CSC2542
SAT-Based Planning

Sheila McIlraith
Department of Computer Science
University of Toronto
Fall 2010

Acknowledgements

Some of the slides used in this course are modifications of Dana Nau's lecture slides for the textbook *Automated Planning,* licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/

Other slides are modifications of slides developed by Malte Helmert, Bernhard Nebel, and Jussi Rintanen.

For this topic, some slides come from Henry Kautz, Ulrich Scholz, and Yiqiao Wang.

I have also used some material prepared by Dan Weld, P@trick Haslum and Rao Kambhampati.

I would like to gratefully acknowledge the contributions of these researchers, and thank them for generously permitting me to use aspects of their presentation material.

Segue

- The problem of finding a valid plan from the planning graph can be encoded on any combinatorial substrate
- Alternatives:
  - CSP [GP-CSP – Do & Kambhampati, 2000]
  - SAT [Blackbox; SATPLAN – Kautz & Selman, 1996+]
  - ASP [Son et al]
  - IP [Vossen et al]
- This is the notion of "Translation to General Problem Solver" that we discussed in our first technical lecture.

Here we discuss SAT as the combinatorial substrate.

Motivation

- Propositional satisfiability (SAT):
  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*
  i.e., an assignment of truth values to the propositions that makes the formula true?
- This was the first problem shown to be NP-complete.
- Lots of research on algorithms for solving SAT.
- Key idea behind SAT-based planning:
  - Translate classical planning problems into satisfiability problems, and solving them using a highly optimized SAT solver.

Basic Approach

- Suppose a plan of length n exists
- Encode this hypothesis in SAT
  - Initial state is true at t_0
  - Goal is true at t_n
  - Actions imply effects, etc
- Look for satisfying assignment
- Decode into plan

Evolution of SAT-based planners

- The success of this approach has largely been the result of impressive advances in the proficiency of SAT solvers.
- A continued limiting factor to this approach is the size of the CNF encoding of some problems.
- Thus, a key challenge to this approach has been how to encode the planning problem effectively. Such encodings have marked the evolution of SAT-based planners.

History…

- 1969 Plan synthesis as theorem proving (Green IJCAI-69)
- 1971 STRIPS (Fikes & Nilsson AIJ-71)
- Decades of work on "specialized theorem provers"

. . .

…History (enter SAT-based planners)…

- 1992 Satplan "approach" (Kautz & Selman ECAI-92)
  - convention for encoding STRIPS-style linear planning in axiom schema
  - Didn't appear practical
- Rapid progress on SAT solving
- 1996 (Kautz & Selman AAAI-96) (Kautz, McAllester & Selman KR-96)
  - Electrifying results (on hand coded formulae)
  - Key technical advance: parallel encodings where noninterfering actions could occur at the same time (i.e., Graphplan ideas) (but no compiler)
- 1997 MEDIC (Ernst *et al.* IJCAI-97)
  - First complete implementation of Satplan (with compiler)
- 1998 Blackbox (Kautz & Selman AIPS98 workshop)
  - Also performed mutex propagation before generating encoding

. . .

…History (IPC)….

- 1998 IPC-1 Blackbox performance comparable to the best
- 2000 IPC-2 Blackbox performance abysmal (Graphplan-style planners dominated)
- 2002 IPC-3 No SAT-based planners entered
- 2004 IPC-4 Satplan04 was clear winner of "optimal propositional planners"
- 2006 IPC-5 Satplan06 & Maxplan* (Chen Xing & Zhang IJCAI-07) dominated**

What accounts for the success in 2004 and 2006?

1) Huge advances in SAT solvers 2000-2004 (e.g., Seige, ZChaff)
   (indeed in 2004 they ran out of time and didn't include mutex propagation)
2) New competition problems that were "intrinsically hard"

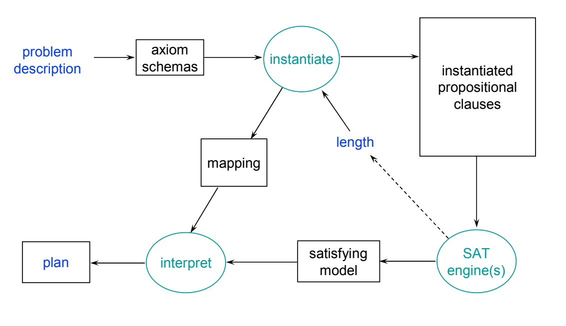
* Also a SAT-based planner

** dominated the "optimal planners" track. Note however that in the so-called "satisficing planners" track, e.g. the heuristic-search based planners that could not guarantee optimal length, satificing planners were able to solve much larger problems!

Outline

- Encoding planning problems as satisfiability problems
- Extracting plans from truth values
- Satisfiability algorithms
- Combining satisfiability with planning graphs
  - Blackbox & SatPlan

The SATPLAN Approach*



* Terminology: "SATPLAN approach" (circa 1992) vs. the SATPLAN planner of 2004, 2006 etc., the successor of Blackbox.

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 as we did with Graphplan:
  - for n = 0, 1, 2, …,
    - 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, 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: 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
  - $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$

13

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, e.g.,
  - 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 = move(r1,loc2,loc1)
      then a_i = move(r1,loc2,loc1,i)

14

Encoding Planning Problems

- Encode (P,n) as a formula Φ such that

  $\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ is a solution for (P,n) if and only if

  There is a satisfying assignment for Φ such that fluents $a_0, \ldots, a_{n-1}$ are true
- Let
  - A = {all actions in the planning domain}
  - S = {all states in the planning domain}
  - L = {all literals in the language}
- Φ is the conjunct of many other formulas …

15

Formulae in Φ

- Formula describing the **initial state**:

  $\bigwedge\{l_0 \mid l \in s_0\} \wedge \bigwedge\{\neg l_0 \mid l \in L - s_0\}$
- Formula describing the **goal**:

  $\bigwedge\{l_n \mid l \in g^+\} \wedge \bigwedge\{\neg l_n \mid l \in g^-\}$
- For every **action** a in A, formulae describing what changes a would make if it were the i'th step of the plan:
  - $a_i \Rightarrow \bigwedge\{p_i \mid p \in \text{Precond}(a)\} \wedge \bigwedge\{e_{i+1} \mid e \in \text{Effects}(a)\}$
- ***Complete exclusion*** axiom:
  - For all actions a and b, formulas saying they can't occur at the same time

    $\neg a_i \vee \neg b_i$
  - this guarantees there can be only one action at a time (i.e., a sequential plan. This is revisted in the blackbox encoding later.
- Is this enough?

16

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:

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

Example

- Planning domain:
  - one robot r1
  - two adjacent locations l1, l2
  - one operator (move the robot)
- Encode (P,n) where $n = 1$
  - Initial state: {at(r1,l1)}
    Encoding: at(r1,l1,0) $\wedge$ ¬at(r1,l2,0)
  - Goal: {at(r1,l2)}
    Encoding: at(r1,l2,1) $\wedge$ ¬at(r1,l1,1)
  - Operator: see next slide

Example (continued)

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

  Encoding:

  move(r1,l1,l2,0) ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)
  move(r1,l2,l1,0) ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)
  move(r1,l1,l1,0) ⇒ at(r1,l1,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l1,1)
  move(r1,l2,l2,0) ⇒ at(r1,l2,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l2,1)
  contradictions (easy to detect)
  move(l1,r1,l2,0) ⇒ …
  move(l2,l1,r1,0) ⇒ …
  move(l1,l2,r1,0) ⇒ …
  move(l2,l1,r1,0) ⇒ …
  nonsensical
- How to avoid generating the last four actions?
  - Assign data types to the constant symbols

Example (continued)

Solution: Add typing of parameters

- Locations: l1, l2
- Robots: r1
- Operator: move(r : robot, l : location, l' : location)
  precond: at(r,l)
  effects: at(r,l'), ¬at(r,l)

  Encoding:

  move(r1,l1,l2,0) ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)
  move(r1,l2,l1,0) ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)

Example (continued)

- Complete-exclusion axiom:
  $\neg$move(r1,l1,l2,0) $\vee$ $\neg$move(r1,l2,l1,0)
- Explanatory frame axioms:
  $\neg$at(r1,l1,0) $\wedge$ at(r1,l1,1) $\Rightarrow$ move(r1,l2,l1,0)
  $\neg$at(r1,l2,0) $\wedge$ at(r1,l2,1) $\Rightarrow$ move(r1,l1,l2,0)
  at(r1,l1,0) $\wedge$ $\neg$at(r1,l1,1) $\Rightarrow$ move(r1,l1,l2,0)
  at(r1,l2,0) $\wedge$ $\neg$at(r1,l2,1) $\Rightarrow$ move(r1,l2,l1,0)

Extracting a Plan

- Suppose we find a satisfying assignment for Φ.
  - This means *P* has a solution of length *n*
- For *i*=1,…,*n*, there will be exactly one action s.t. a_i = *true*
  - This is the *i*'th action of the plan.
- Example (from the previous slides):
  - Φ can be satisfied with move(r1,l1,l2,0) = *true*
  - Thus $\langle$move(r1,l1,l2,0)$\rangle$ is a solution for (*P*,0)
    - It's the only solution - no other way to satisfy Φ

Planning

- How to find an assignment of truth values that satisfies Φ?
  - Use a satisfiability (SAT) algorithm
    - Systematic search e.g., Davis-Putnam-Logemann-Loveland (DPLL)
    - Local search e.g., GSAT, Walksat
- Example: the *Davis-Putnam** 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)
    $\Phi = \{\{D\}, \{\neg D, A, \neg B\}, \{\neg D, \neg A, \neg B\}, \{\neg D, \neg A, B\}, \{A\}\}$
  - Two special cases:
    - If $\Phi = \varnothing$ then Φ is always *true*
    - If $\Phi = \{\ldots, \varnothing, \ldots\}$ then Φ is always *false* (hence unsatisfiable)

****NOTE:** DP is the term used in the text book but is actually a resolution procedure. DPLL(1962) is a refinement of DP(1960). "DP" is sometimes used to refer to "DPLL".*

The Davis-Putnam Procedure

Backtracking search through alternative assignments of truth values to literals

- μ = {literals to which we have assigned the value TRUE}; initially empty
- if Φ contains $\varnothing$ then
  - backtrack
- if Φ is $\varnothing$ then
  - μ is a solution
- while Φ contains a clause that's a single literal *l*
  - Remove clause containing l
  - Remove $\neg$*l* from clauses
- select a Boolean variable *P* in Φ
- do recursive calls on
  - $\Phi \cup P$
  - $\Phi \cup \neg P$

```
Davis-Putnam(Φ,μ)
    if ∅ ∈ Φ then return
    if Φ = ∅ then exit with 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)
end

Unit-Propagate(Φ,μ)
    while there is a unit clause {l} in Φ do
        μ ← μ ∪ {l}
        for every clause C ∈ Φ
            if l ∈ C then Φ ← Φ − {C}
            else if ¬l ∈ C then Φ ← Φ − {C} ∪ {C − {¬l}}
end
```

Local Search

- Let *u* be an assignment of truth values to all of the variables
  - cost(u,Φ) = number of clauses in Φ that are **not** satisfied by *u*
  - flip(P,u) = *u* except that *P*'s truth value is reversed

Boolean variable

- Local search:
  - Select a random assignment *u*
  - while cost(u,Φ) $\neq$ 0
    - if there is a *P* such that cost(flip(P,u),Φ) < cost(u,Φ) then
      - randomly choose any such *P*
      - $u \leftarrow$ 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 (local search algorithm)

- Basic-GSAT:
  - Select a random assignment *u*
  - while cost(u,Φ) $\neq$ 0
    - choose a *P* that minimizes cost(flip(P,u),Φ), and flip it
- Not guaranteed to terminate (in contrast to DPLL)

- WALKSAT
  - Like GSAT but differs in the method used to pick which variable to flip

- Both algorithms may restart with a new random assignment if trapped in local minima.
- Many versions of GSAT/WalkSAT. WalkSAT superior for planning.

But….

GSAT (local search algorithm)

- Basic-GSAT:
  - Select a random assignment *u*
  - while cost(u,Φ) $\neq$ 0
    - choose a *P* that minimizes cost(flip(P,u),Φ), and flip it
- Not guaranteed to terminate (in contrast to DPLL)

- WALKSAT
  - Like GSAT but differs in the method used to pick which variable to flip
  - WalkSAT first picks a clause which is unsatisfied by the current assignment, then flips a variable within that clause. The clause is generally picked at random among unsatisfied clauses. The variable is generally picked that will result in the fewest previously satisfied clauses becoming unsatisfied, with some probability of picking one of the variables at random. When picking at random, WalkSAT is guaranteed at least a chance of one out of the number of variables in the clause of fixing a currently incorrect assignment. When picking a guessed to be optimal variable, WalkSAT has to do less calculation than GSAT because it's considering fewer possibilities.
- The algorithm may restart with a new random assignment if no solution has been found for too long, as a way of getting out of local minima of numbers of numbers of unsatisfied clauses.
- Many versions of GSAT and WalkSat exist.
- WalkSAT superior for planning

BUT best DPLL-based solvers (e.g., currently Siege, previously ZChaff) are currently best!

Bottom Line

Previous discussion notwithstanding, the best solvers for SAT-based planning are currently DPLL-based solvers such as Satzilla, PrecoSAT (and previously RelSAT and before that Siege and before that ZChaff) that have the option of using random restarts and some other local-search “tricks”

Discussion of the '92 Satplan Approach

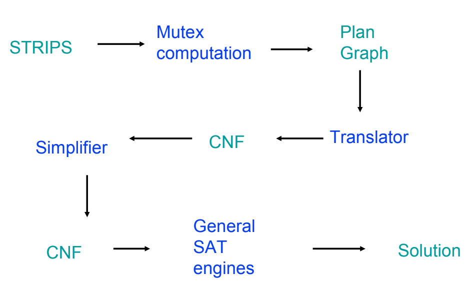
- Recall the overall approach:
  - for n = 0, 1, 2, …,
    - encode (P,n) as a satisfiability problem Φ
    - if Φ is satisfiable, then
      - From the set of truth values that satisfies Φ, extract a solution plan and return it
- How well does this work?

Discussion of the '92 Satplan Approach

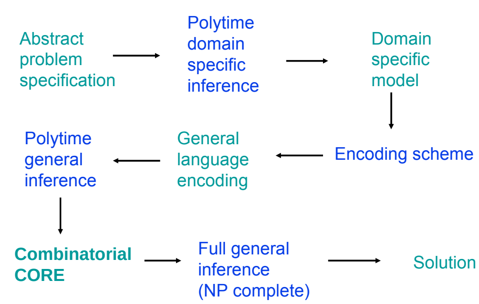
- Recall the overall approach:
  - for n = 0, 1, 2, …,
    - encode (P,n) as a satisfiability problem Φ
    - if Φ is satisfiable, then
      - From the set of truth values that satisfies Φ, extract a solution plan and return it
- How well does this work?
  - By itself, not practical (takes too much memory & time)
  - But it can be combined with other techniques
    - e.g., planning graphs

    (Remember historical discussion at the beginning of this lecture.)

Blackbox



Staged Inference



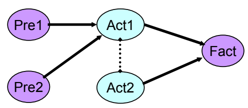
Exploiting the planning graph

The Basic Idea:

- The planning graph approximates the reachability graph by pruning unreachable nodes
- In logical terms, it is actually limiting negative binary propagation

Translation of the Planning Graph

- Fact ⊃ Act1 ∨ Act2
- Act1 ⊃ Pre1 ∧ Pre2
- ¬Act1 ∨ ¬Act2



Improved Encodings

- Translations of Logistics.a:
- STRIPS → Axiom Schemas → SAT
  (Medic system, Weld et. al 1997)
  - 3,510 variables, 16,168 clauses
  - 24 hours to solve
- STRIPS → Plan Graph → SAT
  - 2,709 variables, 27,522 clauses
  - 5 seconds to solve!

SatPlan* (sucessor to Blackbox)

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

**1992 – "Satplan Approach",vs, 2004+ - Satplan implementation, successor to Blackbox*

Improved SAT Encodings for Planning

- As I mentioned at the outset, advances in SAT-based planning have largely been marked by advances in encodings.
  E.g., translations of IPC Logistics.a domain
  - STRIPS → Axiom Schemas → SAT (Medic system, Weld et. al 1997)
    - 3,510 variables, 16,168 clauses
    - **24 hours** to solve
  - STRIPS → Plan Graph → SAT (Blackbox)
    - 2,709 variables, 27,522 clauses
    - **5 seconds** to solve!
- Biggest drawback to Blackbox successors is the enormous sized CNFs
  E.g., Satplan06 encoding of IPC-5 Pipesworld domain with n=19
    - 47,000 variables, 20,000,000 clauses

…. And this is a big reason why heuristic search (aka "satisficing planners") can solve much bigger problems

Action Encoding in Medic* [Ernst et al, IJCAI 1997]

Representation	One Propositional Variable per	Example
Regular	**fully-instantiated action** $n\|F\| + n\|O\|\|D\|^{A_0}$	**move(r1,l1,l2,i)**
Simply-split	**fully-instantiated action's argument** $n\|F\| + n\|O\|\|D\|A_0$	**move1(r1,i) ∧ move2(l1,i) ∧ move3(l2,i)**
Overloaded-split	**fully-instantiated argument** $n\|F\| + n(\|O\|+\|D\|A_0)$	**act(move, i) ∧ act1(r1, i) ∧ act2(l1, i) ∧ act3(l2, i)**
Bitwise	**Binary encodings of actions** $n\|F\| + n[\log_2 \|O\|\|D\|^{A_0}]$	**Bit1**

more vars

more clauses

n – number of steps; |F| - number of fluents; |D| - size of domain
|O| - number of operators; A_0 – maximum arity of predicates

** Recall Medic was pre-Blackbox and had no action parallelism*

Final word for now

- SAT-based planners historically did well in the "optimal" planning track of IPC (as opposed to the satisficing track) because of the iterative nature of the construction of the planning graph representation. In contrast, in the "satisficing" track, heuristic search planners were far outperforming SAT-based planners and scaling to larger problems, while still computing good quality plans. With the advent of heuristic search planners that iterate to find better plans (e.g., LAMA) heuristic searh planners are
- Recent research advances have centred around different encodings and associated query strategies. There have also been interesting advances on using SAT-based planning for cost-optimal planning and the like

REMINDER: Administrative Announcements

- **Tutorial Time:** If you're taking the course for credit, please (re)vist the doodle poll and see whether you can work towards finding a time when we can all meet. We're at an impass!
- I will be posting a schedule with project milestone dates and the due date for the assignment.
- The lecture in 2 weeks will be given by our TA, Christian Muise.
- Suggested readings for next week:
  - Part III introduction of GNT
  - Chapter 9 of GNT
  - A review paper that I will post on our web page.
- Other Issues?