Planning II

CPS 270

Ron Parr

What We Have Learned

- Not all hard problems are the same
- From CSPs, SAT, we have learned:
  - Problem structure gives deep insights into hardness of problem
  - Solution technique heavily influenced by problem structure
  - Structure gives insights into problem independent heuristics

What We Know About Planning

- Planning is hard
- Coming up with good heuristics is hard
- Initial attempts to use problem structure (plan space search) were very messy
  - Plan space search is complicated
  - Ensuring completeness, correctness is tricky

Can our expertise in CSPs help?

- Can planning be reduced to CSPs?
- Need to consider bounded-length plans
  - In general, this isn't too much of a problem because extremely long plans are an indication that we need to reformulate the problem (Towers of Hanoi)
- Our hope: Solve plan as a CSP at let our CSP insights do the work for us

Formulating Planning as a CSP

- Introduce Action(a,i) (binary) to indicate if action a is taken at step i.
  - We introduce |Actions| x plan_length variables
- We also need to represent the statements in our database using proposition(p,i) (binary) to indicate the truth of proposition p at time I
  - This introduces |propositions| x plan_length variables
  - But there's a catch…

Propositionalizing

- Also called "grounding out"
- Recall that domain descriptions an actions involve relations:
  - on(x,table)
  - clear(x)
- Propositions don't take arguments
  - arm_broken

Converting to Propositional Form

- Consider on(x,y)
- Note that we considered this type of issue before when thinking about plan branching factor
- If there are n objects in the world, how many propositions do we need to express all possible realizations of on(x,y)?
- What if there are k relations that each take d variables?

Digression on Propositionalizing

- It turns out that in many planning domains the number of actions (k) is relatively low
- The number of variables involved in each action is usually relatively low too
- Hard to think of an action that involves six or more variables
- In general, propositionalizing is viewed as an inelegant trick that people would like to avoid
- Is fast planning possible w/o this?

Back to CSP formulation

- We now have action(move_x_y_z, i) = t iff we move x from y to z at time i.
- We also have proposition(on_y_z, i) = t iff we y is on z at time i.
- Now we need to set up our constraints so that the problem is satisfiable iff there exists a plan

Plan CSP Constraints

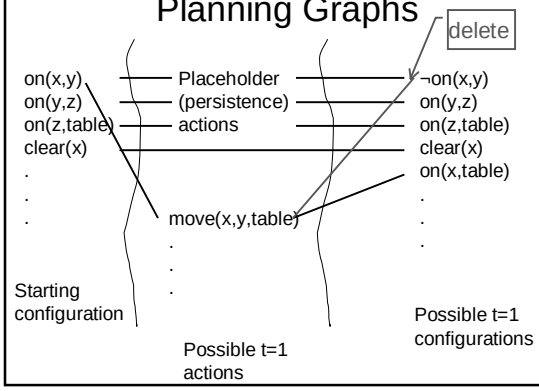
- Actions must be sequential
  - For all a,a' not(action(a,i) and action(a',i))
  - Another quadratic factor!
- Actions' effects on the world. If action(a,i)=t
  - Proposition(p,i-1) = t for each p in preconditions
  - Proposition(p,i)=t for each p in add list
  - Proposition(p,i)=f for each p in delete list
  - This is linear in the new action, proposition space

What's Missing?

- We need to express that propositions persist
  - Proposition(p,i) = f unless
    - It was true in previous step and not deleted
    - It was false in previous step but added
- We need to assert initial and final states
  - Easier than it sounds
  - We just set these variables to have the right values and the CSP does the rest

This works, but…

Planning Graphs



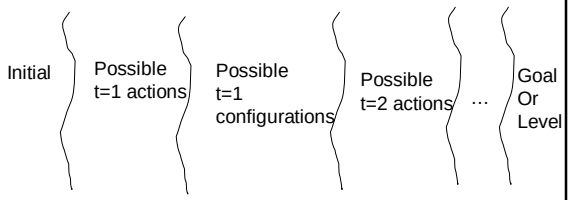
Facts About Planning Graphs

- Similar to CSP constraint graph
- The planning graph includes everything that *might* be true at a particular time
- Includes all actions that *might* be possible at a particular time
- Is a *relaxation* of the original problem

Why this is good

- What do we know about problem relaxations in search?
- Relaxations are a good way of developing admissable heuristics
- A major difficulty with planning is that we have trouble coming up with good heuristics
- Note that plan graphs can provide admissable heuristics for either direction (forward or regression [means/ends] search)

Plan Graph Form



Consider a particular world configuration c
Find the first phase containing all propositions in c
Distance to goal will be an admissable heuristic for forward search
Distance to initial config will be admissable for backward search
Why?

Why this isn't good enough

Mutual Exclusion Between Actions

- Two real (non-persistence) actions can't be taken simultaneously; we mark these mutually exclusive
- Types of mutual exclusion
  - Inconsistent effects/Interference
    - persist(on_x_y,1)
    - action(move_x_y_z, 1)
  - Competing needs
    - Precondition appears positive in one action
    - Appears negated in another

Extending graphs using mutex

For each planning phase:
1. Generate all actions with non-mutex preconditions
2. Mark as mutex all action/maintain pairs that conflict
3. Mark as mutex all action/action pairs with mutex preconds
4. Generate all potential propositions for next time step
5. Mark pairs of propositions that can only be generated by mutex actions as mutex

We now think of everything in terms of mutually compatible sets of propositions.

Plan Graphs with Mutex Constraints

- Extend forward until goal conjunctions appear non-mutex
- This is still a relaxation of the problem
- In essence, we have relaxed the original planning CSP so that we only worry about 2-consistency
- We still have an admissable heuristic
- For any configuration, we search for the earliest one in which the configuration propositions appear in non-mutex form

How do we use this?

- Form basis of a number of algorithms derived from original graphplan algorithm
- All work by constructing a planning graph and then using the graph structure to help guide search in some way
- Despite some apparent complexity, this turns out to be *much, much* cleaner, faster and easier to implement than plan space search algorithms

How well does it work?

- The initial graphplan algorithm was so much faster than competing algorithms it was hard to even compare them on the same scale.
- There is a web page devoted to graphplan:
  - http://www.cs.cmu.edu/~avrim/graphplan.html

Graphplan Summary

- Graphplan combines two concepts:
  - Constraint-based reasoning with a form of 2-consistency
  - Basic search
- Graphplan combines our knowledge of good search methods with our knowledge of good CSP methods

Planning with SAT

- Aome very fast algorithms exist for testing if SAT instances are satisfiable
  - GSAT, WalkSAT
- These don't provide negative proof, but they often find positive proof pretty quickly if it exists
- Planning with a bound on plan length is NP-complete
- All NP complete problems can be transformed to each other in poly time
- Why not transform planning to SAT and try WalkSAT?

Strange things about SATPlan

- This actually works pretty well for some domains
- Details of the transformation are somewhat tricky
- As with the CSP formulation, it tends to produce *very* large problem instances
- Can cause problems for domains with many items
- In a weird way, SATPlan is actually doing plan space search

Things to Ponder

- SATPlan uses a SAT solver.
- Every assignment to the variables of the SAT instances correspond to a (potentially) broken plan
- Changes in the variables correspond to plan repair operations, as in the classical, and often confusing plan-space search algorithms
- So, how do we get performance that is competitive with graph plan?

Modern Planning Conclusion

- Fast planning algorithms seem to rely simple, fast underlying methods
- Ruling out bad things quickly seems to help
  - Unit propagation in SAT formulations
  - Constraint propagation in graphplan variants
- This is still a very open area, not as clean as search/CSPs
- Structure, hierarchy, abstraction and more still under-utilized