Part VI

Propositional Satisfiability Techniques

Introduction

- The general idea is to map a planning problem to a well-known problem for which effective algorithms exist
- More specifically, the idea is to formulate a planning problem as a proposition satisfiability problem
- The approach can be split in 3 steps:
  1. A planning problem is encoded as propositional formula
  2. A satisfiability decision procedure determines whether the formula is satisfiable by assigning truth values to the propositional variables
  3. A plan is extract from the assignments determined by the satisfiability decision procedure

Ouline

- In this chapter we focuse on:
  1. the encoding of planning problem into satisfiability problem
  2. the description of some existing satisfiability procedures used in planning
  3. discussing a way to translate a planning problem to a proposition formula
  4. showing how standard decision procedures can be used as planning procedure
  5. discussing some different ways to encode planning problem

Planning problem as Satisfiability Problems

Planning problem as Satisfiability Problems

- Suppose a classical planning problem $\mathcal{P} = (\Sigma, s_0, S_g)$ where
  - $\Sigma = (S, A, \gamma)$ is the planning domain
  - S the set of states
  - A the set of actions
  - γ the deterministic transition function
  - s_0 the initial state and
  - S_g the set of goal states.
- In planning as satisfiability approach, a problem $\mathcal{P}$ must be encoded as propositional formulate with the property that any its models to solution plan of $\mathcal{P}$
- A model of propositional formula is a truth assignemnent to its variables for which the formula is evaluated to true
- A formula is satisfiable if a model of the formula exists.

States as Propositional Formula

- Similar to classical representation, propositional formulas are used to represent facts that hold in a state
- Suppose we would like to describe the state with one robot r1 and one location l1:

$$\text{at(r1,l1)} \wedge \neg\text{loaded(r1)}$$

- A model μ to this formula is the pne that assigns true to the propositional variable at(r1,l1), and false to loaded(r1) such as

$$\mu = \{\text{at(r1,l1)} \leftarrow \mathit{true}, \text{loaded(r1)} \leftarrow \mathit{false}\}$$

States as Propositional Formula

Intended and Unintended Model

- Suppose we have second location l2, we have a second propositional variable at(r1,l2)
- We want to represent that r1 is at location l1 and not loaded
- We have two models

$$\mu_1 = \{\text{at(r1,l1)} \leftarrow \mathit{true}, \text{loaded(r1)} \leftarrow \mathit{false}, \text{at(r1,l2)} \leftarrow \mathit{true}\}$$
$$\mu_2 = \{\text{at(r1,l1)} \leftarrow \mathit{true}, \text{loaded(r1)} \leftarrow \mathit{false}, \text{at(r1,l2)} \leftarrow \mathit{false}\}$$

- μ_1 is a uninted model (r1 cannot be at two locations at the same time)
- To remove unintended model we have to modify our previous formulas

$$\text{at(r1,l1)} \wedge \neg\text{at(r1,l2)} \wedge \neg\text{loaded(r1)}$$

States as Propositional Formula

Representing a set of states

- A propositional formula can reprsent sets of states rather than a single state, e.g.,

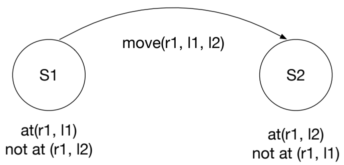
$$(\text{at(r1,l1)} \wedge \neg\text{at(r1,l2)}) \vee (\neg\text{at(r1,l1)} \wedge \text{at(r1,l2)}) \wedge \neg\text{loaded(r1)}$$

Remarks

1. Encoding states as propositional formulas is straightforward
2. Propositional formulas encode states but the encode the dynamics of the system
3. We need to add specific propositional formula to encode the state evolving

States Transitions as Propositional Formulas

- The state resulting from the application of an action is defined by the transition function $\gamma : S \times A \rightarrow S$



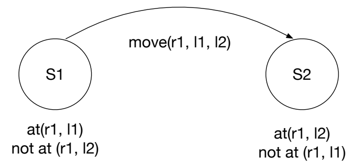
The state s_1 and s_2 can be defined as follows:

$$s_1 = \{\text{at(r1,l1)} \wedge \neg\text{at(r1,l2)}\}$$
$$s_2 = \{\neg\text{at(r1,l1)} \wedge \neg\text{at(r1,l2)}\}$$

⇒ We need to differentiate the propositional variable true in a state

States Transitions as Propositional Formulas

- The transition below



can be represented by the following propositional formula:

$$\text{at(r1,l1,s1)} \wedge \neg\text{at(r1,l2,s1)} \wedge \neg\text{at(r1,l1,s2)} \wedge \text{at(r1,l2,s2)}$$

- A model for this formula is

$$\mu_3 = \{ \text{at(r1,l1,s1)} \leftarrow \mathit{true}, \text{at(r1,l2,s2)} \leftarrow \mathit{false}, \text{at(r1,l1,s2)} \leftarrow \mathit{false}, \text{at(r1,l2,s2)} \leftarrow \mathit{true}\}$$

States Transitions as Propositional Formulas

- We encode the state transition from s_1 to s_2 but ...
- We need to encode the fact that move(r1,l1,l2) causes this transition
- To do this, we have to introduce a new propositional variable move(r1,l1,l2,s1)
- The transition function $\gamma(s_1, moved r1, l1, l2)$ can be encoded as follows:

$$\text{move(r1,l1,l2,s1)} \wedge \text{at(r1,l1,S1)} \wedge \neg\text{at(r1,l2nS1)} \wedge \neg\text{at(r1,l1,s2)} \wedge \text{at(r1,l2,s2)}$$

- A model for this formula is

$$\mu_4 = \{ \text{move(r1,l1,l2,S1)} \leftarrow \mathit{true}, \text{at(r1,l1,s1)} \leftarrow \mathit{true}, \text{at(r1,l2,s2)} \leftarrow \mathit{false}, \text{at(r1,l1,s2)} \leftarrow \mathit{false}, \text{at(r1,l2,s2)} \leftarrow \mathit{true}\}$$

Planning problem as Propositional Formulas

- Now that we know, encode a state and a transition as propositional formulas, we can encode a planning problem to a propositional formula Φ. The construction of Φ is based on three ideas:
  1. Restrict the planning problem to the problem of finding a plan of known length n. This problem is called the **b** ounded planning problem
  2. Transform the bounded planning problem into a satisfiability problem
  3. Try to solve incrementally step by step the satisfiability problem by increasing the size of the bounded planning problem

Planning problem as Propositional Formulas

Encoding predicates

- Predicate symbol with k arguments is translated into a symbol of $k+1$ arguments where the last argument is the step
- In the case of predicate symbols at(r1,l1), we have at(r1,l1,i), $0 \leq i \leq n$
- This means that the robot r1 is at location l1 at step i

Remark

We call fluent the ground atomic formula that describe states at a given step, e.g., at(r1,l1,i).

Planning problem as Propositional Formulas

Encoding actions

- Action symbol with k arguments is translated into a symbol of $k+1$ arguments where the last argument is the step
- In the case of action symbols move(r1,l1,l2), we have move(r1,l1,l2,i), $0 \leq i \leq n-1$
- This means that the robot r1 move from location l1 to location l2 at step i
- The action move(r1,l1,l2,i) executed at step i will produce its effects at step $i+1$

Planning problem as Propositional Formulas

Bound on maximum plan length

- A bounded planning problem can be easily extended to the problem of finding a plan length $\leq n$, with the use of dummy action that does nothing
- If a solution exists, the plan has a maximum length less or equal to the number of sates of the problem
- The number of states of a problem is double exponential in the number of constants symbols and predicates arity

$$n \leq 2^{|D|^{A_p}}$$

  where
  - $|D|$ is the number of constants of the domain
  - A_p is the maximum arity of the predicates
- In practice, we hope find a solution before exploring the whole search space ...

A Complete Encoding

Initial State

- The initial state is encoded as a proposition that is the conjunction of fluents that hold in the initial state and of the negation of those that do not hold, all of them instantiated at step 0:

$$\bigwedge_{f \in s_0} f_0 \wedge \bigwedge_{f \notin s_0} \neg f_0$$

- The initial state is thus fully specified

A Complete Encoding

Goal States

- The set of goal states is encoded as a proposition that is the conjunction of fluents that must hold at step n:

$$\bigwedge_{f \in g^+} f_n \wedge \bigwedge_{f \notin g^-} \neg f_n$$

- The goal state is partially specified by the conjunction of the fluents that hold in all the goal states

A Complete Encoding

Action Effects

- The fact that an action, when applicable, has some effects is encoded with a formula that states that if the action takes place at a given step, then its preconditions must hold at that step and its effects will hold at the next step.
- Let A be the set of all possible actions. For each $a \in A$ and for each $0 \leq i \leq n-1$; we have:

$$a_i \Rightarrow \left(\bigwedge_{p \in \text{precond}(a)} p_i \wedge \bigwedge_{e \in \text{effects}(a)} e_{i+1} \right)$$

A Complete Encoding

Frame Problem

- We need tp state that an action changes only the fluents that are in its effects
- In other words, if a fluent changes, then one of the action that have that fluent in its effects has been executed.
- For each fluent f and for each $0 \leq i \leq n-1$, we have:

$$\neg f_i \wedge f_{i+1} \quad \Rightarrow \quad \left(\bigvee_{a \in A | f_i \in \text{effects}^+(a)} a_i \right) \wedge$$

$$f_i \wedge \neg f_{i+1} \quad \Rightarrow \quad \left(\bigvee_{a \in A | f_i \in \text{effects}^-(a)} a_i \right)$$

A Complete Encoding

Exclusion axiom

- The fact that only one action occurs at each step is garanteed by the following formula, which is called the complete exclusion axiom
- For each for each $0 \leq i \leq n-1$ and for each distinct $a_i, b_i \in A$, we have:

$$\neg a_i \vee \neg b_i$$

A simple concrete example (1/3)

- Consider a simple example, where we have on robot r1 and two location l1 and l2
- Let suppose that the robot can move between two locations
- In the initial state, the robot is at l1
- In the goal state, the robot must be at l2
- The operator that moves the robot is:

  move(r,l,l')

  precond: at(r,l)

  effects: at(r,l'), ¬at(r,l)
- A solution plan of length 1 is enough to reach the goal state

A simple concrete example (2/3)

- The initial and goal states are encoded as formulas (init), and (goal), respectively:

$$(init) \quad \text{at(r1,l1,0)} \wedge \neg \text{at(r1,l2,0)}$$
$$(goal) \quad \text{at(r1,l2,1)} \wedge \neg \text{at(r1,l1,1)}$$

- The action is encoded as:

$$(move1) \quad \text{move(r1,l1,l2,0)} \Rightarrow \text{at(r1,l1,0)} \wedge \text{at(r1,l2,1)} \wedge \neg \text{at(r1,l1,1)}$$
$$(move2) \quad \text{move(r1,l2,l1,0)} \Rightarrow \text{at(r1,l2,0)} \wedge \text{at(r1,l1,1)} \wedge \neg \text{at(r1,l2,1)}$$

A simple concrete example (3/3)

- The frame axioms are expressed as:

$$(\text{at1}) \quad \neg \text{at(r1,l1,0)} \wedge \text{at}(r1, l1, 1) \Rightarrow \text{move(r1,l2,l1,0)}$$
$$(\text{at2}) \quad \neg \text{at(r1,l2,0)} \wedge \text{at}(r1, l2, 1) \Rightarrow \text{move(r1,l1,l2,0)}$$
$$(\text{at3}) \quad \text{at(r1,l1,0)} \wedge \neg \text{at}(r1, l1, 1) \Rightarrow \text{move(r1,l1,l2,0)}$$
$$(\text{at4}) \quad \text{at(r1,l2,0)} \wedge \neg \text{at}(r1, l2, 1) \Rightarrow \text{move(r1,l2,l1,0)}$$

- The exclusion axiom:

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

Encoding Formalisation and Definition

- Let $\Sigma = (S, A, \gamma)$ be a deterministic state transition system
- Let $\mathcal{P} = (\Sigma, s_0, S_g)$ be a classical planning problem where s_0 and S_g are the initial and goal states of the planning problem $\mathcal{P}$
- Let Enc be a function that takes a planning problem $\mathcal{P}$ and a length bound n and returns a propositional formula $\Phi : Enc(\mathcal{P}, \backslash) = \Phi$

Definition

Enc encodes the planning problem $\mathcal{P}$ to a satisfiability problem when the following hold: Φ is satisfiable iff there exist a solution plan of length n to $\mathcal{P}$. We say, in short, that Enc encodes planning to satisfiability.

Planning by Satisfiability

Planning by Satisfiability

- One a bounded planning problem is encoded to a satisfiability problem, a model for the resulting formula can be constructd by a satisfiability decision procedure
- Many procedures have been proposed in particular:
  1. The algorithms based on the Davis-Putnam procedure are sound and complete
  2. The procedures bases on the idea of randomized local search, called stochastic procedures are sound but not complete. This procedures can sometimes scale up better than the complete algorithms.

Davis and Putnam Procedure

- The Davis and Putman procedure is one of the first proposed but still one of the most used
- The procedure takes as input a propositional formula Φ and return a model μ if Φ is satisfiable
- The procedure assumes that Φ is in CNF (Conjunctive Normal Form), i.e., a conjunction of literals (positive or negative propositional variables)
- The procedure performs a depth-first search through the space of all possible assignments until either a model is found or the entire search space without is explored
- The procedure uses a simplification mechanism to reduce the size of the formula when variable are assigned

Davis and Putnam Procedure

Algorithm

Algorithm (Davis-Putnam(Φ, μ))

```
if ∅ ∈ Φ then return failure
if Φ = ∅ then return μ
Unit-Propagate (ϕ, μ)
Select a variable P such that P or ¬P occurs in Φ
Davis-Putnam (ϕ ∪ {P}, μ)
Davis-Putnam (ϕ ∪ {¬P}, μ)
```

Algorithm (Unit-Propagate(Φ, μ))

```
while there is a unit clause {l} ∈ Φ do
    μ ← μ ∪ {l}
    for every clause C ∈ Φ do
        if l ∈ C then Φ ← Φ − {C}
        else if ¬l ∈ C then Φ ← Φ − {C} ∪ {C − {¬l}}
```

Davis and Putnam Procedure

Remarks

Remarks

- The variable section rule may be as simple as choosing the first remaining variable in Φ
- It can select variables occurring in a clause of minimal length
- It can select variables occurring with a maximum number of occurrences in minimum-size clauses

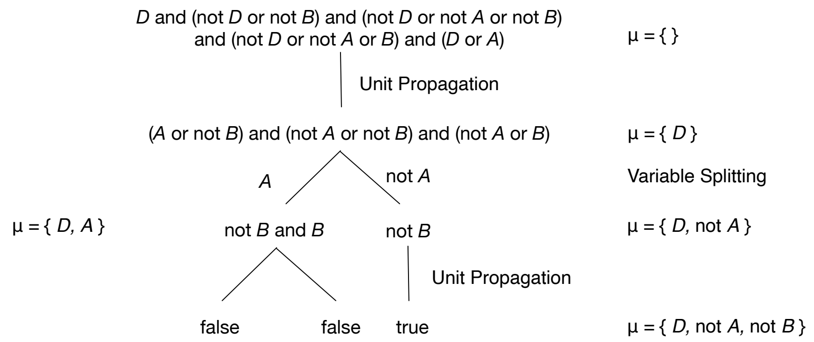
$\Rightarrow$ eliminate clauses as early as possible in the search

Davis and Putnam Procedure

Example

- Consider the following propositional formula in CNF:

$$\Phi = D \wedge (\neg D \vee A \vee \neg B) \wedge (\neg D \vee \neg A \vee \neg B) \wedge (\neg D \vee \neg A \vee B) \wedge (D \vee A)$$



Stochastic Procedures

- Davis-Putman procedire works with partial assignments
  - at each step, not all variables are assign a truth value
  - at the initial step, μ is empty, then it is incrementally constructed by adding assignments to variables
- A alternative idea is to devise algorithms that work from the beginning on total assignments
- A trivial algorithms is the one that
  1. Randomly selects an initial total assignments
  2. Checks wether there is a model and if not
  3. iteratively choose a different assignment until a model is found or all assignments were tested
- This algorithm is sound and complete but not feasible in practice
- This algorithm can be used as basic idea for incomplete satisfiability decision procedures

Local Search Procedure

Algorithm (Local-Search-SAT(Φ))

Select a total assignment μ for Φ randomly
while *μ does not satisfy Φ* **do**
  if μ' *s.t.* Cost $(\mu', \phi) <$ Cost (μ, Φ) *and* $|\mu - \mu'| = 1$ **then**
    $\mu \leftarrow \mu'$
  else
    return Failure
  end
end

Remarks

- The procedure is based on randomized local search
- The cost funnction compute the number of clauses of Φ that is satisfy by μ
- The procedure is incomplete due to local minima

GSAT Algorithm

Algorithm (Basic-GSAT(Φ))

Select a total assignment μ *for* Φ *randomly*
while μ *does not satisfy* Φ **do**
  foreach $P \in \Phi$, $\mu_p \leftarrow$ `Flip` (P, μ) **do**
    $\mu \leftarrow argmin_{\mu_p}$`Cost`$(\mu_p, \Phi)$
  end
end
return μ

Remarks

- The choice of the assignment mechanism helps avoid local minima
- Real implementation of GSAT restart from a new initial assignment when the procedure fails
- The procedure is incomplete

Iterative Repair Approach

- The idea is to iteratively modify a truth assignment such that it satisfies one of the unsatisfied clauses selected according to some criterion
- A unsatisfied clause is seen as a "fault" to be "repair"
- This method differ from previous ones in that at every step the number of clause unsatisfied may increase

Algorithm (Iterative-Repair(Φ))

Select any μ
while μ *does not satisfy* Φ **do**
  if *iteration limit exceeded* **then return** `Failure`
  Select any clause $C \in \Phi$ *not satisfied by* μ
  Modify μ *to satisfy* C
end
return μ

Iterative Repair Approach

Random-Walk

- A well-known version of Iterative-Repair procedure is Random-Walk
- Random-Walk implements the step "Modify μ to satisfy C" in a way that ressembles to GSAT
  - By flipping iteratively one variable in C
- It has been shown that Random-Walk suffers several problems on formulas of a certain complexity
- A probabilistic greedy version of Random-Walk has been proposed, called Walksat
- After C is selected randomly, Walksat selects randomly the varaible to flipped among the following possibilities to mix non greedy and greedy search:
  1. a random variable in C or
  2. the variable C that lead to the greatest number of satisfied clauses when flipped

Different Encodings

Different Encodings

- The encoding presented previously is one encoding
- Since the SAT search procedure takes time exponential in the number of variables, the choice of encoding is critical

Action Representation

- The encoding presented previously, each action is represented by a different logical variable at each step
- This results in $|A| = n|O||D|_0^A$ propositional variables to encode actions with
  - n the number of steps
  - O the number of operators
  - D the number of constant in the domain and
  - A_0 the maximum arity of operators

Action Representation

Simple Operator Splitting

- The idea is to replace each n-ay action proposition with n unary propositions
- For instance, a proposition variable move(r1,l1,l2,i) is replaced by

  move(r1,i) $\wedge$ move(l1,i) $\wedge$ move(l2,i)
- The advantage is that each operator share the same variable
- Simple operator splitting results in $|A| = n|O||D|A_0$

Action Representation

Overloaded Operator Splitting

- Thus generalize the idea if simple operator splitting by allowing different operator to share the same variable
- This done by representing the action, e.g., move, as the argument of a general action predicate Act
- For instance, move(r1,l1,l2,i) is replaced by

  Act(move, i) $\wedge$ Act1(r1,i) $\wedge$ Act2(l1,i) $\wedge$ Act3(l2,i)
- An action for instance fly(r1,l1,l2,i) can share variables Act1(r1,i), Act2(l1,i) and Act3(l2,i) with move(r1,l1,l2,i)
- Overloaded operator splitting results in $|A| = n()|O| + |D|A_0$

Action Representation

Bitwise

- The idea is to provide m bits that encode each action
- For instance, if we have 4 actions:
  - a_1 = move(r1,l1,l2,i)
  - a_2 = move(r1,l2,l1,i)
  - a_3 = move(r2,l1,l2,i)
  - a_4 = move(r2,l2,l1,i)
- We can use just two bits : bit1(i) and bit2(i)
- The formula bit1(i) $\wedge$ bit2(i) can represent a_1, bit1(i) $\wedge \neg$bit2(i) a_2, etc.
- Bitwise representation results in reducing the number of variables to $\lceil log_2|A|\rceil$

Frame Axiom

Classical Frame Axiom

- This is the most obvious formalization of the fact that actions change only what is explicitly states
- For each action a, for each fluent $f \notin$ effects(a), and for each $0 \leq i \leq n-1$ we have:

$$f_i \wedge a_i \Rightarrow f_{i+1}$$

- Problem if a_i does not occurs at step i, a_i is false and the frame axiom does not constraints the value of f_{i+1} which can therefore takes an arbitrary value

Frame Axiom

Classical Frame Axiom

- For instance; consider this classical frame axiom:

  unloaded(r1,i) $\wedge$ move(r1,l1,l2,i) $\Rightarrow$ unloaded(r1, $i+1$)
- When the robot is move from l1 to l2 at step i the robot might be loaded magically
- A solution is to add the at-least-one axioms, i.e., a disjunction of every possible action at step i, that assures that least one action is performed:

$$\bigvee_{a \in A} a_i$$

Frame Axiom

Explanatory Frame Axiom

- In our first encoding, Explanatory Frame Axiom was used to encode that just one action occurs at a given step.
- Thus solution plan are totally ordered
- It could be interested to have concurrent plan
- Explanatory Frame Axiom can be relaxed by defining only inconsistent actions

Size of the different encodings

Actions	Number of variables
Regular	$n\|F\| + n\|O\|\|D\|_0^A$
Simple Splitting	$n\|F\| + n\|O\|\|D\|A_0$
Overloaded Splitting	$n\|F\| + n(\|O\| + \|D\|A_0)$
Bitwise	$n\|F\| + n\lceil log_2\|O\|\|D\|_0^A\rceil$

- n the number of steps
- O the number of operators
- D the number of constant in the domain and
- A_0 the maximum arity of operators
- $|F|$ is the number of fluents with $|F| = |P||D|_p^A$ with $|P|$ the number of predicate and A_p the maximum arity of predicates

Size of the different encodings

Actions	Frame axiom	Number of variables
Regular	Classical	$O(n\|F\|\|A\|)$
Regular	Explanatory	$O(n\|F\|\|A\| + n\|A\|^2)$
Simple Splitting	Classical	$O(n\|F\|\|A\|A_0 + n\|A\|A_0^{\|A\|})$
Simple Splitting	Explanatory	$O(n\|F\|A_0^{\|A\|} + n(\|A\|A_0)^2)$
Overloaded Splitting	Classical	$O(n\|F\|\|A\|A_0) + n(\|A\|A_0)^{\|A\|})$
Overloaded Splitting	Explanatory	$O(n\|F\|(\|A\|A_0)^2 + n(\|F\|\|A\|A_0)^{\|A\|})$
Bitwise	Classical	$O(n\|F\|\|A\|log_2\|A\|)$
Bitwise	Explanatory	$O(n\|F\|\|A\|(log_2\|A\|)^{\|A\|})$

- $|A| = |O||D|^{A_0}$ is the number of actions of the problem

To go further

Exercices

Exercice 1

Are the following formulas satisfied ?

$$(\neg D \vee A \vee \neg B) \wedge (\neg D \vee \neg A \vee \neg B) \wedge (\neg D \vee \neg A \vee B) \wedge (D \vee A)$$
$$(D \rightarrow (A \rightarrow \neg B)) \wedge (D \vee (\neg A \rightarrow \neg B)) \wedge (\neg D \vee \neg A \vee B) \wedge (D \leftarrow A)$$

Run the Davis-Putnam procedure on them and explain the result. Also run a stochastic procedure.

To go further

Further readings

- H. Kautz, B. Selman
  Planning as Satisfiability.
  ECAI 1992: 359-363
- J. Rintanen
  Planning and SAT.
  Handbook of Satisfiability 2021: 765-789