Chapter 7
Propositional Satisfiability Techniques
Motivation

- Propositional satisfiability: given a boolean formula
  - e.g., \((P \lor Q) \land (\neg Q \lor R \lor S) \land (\neg R \lor \neg P)\), does there exist a model
  - i.e., an assignment of truth values to the propositions that makes the formula true?
- This was the very first problem shown to be NP-complete
- Lots of research on algorithms for solving it
  - 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
SATPLAN

Henry Kautz and Bart Selman (1996)

Idea

- Transform a planning problem into a satisfiability problem.
- Use a general-purpose SAT solver to find a satisfying assignment.
- Translate the satisfying assignment back to a plan for the original problem.

Results

- Efficient.
- Key issue: SAT encoding of the planning problem.
- Huge SAT instances (around 10,000 variables)
The Satisfiability (SAT) Problem

Given

- A boolean formula over $n$ variables.

Find

- A satisfying assignment, that is, an assignment to all variables such that the formula evaluates to TRUE.

The formula can be unsatisfiable, satisfiable, or a tautology.

Example

- $(x_1 + x_2 + \bar{x}_3)(\bar{x}_2 + x_4)(\bar{x}_1 + \bar{x}_5)$
  
- $(\bar{x}_1 + x_2 + x_3 + \bar{x}_4 + x_5)(\bar{x}_3)(x_2 + x_5)(\bar{x}_5)(x_1)$

- $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 1, 0)$
SAT as CSP

SAT is a Constraint Satisfaction Problem!

- **Variables**: Boolean variables in the formula
- **Domains**: Each variable can take values from \{0, 1\}
- **Constraints**: Each clause of size \(k\) is a \(k\)-ary constraint

\[
(\overline{x_2} + x_3 + x_4)
\]

\(x_2, x_3,\) and \(x_4\) cannot be simultaneously \(x_2 = 1, x_3 = 0,\) and \(x_4 = 0\)

- **Solution**: An assignment that satisfies all constraints (clauses).
Architecture of a SAT-based planner

Problem Description
- Init State
- Goal State
- Actions

Compiler (encoding) → CNF

Simplifier (polynomial inference)

Solver (SAT engine/s)

Decoder

Plan

Increment plan length
If unsatisfiable

satisfying model

mapping
Compiler
- take a planning problem as input, guess a plan length, and generate a propositional logic formula, which if satisfied, implies the existence of a solution plan

Simplifier
- use fast techniques such as unit clause propagation and pure literal elimination to shrink the CNF formula

Solver
- use systematic or stochastic methods to find a satisfying assignment. If the formula is unsatisfiable, then the compiler generates a new encoding reflecting a longer plan length

Decoder
- translate the result of solver into a solution plan.
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 \(\Phi\)
    - if \(\Phi\) is satisfiable, then
      - From the set of truth values that satisfies \(\Phi\), a solution plan can be constructed, so 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, for example, `at(r1,loc1)` into `at-r1-loc1`
- We do the same thing here, but we don’t bother to rewrite
  - Just write the proposition as `at(r1,loc1)`
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
  - \( \text{at}(r_1, \text{loc}_1, i) \) is a fluent that’s true iff \( \text{at}(r_1, \text{loc}_1) \) is in \( s_i \)
  - We’ll use \( l_i \) to denote the fluent for literal \( l \) in state \( s_i \)
    - e.g., if \( l = \text{at}(r_1, \text{loc}_1) \) then \( l_i = \text{at}(r_1, \text{loc}_1, i) \)
  - \( a_i \) is a fluent saying that \( a \) is the \( i \)th step of \( \pi \)
    - e.g., if \( a = \text{move}(r_1, \text{loc}_2, \text{loc}_1) \) then \( a_i = \text{move}(r_1, \text{loc}_2, \text{loc}_1, i) \)
Encoding Planning Problems

- Encode \((P,n)\) as a formula \(\Phi\) such that
  \[\pi = \langle a_0, a_1, \ldots, a_{n-1} \rangle\] is a solution for \((P,n)\) if and only if
  \(\Phi\) can be satisfied in a way that makes the fluents \(a_0, \ldots, a_{n-1}\) true.

- Let
  - \(A = \{\text{all actions in the planning domain}\}\)
  - \(S = \{\text{all states in the planning domain}\}\)
  - \(L = \{\text{all literals in the language}\}\)

- \(\Phi\) is the conjunct of many other formulas …
Formulas in $\Phi$

- Formula describing the initial state:
  \[ \land \{l_0 \mid l \in s_0\} \land \land \{\neg l_0 \mid l \in L - s_0\} \]

- Formula describing the goal:
  \[ \land \{l_n \mid l \in g^+\} \land \land \{\neg l_n \mid l \in g^-\} \]

- For every action $a$ in $A$, formulas describing what changes $a$ would make if it were the $i$'th step of the plan:
  \[ a_i \implies \land \{p_i \mid p \in \text{Precond}(a)\} \land \land \{e_{i+1} \mid e \in \text{Effects}(a)\} \]

- Complete exclusion axiom:
  - For all actions $a$ and $b$, formulas saying they can’t occur at the same time
    \[ \neg a_i \lor \neg b_i \]
  - this guarantees there can be only one action at a time

- Is this enough?
Frame Axioms

- **Classical Frame axioms:**
  - Formulas describing what *doesn’t* change between steps $i$ and $i+1$

- **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:

$$
(\neg l_i \land l_{i+1} \Rightarrow \forall a \in A \{a_i/l \in \text{effects}^+(a)\})
\land
(l_i \land \neg l_{i+1} \Rightarrow \forall a \in A \{a_i/l \in \text{effects}^-(a)\})
$$
Encodings of Planning to SAT
Frame Axioms

- **Classical**: (McCarthy & Hayes 1969)
  - State what fluents are left unchanged by an action
  - \( \text{clear}(d, i) \land \text{move}(a, b, c, i) \Rightarrow \text{clear}(d, i+1) \)
  - Problem: if no action occurs at step \( i \) nothing can be inferred about propositions at level \( i+1 \)
  - Sol: at-least-one axiom: at least one action occurs

- **Explanatory**: (Haas 1987)
  - State the causes for a fluent change
  - \( \text{clear}(d, i) \land \neg \text{clear}(d, i+1) \Rightarrow \)
    
    \[
    (\text{move}(a, b, d, i) \lor \text{move}(a, c, d, i) \lor \ldots \lor \text{move}(c, \text{Table}, d, i))
    \]
Example

- Planning domain:
  - one robot \(r_1\)
  - two adjacent locations \(l_1, l_2\)
  - one operator (move the robot)

- Encode \((P,n)\) where \(n = 1\)

  - Initial state: \(\{\text{at}(r_1,l_1)\}\)
    Encoding: \(\text{at}(r_1,l_1,0) \land \neg\text{at}(r_1,l_2,0)\)

  - Goal: \(\{\text{at}(r_1,l_2)\}\)
    Encoding: \(\text{at}(r_1,l_2,1) \land \neg\text{at}(r_1,l_1,1)\)

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

- How to avoid generating the last four actions?
  - Assign data types to the constant symbols like we did for state-variable representation

contradictions (easy to detect)
nonsensical
Example (continued)

- 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:
  \[ \neg \text{move}(r1,l1,l2,0) \lor \neg \text{move}(r1,l2,l1,0) \]

- Explanatory frame axioms:
  \[ \neg \text{at}(r1,l1,0) \land \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l2,l1,0) \]
  \[ \neg \text{at}(r1,l2,0) \land \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l1,l2,0) \]
  \[ \text{at}(r1,l1,0) \land \neg \text{at}(r1,l1,1) \Rightarrow \text{move}(r1,l1,l2,0) \]
  \[ \text{at}(r1,l2,0) \land \neg \text{at}(r1,l2,1) \Rightarrow \text{move}(r1,l2,l1,0) \]
Extracting a Plan

- Suppose we find an assignment of truth values that satisfies $\Phi$.
  - 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 $i$’th action of the plan.

- Example (from the previous slides):
  - $\Phi$ can be satisfied with $\text{move}(r1,l1,l2,0) = true$
  - Thus $\langle \text{move}(r1,l1,l2,0) \rangle$ is a solution for $(P, 1)$
    » It’s the only solution - no other way to satisfy $\Phi$
Birthday Dinner Example

- Goal: \( \neg \text{garb} \land \text{dinner} \land \text{present} \)
- Init: \( \text{garb} \land \text{clean} \land \text{quiet} \)
- Actions:
  - Cook
    - Pre: clean
    - Effect: dinner
  - Wrap
    - Pre: quiet
    - Effect: present
  - Carry
    - Pre:
    - Effect: \( \neg \text{garb} \land \neg \text{clean} \)
  - Dolly
    - Pre:
    - Effect: \( \neg \text{garb} \land \neg \text{quiet} \)
Constructing SATPLAN sentence

- Initial sentence (clauses): garb\(_0\), clean\(_0\), quiet\(_0\), \neg\text{present}\(_0\), \neg\text{dinner}\(_0\)
- Goal (at depth 2): \neg\text{garb}\(_2\), \text{present}\(_2\), \text{dinner}\(_2\)
- Action\(_t\) \rightarrow (\text{Pre}_t \land \text{Eff}_{t+1}) \text{ [in clause form]}  
  - Cook\(_0\) \rightarrow (\text{clean}\(_0\) \land \text{dinner}\(_1\) )
- Explanatory Frame Axioms: For every state change, say what could have caused it  
  - garb\(_0\) \land \neg \text{garb}\(_1\) \rightarrow (\text{dolly}\(_0\) \lor \text{carry}\(_0\) ) \text{ [in clause form]} 
- Complete exclusion axiom: For all actions a and b, add \neg a\(_t\) \lor \neg b\(_t\)  
  - this guarantees there can be only one action at a time
Planning

- How to find an assignment of truth values that satisfies $\Phi$?
  - Use a satisfiability algorithm

- Example: the *Davis-Putnam* algorithm

  - First need to put $\Phi$ into conjunctive normal form
    
    $\Phi = D \land (\neg D \lor A \lor \neg B) \land (\neg D \lor \neg A \lor \neg B) \land (\neg D \lor \neg A \lor B) \land A$

  - Write $\Phi$ as a set of *clauses* (disjuncts of literals)
    
    $\Phi = \{D, (\neg D \lor A \lor \neg B), (\neg D \lor \neg A \lor \neg B), (\neg D \lor \neg A \lor B), A\}$

  - Two special cases:
    
    - $\Phi = \{\}$ is a formula that’s always *true*
    - $\Phi = \{\ldots, (), \ldots\}$ is a formula that’s always *false*
Algorithms for the Satisfiability Problem

Complete or Systematic Methods

- Explore the space of all possible assignments systematically.
- If there is a satisfying assignment, it *will* be found.
- If no satisfying assignment is found, the formula *is* unsatisfiable.

Incomplete or Stochastic Methods

- Stochastic moves in the space of all possible assignments.
- If there is a satisfying assignment, it *may* be found.
- If no satisfying assignment is found, the formula *may* or *may not* be unsatisfiable.
**SAT solvers**

- **Systematic SAT solvers**
  - DPLL algorithm
  - Perform a backtracking depth-first search through the space of partial truth assignment, using unit-clause and pure-literal heuristics

- **Stochastic SAT solvers**
  - Search locally using random moves to escape from local minima.
  - incomplete
  - GSAT: perform a greedy search. After hill climbing for a fixed amount of flips, it starts anew with a freshly generated, random assignment.
  - WALKSAT: improve GSAT by adding additional randomness akin to simulated annealing
The Davis-Putnam Procedure

Depth-first backtracking through combinations of truth values

- Select a variable $P$ in $\Phi$
- Recursive call on $\Phi \land P$
  - Simplify: remove all occurrences of $\neg P$
  - If $\Phi = \{\ldots, (), \ldots\}$ then backtrack
  - Else if $\Phi = \{\}$ then have a solution
- Recursive call on $\Phi \land \neg P$
  - Simplify: remove all occurrences of $P$
  - If $\Phi = \{\ldots, (), \ldots\}$ then backtrack
  - Else if $\Phi = \{\}$ then have a solution

Davis-Putnam($\Phi, \mu$)

- if $\emptyset \in \Phi$ then return
- if $\Phi = \emptyset$ then exit with $\mu$
- Unit-Propagate($\Phi, \mu$)
  - select a variable $P$ such that $P$ or $\neg P$ occurs in $\phi$
  - Davis-Putnam($\Phi \cup \{P\}, \mu$)
  - Davis-Putnam($\Phi \cup \{\neg P\}, \mu$)

end

Unit-Propagate($\Phi, \mu$)

- while there is a unit clause $\{l\}$ in $\Phi$ do
  - $\mu \leftarrow \mu \cup \{l\}$
  - for every clause $C \in \Phi$
    - if $l \in C$ then $\Phi \leftarrow \Phi - \{C\}$
    - else if $\neg l \in C$ then $\Phi \leftarrow \Phi - \{C\} \cup \{C - \{\neg l\}\}$

end
DPLL: Unit Propagation and Purification

Unit Propagation

- If there is a unit clause, there is only one promising assignment to the corresponding variable (unary constraint).

- Example: \((x_1 + x_2 + x_3)(\overline{x}_2)(\overline{x}_1 + \overline{x}_5)(\overline{x}_2 + x_4)\)

- Make that assignment \((x_2 = 0)\) and eliminate the variable \((x_2)\).

Purification

- A pure variable appears purely in positive \((x_i)\) or negative \((\overline{x}_i)\) form.

- Assign \(x_i = 1\) in the positive case and \(x_i = 0\) in the negative case.

- Example: \(x_3\) and \(x_5\) are pure. Assign \(x_3 = 1\) and \(x_5 = 0\).


**DPLL: Branching**

**Branching or Splitting**

- If there are not unit clauses or pure variables, select an unassigned variable and try in turn the two possible assignments.
- Create a reduced formula in each case and continue recursively.

**Example**

- \((x_1 + x_2 + x_3)(\bar{x}_1 + \bar{x}_5)(\bar{x}_2 + x_4)(x_1 + \bar{x}_3 + x_5)\)

- Assume that \(x_1\) is selected.
- \(x_1 = 1\) gives \((\bar{x}_5)(\bar{x}_2 + x_4)\)
- \(x_1 = 0\) gives \((x_2 + x_3)(\bar{x}_2 + x_4)(\bar{x}_3 + x_5)\)
Local Search

- Let $u$ be an assignment of truth values to all of the variables
  - $\text{cost}(u, \Phi) =$ number of clauses in $\Phi$ that aren’t satisfied by $u$
  - $\text{flip}(P, u) = u$ with the truth value of $P$ reversed

- Local search:
  - Select a random assignment $u$
  - while $\text{cost}(u, \Phi) \neq 0$
    - if there is a $P$ such that $\text{cost}(\text{flip}(P, u), \Phi) < \text{cost}(u, \Phi)$ then
      - randomly choose any such $P$
      - $u \leftarrow \text{flip}(P, u)$
    - 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

- Basic-GSAT:
  - Select a random assignment $u$
  - while cost($u, \Phi$) $\neq 0$
    - choose the $P$ that minimizes cost(flip($P, u$), $\Phi$)
- Not guaranteed to terminate

- GSAT:
  - restart after a max number of flips
  - return failure after a max number of restarts

- Walksat
  - works better than both local search and GSAT
Walksat

For i=1 to max-tries
   A:= random truth assignment
For j=1 to max-flips
   If solution?(A) then return A else
   C:= random unsatisfied clause
   With probability p flip a random variable in C
   With probability (1- p) flip the variable in C that minimizes the number of unsatisfied clauses
Example

- Consider the following formula (Solution = \{D, \neg A, \neg B\})
  \[ \Phi = D \land (\neg D \lor A \lor \neg B) \land (\neg D \lor \neg A \lor \neg B) \land (\neg D \lor \neg A \lor B) \land (D \lor A) \]

- Local search select a random total assignment, e.g., \[ u = \{D, A, B\} \], under which only \((\neg D \lor \neg A \lor \neg B)\) is unsatisfied, and \(\text{Cost}(u, \Phi) = 1\)

- There is no \(u'\) such that \(|u - u'| = 1\) and \(\text{Cost}(u', \Phi) < 1\)

- Therefore search stops with failure

- If the initial guess was \(\{\neg D, \neg A, \neg B\}\), the search could find solution in one step
Example (Cont.)

- Consider the following formula (Solution = \{D, \neg A, \neg B\})
  \[ \Phi = D \land (\neg D \lor A \lor \neg B) \land (\neg D \lor \neg A \lor \neg B) \land (\neg D \lor \neg A \lor B) \land (D \lor A) \]

- Basic GSAT selects a random total assignment, say, \( u = \{D, A, B\} \),
- It has two alternatives:
  - flipping D or B (because corresponding cost is 1)
    (Flipping A has a cost of 2),
- if B is flipped, next A will be chosen, and solution is found
Example (Cont.)

- Consider the following formula (Solution = \{D, \neg A, \neg B\})
  \[ \Phi = D \land (\neg D \lor A \lor \neg B) \land (\neg D \lor \neg A \lor \neg B) \land (\neg D \lor \neg A \lor B) \land (D \lor A) \]

- Iterative-Repair (A simplified version of Walksat) with the same initial guess \( u = \{D, A, B\} \),

- It has to repair clause \((\neg D \lor \neg A \lor \neg B)\)

- Different assignments can repair this, one is the solution.

- Suppose \( u = \{\neg D, \neg A, \neg B\} \) is selected,

- it then must repair two clauses \( D \) and \((D \lor A)\),

- If \( D \) is selected then iterative repair finds the solution, but if \((D \lor A)\) is selected, it will have two choices one of the leads to solution
Recall the overall approach:

- for $n = 0, 1, 2, \ldots$,
  - encode $(P,n)$ as a satisfiability problem $\Phi$
  - if $\Phi$ is satisfiable, then
    - From the set of truth values that satisfies $\Phi$, extract a solution plan and return it

How well does this work?
Recall the overall approach:

- for \( n = 0, 1, 2, \ldots \),
  - encode \((P,n)\) as a satisfiability problem \(\Phi\)
  - if \(\Phi\) is satisfiable, then
    - From the set of truth values that satisfies \(\Phi\), extract a solution plan and return it

How well does this work?

- By itself, not very practical (takes too much memory and time)

But it can be combined with other techniques

- e.g., planning graphs
Parameters of SAT-based planner

- Encoding of Planning Problem into SAT
  - Frame Axioms
  - Action Encoding

- Encoding is important to the performance of Solver, since solver speed can be exponential in the size of the formula.

- Simplification

- SAT Solver(s)
# Action Encoding

<table>
<thead>
<tr>
<th>Representation</th>
<th>One Propositional Variable per</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Regular</strong></td>
<td>fully-instantiated action</td>
<td>Move(r1, l1, l2)</td>
</tr>
<tr>
<td><strong>Simply Splitting</strong></td>
<td>fully-instantiated action’s argument</td>
<td>Move(Arg1, r1) ( \land ) Move(Arg2, l1) ( \land ) Move(Arg3, l2)</td>
</tr>
<tr>
<td><strong>Overloaded Splitting</strong></td>
<td>fully-instantiated argument</td>
<td>Act(Move) ( \land ) Arg1(r1) ( \land ) Arg2(l1) ( \land ) Arg3(l2)</td>
</tr>
<tr>
<td><strong>Bitwise</strong></td>
<td>Binary encodings of actions</td>
<td>Bit1 ( \land ) ~Bit2 ( \land ) Bit3 ( (Move(r1, l1, l2) = 5) )</td>
</tr>
</tbody>
</table>

| more vars              | more clauses                            |
Linear Encoding

- Initial and Goal States
- Action implies both preconditions and its effects
- Only one action at a time
- Some action occurs at each time
  (allowing for do-nothing actions)
- Classical frame axioms
- Operator Splitting
Graphplan-based Encoding

- Goal holds at last layer (time step)
- Initial state holds at layer 0
- Fact at level i implies disjunction of all operators at level i–1 that have it as an add-effect
- Operators imply their preconditions
- Conflicting Actions (only action mutex explicit, fact mutex implicit)
Graphplan Encoding

Fact $\Rightarrow$ Act1 $\lor$ Act2

Act1 $\Rightarrow$ Pre1 $\land$ Pre2

$\neg$Act1 $\lor$ $\neg$Act2
Compare Graphplan with SAT

- Both approaches convert parameterized action schemata into a finite propositional structure representing the space of possible plans up to a given length
  - The planning graph
  - a CNF formula
- Both approaches use local consistency methods before resorting to exhaustive search
  - mutex propagation
  - Propositional simplification
- Both approaches iteratively expand their propositional structure until they find a solution
  - planning graph is extended when no solution is found
  - propositional logic formula is recreated for a longer plan length
- Planning graph can be automatically converted into CNF notation for solution with SAT solvers
Comparison with Plan Space Planning

- Plan Space planning
  - <5 primitive actions in solutions
  - Works best if few interactions between goals

- Constraint-based planning
  - Graphplan, SATPLAN, + descendents
  - 100+ primitive actions in solutions
  - Moderate time horizon <30 time steps
  - Handles interacting goals well
BlackBox

- The BlackBox procedure combines planning-graph expansion and satisfiability checking
  - It is roughly as follows:

- for $n = 0, 1, 2, \ldots$
  - **Graph expansion:**
    - create a “planning graph” that contains $n$ “levels”
  - Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
  - If it does, then
    - Encode $(P,n)$ as a satisfiability problem $\Phi$ but include only the actions in the planning graph
    - If $\Phi$ is satisfiable then return the solution
More about BlackBox

- Memory requirement still is combinatorially large, but less than satisifiability alone
- It was one of the two fastest planners in the 1998 planning competition
Graph Search vs. SAT

Time

Problem size / complexity

Graphplan

SATPLAN

Blackbox with solver schedule