Lecture slides for

Automated Planning: Theory and Practice

Chapter 7
Propositional Satisfiability Techniques

Dana S. Nau

University of Maryland

12:58 PM February 15, 2012

Motivation

- Propositional satisfiability: given a boolean formula
  - » e.g., $(P \lor Q) \land (\neg Q \lor R \lor S) \land (\neg R \lor \neg P)$,

  does there exist a model
  - » i.e., an assignment of truth values to the propositions

  that makes the formula true?
- This was the very first problem shown to be NP-complete
- Lots of research on algorithms for solving it
  - Algorithms are known for solving all but a small subset in average-case polynomial time
- Therefore,
  - Try translating classical planning problems into satisfiability problems, and solving them that way

Outline

- Encoding planning problems as satisfiability problems
- Extracting plans from truth values
- Satisfiability algorithms
  - Davis-Putnam
  - Local search
  - GSAT
- Combining satisfiability with planning graphs
  - SatPlan

Overall Approach

- A *bounded planning problem* is a pair (P,n):
  - P is a planning problem; n is a positive integer
  - Any solution for P of length n is a solution for (P,n)

- Planning algorithm:
- Do iterative deepening like we did with Graphplan:
  - for $n = 0, 1, 2, \ldots,$
    - » encode (P,n) as a satisfiability problem Φ
    - » if Φ is satisfiable, then
      - From the set of truth values that satisfies Φ, a solution plan can be constructed, so return it and exit

Notation

- For satisfiability problems we need to use propositional logic
- Need to encode ground atoms into propositions
  - For set-theoretic planning we encoded atoms into propositions by rewriting them as shown here:
    - Atom: at(r1,loc1)
    - Proposition: at-r1-loc1
- For planning as satisfiability we'll do the same thing
  - But we won't bother to do a syntactic rewrite
  - Just use at(r1,loc1) itself as the proposition

- Also, we'll write plans starting at a_0 rather than a_1
  - $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$

Fluents

- If $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is a solution for (P,n), it generates these states:

  $s_0, \quad s_1 = \gamma(s_0,a_0), \quad s_2 = \gamma(s_1,a_1), \quad \ldots, \quad s_n = \gamma(s_{n-1}, a_{n-1})$

- *Fluent*: proposition saying a particular atom is true in a particular state
  - at(r1,loc1,i) is a fluent that's true iff at(r1,loc1) is in s_i
  - We'll use l_i to denote the fluent for literal l in state s_i
    - » e.g., if l = at(r1,loc1)
      then l_i = at(r1,loc1,i)
  - a_i is a fluent saying that a is the i'th step of π
    - » e.g., if a = move(r1,loc2,loc1)
      then a_i = move(r1,loc2,loc1,i)

Encoding Planning Problems

- Encode (P,n) as a formula Φ such that
  - $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is a solution for (P,n) if and only if
    Φ can be satisfied in a way that makes the fluents $a_0, \ldots, a_{n-1}$ true
- Let
  - A = {all actions in the planning domain}
  - S = {all states in the planning domain}
  - L = {all literals in the language}
- Φ is the conjunct of many other formulas …

Formulas in Φ

1. Formula describing the initial state:
   - $\bigwedge\{l_0 \mid l \in s_0\} \wedge \bigwedge\{\neg l_0 \mid l \in L - s_0\}$
2. Formula describing the goal:
   - $\bigwedge\{l_n \mid l \in g^+\} \wedge \bigwedge\{\neg l_n \mid l \in g^-\}$
3. For every action a in A and for $i = 1, \ldots, n$, a formula describing what changes a would make if it were the i'th step of the plan:
   - $a_i \Rightarrow \bigwedge\{p_i \mid p \in \text{Precond}(a)\} \wedge \bigwedge\{e_{i+1} \mid e \in \text{Effects}(a)\}$
4. *Complete exclusion* axiom:
   - For every pair of actions a and b, and for $i = 0, \ldots, n{-}1$, a formula saying they can't both be the i'th step of the plan

     $\neg a_i \vee \neg b_i$
   - this guarantees there can be only one action at a time

- Is this enough?

Frame Axioms

5. *Frame axioms*:
   - Formulas describing what *doesn't* change between steps i and i+1
- Several ways to write these

- One way: *explanatory frame axioms*
   - For $i = 0, \ldots, n-1$, an axiom for every literal l
      - Says that if l changes between s_i and s_{i+1}, then the action at step i must be responsible:

$$(\neg l_i \wedge l_{i+1} \Rightarrow \bigvee_{a \text{ in } A} \{a_i \mid l \in \text{effects}^+(a)\})$$
$$\wedge\ (l_i \wedge \neg l_{i+1} \Rightarrow \bigvee_{a \text{ in } A} \{a_i \mid l \in \text{effects}^-(a)\})$$

Example

- Planning domain:
  - one robot r1
  - two adjacent locations l1, l2
  - one planning operator (to move the robot from one location to another)

- Encode (P,n) where $n = 1$

  1. Initial state: {at(r1,l1)}
     Encoding: at(r1,l1,0) ∧ ¬at(r1,l2,0)

  2. Goal: {at(r1,l2)}
     Encoding: at(r1,l2,1) ∧ ¬at(r1,l1,1)

  3. Operator: see next slide

Example (continued)

- Operator: $\mathsf{move}(r,l,l')$

  precond: $\mathsf{at}(r,l)$

  effects: $\mathsf{at}(r,l')$, $\neg\mathsf{at}(r,l)$

  Encoding:

  $\mathsf{move(r1,l1,l2,0)} \Rightarrow \mathsf{at(r1,l1,0)} \wedge \mathsf{at(r1,l2,1)} \wedge \neg\mathsf{at(r1,l1,1)}$

  $\mathsf{move(r1,l2,l1,0)} \Rightarrow \mathsf{at(r1,l2,0)} \wedge \mathsf{at(r1,l1,1)} \wedge \neg\mathsf{at(r1,l2,1)}$

  $\left.\begin{array}{l}\mathsf{move(r1,l1,l1,0)} \Rightarrow \mathsf{at(r1,l1,0)} \wedge \mathsf{at(r1,l1,1)} \wedge \neg\mathsf{at(r1,l1,1)} \\ \mathsf{move(r1,l2,l2,0)} \Rightarrow \mathsf{at(r1,l2,0)} \wedge \mathsf{at(r1,l2,1)} \wedge \neg\mathsf{at(r1,l2,1)}\end{array}\right\}$ contradictions (easy to detect)

  $\left.\begin{array}{l}\mathsf{move(l1,r1,l2,0)} \Rightarrow \ldots \\ \mathsf{move(l2,l1,r1,0)} \Rightarrow \ldots \\ \mathsf{move(l1,l2,r1,0)} \Rightarrow \ldots \\ \mathsf{move(l2,l1,r1,0)} \Rightarrow \ldots\end{array}\right\}$ nonsensical, and we can avoid generating them if we use data types like we did for state-variable representation

- Operator: $\mathsf{move}(r : \mathsf{robot},\ l : \mathsf{location},\ l' : \mathsf{location})$

  precond: $\mathsf{at}(r,l)$

  effects: $\mathsf{at}(r,l')$, $\neg\mathsf{at}(r,l)$

Example (continued)

4. Complete-exclusion axiom:

   $\neg$move(r1,l1,l2,0) $\vee$ $\neg$move(r1,l2,l1,0)

5. Explanatory frame axioms:

   $\neg$at(r1,l1,0) $\wedge$ at(r1,l1,1) $\Rightarrow$ move(r1,l2,l1,0)

   $\neg$at(r1,l2,0) $\wedge$ at(r1,l2,1) $\Rightarrow$ move(r1,l1,l2,0)

   at(r1,l1,0) $\wedge$ $\neg$at(r1,l1,1) $\Rightarrow$ move(r1,l1,l2,0)

   at(r1,l2,0) $\wedge$ $\neg$at(r1,l2,1) $\Rightarrow$ move(r1,l2,l1,0)

- Φ is the conjunct of all of these

Summary of the Example

- P is a planning problem with one robot and two locations
  - initial state {at(r1,l1)}
  - goal {at(r1,l2)}
- Encoding of $(P, 1)$
  - Φ = [at(r1,l1,0) ∧ ¬at(r1,l2,0)] (initial state)

    ∧ [at(r1,l2,1) ∧ ¬at(r1,l1,1)] (goal)

    ∧ [move(r1,l1,l2,0)
    ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)] (action)

    ∧ [move(r1,l2,l1,0)
    ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)] (action)

    ∧ [¬move(r1,l1,l2,0) ∨ ¬move(r1,l2,l1,0)] (complete exclusion)

    ∧ [¬at(r1,l1,0) ∧ at(r1,l1,1) ⇒ move(r1,l2,l1,0)] (frame axiom)

    ∧ [¬at(r1,l2,0) ∧ at(r1,l2,1) ⇒ move(r1,l1,l2,0)] (frame axiom)

    ∧ [at(r1,l1,0) ∧ ¬at(r1,l1,1) ⇒ move(r1,l1,l2,0)] (frame axiom)

    ∧ [at(r1,l2,0) ∧ ¬at(r1,l2,1) ⇒ move(r1,l2,l1,0)] (frame axiom)

Extracting a Plan

- Let Φ be an encoding of (*P,n*)
- Suppose we find an assignment of truth values that satisfies Φ.
  - This means *P* has a solution of length *n*
- For *i*=1,…,*n*, there will be exactly one action *a* such that a_i = *true*
  - This is the *i*'th action of the plan.
- Example
- The formula on the previous slide
  - Φ can be satisfied with move(r1,l1,l2,0) = *true*
    - Thus ⟨move(r1,l1,l2,0)⟩ is a solution for (*P*,1)
  - It's the only solution - no other way to satisfy Φ

Planning

- How to find an assignment of truth values that satisfies Φ?
  - Use a satisfiability algorithm

- Example: the *Davis-Putnam* algorithm

  - First need to put Φ into conjunctive normal form

    e.g., $\Phi = D \wedge (\neg D \vee A \vee \neg B) \wedge (\neg D \vee \neg A \vee \neg B) \wedge (\neg D \vee \neg A \vee B) \wedge A$

  - Write Φ as a set of *clauses* (disjuncts of literals)

    $\Phi = \{\{D\}, \ \{\neg D, A, \neg B\}, \ \{\neg D, \neg A, \neg B\}, \ \{\neg D, \neg A, B\}, \ \{A\}\}$

  - Some special cases:
    - If $\Phi = \varnothing$ then Φ is always *true*
    - If $\Phi = \{\ldots, \varnothing, \ldots\}$ then Φ is always *false* (hence unsatisfiable)
    - If Φ contains a *unit clause*, *l*, then *l* must be true in order to satisfy Φ

The Davis-Putnam Procedure

Backtracking search through alternative assignments of truth values to literals

- μ = {literals to which we have assigned the value TRUE}
  - initially empty
- For every unit clause l
  - add l to μ
  - remove clauses that contain l
  - modify clauses that contain $\neg l$
- If Φ contains $\varnothing$, μ fails
- If $\Phi = \varnothing$, μ is a solution
- Select a Boolean variable P in Φ
- do two recursive calls
  - $\Phi \wedge P$
  - $\Phi \wedge \neg P$

Davis-Putnam(Φ,μ)
    Unit-propagate(Φ,μ)
    if $\varnothing \in \Phi$ then return
    if $\Phi = \varnothing$ then exit with μ
    select a variable P such that P or $\neg P$ occurs in ϕ
    Davis-Putnam($\Phi \cup \{P\},\mu$)
    Davis-Putnam($\Phi \cup \{\neg P\},\mu$)
end

error in the book here

Unit-Propagate(Φ,μ)
    while there is a unit clause $\{l\}$ in Φ do
        $\mu \leftarrow \mu \cup \{l\}$
        for every clause $C \in \Phi$
            if $l \in C$ then $\Phi \leftarrow \Phi - \{C\}$
            else if $\neg l \in C$ then $\Phi \leftarrow \Phi - \{C\} \cup \{C - \{\neg l\}\}$
end

Local Search

- Let u be an assignment of truth values to all of the variables
  - cost(u,Φ) = number of clauses in Φ that aren't satisfied by u
  - flip(P,u) = u except that P's truth value is reversed

Boolean variable

- Local search:
  - Select a random assignment u
  - while cost(u,Φ) $\neq 0$
    - » if there is a P such that cost(flip(P,u),Φ) $<$ cost(u,Φ) then
      - randomly choose any such P
      - $u \leftarrow$ flip(P,u)
    - » else return failure
- Local search is sound
- If it finds a solution it will find it very quickly
- Local search is not complete: can get trapped in local minima

GSAT

- Basic-GSAT:
  - Select a random assignment u
  - while $\text{cost}(u,\Phi) \neq 0$
    - » choose a P that minimizes $\text{cost}(\text{flip}(P,u),\Phi)$, and flip it
- Not guaranteed to terminate

- GSAT:
  - restart after a max number of flips
  - return failure after a max number of restarts

- The book discusses several other stochastic procedures
  - One is Walksat
    - » works better than both local search and GSAT
  - I'll skip the details

Discussion

- Recall the overall approach:
  - for $n = 0, 1, 2, \ldots,$
    - encode (P,n) as a satisfiability problem Φ
    - if Φ is satisfiable, then
      - From the set of truth values that satisfies Φ, extract a solution plan and return it
- By itself, not very practical (takes too much memory and time)
- But it can work well if combined with other techniques
  - e.g., planning graphs

SatPlan

- SatPlan combines planning-graph expansion and satisfiability checking
- Works roughly as follows:
  - for k = 0, 1, 2, …
    - Create a planning graph that contains k levels
    - Encode the planning graph as a satisfiability problem
    - Try to solve it using a SAT solver
      - If the SAT solver finds a solution within some time limit,
        - Remove some unnecessary actions
        - Return the solution
- Memory requirement still is combinatorially large
  - but less than what's needed by a direct translation into satisfiability
- BlackBox (predecessor to SatPlan) was one of the best planners in the 1998 planning competition
- SatPlan was one of the best planners in the 2004 and 2006 planning competitions

Other Translation Approaches

- Translate planning problems into 0-1 integer programming problems
  - Then solve them using an integer programming package such as CPLEX
  - Techniques are somewhat similar to translation of planning to satisfiability

- Translate planning problems into constraint satisfaction problems
  - Then solve them using CSP techniques such as arc consistency and path consistency
  - For details, see Chapter 8