause.
  - e.g. $(A, B, \neg A)$
- Clause bound to be true.
  - e.g. whether A is true or false.
  - Therefore, can be deleted.

Mid-Lecture Exercise

- Apply DPLL heuristics to the following sentence:

$(S_{2,1})$, $(\neg S_{1,1})$, $(\neg S_{1,2})$,
$(\neg S_{2,1}, W_{2,2})$, $(\neg S_{1,1}, W_{2,2})$, $(\neg S_{1,2}, W_{2,2})$,
$(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2})$.

- Use case splits if model not found by these heuristics.

Solution

Symbols: $S_{1,1}$, $S_{1,2}$, $S_{2,1}$, $W_{2,2}$

$(S_{2,1})$, $(\neg S_{1,1})$, $(\neg S_{1,2})$,
$(\neg S_{2,1}, W_{2,2})$,
$(\neg S_{1,1}, W_{2,2})$, $(\neg S_{1,2}, W_{2,2})$,
$(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2})$.

– Pure symbol heuristic:
  - No literal is pure.
– Unit clause heuristic:
  - $S_{2,1}$ is true; $S_{1,1}$ and $S_{1,2}$ are false.
– Early termination heuristic:
  - $(\neg S_{1,1}, W_{2,2})$, $(\neg S_{1,2}, W_{2,2})$ are both true.
  - $(\neg W_{2,2}, S_{2,1}, S_{1,1}, S_{1,2})$ is true.
– Unit clause heuristic:
  - $\neg S_{2,1}$ is false, so $(\neg S_{2,1}, W_{2,2})$ becomes unit clause.
  - $W_{2,2}$ must be true.

The `WalkSAT` algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness

The `WalkSAT` algorithm

```
function WALKSAT(clauses, p, max-flips) returns a satisfying model or failure
   inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a "random walk" move
           max-flips, number of flips allowed before giving up

   model ← a random assignment of true/false to the symbols in clauses
   for i = 1 to max-flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      with probability p flip the value in model of a randomly selected symbol
            from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
   return failure
```

Algorithm checks for satisfiability by randomly flipping the values of variables

Hard satisfiability problems

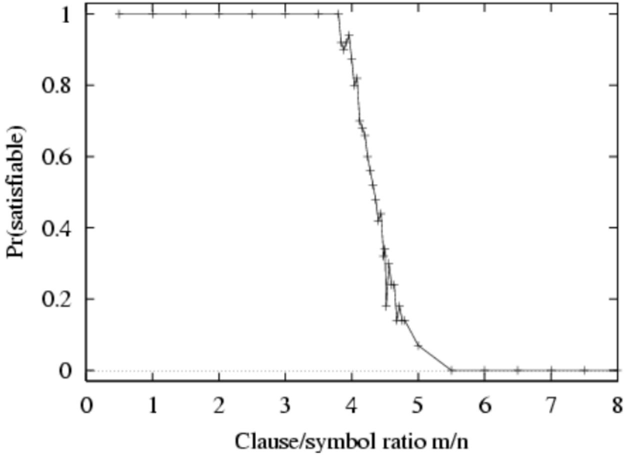
- Consider random 3-CNF sentences: 3SAT problem
- Example:

$$(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$$

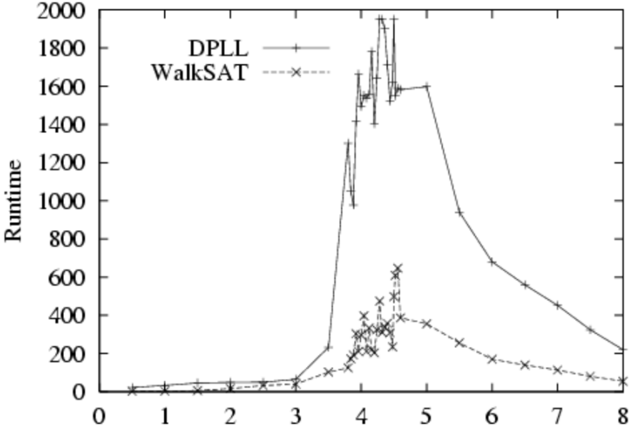
m: number of clauses
n: number of symbols

- Hard problems seem to cluster near $m/n = 4.3$ (critical point)

Hard satisfiability problems



Hard satisfiability problems



Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$

Inference-based agents in the wumpus world

– A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,4}$
$\neg W_{1,1} \vee \neg W_{1,2}$
$\neg W_{1,1} \vee \neg W_{1,3}$
...
$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

The Wumpus Agent (1)

```
function HYBRID-WUMPUS-AGENT (percept) returns an action
  inputs: percept, a list, [stench, breeze, glitter, bump, scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, intially 0, indicating time
              plan, an action sequence, initially empty
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x,y] : ASK (KB,OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
     plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
     unvisited ← {[x,y] : ASK(KB,L^{t'}_{x,y} ) = false for all t' ≤ t }
     plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB,HaveArrow^t ) = true then
     possible_wumpus ← {[x,y] : ASK(KB,¬W_{x,y} ) = false }
     plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then          // no choice but to take a risk
     not_unsafe ← {[x,y] : ASK(KB,¬OK^t_{x,y}) = false }
     plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
     plan ← PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t+1
  return action
```

The Wumpus Agent (2)

```
function PLAN-ROUTE(current, goals, allowed) returns an action sequence
   inputs: current, the agent's current position
           goals, a set of squares; try to plan a route to one of them
           allowed, a set of squares that can form part of the route

   problem ← ROUTE-PROBLEM(current, goals, allowed)
   return A*-GRAPH-SEARCH(problem)
```

We need more!

– Effect axioms:

$$L_{1,1}^0 \wedge \text{FacingEast}^0 \wedge \text{Forward}^0 \Rightarrow L_{2,1}^1 \wedge \neg L_{11}^1$$

– We need extra axioms about the world.

– Frame problem

  - Frame axioms:

$$\text{Forward}^t \Rightarrow \left(\text{HaveArrow}^t \Leftrightarrow \text{HaveArrow}^{t+1}\right)$$
$$\text{Forward}^t \Rightarrow \left(\text{WumpusAlive}^t \Leftrightarrow \text{WumpusAlive}^{t+1}\right)$$

  - Successor-state axioms:

$$\text{HaveArrow}^{t+1} \Leftrightarrow (\text{HaveArrow}^t \wedge \neg \text{Shoot}^t)$$

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square
- For every time t and every location $[x, y]$,

$$L^t_{x,y} \land \text{FacingRight}^t \land \text{Forward}^t \Rightarrow L^{t+1}_{x+1,y}$$

- Rapid proliferation of clauses

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions.
- Two algorithms: DPLL & `WalkSAT`
- Hard satisfiability problems
- Applications to Wumpus World.
- Propositional logic lacks expressive power