SAT-Based Planning

Using Propositional SAT-Solvers to Search for Plans

Literature

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 $\mathcal{P}=(\Sigma,s,g)$ into a propositional formula $\Phi$ such that:
  - $\mathcal{P}$ has a solution if and only if $\Phi$ is satisfiable, and
  - every model $\mu$ of $\Phi$ corresponds to a solution plan $\pi$ 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 \( r2 \): \( \text{at}(\text{robr},\text{loc}1) \) and \( \text{at}(\text{robr},\text{loc}2) \)
  - \( q1 \) and \( q2 \): \( \text{at}(\text{robq},\text{loc}1) \) and \( \text{at}(\text{robq},\text{loc}2) \)
  - \( ur \) and \( uq \): \( \text{unloaded}(\text{robr}) \) and \( \text{unloaded}(\text{robq}) \)

- containers:
  - \( a1, a2, ar, \) and \( aq \): \( \text{in}(\text{conta},\text{loc}1) \), \( \text{in}(\text{conta},\text{loc}2) \), \( \text{loaded}(\text{conta},\text{robr}) \), and \( \text{loaded}(\text{conta},\text{robq}) \)
  - \( b1, b2, br, \) and \( bq \): \( \text{in}(\text{contb},\text{loc}1) \), \( \text{in}(\text{contb},\text{loc}2) \), \( \text{loaded}(\text{contb},\text{robr}) \), and \( \text{loaded}(\text{contb},\text{robq}) \)

- 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: \( \{r1 \iff \text{true}, q2 \iff \text{true}, a1 \iff \text{true}, b2 \iff \text{true}, ur \iff \text{true}, uq \iff \text{true}\} \)
**Intended vs. Unintended Models**

- possible models:
  - intended model: $\{r_1 \leftarrow \text{true}, r_2 \leftarrow \text{false}, q_1 \leftarrow \text{false}, q_2 \leftarrow \text{true}, u_r \leftarrow \text{true}, u_q \leftarrow \text{true}, a_1 \leftarrow \text{true}, a_2 \leftarrow \text{false}, a_r \leftarrow \text{false}, a_q \leftarrow \text{false}, b_1 \leftarrow \text{false}, b_2 \leftarrow \text{true}, b_r \leftarrow \text{false}, b_q \leftarrow \text{false}\}$
  - unintended model: $\{r_1 \leftarrow \text{true}, r_2 \leftarrow \text{true}, q_1 \leftarrow \text{false}, q_2 \leftarrow \text{true}, u_r \leftarrow \text{true}, u_q \leftarrow \text{true}, a_1 \leftarrow \text{true}, a_2 \leftarrow \text{false}, a_r \leftarrow \text{true}, a_q \leftarrow \text{false}, b_1 \leftarrow \text{false}, b_2 \leftarrow \text{true}, b_r \leftarrow \text{false}, b_q \leftarrow \text{false}\}$
- encoding: add negated propositions not in state
  - example:
    - $r_1 \land \neg r_2 \land \neg q_1 \land q_2 \land u_r \land u_q \land a_1 \land \neg a_2 \land \neg a_r \land \neg a_q \land \neg b_1 \land b_2 \land \neg b_r \land \neg b_q$

**Encoding the Set of Goal States**

- goal: defined as set of states
  - example:
    - swap the containers
    - all states in which $a_2$ and $b_1$ are true
- propositional formula can encode multiple states:
  - example: $a_2 \land b_1$ ($2^{12}$ 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: $\gamma(s_1,\text{Mr12}) = s_2$ where:
  - $s_1$ described by $r_1 \land \neg r_2$
  - $s_2$ described by $\neg r_1 \land r_2$
- problem: $r_1 \land \neg r_2 \land \neg r_1 \land r_2$ has no model
- idea: extend propositions with state index
  - example: $r_{1\_1} \land \neg r_{2\_1} \land \neg r_{1\_2} \land r_{2\_2}$
  - model: $\{r_{1\_1}\leftarrow\text{true}, r_{2\_1}\leftarrow\text{false}, r_{1\_2}\leftarrow\text{false}, r_{2\_2}\leftarrow\text{true}\}$
**Extended Action Propositions**

- use same mechanism to describe actions applied in different states:
  - example: $\text{Mr12}_1$: move robot $r$ from location 1 to location 2 in state $s_2$
  - action encoding: $\text{Mr12}_1 \Rightarrow (r_{1_1} \land \neg r_{2_1} \land \neg r_{1_2} \land r_{2_2})$

**Bounded Planning Problems**

- encoding in two steps:
  - bounded planning problem: for a given planning problem $\mathcal{P}=(\Sigma, s, g)$ 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:
  - $\land_{f \in si} f_0 \land \land_{f \in si} \neg f_0$

- goal states:
  - $\land_{f \in g^+} f_n \land \land_{f \in g^-} \neg f_n$
**Encoding Actions**

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

For $0 \leq i \leq n-1$:
- $a_i \Rightarrow (\land_{f \in \text{precond}(a)} f_i \land \land_{f \in \text{effects}^+(a)} f_{i+1} \land \land_{f \in \text{effects}^-(a)} \neg f_{i+1})$

**Encoding Frame Axioms**

- Use explanation closure axioms for more compact SAT problem

For $0 \leq i \leq n-1$:
- $(f_i \land \neg f_{i+1}) \Rightarrow (\lor_{a \in A \land f \in \text{effects}^-(a)} a_i) \land$
- $(\neg f_i \land f_{i+1}) \Rightarrow (\lor_{a \in A \land f \in \text{effects}^+(a)} a_i)$
Encoding Exclusion Axioms

- allow only exactly one action at each step

- for 0 ≤ i ≤ n-1 and a≠a’, a,a’∈A:
  - ¬ a_i v ¬ a’_i

Overview

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

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

Conjunctive Normal Form

- formula $F$ is in conjunctive normal form (CNF) iff:
  - $F$ has the form $F_1 \land \ldots \land F_n$ and
  - each $F_i$, $i \in 1\ldots n$, 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 $I$, $l(F) = l(F')$
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
    - $(1 \text{ success} + 1/p-1 \text{ failures})$
Hill Climbing:
getBestSuccessors

getBestSuccessors(i, clauses)

\[ tc \leftarrow -1; \ succs \leftarrow \{} \]

for every proposition \( p \) in \( i \)

\[ i' \leftarrow i.\text{flipValueOf}(p) \]

\[ n \leftarrow \text{number of clauses true under } i' \]

if \( n > tc \) then \( tc \leftarrow n; \ succs \leftarrow \{} \)

if \( n = tc \) then \( succs \leftarrow succs + i' \)

return \( succs \)

GSAT: Pseudo Code

function GSAT(clauses)

\[ props \leftarrow clauses.\text{getPropositions()} \]

loop at most MAXLOOP times

\[ i \leftarrow \text{randomInterpretation}(props) \]

while not \( clauses.\text{evaluate}(i) \) do

\[ succs \leftarrow \text{getBestSuccessors}(i, clauses) \]

\[ i \leftarrow succs.\text{selectOne()} \]

if \( clauses.\text{evaluate}(i) \) 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:
    - $\delta$ = 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