

# Foundations of Artificial Intelligence

## 11. Action Planning

Solving Logically Specified Problems using a General Problem Solver

Joschka Boedecker and Wolfram Burgard and Bernhard Nebel



Albert-Ludwigs-Universität Freiburg

June 28, 2017

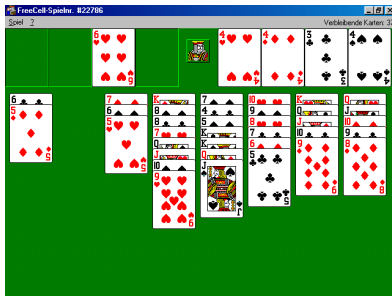
# Contents

- 1 What is Action Planning?
- 2 Planning Formalisms
- 3 Basic Planning Algorithms
- 4 Computational Complexity
- 5 Current Algorithmic Approaches
- 6 Summary

- 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.
- The outcome is usually a **set of actions**, with temporal and other constraints on them, for **execution** by some agent or agents.
- As a **core aspect** of **human intelligence**, planning has been studied since the earliest days of AI and cognitive science. Planning research has led to many useful tools for real-world applications, and has yielded significant insights into the organization of behavior and the nature of reasoning about actions. [Tate 1999]

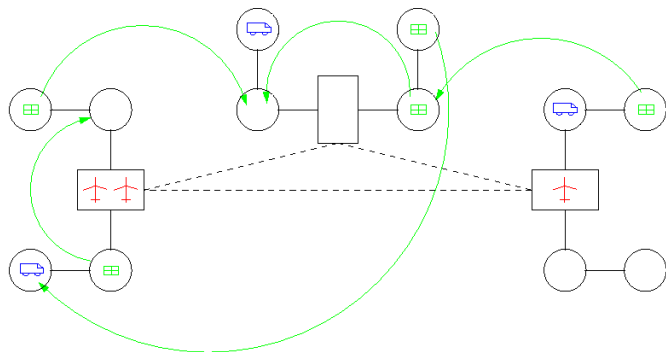
# Planning Tasks

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.



# 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!

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

## Initial state:

$$At(truck1, loc1, s_0) \wedge At(package1, loc3, s_0)$$

## Operators (successor-state axioms):

$$\forall a, s, l, p, t \text{ } At(t, p, Do(a, s)) \Leftrightarrow \{a = Drive(t, l, p) \wedge Poss(Drive(t, l, p), s) \\ \vee At(t, p, s) \wedge (a \neq \neg Drive(t, p, l, s) \vee \neg Poss(Drive(t, p, l, s)))\}$$

## Goal conditions (query):

$$\exists s \text{ } At(package1, loc2, s)$$

The **constructive** proof of the existential query (computed by an automatic theorem prover) delivers a plan that does what is desired. Can be quite **inefficient**!



# The Basic STRIPS Formalism

## STRIPS: STanford Research Institute Problem Solver

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

- **Operator:**

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

with  $para \subseteq \mathbf{V}$ ,  $pre \subseteq \Sigma_{S, \mathbf{V}}$ ,  $eff \subseteq \Sigma_{S, \mathbf{V}} \cup \neg \Sigma_{S, \mathbf{V}}$  (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:  $v1, s, t$  with  $truck(v1)$  and  $street(s, t)$
- Action:  $drive(v1, s, t)$

- A **plan**  $\Delta$  is a sequence of actions
- State resulting from **executing a plan**:

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

- **Plan**  $\Delta$  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:**  $G = \{ at(p1, s), at(p2, c) \}$

**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

# Simplifications: DATALOG- and Propositional STRIPS

- STRIPS as described above allows for unrestricted **first-order terms**, i.e., arbitrarily nested **function terms**
- **Infinite state space**
- Simplification: No function terms (only 0-ary = constants)
- **DATALOG-STRIPS**
- Simplification: No variables in operators (= actions)
- **Propositional STRIPS**
- Propositional STRIPS used in planning algorithms nowadays (but specification is done using DATALOG-STRIPS)

# 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
- **Numerical resources**: Resources (such as fuel or time) can be effected and be used in preconditions
- **Durative actions**: Actions can have duration and can be executed concurrently
- **Axioms/Constraints**: The domain is not only described by operators, but also by additional laws

# 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-2011) with most of the features mentioned – and many sub-dialects and extensions.
- Sort of “standard” by now.



# PDDL Logistics Example

(define (domain logistics)

  (:types truck airplane - vehicle  
    package vehicle - physobj  
    airport location - place  
    city place physobj - object)

  (:predicates (in-city ?loc - place ?city - city)  
    (at ?obj - physobj ?loc - place)  
    (in ?pkg - package ?veh - vehicle))

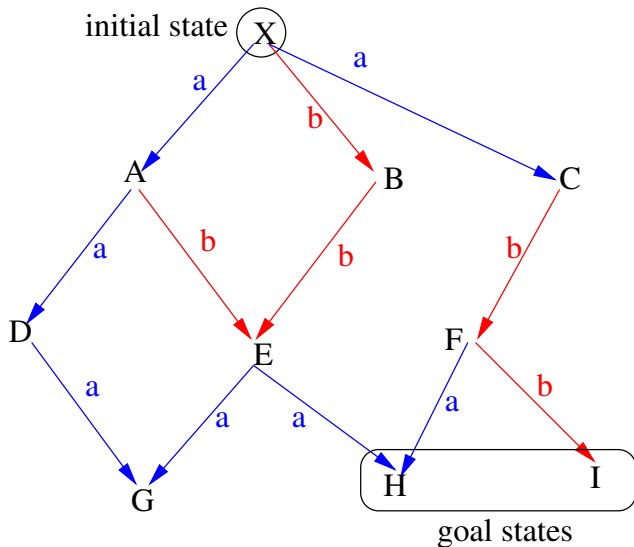
  (:action LOAD-TRUCK

    :parameters  (?pkg - package ?truck - truck ?loc - place)  
    :precondition  (and (at ?truck ?loc) (at ?pkg ?loc))  
    :effect       (and (not (at ?pkg ?loc)) (in ?pkg ?truck)))  
    ...)

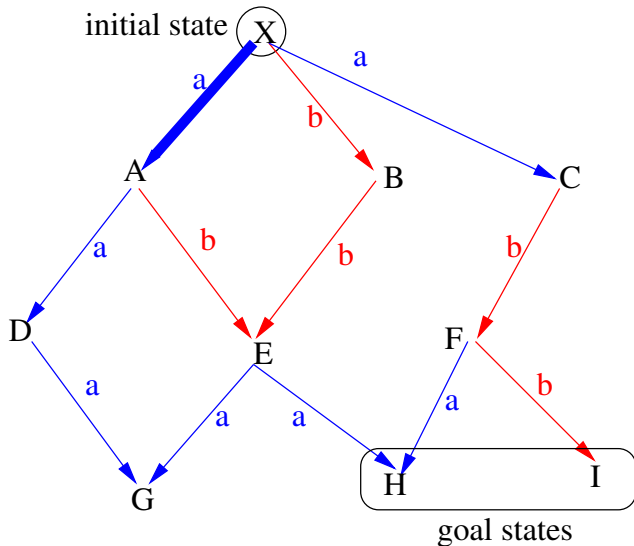
# Planning Problems as Transition Systems

- We can view planning problems as searching for goal nodes in a large **labeled graph** (**transition system**)
  - **Nodes** are defined by the value assignment to the fluents = **states**
  - **Labeled edges** are defined by actions that change the appropriate fluents
  - Use graph search techniques to find a (shortest) path in this graph!
  - **Note:** The graph can become **huge**: 50 Boolean variables lead to  $2^{50} = 10^{15}$  **states**
- **Create the transition system on the fly and visit only the parts that are *necessary***

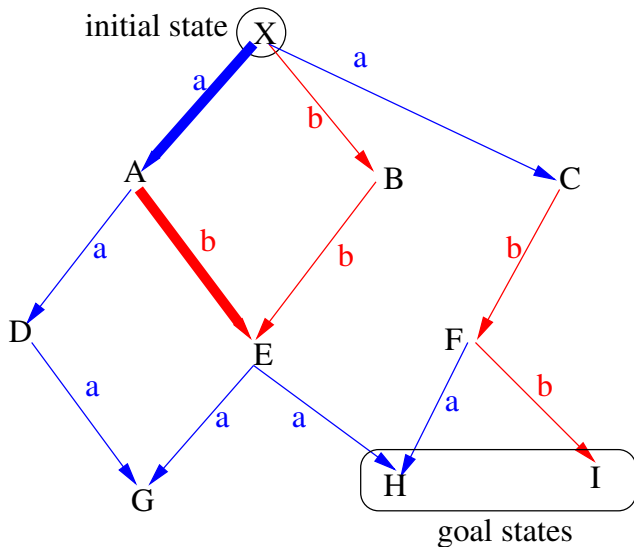
# Transition System: Searching Through the State Space



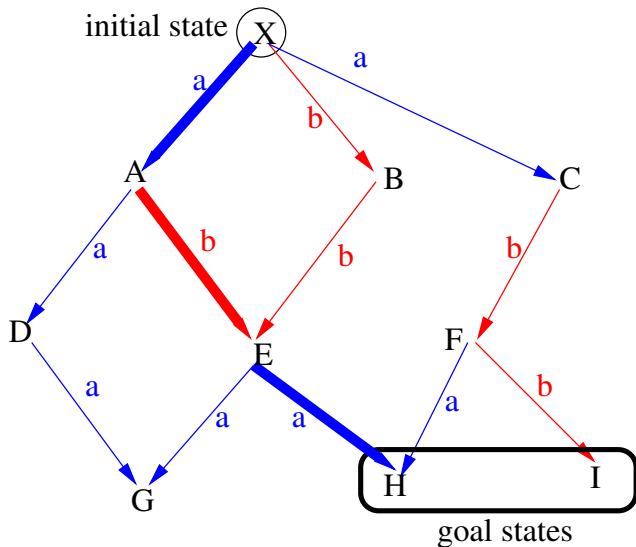
# Transition System: Searching Through the State Space



# Transition System: Searching Through the State Space



# Transition System: Searching Through the State Space



# Progression Planning: Forward Search

Search through transition system starting at **initial state**

- 1 Initialize partial plan  $\Delta := \langle \rangle$  and **start** at the unique **initial state**  $I$  and make it the current state  $S$
- 2 **Test** whether we have reached a **goal state** already:  $G \subseteq S$ ? If so, return plan  $\Delta$ .
- 3 **Select one applicable action**  $o_i$  **non-deterministically** and
  - compute successor state  $S := App(S, o_i)$ ,
  - extend plan  $\Delta := \langle \Delta, o_i \rangle$ , and continue with step 2.

Instead of non-deterministic choice use some **search strategy**.

Progression planning can be **easily extended** to more expressive planning languages

# Progression Planning: Example

$$\mathcal{S} = \{a, b, c, d\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \}$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$

{a,b}





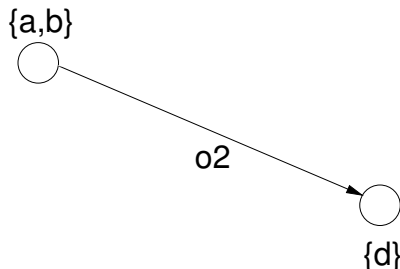
# Progression Planning: Example

$$\mathcal{S} = \{a, b, c, d\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle,$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$



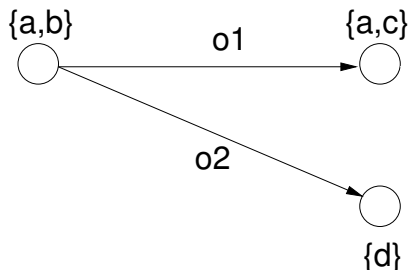
# Progression Planning: Example

$$\mathcal{S} = \{a, b, c, d\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{a, b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{a, b\}, \{\neg a, \neg b, d\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle,$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$



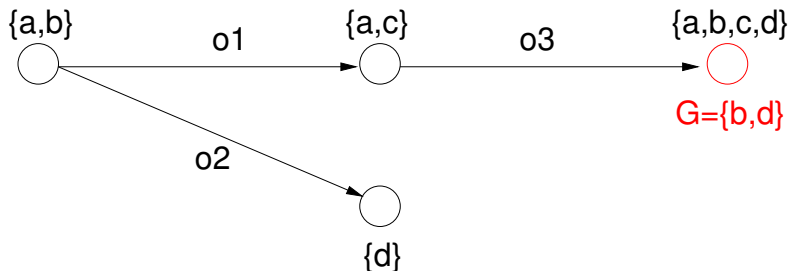
# Progression Planning: Example

$$\mathcal{S} = \{a, b, c, d\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{a, b\}, \{-b, c\} \rangle, \\ o_2 = \langle \emptyset, \{a, b\}, \{-a, \neg b, d\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d\} \rangle, \}$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$



# Regression Planning: Backward Search

Search through transition system starting at **goal states**. Consider **sets of states**, which are **described** by the atoms that are **necessarily true** in them

- 1 Initialize partial plan  $\Delta := \langle \rangle$  and set  $\mathbf{S} := \mathbf{G}$
- 2 **Test** whether we have reached the unique **initial state** already:  $\mathbf{I} \supseteq \mathbf{S}$ ? If so, return plan  $\Delta$ .
- 3 **Select one action**  $o_i$  **non-deterministically** which does not make (sub-)goals false ( $\mathbf{S} \cap \neg \text{eff}^-(o_i) = \emptyset$ ) and
  - compute the **regression** of the description  $\mathbf{S}$  through  $o_i$ :

$$\mathbf{S} := \mathbf{S} - \text{eff}^+(o_i) \cup \text{pre}(o_i)$$

- extend plan  $\Delta := \langle o_i, \Delta \rangle$ , and continue with step 2.

Instead of non-deterministic choice use some **search strategy**

Regression becomes much more complicated, if e.g. **conditional effects** are allowed. Then the result of a regression can be a general Boolean formula

# Regression Planning: Example

$$\mathcal{S} = \{a, b, c, d, e\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \}$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$

$\{b, d\}$   
○

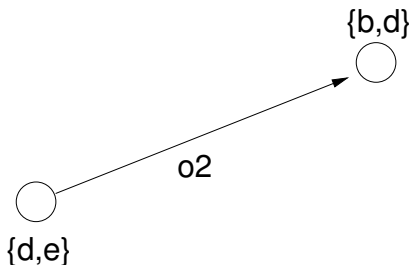
# Regression Planning: Example

$$\mathcal{S} = \{a, b, c, d, e\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle,$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$



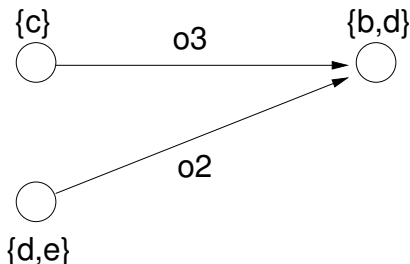
# Regression Planning: Example

$$\mathcal{S} = \{a, b, c, d, e\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \}$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$



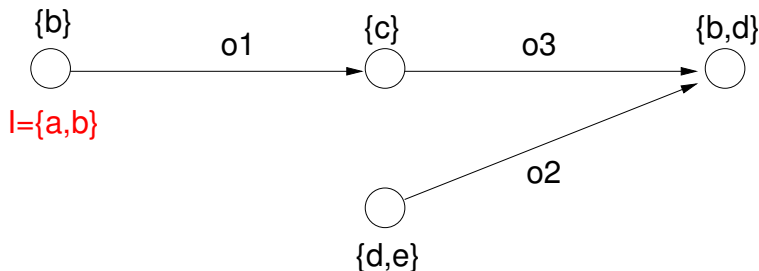
# Regression Planning: Example

$$\mathcal{S} = \{a, b, c, d, e\},$$

$$\mathbf{O} = \{ o_1 = \langle \emptyset, \{b\}, \{\neg b, c\} \rangle, \\ o_2 = \langle \emptyset, \{e\}, \{b\} \rangle, \\ o_3 = \langle \emptyset, \{c\}, \{b, d, \neg e\} \rangle, \}$$

$$\mathbf{I} = \{a, b\}$$

$$\mathbf{G} = \{b, d\}$$





# Other Types of Search

- Of course, other types of search are possible.
- Change perspective: Do not consider the **transition system** as the space we have to explore, but consider the search through the space of (incomplete) plans:
  - **Progression search**: Search through the space of plan **prefixes**
  - **Regression search**: Search through **plan suffixes**
- **Partial order planning**:
  - Search through partially ordered plans by starting with the empty plan and trying to satisfy (sub-)goals by introducing new actions (or using old ones)
  - Make ordering choices only when necessary to resolve conflicts

# The Planning Problem – Formally

## Definition (Plan existence problem (PLANEX))

**Instance:**  $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$ .

**Question:** Does there exist a plan  $\Delta$  that solves  $\Pi$ , i.e.,  $Res(\mathbf{I}, \Delta) \supseteq \mathbf{G}$ ?

## Definition (Bounded plan existence problem (PLANLEN))

**Instance:**  $\Pi = \langle \mathcal{S}, \mathbf{O}, \mathbf{I}, \mathbf{G} \rangle$  and a positive integer  $n$ .

**Question:** Does there exist a plan  $\Delta$  of length  $n$  or less that solves  $\Pi$ ?

From a practical point of view, also **PLANGEN** (*generating* a plan that solves  $\Pi$ ) and **PLANLENGEN** (*generating* a plan of length  $n$  that solves  $\Pi$ ) and **PLANOPT** (generating an optimal plan) are interesting (but at least as hard as the decision problems).

# Basic STRIPS with First-Order Terms

- The state space for STRIPS with general first-order terms is **infinite**
- We can use function terms to describe (the index of) **tape cells of a Turing machine**
- We can use operators to describe the **Turing machine control**
- The existence of a plan is then equivalent to the existence of a **successful computation** on the Turing machine
- PLANEX for STRIPS with first-order terms can be used to decide the **Halting problem**

## Theorem

*PLANEX for STRIPS with first-order terms is **undecidable**.*

## Theorem

*PLANEX is **PSPACE-complete** for propositional STRIPS.*

- Membership follows because we can successively guess operators and compute the resulting states (needs only polynomial space)
- Hardness follows using again a **generic reduction** from TM acceptance. Instantiate polynomially many tape cells with no possibility to extend the tape (only poly. space, can all be generated in poly. time)
- PLANLEN is also PSPACE-complete (membership is easy, hardness follows by setting  $k = 2^{|\Sigma|}$ )

# Restrictions on Plans

- If we restrict the length of the plans to be only **polynomial** in the size of the planning task, **PLANEX** becomes **NP-complete**
- Similarly, if we use a **unary** representation of the natural number  $k$ , then **PLANLEN** becomes **NP-complete**
- Membership obvious (guess & check)
- Hardness by a straightforward reduction from SAT or by a generic reduction.
- One source of complexity in planning stems from the fact that plans can become **very long**
- We are only interested in short plans!
- We can use methods for NP-complete problems if we are only looking for “short” plans.

## Theorem

*The problem of deciding plan existence for precondition-free, propositional STRIPS is in P.*

## Proof.

Do a backward greedy plan generation. Choose all operators that make some goals true and that do not make any goals false. Remove the satisfied goals and the operators from further consideration and iterate the step. Continue until all remaining goals are satisfied by the initial state (**succeed**) or no more operators can be applied (**fail**).

# Propositional, Precondition-free STRIPS and Plan Length

## Theorem

*The problem of deciding whether there exists a plan of length  $k$  for precondition-free, propositional STRIPS is NP-complete, even if all effects are positive.*

## Proof.

Membership in NP is obvious. Hardness follows from a straightforward reduction from the MINIMUM-COVER problem [Garey & Johnson 79]:

Given a collection  $C$  of subsets of a finite set  $S$  and a positive integer  $k$ , does there exist a cover for  $S$  of size  $k$  or less, i.e., a subset  $C' \subseteq C$  such that  $\bigcup C' \supseteq S$  and  $|C'| \leq k$ ?



We will use this result later

# Current Approaches

- In 1992, Kautz and Selman introduced the idea of **planning as satisfiability**
- Encode possible  $k$ -step plans as Boolean formulas and use an **iterative deepening** search approach
- In 1995, Blum and Furst came up with the **planning graph** approach
- **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 the original GPS idea
- Geffner (1997) followed up with a propositional, simplified version (**HSP**) and Hoffmann & Nebel (2001) with an extended version integrating strong pruning (**FF**). Most recent: **Fast Downward** (Helmert et al.).
- Heuristic planners seem to be the **most efficient** non-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  $1, 2, \dots, k - 1$  before a plan of length  $k$  is produced.

- Traditionally, planning has been viewed as a special kind of **deductive** problem
  - Given
    - a formula describing possible **state changes**
    - a formula describing the **initial state** and a formula characterizing the **goal conditions**
    - try to prove the existential formula *there exists a sequence of state changes transforming the initial state into the final one*
- Since the proof is done constructively, the **plan** is constructed as a by-product

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

# Advantages of the Approach

- Has a more 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

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

$at(p1, c)$

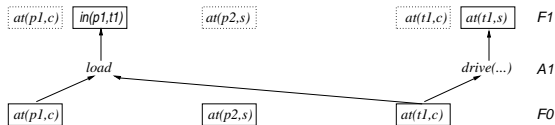
$at(p2, s)$

$at(t1, c)$

$F0$

# Example Graph

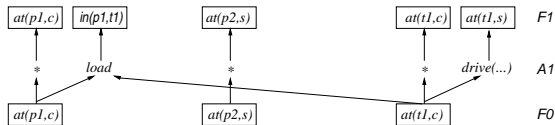
- $\mathbf{I} = \{at(p1, c), at(p2, s), at(t1, c)\}$ ,  $\mathbf{G} = \{at(p1, s), in(p2, t1)\}$
- All **applicable** actions and their positive effects are included





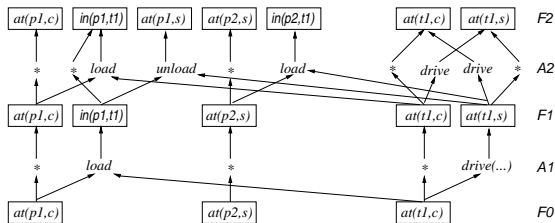
# Example Graph

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



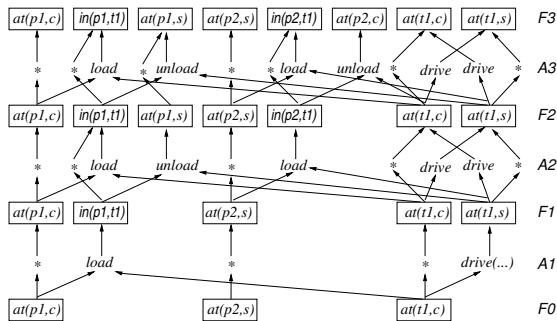
# Example Graph

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



# Example Graph

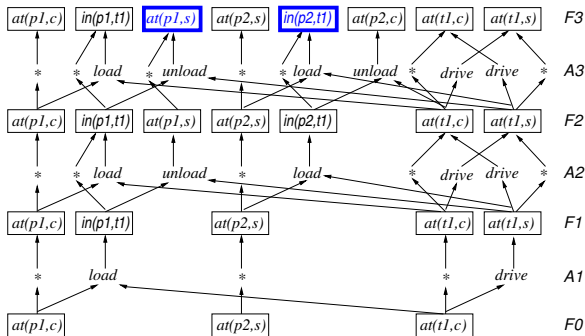
- $I = \{at(p1, c), at(p2, s), at(t1, c)\}$ ,  
 $G = \{at(p1, s), in(p2, t1)\}$
- All **applicable** actions and their positive effects are included
- In order to **propagate unchanged properties**, use *noop* action, denoted by \*
- **Expand** graph as long as not all goal atoms are in the fact level



- 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.

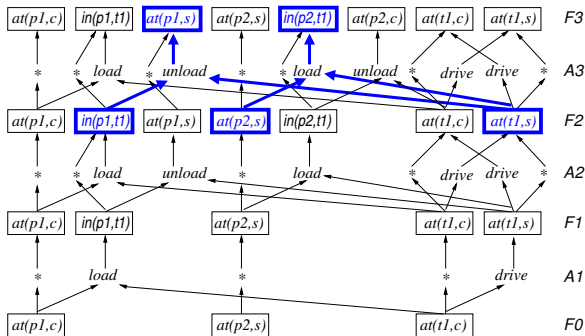
# Extracting From the Example Graph

Start with **goals** at highest fact level



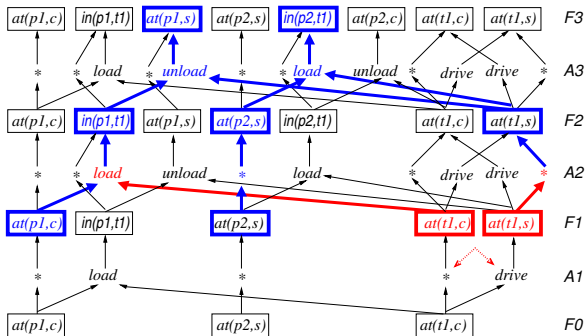
# Extracting From the Example Graph

Select minimal set of actions & corresponding **subgoals**



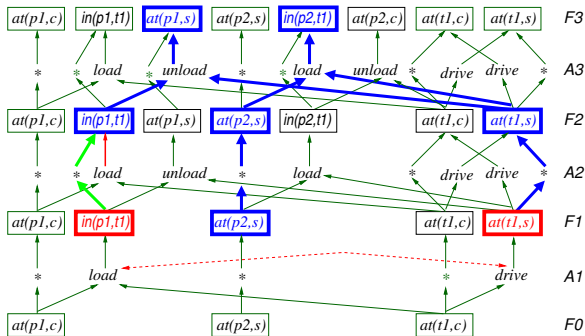
# Extracting From the Example Graph

Wrong choice leading to conflicting actions



# Extracting From the Example Graph

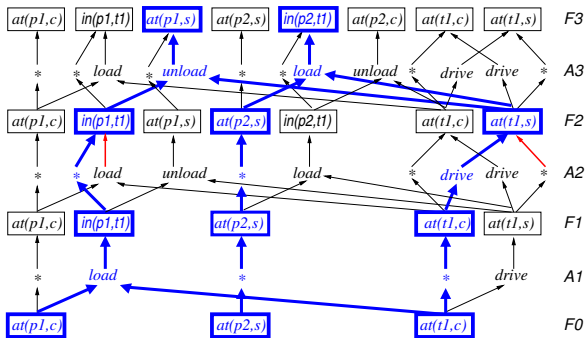
*Other choice, but no further selection possible*





# Extracting From the Example Graph

## Final selection



# 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

# Satisfiability-Based Planning based on Planning Graphs

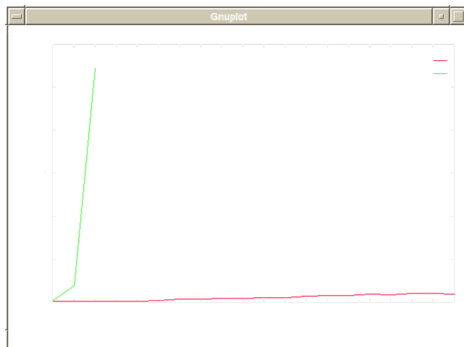
- Use **planning graph** in order to generate Boolean formula
  - The **initial facts** in layer  $F_0$  and the goal atoms in layer  $F_k$  are true
  - Each **fact** in layer  $F_i$  implies the **disjunction** of the actions having the fact as an effect
  - Each **action** implies the **conjunction** of the preconditions of the action
  - Conflicting actions cannot be executed at the same time.
- Turns out to be empirically **more efficient** than the earlier coding (because plans can be much shorter)
- Other codings are possible, e.g., purely **action-** or **state-based** codings

# 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**

# 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**

- One can use **progression** or **regression** search, or even search in the space of **incomplete partially ordered plans**
- One can use **local** or **global, systematic** search strategies
- One can use different **heuristics**, which can be compared along the dimension of being
  - **efficiently computable**, i.e., should be computable in poly. time
  - **informative**, i.e., should make reasonable distinctions between search nodes
  - and **admissible**, i.e., should underestimate the real costs (useful in  $A^*$  search).

- Consider all states that are **reachable** by executing one action
  - Try to improve the heuristic value
  - **Hill climbing**: Select the successor with the minimal heuristic value
  - **Enforced hill climbing**: Do a **breadth-first search** until you find a node that has a better evaluation than the current one.
- **Note**: Because these algorithms are not **systematic**, they cannot be used to prove the absences of a solution

- Maintain a list of **open nodes** and select always the one which is **best** according to the heuristic
- **Weighted A\***: combine estimate  $h(S)$  for state  $S$  and costs  $g(S)$  for reaching  $S$  using the weight  $w$  with  $0 \leq w \leq 1$ :

$$f(S) = w * g(S) + (1 - w) * h(S).$$

- If  $w = 0.5$ , we have ordinary **A\***, i.e., the algorithm finds the shortest solution provided  $h$  is **admissible**, i.e., the heuristics never overestimates
- If  $w < 0.5$ , the algorithm is **greedy**
- If  $w > 0.5$ , the algorithm behaves more like **best-first search**



# 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  $\rightsquigarrow h^+(s)$
- How difficult is **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?
- **PLANLEN** for precondition-free operators with only positive effects is **NP-complete**
- Consider approximations to  $h^+$ .

# The HSP Heuristic

- The first idea of estimating the distance to the goal for monotonic planning might be to count the number of unsatisfied goals atoms
- Neither **admissible** nor very **informative**
- Estimate the costs of making an atom  $p$  true in state  $S$ :

$$h(S, p) = \begin{cases} 0 & \text{if } p \in S \\ \min_{a \in \mathbf{O}, p \in \text{eff}^+(a)} (1 + \max_{q \in \text{pre}(a)} h(S, q)) & \text{otherwise} \end{cases}$$

- Estimate distance from  $S$  to  $S'$ :  $h(S, S') = \max_{p \in S'} h(S, p)$
  - Is **admissible**, because only the longest chain is taken, but it is not very *informative*
  - Use  $\sum$  instead of **max** (this is the HSP heuristics)
  - Is not *admissible*, but more **informative**. However, it ignores **positive** interactions!
- Can be computed by using a **dynamic programming technique**

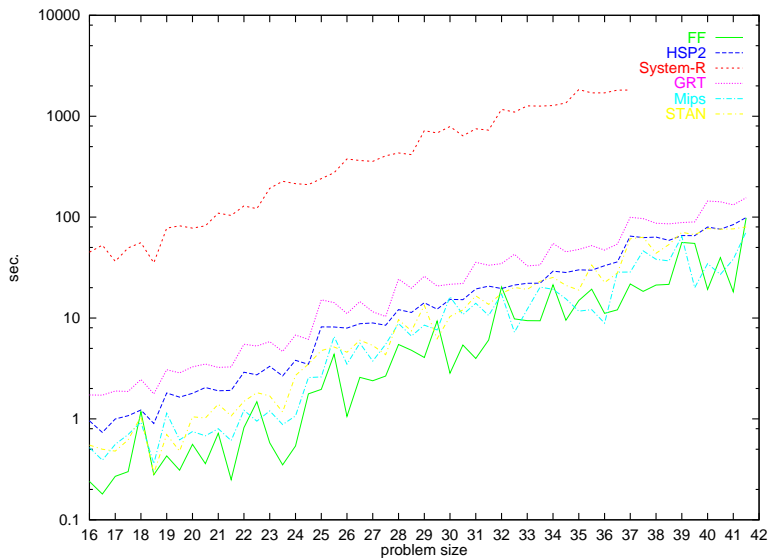
# The FF Heuristic

- 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

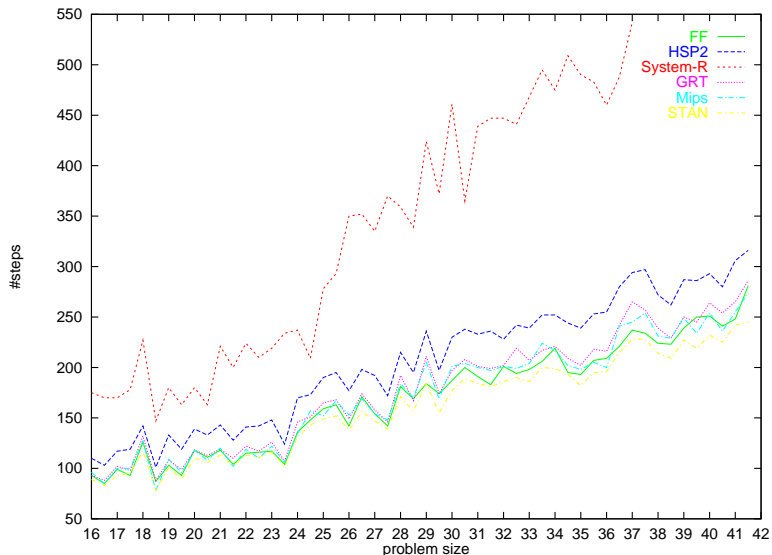
# The FF System

- 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 is one of the fastest planners around
- Meanwhile, faster systems such as FDD and LAMA, also designed in our group

# Runtime: *Logistics* in the 2000 competition



# Solution Quality: *Logistics* in the 2000 competition

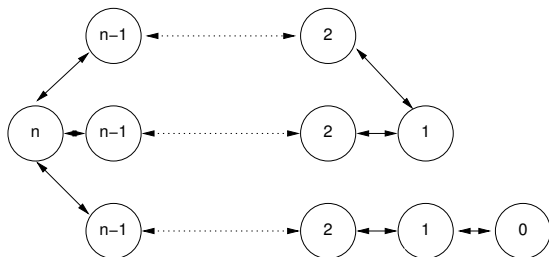




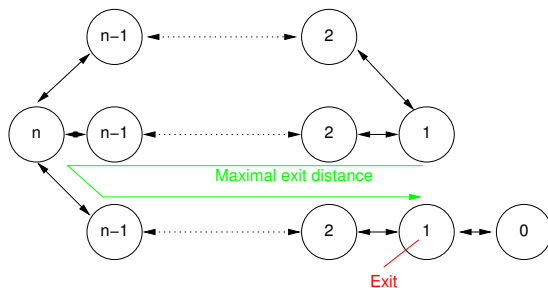
# FF – Why is it so Fast?

- FF was the fastest planner at the competition in 2000 across all planning domains – and still is a benchmark system
- Further experiments showed that this extends to most other planning domains in the literature
- What is the **search space topology** under the used heuristic estimator?
- Problematical issues in the search space topology:
  - local minima
  - benches
  - dead ends

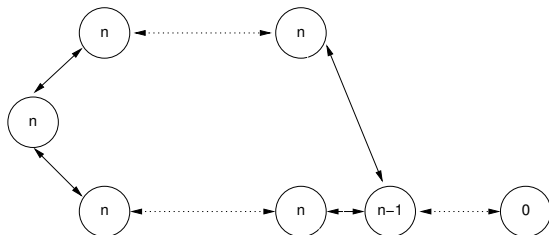
We have to go “upwards” before we can leave



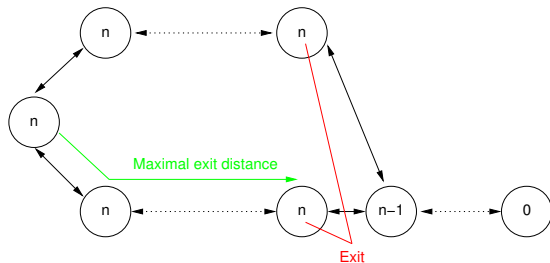
We have to go “upwards” before we can leave



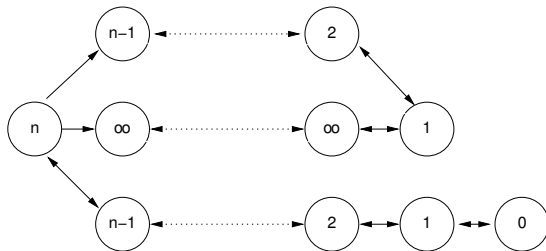
All neighboring states look the same



All neighboring states look the same

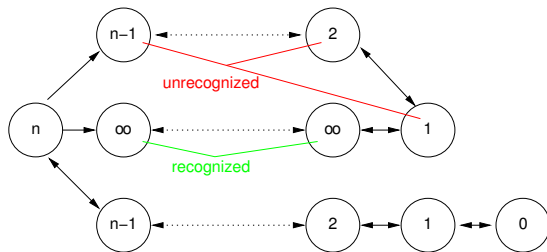


There is no path to a solution

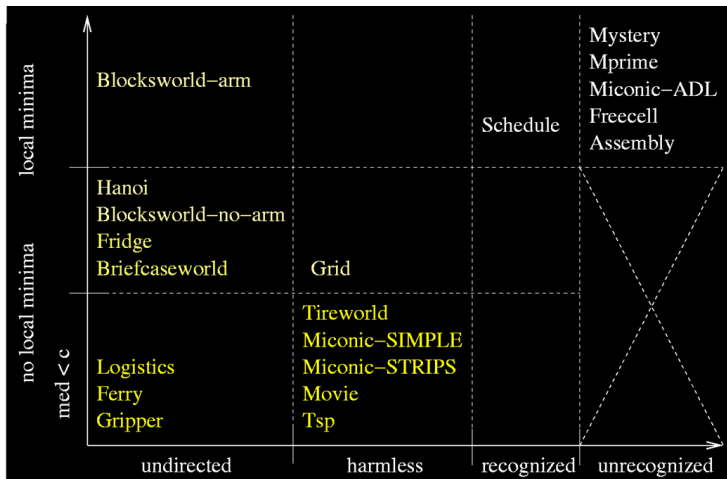


# Dead Ends

There is no path to a solution



# Classification of Benchmark Domains



These properties have been analytically proven for  $h^+$ , but apply empirically also to the FF heuristic



- Foundation / theory
- Extending planning technology in order to cope with multi-agent scenarios and epistemic goals
- Highly efficient planning systems (that are competitive at the IPC)
- Using planning techniques and extending them for robot control
- Using planning methodology in application in general

# Summary

- Rational agents need to **plan** their course of action
- In order to describe planning tasks in a domain-independent, declarative way, one needs **planning formalisms**
- Basic **STRIPS** is a simple planning formalism, where actions are described by their preconditions in form of a conjunction of atoms and the effects are described by a list of literals that become true and false
- **PDDL** is the current “standard language” that has been developed in connection with the **international planning competition**
- Basic planning algorithms search through the space created by the **transition system** or through the **plan space**.
- Planning with **STRIPS** using **first-order** terms is **undecidable**
- Planning with **propositional STRIPS** is **PSPACE-complete**
- Since 1992, we have reasonably efficient planning method for **propositional, classical STRIPS planning**
- You can learn more about it in our **planning class** next term.