SAT-Based Planning

Using Propositional SAT-Solvers to Search for Plans

Literature

- Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 7. Elsevier/Morgan Kaufmann, 2004.

The General Idea

- idea: transform planning problem into other problem for which efficient solvers are known
- approach here:
  - transform planning problem into propositional satisfiability problem (SAT)
  - solve transformed problem using (efficient) SAT solver, e.g. GSAT
  - extract a solution to the planning problem from the solution to transformed problem

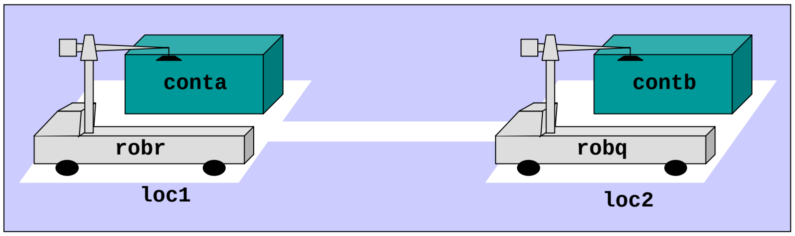
Overview

- Encoding Planning Problems as Satisfiability Problems (SAT)
- Efficient SAT Solving Algorithms

Encoding a Planning Problem

- aim: encode a propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$ into a propositional formula Φ such that:
  - $\mathcal{P}$ has a solution if and only if Φ is satisfiable, and
  - every model μ of Φ corresponds to a solution plan π of $\mathcal{P}$.
- key elements to encode:
  - world states
  - state-transitions (actions)

Example: Simplified DWR Problem



- robots can load and unload autonomously
- locations may contain unlimited number of robots and containers
- problem: swap locations of containers

Simplified DWR Problem: State Proposition Symbols

- robots:
  - *r1* and *r2*: at(robr,loc1) and at(robr,loc2)
  - *q1* and *q2*: at(robq,loc1) and at(robq,loc2)
  - *ur* and *uq*: unloaded(robr) and unloaded(robq)
- containers:
  - *a1*, *a2*, *ar*, and *aq*: in(conta,loc1), in(conta,loc2), loaded(conta,robr), and loaded(conta,robq)
  - *b1*, *b2*, *br*, and *bq*: in(contb,loc1), in(contb,loc2), loaded(contb,robr), and loaded(contb,robq)
- initial state: {*r1*, *q2*, *a1*, *b2*, *ur*, *uq*}

Encoding World States

- use conjunction of propositions that hold in the state
- example:
  - initial state: {*r1*, *q2*, *a1*, *b2*, *ur*, *uq*}
  - encoding: $r1 \land q2 \land a1 \land b2 \land ur \land uq$
  - model: {*r1*←true, *q2*←true, *a1*←true, *b2*←true, *ur*←true, *uq*←true}

Intended vs. Unintended Models

- possible models:
  - intended model: {*r1*←true, *r2*←false, *q1*←false, *q2*←true, *ur*←true, *uq*←true, *a1*←true, *a2*←false, *ar*←false, *aq*←false, *b1*←false, *b2*←true, *br*←false, *bq* ←false}
  - unintended model: {*r1*←true, *r2*←true, *q1*←false, *q2*←true, *ur*←true, *uq*←true, *a1*←true, *a2*←false, *ar*←true, *aq*←false, *b1*←false, *b2*←true, *br*←false, *bq* ←false}
- encoding: add negated propositions not in state
  - example: $r1 \land \neg r2 \land \neg q1 \land q2 \land ur \land uq \land a1 \land \neg a2 \land \neg ar \land \neg aq \land \neg b1 \land b2 \land \neg br \land \neg bq$

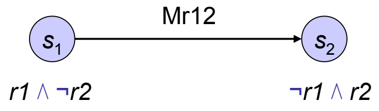
Encoding the Set of Goal States

- goal: defined as set of states
  - example:
    - swap the containers
    - all states in which *a2* and *b1* are true
- propositional formula can encode multiple states:
  - example: $a2 \land b1$ (2^{12} possible models)
  - use disjunctions for other types of goals

Simplified DWR Problem: Action Symbols

- move actions:
  - Mr12: move(robr,loc1,loc2), Mr21: move(robr,loc2,loc1), Mq12: move(robq,loc1,loc2), Mq21: move(robq,loc2,loc1)
- load actions:
  - Lar1: load(conta,robr,loc1); Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, and Lbq2 correspondingly
- unload actions:
  - Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, and Ubq2 correspondingly

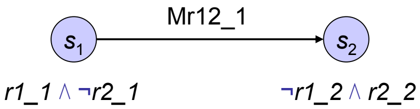
Extended State Propositions



- state transition: $\gamma(s_1,\mathrm{Mr12}) = s_2$ where:
  - s_1 described by $r1 \land \neg r2$ and
  - s_2 described by $\neg r1 \land r2$
- problem: $r1 \land \neg r2 \land \neg\ r1 \land r2$ has no model
- idea: extend propositions with state index
  - example: $r1\_1 \land \neg r2\_1 \land \neg r1\_2 \land r2\_2$
  - model: {$r1\_1$←true, $r2\_1$←false, $r1\_2$←false, $r2\_2$←true}

Extended Action Propositions

- use same mechanism to describe actions applied in different states:
  - example: Mr12_1: move robot *r* from location 1 to location 2 in state s_2
  - action encoding:
    Mr12_1 ⇒ (*r1_1* ∧ ¬*r2_1* ∧ ¬*r1_2* ∧ *r2_2*)



Bounded Planning Problems

- encoding in two steps:
  - <u>bounded planning problem</u>: for a given planning problem $\mathcal{P}=(\Sigma,s_i,g)$ find a solution plan of a fixed length *n*
  - encode the bounded planning problem into a satisfiability problem
    - state propositions with index 0 … *n*
    - action propositions with index 0 … *n*-1

Encoding Bounded Planning Problems

- conjunction of formulas describing:
  - the initial state
  - the goal states
  - actions (applicability and effects)
  - frame axioms
  - one action at a time

Encoding Initial and Goal States

- Let *F* be the set of state propositions (fluents). Let $f \in F$.

- initial state:
  - $\bigwedge_{f \in si} f\_0 \wedge \bigwedge_{f \notin si} \neg f\_0$
- goal states:
  - $\bigwedge_{f \in g+} f\_n \wedge \bigwedge_{f \in g-} \neg f\_n$

Encoding Actions

- Let A be the set of action propositions. Let $a \in A$.

- for $0 \leq i \leq n$-1:
  - $a\_i \Rightarrow \Big(\bigwedge_{f \in precond(a)} f\_i \wedge$
    $\bigwedge_{f \in effects+(a)} f\_i\text{+1} \wedge$
    $\bigwedge_{f \in effects-(a)} \neg f\_i\text{+1} \Big)$

Encoding Frame Axioms

- use explanation closure axioms for more compact SAT problem

- for $0 \leq i \leq n$-1:
  - $(f\_i \wedge \neg f\_i\text{+1}) \Rightarrow \Big(\bigvee_{a \in \mathrm{A} \wedge f \in effects-(a)} a\_i\Big) \wedge$
  - $(\neg f\_i \wedge f\_i\text{+1}) \Rightarrow \Big(\bigvee_{a \in \mathrm{A} \wedge f \in effects+(a)} a\_i\Big)$

Encoding Exclusion Axioms

- allow only exactly one action at each step
- for $0 \leq i \leq n\text{-}1$ and $a \neq a'$, $a,a' \in A$:
  - $\neg a\_i \vee \neg a'\_i$

Overview

- Encoding Planning Problems as Satisfiability Problems (SAT)
- Efficient SAT Solving Algorithms

Generic SAT Problem

- given: set of *m* propositional formulas: $\{F_1 \dots F_m\}$
  - containing *n* proposition symbols: $P_1 \dots P_n$
- find: an interpretation *I*
  - that assigns truth values (T, F) to $P_1 \dots P_n$, i.e. $I(F_j)$ = T or $I(F_j)$ = F, and
  - under which all the formulas evaluate to T, i.e. $I(F_1 \wedge \dots \wedge F_m)$ = T

Conjunctive Normal Form

- formula *F* is in conjunctive normal form (CNF) iff:
  - *F* has the form $F_1 \wedge \dots \wedge F_n$ and
  - each F_i, $i \in 1 \dots n$, is a disjunction of literals
- **Proposition**: Let *F* be a propositional formula. Then there exists a propositional formula *F*' in CNF such that:
  - *F* and *F*' are equivalent, i.e.
  - for every interpretation *I*, *I*(*F*) = *I*(*F*')

Transformation into CNF

- eliminate implications:
  - $F \leftrightarrow G = F \rightarrow G \land G \rightarrow F$
  - $F \rightarrow G = \neg F \lor G$
- bring negations before atoms:
  - $\neg(F \lor G) = \neg F \land \neg G$
  - $\neg(F \land G) = \neg F \lor \neg G$
  - $\neg(\neg F) = F$
- apply distributive laws:
  - $F \land (G \lor H) = (F \land G) \lor (F \land H)$
  - $F \lor (G \land H) = (F \lor G) \land (F \lor H)$

SAT Solving Procedures

- systematic:
  - Davis-Putnam algorithm
    - extend partial assignment into complete assignment
    - sound and complete
- stochastic:
  - local search algorithms (GSAT, WalkSAT)
    - modify randomly chosen total assignment
    - sound, not complete, very fast

Local Search Algorithms

- basic principles:
  - keep only a single (complete) state in memory
  - generate only the neighbours of that state
  - keep one of the neighbours and discard others
- key features:
  - no search paths
  - neither systematic nor incremental
- key advantages:
  - use very little memory (constant amount)
  - find solutions in search spaces too large for systematic algorithms

Random-Restart Hill-Climbing

- method:
  - conduct a series of hill-climbing searches from randomly generated initial states
  - stop when a goal is found
- analysis:
  - complete with probability approaching 1
  - requires 1/*p* restarts where *p* is the probability of success
    (1 success + 1/*p*-1 failures)

Hill Climbing: getBestSuccessors

```
getBestSuccessors(i,clauses)
   tc ← -1; succs ← {}
   for every proposition p in i
      i’ ← i.flipValueOf(p)
      n ← number of clauses true under i’
      if n > tc then tc ← n; succs ← {}
      if n = tc then succs ← succs + i’
   return succs
```

GSAT: Pseudo Code

```
function GSAT(clauses)
props ← clauses.getPropositions()
loop at most MAXLOOP times
   i ← randomInterpretation(props)
   while not clauses.evaluate(i) do
      succs ← getBestSuccessors(i,clauses)
      i ← succs.selectOne()
   if clauses.evaluate(i) return i
return unknown
```

GSAT Evaluation

- experimental results:
  - solved every problem correctly that Davis-Putnam could solve, only much faster
  - begins to return “unknown” on problems orders of magnitude larger than Davis-Putnam can solve
- analysis:
  - problems with many local maxima are difficult for GSAT

WalkSAT

- idea:
  - start with random interpretation
  - choose a random proposition to flip
  - accept if it represents an uphill or level move
  - otherwise accept it with probability $e^{-\delta/T(s)}$ where:
    - δ = decrease in number of true clauses under i’
    - T(s) = monotonically decreasing function from number of steps taken to temperature value

Overview

- Encoding Planning Problems as Satisfiability Problems (SAT)

➡ Efficient SAT Solving Algorithms