Chapter 7
Propositional Satisfiability Techniques

Dana S. Nau
University of Maryland
Fall 2009
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 \(\Phi\)
    - if \(\Phi\) is satisfiable, then
      - From the set of truth values that satisfies \(\Phi\), 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: $\text{at}(r1,\text{loc}1)$
    - Proposition: $\text{at-r1-loc1}$
- For planning as satisfiability we’ll do the same thing
  - But we won’t bother to do a syntactic rewrite
  - Just use $\text{at}(r1,\text{loc}1)$ 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 \(l = \text{at}(r1,loc1)\)
    then \(l_i = \text{at}(r1,loc1,i)\)
  ◆ \(a_i\) is a fluent saying that \(a\) is the \(i\)’th step of \(\pi\)
    » e.g., if \(a = \text{move}(r1,loc2,loc1)\)
    then \(a_i = \text{move}(r1,loc2,loc1,i)\)
Encoding Planning Problems

- Encode \((P, n)\) as a formula \(\Phi\) such that
  \(\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle\) is a solution for \((P, n)\) if and only if
  \(\Phi\) can be satisfied in a way that makes the fluents \(a_0, \ldots, a_{n-1}\) true

- Let
  - \(A = \{\text{all actions in the planning domain}\}\)
  - \(S = \{\text{all states in the planning domain}\}\)
  - \(L = \{\text{all literals in the language}\}\)

- \(\Phi\) is the conjunct of many other formulas …
Formulas in $\Phi$

- Formula describing the initial state:
  \[ \land \{ l_0 \mid l \in s_0 \} \land \land \{ \neg l_0 \mid l \in L - s_0 \} \]

- Formula describing the goal:
  \[ \land \{ l_n \mid l \in g^+ \} \land \land \{ \neg l_n \mid l \in g^- \} \]

- For every action $a$ in $A$, formulas describing what changes $a$ would make if it were the $i$’th step of the plan:
  \[ a_i \Rightarrow \land \{ p_i \mid p \in \text{Precond}(a) \} \land \land \{ e_{i+1} \mid e \in \text{Effects}(a) \} \]

- Complete exclusion axiom:
  - For all actions $a$ and $b$, formulas saying they can’t occur at the same time
    \[ \neg a_i \lor \neg b_i \]
  - this guarantees there can be only one action at a time

- Is this enough?
Frame Axioms

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

- One way: *explanatory frame axioms*
  - One 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:

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

- Planning domain:
  - one robot $r_1$
  - two adjacent locations $l_1, l_2$
  - one operator (move the robot)

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

  - Initial state: $\{at(r_1,l_1)\}$
    Encoding: $at(r_1,l_1,0) \land \neg at(r_1,l_2,0)$

  - Goal: $\{at(r_1,l_2)\}$
    Encoding: $at(r_1,l_2,1) \land \neg at(r_1,l_1,1)$

  - Operator: see next slide
Example (continued)

- Operator: move(r,l,l’)
  - precond: at(r,l)
  - effects: at(r,l’), ¬at(r,l)

Encoding:

\[
\begin{align*}
\text{move}(r1,l1,l2,0) & \Rightarrow \text{at}(r1,l1,0) \land \text{at}(r1,l2,1) \land \neg \text{at}(r1,l1,1) \\
\text{move}(r1,l2,l1,0) & \Rightarrow \text{at}(r1,l2,0) \land \text{at}(r1,l1,1) \land \neg \text{at}(r1,l2,1) \\
\text{move}(r1,l1,l1,0) & \Rightarrow \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \land \neg \text{at}(r1,l1,1) \\
\text{move}(r1,l2,l2,0) & \Rightarrow \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \land \neg \text{at}(r1,l2,1) \\
\text{move}(l1,r1,l2,0) & \Rightarrow \ldots \\
\text{move}(l2,l1,r1,0) & \Rightarrow \ldots \\
\text{move}(l1,l2,r1,0) & \Rightarrow \ldots \\
\text{move}(l2,l1,r1,0) & \Rightarrow \ldots
\end{align*}
\]

- How to avoid generating the last four actions?
  - Assign data types to the constant symbols like we did for state-variable representation
Example (continued)

- **Locations:** \( l_1, l_2 \)
- **Robots:** \( r_1 \)
- **Operator:** \( \text{move}(r : \text{robot}, l : \text{location}, l' : \text{location}) \)
  
  - **precond:** \( \text{at}(r,l) \)
  - **effects:** \( \text{at}(r,l'), \neg \text{at}(r,l) \)

**Encoding:**

\[
\begin{align*}
\text{move}(r_1,l_1,l_2,0) \Rightarrow & \quad \text{at}(r_1,l_1,0) \land \text{at}(r_1,l_2,1) \land \neg \text{at}(r_1,l_1,1) \\
\text{move}(r_1,l_2,l_1,0) \Rightarrow & \quad \text{at}(r_1,l_2,0) \land \text{at}(r_1,l_1,1) \land \neg \text{at}(r_1,l_2,1)
\end{align*}
\]
Example (continued)

- Complete-exclusion axiom:
  \[ \neg \text{move}(r1,l1,l2,0) \lor \neg \text{move}(r1,l2,l1,0) \]

- Explanatory frame axioms:
  \[ \neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l2,l1,0) \]
  \[ \neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l1,l2,0) \]
  \[ \text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0) \]
  \[ \text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l2,l1,0) \]
Extracting a Plan

- Suppose we find an assignment of truth values that satisfies \( \Phi \).
  - This means \( P \) has a solution of length \( n \)

- For \( i=1,\ldots,n \), there will be exactly one action \( a \) such that \( a_i = true \)
  - This is the \( i \)'th action of the plan.

- Example (from the previous slides):
  - \( \Phi \) can be satisfied with \( \text{move}(r1,l1,l2,0) = true \)
  - Thus \( \langle \text{move}(r1,l1,l2,0) \rangle \) is a solution for \( (P,0) \)
    - It’s the only solution - no other way to satisfy \( \Phi \)
Planning

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

- Example: the *Davis-Putnam* algorithm

  - First need to put $\Phi$ into conjunctive normal form
    
    $$
    \Phi = D \land (\neg D \lor A \lor \neg B) \land (\neg D \lor \neg A \lor \neg B) \land (\neg D \lor \neg A \lor B) \land A
    $$

  - Write $\Phi$ as a set of *clauses* (disjuncts of literals)
    
    $$
    \Phi = \{\{D\}, \{\neg D, A, \neg B\}, \{\neg D, \neg A, \neg B\}, \{\neg D, \neg A, B\}, \{A\}\}
    $$

  - Two special cases:
    
    - If $\Phi = \emptyset$ then $\Phi$ is always *true*
    - If $\Phi = \{\ldots, \emptyset, \ldots\}$ then $\Phi$ is always *false* (hence unsatisfiable)
The Davis-Putnam Procedure

Backtracking search through alternative assignments of truth values to literals

- $\mu = \{\text{literals to which we have assigned the value TRUE}\}; \text{ initially empty}$
- if $\Phi$ contains $\emptyset$ then
  - $\triangleright$ backtrack
- if $\Phi$ contains $\emptyset$ then
  - $\triangleright$ $\mu$ is a solution
- while $\Phi$ contains a clause that’s a single literal $l$
  - $\triangleright$ add $l$ to $\mu$
  - $\triangleright$ remove $l$ from $\Phi$
- select a Boolean variable $P$ in $\Phi$
- do recursive calls on
  - $\Phi \land P$
  - $\Phi \land \neg P$

```
Davis-Putnam(\Phi, \mu)
  if \emptyset \not\in \Phi \text{ then return}
  if \Phi = \emptyset \text{ then exit with } \mu
  Unit-Propagate(\Phi, \mu)
  select a variable $P$ such that $P$ or $\neg P$ occurs in $\phi$
  Davis-Putnam(\Phi \cup \{P\}, \mu)
  Davis-Putnam(\Phi \cup \{-P\}, \mu)
```

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

- Let $u$ be an assignment of truth values to all of the variables
  - $\text{cost}(u, \Phi) =$ number of clauses in $\Phi$ that aren’t satisfied by $u$
  - $\text{flip}(P, u) = u$ except that $P$’s truth value is reversed

- Local search:
  - Select a random assignment $u$
  - while $\text{cost}(u, \Phi) \neq 0$
    - if there is a $P$ such that $\text{cost}(\text{flip}(P, u), \Phi) < \text{cost}(u, \Phi)$ then
      - randomly choose any such $P$
      - $u \leftarrow \text{flip}(P, u)$
    - 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 $\Phi$
    - if $\Phi$ is satisfiable, then
      - From the set of truth values that satisfies $\Phi$, extract a solution plan and return it

- How well does this work?
Discussion

- Recall the overall approach:
  - for $n = 0, 1, 2, \ldots$
    - encode $(P,n)$ as a satisfiability problem $\Phi$
    - if $\Phi$ is satisfiable, then
      - From the set of truth values that satisfies $\Phi$, extract a solution plan and return it

- How well does this work?
  - By itself, not very practical (takes too much memory and time)
  - But it can be combined with other techniques
    - e.g., planning graphs
SatPlan

- SatPlan combines planning-graph expansion and satisfiability checking, roughly as follows:
  - for $k = 0, 1, 2, \ldots$
    - 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