Foundations of AI

15. Planning

The art and practice of thinking before acting

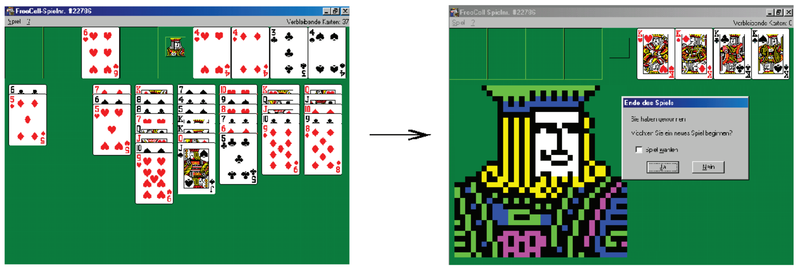
Wolfram Burgard, Andreas Karwath, Bernhard Nebel, and Martin Riedmiller

What is planning?

- Planning is the process of generating (possibly partial) representations of future behavior prior to the use of such plans to constrain or control that behavior:
- *Planning is the art and practice of thinking before acting* [Haslum]
- The outcome is usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents.

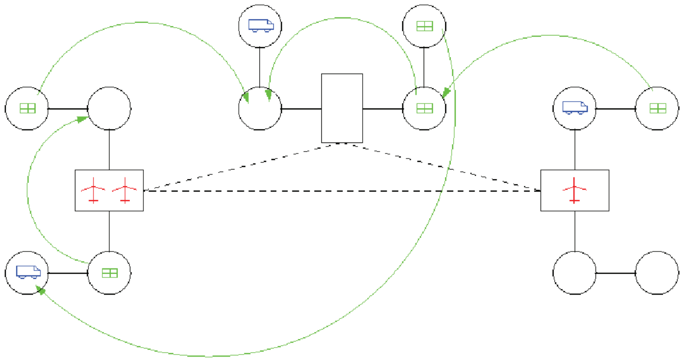
Planning tasks

Given a **current state**, a set of possible **actions**, a specification of the **goal conditions**, which **plan** transforms the *current state* into a *goal state*?



Another planning task: *Logistics*

Given a road map, and a number of trucks and airplanes, make a plan to transport objects from their start to their goal destinations.



Domain-independent action planning

- Start with a **declarative specification** of the planning problem
- Use a **domain-independent planning** system to solve the planning problem

⇝ Domain-independent planners are *generic problem solvers*

- Issues:
  - Good for evolving systems and those where performance is not critical
  - Running time should be comparable to specialized solvers
  - Solution quality should be acceptable
  - . . . at least for all the problems we care about

Planning problem classes

Effects: deterministic, non-deterministic, probabilistic

Observability of the environment: complete, partial, not observable

Horizon: finite, infinite

Objective: reach goal, maintain property, maximize probability of reaching a state, maximize expected reward

Classical Planning: deterministic actions, complete observability (in the beginning), finite horizon, reach goal

Conditional Planning: non-deterministic actions, complete observability, finite horizon, reach goal

Markov Decision Processes (MDP): probabilistic actions, complete obs., maximize expected reward

. . .

Action planning is not . . .

- *Problem solving by search*, where we describe a problem by a state space and then implement a program to search through this space
  - in action planning, we specify the problem declaratively (using logic) and then solve it by a general planning algorithm
- *Program synthesis*, where we generate programs from specifications or examples
  - in action planning we want to solve just one instance and we have only very simple action composition (i.e., sequencing, perhaps conditional and iteration)
- *Scheduling*, where all jobs are known in advance and we only have to fix time intervals and machines
  - instead we have to find the right actions and to sequence them

⇝ Of course, there is interaction with these areas!

The basic STRIPS formalism

STRIPS: **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

- $\mathcal{S}$ is a *first-order signature* and $\Sigma_{\mathcal{S}}$ denotes the set of *ground atoms* over the signature (also called **facts** or **fluents**).
- $\Sigma_{\mathcal{S},\mathbf{V}}$ is the set of atoms over $\mathcal{S}$ using variable symbols from the set of variables **V**.
- A **first-order STRIPS state** S is a subset of $\Sigma_{\mathcal{S}}$ denoting a *complete theory* or *model* (using CWA).
- A **planning task** (or **planning instance**) is a 4-tuple $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$, where
  - **O** is a set of **operator** (or *action types*)
  - $\mathbf{I} \subseteq \Sigma_{\mathcal{S}}$ is the **initial state**
  - $\mathbf{G} \subseteq \Sigma_{\mathcal{S}}$ is the **goal specification**
- **No domain constraints** (although present in original formalism)

Operators, actions & state change

- **Operator:**

$$o = \langle para, pre, eff \rangle,$$

  with $para \subseteq \mathbf{V}$, $pre \subseteq \Sigma_{\mathcal{S},\mathbf{V}}$, $eff \subseteq \Sigma_{\mathcal{S},\mathbf{V}} \cup \neg\Sigma_{\mathcal{S},\mathbf{V}}$ (element-wise negation) and all variables in *pre* and *eff* are listed in *para*.
  Also: $pre(o)$, $eff(o)$.
  eff^{+} = positive effect literals
  eff^{-} = negative effect literals
- **Operator instance** or **action**: Operator with empty parameter list (*instantiated schema!*)
- **State change** induced by action:

$$App(S, o) = \begin{cases} S \cup eff^{+}(o) - \neg eff^{-}(o) & \text{if } pre(o) \subseteq S \text{ \& } eff(o) \text{ is cons.} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Example formalization: *Logistics*

- Logical atoms: $at(O, L)$, $in(O, V)$, $airconn(L1, L2)$, $street(L1, L2)$, $plane(V)$, $truck(V)$
- Load into truck: *load*

Parameter list:	(O, V, L)
Precondition:	$at(O, L), at(V, L), truck(V)$
Effects:	$\neg at(O, L), in(O, V)$

- Drive operation: *drive*

Parameter list:	$(V, L1, L2)$
Precondition:	$at(V, L1), truck(V), street(L1, L2)$
Effects:	$\neg at(V, L1), at(V, L2)$

- ...
- Some constant symbols: $t1, s, c, p1$ with $truck(t1)$ and $street(s, c)$
- Action: $drive(t1, s, c)$

Plans & successful executions

- A **plan** Δ is a sequence of actions
- State resulting from **executing a plan**:

$$Res(S, \langle\rangle) = S$$

$$Res(S, (o; \Delta)) = \begin{cases} Res(App(S, o), \Delta) & \text{if } App(S, o) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Plan Δ is successful** or **solves** a planning task if $Res(\mathbf{I}, \Delta)$ is defined and $\mathbf{G} \subseteq Res(\mathbf{I}, \Delta)$.

A small *Logistics* example

Initial state: $S = \left\{ \begin{array}{l} at(p1, c), at(p2, s), at(t1, c), \\ at(t2, c), street(c, s), street(s, c) \end{array} \right\}$

Goal: $\mathbf{G} = \{ at(p1, s), at(p2, c) \}$

Successful plan: $\Delta = \langle load(p1, t1, c), drive(t1, c, s), unload(p1, t1, s), load(p2, t1, s), drive(t1, s, c), unload(p2, t1, c) \rangle$

Other successful plans are, of course, possible

Beyond STRIPS

Even when keeping all the restrictions of classical planning, one can think of a number of *extensions* of the planning language.

- **General logical formulas as preconditions**: Allow all Boolean connectors and quantification
- **Conditional effects**: Effects that happen only if some additional conditions are true. For example, when *pressing the accelerator pedal*, the effects depends on which gear has been selected (no, reverse, forward).
- **Multi-valued state variables**: Instead of 2-valued Boolean variables, multi-valued variables could be used
- ...

Simplifications: DATALOG- and propositional STRIPS

- STRIPS as described above allows for unrestricted first-order terms, i.e., arbitrarily nested **function terms**

$\rightsquigarrow$ **Infinite state space**

- Simplification: No function terms (only 0-ary = constants)

$\rightsquigarrow$ **DATALOG-STRIPS**

- Simplification: No variables in operators (= actions)

$\rightsquigarrow$ **Propositional STRIPS**

$\rightarrow$ used in planning algorithms nowadays (but specification is done using DATALOG-STRIPS)

PDDL: The planning domain description language

- Since 1998, there exists a bi-annual *scientific competition* for action planning systems.
- In order to have a common language for this competition, **PDDL** has been created (originally by Drew McDermott)
- Meanwhile, version 3.1 (IPC-2008) with most of the features mentioned.
- Sort of standard language by now.
- We will stick to STRIPS here.

Current Approaches to Planning

- In 1992, Kautz and Selman introduced **planning as satisfiability**
- ⇝ Encode possible k-step plans as Boolean formulas and use an iterative deepening search
- In 1995, Blum and Furst introduced **planning graphs**
- ⇝ iterative deepening approach that prunes the search space using a graph-structure
- In 1996, McDermott proposed to use (again) an **heuristic estimator** to control the selection of actions, similar to GPS
- Geffner (1997) followed up with a propositional, simplified version (**HSP**) and Hoffmann & Nebel (2001) with an extended version integrating strong pruning. (**FF**)
- Even better system is **FD** by Helmert
- ⇝ Heuristic planners seem to be the most efficient sub-optimal planners these days

Iterative Deepening Search

1. Initialize $k = 0$
2. Try to construct a plan of length k exhaustively
3. If unsuccessful, increment k and goto step 2.
4. Otherwise return plan

- ⇝ Finds shortest plan
- ⇝ Needs to prove that there are no plans of length $1, 2, \ldots k - 1$ before a plan of length k is produced.

Planning as Satisfiability

- Take the dual perspective: Consider all models satisfying a *particular formula* as plans
- → Similar to what is done in the generic reduction that shows NP-hardness of *SAT* (simulation of a computation on a Turing machine)
- Build formula for k steps, check **satisfiability**, and increase k until a satisfying assignment is found
- Use time-indexed propositional atoms for facts and action occurrences
- Formulate constraints that describe what it means that a plan is successfully executed:
  - Only one action per step
  - If an action is executed then their preconditions were true and the effects become true after the execution
  - If a fact is not affected by an action, it does not change its value (frame axiom)

Planning as Satisfiability: Example

- Fact atoms:
  $at(p1, s)_i, at(p1, c)_i, at(t1, s)_i, at(t1, c)_i, in(p1, t1)_i$
- Action atoms:
  $move(t1, s, c)_i, move(t1, c, s)_i, load(p1, s)_i, \ldots$
- Initial state: $at(p1, c)_1, at(p2, s)_1, at(t1, c)_1$
- Only one action per step:
  $\bigwedge_{i,x,y} \neg(unload(t1, p1, x)_i \wedge load(p1, t1, y)_i) \wedge \ldots$
- Preconditions: $\bigwedge_{i,x}(unload(p1, t1, x)_i \rightarrow in(p1, t1)_{i-1}) \wedge \ldots$
- Effects:
  $\bigwedge_{i,x}(unload(p1, t1, x)_i \rightarrow \neg in(p1, t1)_i \wedge at(p1, x)_i) \wedge \ldots$
- Frame axioms:
  $\bigwedge_{i,x,y,z}(\neg move(t1, x, y)_i \rightarrow (at(t1, z)_{i-1} \leftrightarrow at(t1, z)_i)) \wedge \ldots$
- ⇝ A satisfying truth assignment corresponds to a **plan** (use the true action atoms)

Advantages of the Approach

- Flexible search strategy
- Can make use of **SAT solver** technology
- . . . and automatically profits from advances in this area
- Can express constraints on intermediate states
- Can use logical axioms to express additional constraints, e.g., to prune the search space

Planning Based on Planning Graphs

Main ideas:

- Describe *possible* developments in a graph structure (use only positive effects)
  - Layered graph structure with fact and action levels
  - **Fact level (F level):** positive atoms (the first level being the initial state)
  - **Action level (A level):** actions that can be applied using the atoms in the previous fact level
  - *Links:* precondition and effect links between the two layers
- Record **conflicts** caused by negative effects and propagate them
- **Extract a plan** by choosing only non-conflicting parts of the graph (allowing for **parallel** actions)

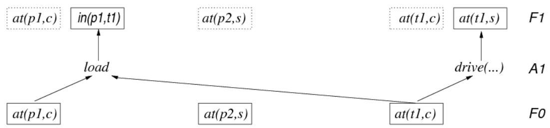
⇝ Parallelism (for non-conflicting actions) is a great boost for the efficiency.

Example Graph

- $\underline{\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}}$, $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$

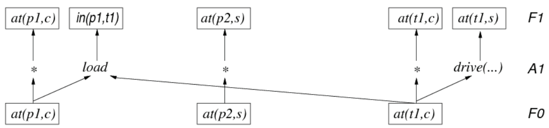
Example Graph

- $\underline{\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}}$, $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$
- All applicable actions are included



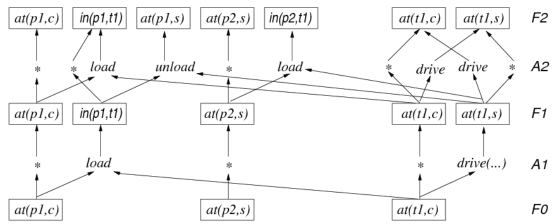
Example Graph

- $\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}$, $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$
- All applicable actions are included
- In order to propagate unchanged properties, use *noop* action, denoted by *



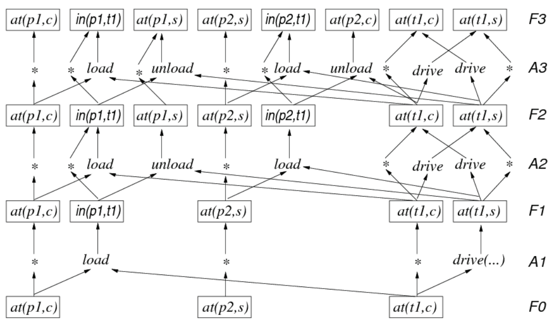
Example Graph

- $\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}$, $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$
- All applicable actions are included
- In order to propagate unchanged properties, use *noop* action, denoted by *
- Expand graph



Example Graph

- $\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}$, $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$
- All applicable actions are included
- In order to propagate unchanged properties, use *noop* action, denoted by *
- Expand graph as long as not all goal atoms are in the fact level

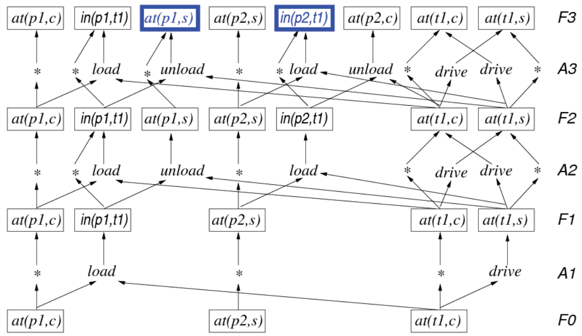


Plan Extraction

- Start at last fact level with goal atoms
- Select a minimal set of **non-conflicting actions** that generate the goal atoms
  - Two actions are **conflicting** if they have complementary effects or if one action deletes or asserts a precondition of the other action
- Use the preconditions of the selected actions as **(sub-)goals** on the next lower fact level
- **Backtrack** if no non-conflicting choice is possible
- If all possibilities are exhausted, the graph has to be **extended** by another level.

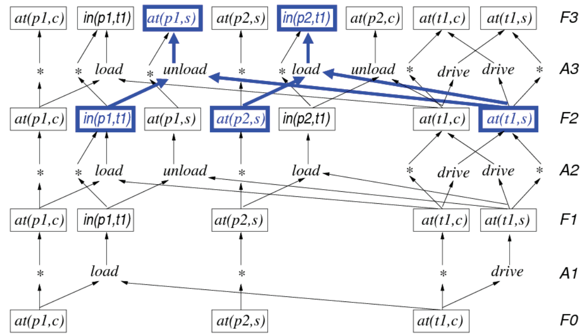
Extracting From the Example Graph

Start with *goals* at highest fact level



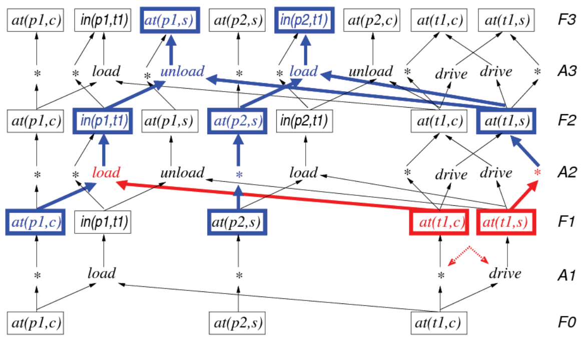
Extracting From the Example Graph

Select minimal set of actions & corresponding *subgoals*



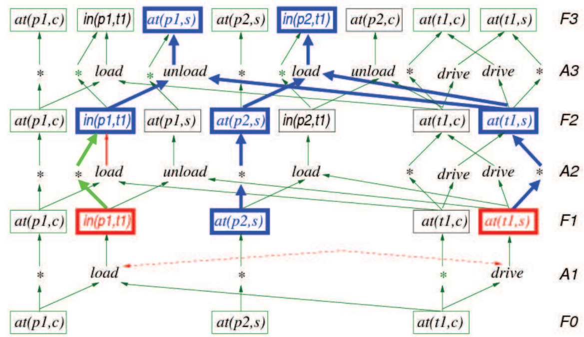
Extracting From the Example Graph

Wrong choice leading to conflicting actions



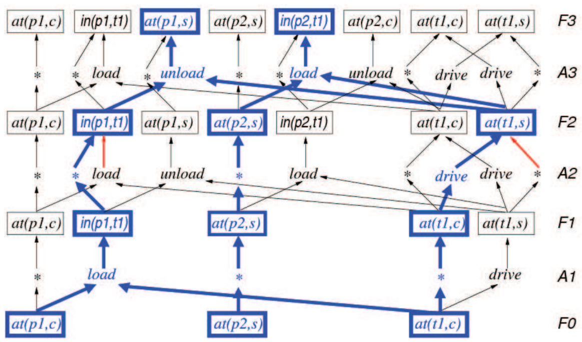
Extracting From the Example Graph

Other choice, but no further selection possible



Extracting From the Example Graph

Final selection



Disadvantages of Iterative Deepening Planners

- If a domain contains many symmetries, proving that there is no plan up to length of $k-1$ can be very costly.
- Example: *Gripper* domain:
  - there is one robot with two grippers
  - there is room A that contains n balls
  - there is another room B connected to room A
  - the goal is to bring all balls to room B
- Obviously, the plan must have a length of at least $n/2$, but ID planners will try out all permutations of actions for shorter plans before noting this.

⇝ Give better *guidance*

Propagation of Conflict Information: Mutex pairs

Idea: Try to identify as many pairs of conflicting choices as possible in order to **prune** the search space

- Any pair of conflicting actions is *mutex* (mutually exclusive)
- A pair of atoms is *mutex* at F-level $i > 0$ if all ways of making them true involve actions that are *mutex* at the A-level i
- A pair of actions is also *mutex* if their preconditions are
- ...

→ Actions that are *mutex* cannot be executed at the same time

→ Facts that are *mutex* cannot be both made true at the same time

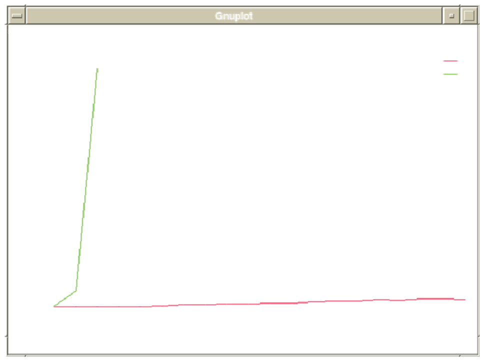
⇝ Never choose *mutex pairs* during *plan extraction*

Plan graph search and **mutex propagation** make planning 1–2 orders of magnitude more **efficient** than conventional methods

Heuristic Search Planning

- Use an **heuristic estimator** in order to select the next action or state
- Depending on the **search scheme** and the heuristic, the plan might not be the shortest one

→ It is often easier to go for *sub-optimal* solutions (remember *Logistics*)



Heuristic search planner vs. iterative deepening on *Gripper*

Deriving Heuristics: Relaxations

- General principle for deriving heuristics:
  - Define a **simplification** (relaxation) of the problem and take the difficulty of a solution for the simplified problem as an *heuristic estimator*
- Example: *straight-line distance* on a map to estimate the travel distance
- Example: *decomposition* of a problem, where the components are solved ignoring the interactions between the components, which may incur additional costs
- In planning, one possibility is to ignore *negative effects*

Ignoring Negative Effects: Example

- In *Logistics*: The negative effects in *load* and *drive* are ignored:
- *Simplified* load operation: $load(O, V, P)$
  Precondition: $at(O, P), at(V, P), truck(V)$
  Effects: ~~$\neg at(O, P)$~~, $in(O, V)$

⇝ After loading, the package is still at the place and also inside the truck

- *Simplified* drive operation: $drive(V, P1, P2)$
  Precondition: $at(V, P1), truck(V), street(P1, P2)$
  Effects: ~~$\neg at(V, P1)$~~, $at(V, P2)$
- After driving, the truck is in two places!

→ We want the length of the shortest *relaxed* plan ⇝ $h^+(s)$

⇝ How difficult is monotonic planning?

Monotonic Planning

Assume that all effects are positive

- finding **some plan** is easy:
  - Iteratively, execute all actions that are executable and have not all their effects made true yet
  - If no action can be executed anymore, check whether the goal is satisfied
  - If not, there is no plan
  - Otherwise, we have a plan containing each action only once
- Finding the **shortest plan**: easy or difficult?

→ NP-hard

⇝ Consider approximations to h^+.

The FF Heuristic

- Use the *planning graph method* to construct a plan for the monotone planning problem
- Can be done in poly. time (and is *empirically very fast*)
- Generates an *optimal parallel plan* that might not be the best sequential plan

→ The number of actions in this plan is used as the heuristic estimate (more *informative* than the parallel plan length, but not *admissible*)

⇝ Appears to be a good approximation

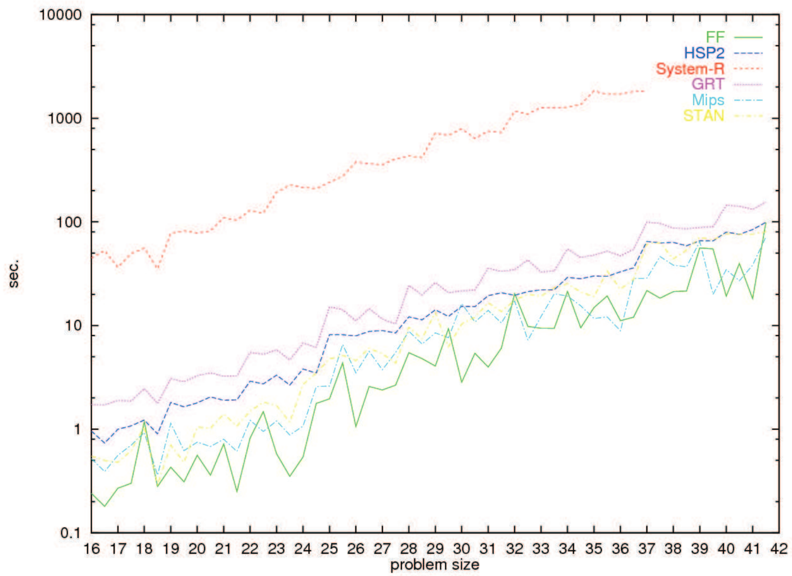
The FF System

- **FF** (*Fast Forward*) is a **heuristic search planner** developed in Freiburg
- **Heuristic**: Goal distances are estimated by *solving a relaxation* of the task in every search state (ignoring negative effects) – the solution is not minimal, however!
- **Search strategy**: *Enforced hill-climbing*
- **Pruning**: Only a fraction of each states successors are considered: only those*successors* that would be *generated by the relaxed solution* – with a fall-back strategy considering all successors if we are unsuccessful

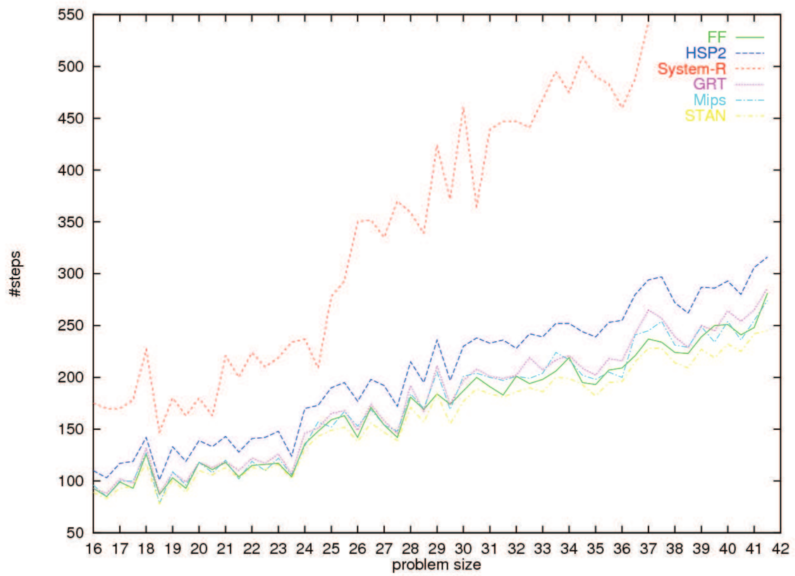
⇝ FF used to be one of the fastest planners around

→ Meanwhile, there is **FD**, which contains more domain analysis and which is faster because of this

Runtime: *Logistics* in the 2000 competition



Solution Quality: *Logistics* in the 2000 competition



Summary and Outlook

- Planning generates representation of future behavior
- Classical planning assumes full observability and deterministic actions
- Compared with *MDPs*, one can deal with much larger state spaces
- Current algorithmic approaches are
  - planning as satisfiability
  - planning graphs
  - heuristic search planning, which seems to be the most promosing approach for satisficing planning
- Many possible extensions . . .
- Applications in robotic, video games, . . .

⇝ Come to the Foundations of AI group, if you are interested in pursuing research in this area