Foundations of AI

13. Planning

Solving Logically Specified Problems
Step by Step

Wolfram Burgard, Andreas Karwath,
Bernhard Nebel, and Martin Riedmiller

Contents

Planning

- Given an *logical description* of the initial situation,
- a *logical description* of the goal conditions, and
- a *logical description* of a set of possible actions,

→ find a sequence of actions (a plan) that brings us from the initial situation to a situation in which the goal conditions hold.

Planning vs. Problem-Solving

Basic difference: **Explicit, logic-based representation**

- States/Situations: Through descriptions of the world by logical formula vs. data structures
  This way, the agent can explicitly think about and communicate
- Goal conditions as logical formulae vs. goal test (black box)
  The agent can also reflect on its goals.
- Operators: Axioms or transformation on formulae vs. modification of data structures by programs
  The agent can gain information about the effects of actions by inspecting the operators.

Planning vs. Automatic Programming

Difference between planning and automatic programming (generating programs):

- In planning, one uses a logic-based description of the environment.
- Plans are usually only linear programs (no control structures).

Planning as Logical Inference (1)

Planning can be elegantly formalized with the help of the *situation calculus*.

Initial state:

$$At(Home,s_0) \wedge \neg Have(milk,s_0) \wedge \neg Have(banana,s_0) \wedge \neg Have(drill,s_0)$$

Operators (successor-state axioms):

$$\forall a,s\ Have(milk, do(a,s)) \Leftrightarrow$$
$$[a = buy(milk) \wedge Poss(buy(milk), s) \vee Have(milk,s) \wedge a \neq \neg drop(milk)]$$

Goal conditions (query):

$$\exists s\ At(home, s) \wedge Have(milk, s) \wedge Have(banana,s) \wedge Have(drill,s)$$

When the initial state, all prerequisites and all successor-state axioms are given, the **constructive** proof of the existential query delivers a plan that does what is desired.

Planning as Logical Inference (2)

The variable bindings for s could be as follows:

$$do(go(home), do(buy(drill), do(go(hardware\_store), do(buy(banana), do(buy(milk), do(go(supermarket), s0))))))$$

I.e. the plan (term) would be

$$\langle go(super\_market), buy(milk), \ldots \rangle$$

However, the following plan is also correct:

$$\langle go(super\_market), buy(milk), drop(milk), buy(milk), \ldots \rangle$$

In general, planning by theorem proving is very inefficient

Specialized inference system for limited representation.

→ Planning algorithm

The STRIPS Formalism

STRIPS: STanford Research Institute Problem Solver (early 70s)

The system is obsolete, but the formalism is still used. Usually simplified version is used:

World state (including initial state): Set of ground atoms (called fluents), no function symbols except for constants, interpreted under closed world assumption (CWA). Sometimes also standard interpretation, i.e. negative facts must be explicitly given

Goal conditions: Set of ground atoms

Note: No explicit state variables as in sitation calculus. Only the current world state is accessible.

STRIPS Operators

Operators are triples, consisting of

Action Description: Function name with parameters (as in situation calculus)

Preconditions: Conjunction of positive literals; must be true before the operator can be applied (after variables are instantiated)

Effects: Conjunction of positive and negative literals; positive literals are added (ADD list), negative literals deleted (DEL list) (no **frame** problem!).

Op(Action*: Go(here,there),*
Precond*: At(here) , Path(here, there),*
Effect*: At(there) , ¬At(here))*

Actions and Executions

- An **action** is an operator, where all variables have been *instantiated*:
- *Op(* Action: *Go(),*
  Precond: *At(Home)* , *Path(Home, SuperMarket),*
  Effect: *At(Supermarket)* , *¬At(Home)*)
- An action can be **executed** in a state, if its precondition is satisfied. It will then bring about its effects.

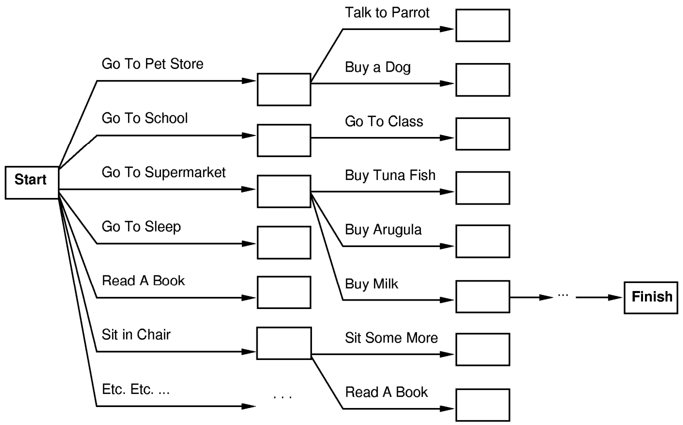
Linear Plans

- A sequence of actions is a **plan**
- For a given initial state ***I*** and goal conditions ***G***, such a plan ***P*** can be **successfully executed** in ***I*** iff there exists a sequence of states $s_0, s_1, \ldots, s_n$ such that
  - the *i*th action in ***P*** can be executed in s_{i-1} and results in s_i
  - s_0 = ***I*** and s_n satisfies ***G***
- ***P*** is called a **solution** to the **planning problem** specified by the *operators*, ***I*** and ***G***

Searching in the State Space

We can now search through the state space (the set of all states formed by truth assignments to fluents) – and in this way reduce planning to searching.

We can search forwards (**progression planning**):



Or alternatively, we can start at the goal and work backwards (**regression planning**).

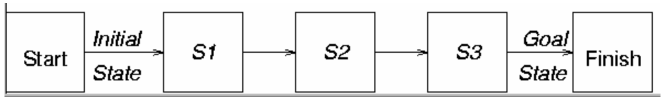
Possible since the operators provide enough information

Searching in the Plan Space

Instead of searching in the state space, we can search in the *space of all plans*.

The initial state is a **partial plan** containing only start and goal states:

The goal state is a **complete plan** that solves the given problem:



Operators in the plan space:

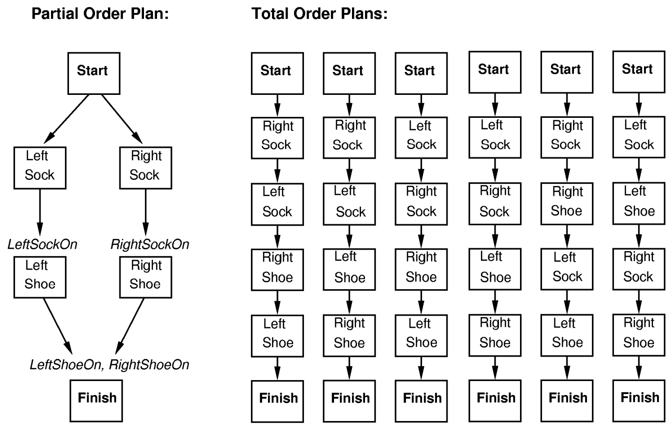
Refinement operators make the plan more complete (more steps etc.)

Modification operators modify the plan (in the following, we use only refinement operators)

Plan = Sequence of Actions?

Often, however, it is neither meaningful nor possible to commit to a specific order early-on (put on socks and shoes).

→ **Non-linear** or **partially-ordered plans** (**least-commitment planning**)



Representation of Non-linear Plans

A plan step = STRIPS operator (or action in the final plan)

A **plan** consists of

- A set of **plan steps** with partial ordering (<), where $S_i < S_j$ implies S_i must be executed before S_j.
- A set of **variable assignments** $x = t$, where x is a variable and t is a constant or a variable.
- A set of **causal relationships** $S_i \rightarrow S_j$ means "S_i produces the precondition c for S_j" (implies $S_i < S_j$).

Solutions to planning problems must be **complete** and **consistent**.

Completeness and Consistency

Complete Plan:

Every precondition of a step is fulfilled:

$$\forall S_j \ \forall c \in Precond(S_j):$$

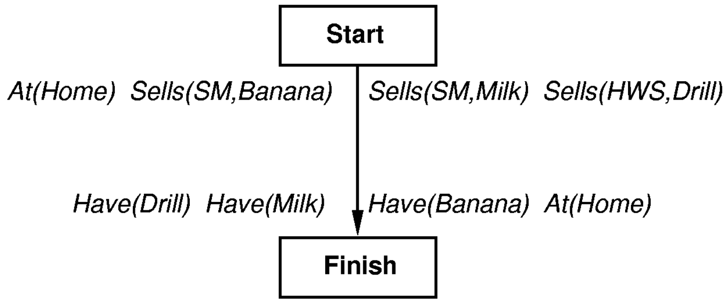
$$\exists S_i \text{ with } S_i < S_j \text{ and } c \in Effects(S_i) \text{ and}$$
$$\text{for every linearization of the plan:}$$
$$\forall S_k \text{ with } S_i < S_k < S_j, \neg c \notin Effect(S_k).$$

Consistent Plan:

if $S_i < S_j$, then S_j S_i and
if $x = A$, then $x \neq B$ for distinct A and B for a variable x. (*unique name assumption* = UNA)

A **complete, consistent plan** is called a **solution** to a planning problem (all linearizations are *executable linear plans*)

Example



Actions:

Op(`Action`: Go(here, there),
`Precond`: At(here) ∧ Path(here, there),
`Effect`: At(there) ∧ ¬At(here))

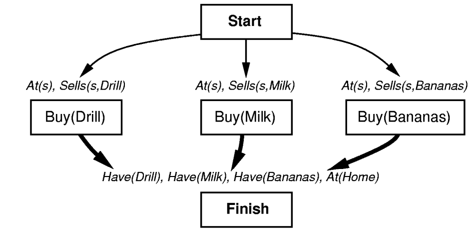
Op(`Action`: Buy(store, x),
`Precond`: At(store) ∧ Sells(store, x),
`Effect`: Have(x))

Note: there, here, x, store are variables.

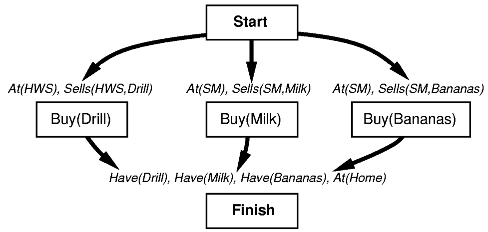
Note: In figures, we may just write *Buy(Banana)* instead of *Buy(SM, Banana)*

Plan Refinement (1)

Regression Planning:
Fulfils the **Have** predicates:



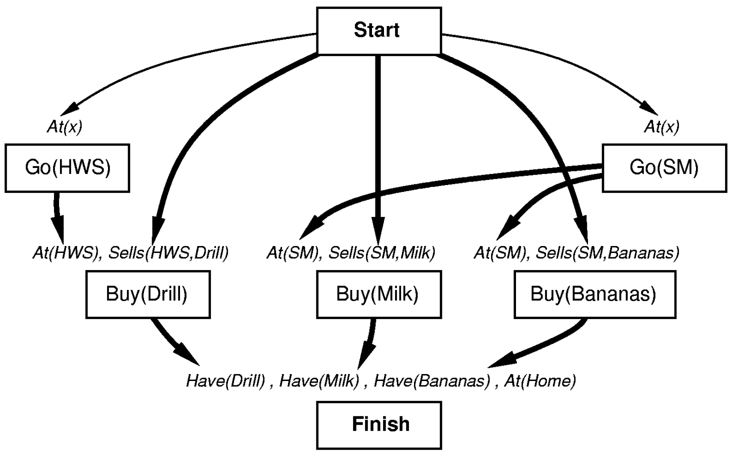
... after instantiation of the variables:



Thin arrow = <, thick arrow = causal relationship + <

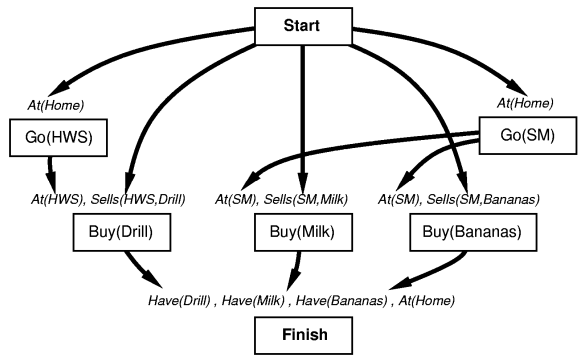
Plan Refinement (2)

Shop at the right store...



Plan Refinement (3)

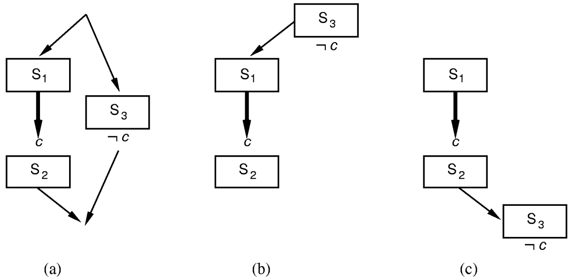
First, you have to go there...



Note: So far no searching, only simple backward chaining.

Now: Conflict! If we have done **go(HWS)**, we are no longer **At(home)**. Likewise for **go(SM)**.

Protection of Causal Links



(a) Conflict: S_3 threatens the causal relationship between S_1 and S_2.

Conflict solutions:

(b) **Demotion**: Place the threatening step before the causal relationship.

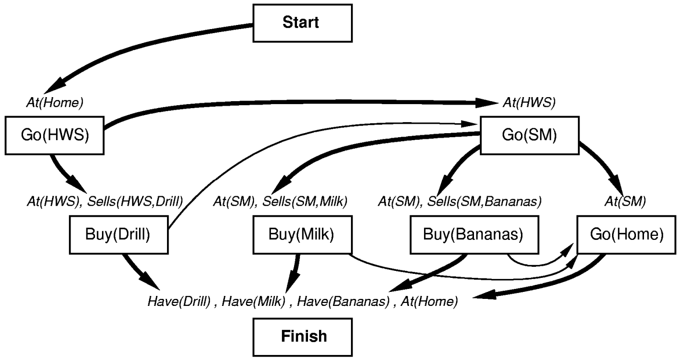
(c) **Promotion**: Place the threatening step after the causal relationship.

A Different Plan Refinement...

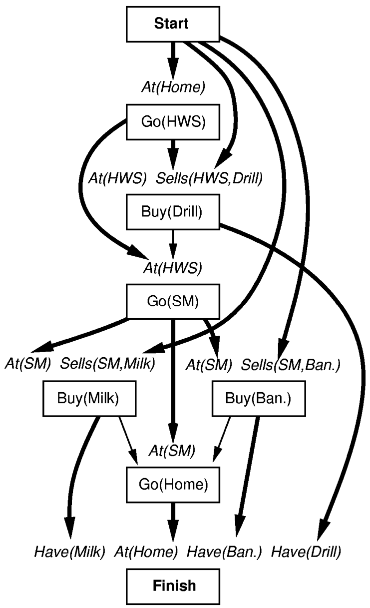
- We cannot resolve the conflict by "protection".

→ It was a mistake to choose to refine the plan.

- **Alternative:** When instantiating *At(x)* in *go(SM)*, choose *x* = HWS (with causal relationship)
- **Note:** This threatens the purchase of the drill → promotion of *go(SM)*.



The Complete Solution



The POP Algorithm

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, c ← SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

---

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

Properties of the POP Algorithm

Correctness: Every result of the POP algorithm is a complete, correct plan.

Completeness: If breadth-first-search is used, the algorithm finds a solution, given one exists.

Systematicity: Two distinct partial plans do not have the same total ordered plans as a refinement provided the partial plans are not refinements of one another (and totally ordered plans contain causal relationships).

Problems: Informed choices are difficult to make & data structure is expensive

→ Instantiation of variables is not addressed.

New Approaches

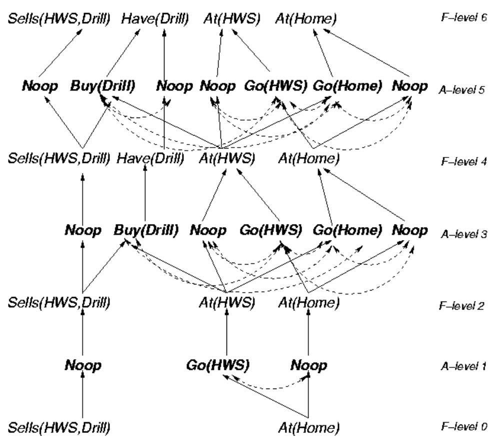
- Since 1995, a number of new algorithmic approaches have been developed, which are much faster than the POP algorithm:
  - Planning based on planning graphs
  - Satisfiability based planning
  - BDD-based approaches (good for multi-state problems)
  - Heuristic-search based planning
- Note: all approaches work on propositional representations, i.e., all operators are already instantiated!

Planning Graphs

- Parallel execution of actions possible
- Assumption: Only positive preconditions
- Describe possible developments in a layered graph (fact level/action level)
  - links from (positive) facts to preconditions
  - positive effects generate (positive) facts
  - negative effects are used to mark conflicts
- Extract plan by choosing only non-conflicting parts of graph

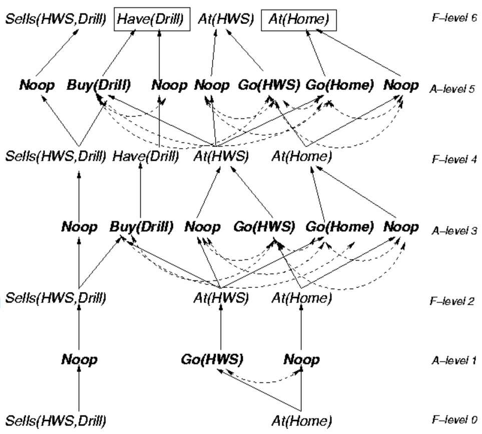
Generating a Planning Graph

- Start with initial fact level 0.
- Add all applicable actions
- In order to propagate unchanged property *p*, use special action $noop_p$
- Generate all positive effects on next fact level
- Mark conflicts (between actions that cannot be executed in parallel)
- Expand planning graph as long as not all atoms in fact level



Extract a Plan

- Start at last fact level with goal facts
- Select minimal set of **non-conflicting** actions generating the goals
- Use preconditions of these actions as goals on next lower level
- Backtrack if no non-conflicting choice is possible



Conflict Information

- Two actions interfere (cannot be executed in parallel):
  - one action deletes or asserts the precondition of the other action
  - they have opposite effects on one atomic fact
- They are marked as conflicting
  - and this information is propagated to prune the search early on

Mutex Pairs: Mutually exclusive action or fact pairs

- No pair of facts is *mutex* at fact level 0
- A pair of facts is *mutex* at fact level $i > 0$ if all ways of making them true involve actions that are *mutex* at the action level i-1
- A pair of actions is *mutex* at action level i if
  - they interfere or
  - one precondition of one action is *mutex* to a precondition of the other action at fact level i-1

➢ *Mutex* pairs cannot be true/executed at the same time

➢ Note that we do not find all pairs that cannot be true/executed at the same time, but only the easy to spot pairs with the procedure sketched above

Planning Graphs: General Method

- Expand planning graph until all goal atoms are in fact level and they are not mutex
- If not possible, terminate with failure
- Iterate:
  - Try to extract plan and terminate with plan if successful
  - Expand by another action and fact level
- Termination for unsolvable planning problems can be guaranteed – but is complex

Properties of the *Planning Graph* Approach

- Finds an optimal solution (for parallel plans)
- Terminates on unsolvable planning instances
- Is ***much*** faster than POP planning
- Has problems with symmetries:
  - Example: Transport *n* objects from room A to room B using one gripper
  - If shortest plan has *k* steps, it proves that there is no *k-1* step plans (iterating over all permutations of k-1 objects!)

Planning as Satisfiability

- Based on planning graphs of depth *k*, one can generate a set of propositional CNF formulae
  - such that each model of these formulae correspond to a *k*-step plan
  - very similar to modeling a non-det. TM using CNFs in the proof of NP-hardness of propositional satisfiability!
  - basically, one performs a different kind of search in the planning graph (middle out instead of regression search)
  - can be considerable faster, sometimes ...

Heuristic Search Planning

- Forward state-space search is often considered as too inefficient because of the high branching factor
- Why not use a heuristic estimator to guide the search?
- Could that be *automatically derived* from the representation of the planning instance?

➢ Yes, since the actions are not “black boxes” as in search!

Ignoring Negative Effects

- Ignore all negative effects (assuming again we have only positive preconditions)
  - *monotone planning*
- Example for the buyer’s domain:
  - Only *Go* and *Drop* have negative effects (perhaps also *Buy*)
  - Minimal length plan: *<Go(HWS), Buy(Drill), Go(SM), Buy(Bananas), Buy(Milk), Go(Home)>*
  - Ignoring negative effects: *<Go(HWS), Buy(Drill), Go(SM), Buy(Bananas), Buy(Milk) >*
- Usually plans with simplified ops. are shorter

Monotone Planning

- Monotone planning is easy, i.e., can be solved in polynomial time:
  - While we have not made all goal atoms true:
    - Pick any action that
      - is applicable and
      - has not been applied yet
    - and apply it
    - If there is no such action, return failure
    - otherwise continue
- Planning time and plan length bounded by number of actions times number of facts

Monotone *Optimal* Planning

- Finding the *shortest plan* is what we need to get an admissible heuristic, though!
- This is NP-hard, even if there are no preconditions!

➢ Reason: *Minimum Set Cover,* which is NP-complete, can be reduced to this problem

Minimum Set Cover

- **Given:** A set *S*, a collection of subsets *C* = $\{C_1, \ldots, C_n\}$, $C_i \subseteq S$, and a natural number *k*.
- **Question:** Does there exist a subset of *C* of size *k* covering *S*?

➢ Problem is NP-complete

➢ and obviously a special case of the monotone planning optimization problem

Simplifying it Further …

- Since the monotone planning heuristic is computationally too expensive, simplify it further:
  - compute heuristic distance for each atom (recursively) by assuming independence of sub-goals
  - solve the problem with any planner (i.e. the planning graph approach) and use this as an approximative solution

❖ both approaches may over-estimate, i.e., it is not an admissible heuristic any longer

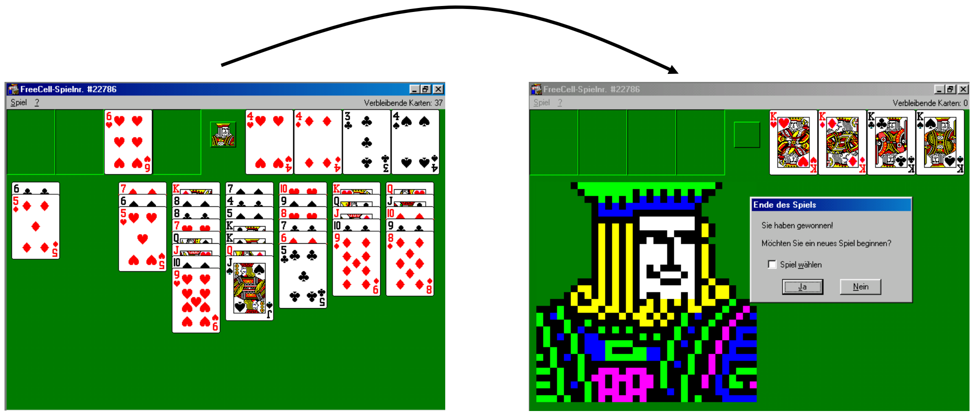
The Fast-Forward (FF) System

- Heuristic: Solve the monotone planning problem resulting from the relaxation using a planning graph approach
- Search: Hill-climbing extended by breadth-first search on plateaus
- Pruning: Only those successors are considered that are part of a relaxed solution
- Fall-back strategy: complete best-first search

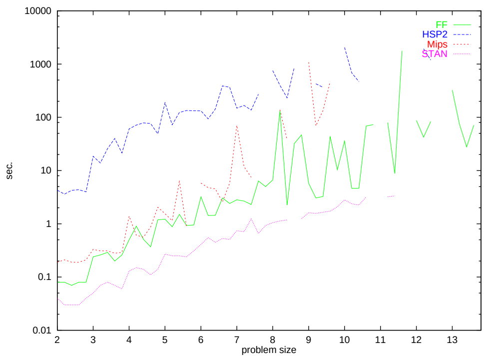
Relative Performance of FF

- FF performs very well on the planning benchmarks that are used for planning competitions (IPC = *International Planning Competition*)
- Examples:
  - Blocks world
  - Logistics
  - Freecell
- Meanwhile refined and also new planners such as FDD

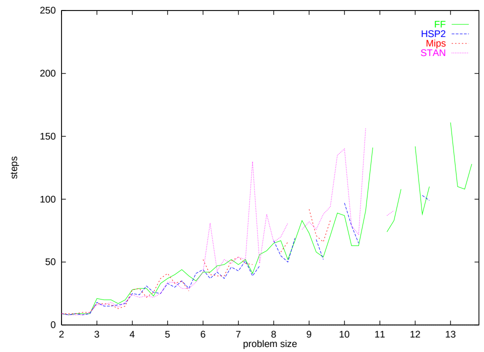
Example: Freecell



Freecell: Performance



CPU time



Solution size

One Possible Explanation …

- Search space topology
- Look for search space properties such as
  - local minima
  - size of plateaus
  - dead ends (detected & undetected)
- Estimate by
  - exploring small instances
  - sampling large instance
- Try to prove conjectures found this way

➢ Goes some way in understanding problem structure

Outlook

- More expressive action languages
- More expressive domains: numerical values / time
- Non-classical planning: Dropping the single-state assumption
- Multi-agent planning

Extensions: More Powerful Action Language

- Conditional actions
  - Often the effects are dependent on the context the action is executed in
  - Example: *press accelerator pedal*
    - If in “forward gear”: car goes forward
    - If in “neutral gear”: car does nothing
    - If in “reverse gear”: car goes backward
- More powerful conditions:
  - General propositional connectors
  - First-order formulas (over finite domains)

Extensions: Domain Modelling

- Considered so far: fluents that can be true or false
- Often needed: numerical values
  - Resource consumption
  - Profit
  - Cost-optimal planning
  - ➢ Leads easily to undecidability
- Special case of resource: time
  - Parallel execution of actions with duration
  - Needs refined semantics (when do effects occur etc.)

Non-classical Planning

- Classical planning assumes:
  - Complete knowledge about the initial state
  - Deterministic effects
  - No exogenous actions
  - ➢ Single state after each action execution
- Non-classical planning:
  - Drop single-state assumption
  - Sensing actions
  - ➢ Conditional planning
  - Perhaps limited observability (none, partial, full)
  - No observability: Conformant planning (as in the vacuum cleaner example)
  - ➢ Computational complexity of non-classical planning is much higher (because it is a multi-state problem)

Planning and Execution

- Realistic environments (aka "the real world")
  - dynamically changing due to other agents
  - only partially observable
  - → many possible world states
- Conditional planning:
  - Very costly
  - Plan for every possible world state in advance
  - Most of the conditional plan becomes obsolete as soon as a perception is made
  - Often no (good) model of contingencies
- Alternative:
  - Planning, execution, monitoring, replanning, ...

Monitoring and Replanning

- Things that may happen during execution
  - Everything works like a charm!
  - Failures
  - Unexpected observations
  - Unexpected events (other agents or nature)
- Monitoring
  - Action monitoring: check if
    - preconditions are satisfied
    - intended effects occured
  - Plan monitoring: check if
    - whole plan is still executable in current state and
    - will reach goal state
  - Serendipity
- Replanning: several variants
  - Start planning again from scratch → find optimal plan (again)
  - Determine where plan will fail and replan only from there → maxime plan stability
  - Plan repair by local search → maximize some other similarity metric

Continual Planning

- Continual Planning:
  - Suspend planning
    - for partial plan execution
    - for sensing → for resolving contingencies
  - Then plan again in light of new knowledge.
- How do agents decide when to switch between planning and execution?
  - Model sensing actions
  - Reason about how they can reduce uncertainty
  - → Active knowledge gathering

Multi-Agent Planning

- Planning for multiple agents
  - Concurrent execution
  - Execution synchronisation
- Planning by multiple agents
  - Distributed planning
- Various degrees of cooperativity → game theory
- Distributed Continual Planning
  - Agents continually interleave planning, acting, sensing and interacting
  - Agents negotiate common goals and plans over time

Summary

- Planning differs from problem-solving in that the **representation is more flexible**.
- We can search in the **plan space** instead of the state space
- The POP algorithm realizes non-linear planning and is **complete** and **correct**, but it is difficult to design good heuristics
- Recent approaches to planning have **boosted** the efficiency of planning methods significantly
- **Heuristic search planning** appears to be one of the fastest (non-optimal) methods
- **Non-classical planning** makes more realistic assumptions, but the planning problem becomes much more complex
- **Continual planning** can be used to address the expressivity/efficiency tradeoff
- **Multi-agent planning** is important if groups of cooperating or competing agents strive to achieve goals