SAT-based planners

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Encoding planning problems as satisfiability problems

Source

- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons
- License https://creativecommons.org/licenses/by-nc-sa/2.0/

Outline

- **Motivation**
- Overall approach
- Notation
- Example: GoTo
- SAT planners
- Conclusions

Motivation

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

  does there exist a model (an assignment of truth values to the propositions) that makes the formula true?
- This was the very first problem shown to be NP-complete
- 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

- Motivation
- **Overall approach**
- Notation
- Example: GoTo
- SAT planners
- Conclusions

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

Outline

- Motivation
- Overall approach
- **Notation**
- Example: GoTo
- SAT planners
- Conclusions

Notation: propositions

- For satisfiability problems we need to use propositional logic
- Need to encode ground atoms into propositions
  - For set-theoretic planning we encoded predicates into propositions by rewriting them as shown here:
    - Predicate: 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
  - $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$

Notation: fluents

- Proposition saying a particular predicate 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 = GoTo(r1,loc2,loc1)
      then a_i = GoTo(r1,loc2,loc1,i)

Outline

- Motivation
- Overall approach
- Notation
- **Example: GoTo**
- SAT planners
- Conclusions

Example (I)

- 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: $\text{at(r1,l1,0)} \wedge \neg\text{at(r1,l2,0)}$
  2. Goal: {at(r1,l2)}
     Encoding: $\text{at(r1,l2,1)} \wedge \neg\text{at(r1,l1,1)}$
  3. Operator: GoTo (see next slide)

Example (II)

- Operator: GoTo (r,l,l')
  precond: at(r,l)
  effects: at(r,l'), $\neg$at(r,l)

  Encoding:

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

  contradictions (easy to detect)

  GoTo(l1,r1,l2,0) $\Rightarrow$ …
  GoTo(l2,l1,r1,0) $\Rightarrow$ …
  GoTo(l1,l2,r1,0) $\Rightarrow$ …
  GoTo(l2,l1,r1,0) $\Rightarrow$ …

  nonsensical, and we can avoid generating them if we use data types like we did for state-variable representation

- Operator: move(r : robot, l : location, l' : location)
  precond: at(r,l)
  effects: at(r,l'), $\neg$at(r,l)

Example (III)

4. Complete-exclusion axiom:

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

5. Explanatory frame axioms:

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

Extracting a Plan

- Suppose we find an assignment of truth values that satisfies it
  - 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 ith action of the plan.
- The formula on the previous slide
  - It can be satisfied with GoTo(r1,l1,l2,0) = *true*
    - Thus ⟨GoTo(r1,l1,l2,0)⟩ is a solution for (P,1)
  - It's the only solution - no other way to satisfy it

Planning

- How to find an assignment of truth values that satisfies Φ?
  - Use a satisfiability algorithm
- Example: the *DPLL* 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 $\Phi = \emptyset$ then Φ is always *true*
    - If $\Phi = \{\ldots, \emptyset, \ldots\}$ then Φ is always *false* (hence unsatisfiable)
    - If Φ contains a *unit clause*, *l*, then *l* must be true in order to satisfy Φ

Outline

- Motivation
- Overall approach
- Notation
- Example: GoTo
- **SAT planners**
- Conclusions

SAT planners

- SATPLAN: builds a GP, translates manually the graph constraints to {} axioms, then uses a SAT and if no solution is found → length increases
- BLACKBOX: combines GP and SATPLAN
- LPSAT: uses a backtrack random algorithm with a new formalism (LCNF) that combines propositional logic with a set of metric constraints
- LPG
- …

Outline

- Motivation
- Overall approach
- Notation
- Example: GoTo
- SAT planners
- **Conclusions**

Conclusion

- The philosophy of the algorithm is:
  - The planning problem is translated to CNF
  - Guess the length which aims to achieve the goal
  - A set of propositional clauses is generated to check the satisfiability
  - Apply algorithms for propositional logic (DPLL, WALSAT, LMTS-style)