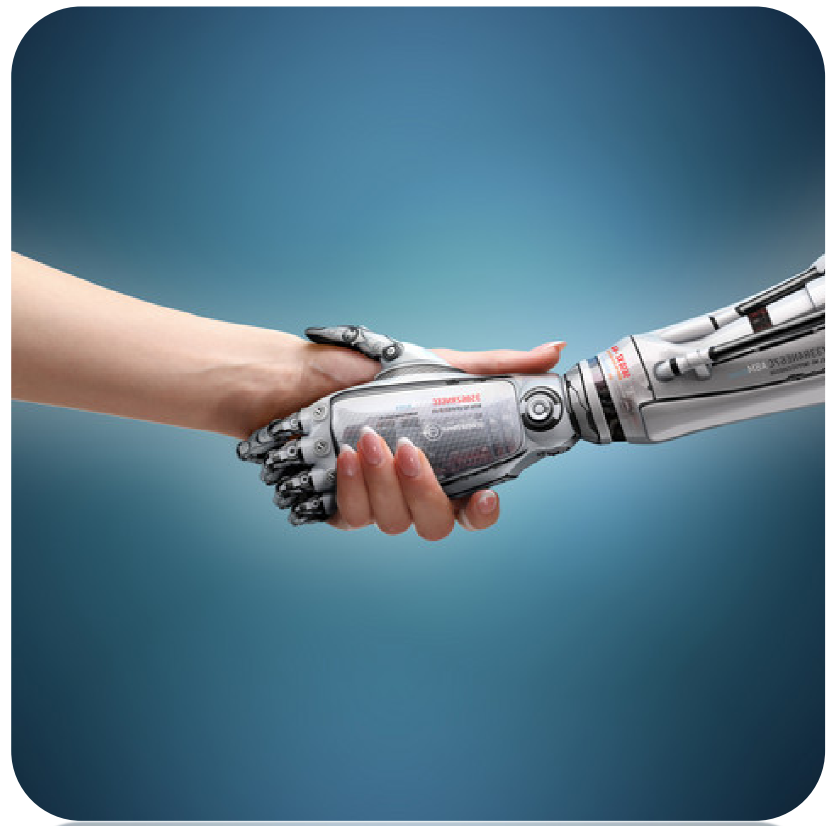
Zurich University
of Applied Sciences

Artificial Intelligence
V07: Planning

Planning as search
Algorithms for classical planning
Next steps



Based on material by
- Stuart Russell, UC Berkeley
- Peter Ljunglöf, U Gothenburg / Chalmers
- Malte Helmert, U Basel

Zurich University of Applied Sciences and Arts
CAI Centre for Artificial Intelligence (stdm)

Educational objectives

- **Remember PDDL** semantics
- **Explain** in which regard and why **planning is** the **large**st part **in AI**
- **Comprehend** and **extend plans** given **in PDDL**
- **Know** the **road ahead** for more **complicated** planning **problems including hierarchical planning**

"In which we see how an agent can take advantage of the structure of a problem to efficiently construct complex plans of action."

➔ Reading: AIMA, ch. 11 (excluding 11.5,11.6)

1. PLANNING AS SEARCH

Planning and AI

Classical planning

- *«Planning is the **art and practice of thinking before acting**»* Patrik Haslum
  «... finding a sequence of actions to accomplish a goal ...» AIMA ch. 11.1
- Planning agents **seen so far**:
  - **Problem solving** agent (V03/V04): atomic representation → needs domain-specific heuristics
  - Hybrid propositional **logic** agent (V06a): ground (i.e., variable-free) sentences → may get swamped

➔ The part of AI being **conducted by most** researchers today calling themselves ***«AI guys»***

Entirely true 2016 – changed with the current hype

Why is planning so big?

- Solved **applications**: Large logistics problems, operational planning, robotics, scheduling, …
- Community: **Search** is its basis; **logic** & **knowledge** representation is **part of it**
  → treated at specialized (ICAPS) and major AI (IJCAI, AAAI, ECCAI) international conferences
- **AI**'s tendency of **spawning new disciplines**:
  - Many now autonomous disciplines started as a field of study within AI
  - Examples: Computer vision, robotics, information retrieval, automatic speech recognition
  - Machine learning could take this path, but currently the "buzzword status of AI keeps it
- Other universalist tendencies: *"**Everything** is **search**", "everything is **optimization**"*

One of planning's big shots: Malte Helmert of University of Basel (→ see http://ai.cs.unibas.ch/misc/tutorial_aaai2015/)

Automated planning

Zurich University
of Applied Sciences

Setting

- a **single agent** in a (→ multi-agent / game-playing possible)
- **fully observable**, (→ conformant planning possible)
- sequential and discrete, (→ temporal and real-time planning possible)
- **deterministic** and (→ probabilistic planning possible)
- **static** (offline) environment (→ online possible)

"Tower of Hanoi" is a classic planning example

Tool: Planning Domain Definition Language (PDDL)

- A **subset of FOL**, more expressive than propositional logic
- Used to **define** the **planning task as** a **search** problem:
  - Initial states and goal states
  - A set of $Action(s)$ in terms of **preconditions** and **effects** → $Result(s, a)$
  - Closed world assumption: Unmentioned state variables are assumed false
- It allows for **factored** representation (collection of variables)
- Derived from the STRIPS planning language

PDDL / STRIPS operators
Tidily arranged action descriptions, restricted language

Note that capitalization of atoms (predicates & terms) is different here as compared to Datalog (V06b), to be consistent with AIMA.

From action schema to ground action: 2 examples

- Action schema (variables are universally quantified [∀]):
  $Action(\ Fly(p, from, to),$
  $\quad Precondition\colon\ At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
  $\quad Effect\colon\ \neg At(p, from) \land At(p, to))$
- Ground action (all variables have been substituted with values):
  Upper-case constants
  $Action(\ Buy(MarshallGuitarBox, BackstageMusic),$
  $\quad Precondition\colon\ At(BackstageMusic) \land Sells(BackstageMusic, MarshallGuitarBox)$
  $\quad Effect\colon\ Have(MarshallGuitarBox))$
  →Note: this abstracts away many important details of buying

Restricted language allows for **efficient** algorithms

- Action precondition: conjunction of positive literals
- Action effect: conjunction of literals
- Applicability of action a in state s: $iff\ s \vDash Precondition(a)$
  → E.g., $\forall p, from, to\ \left(Fly(p, from, to) \in Actions(s)\right) \Leftrightarrow s \vDash (At(p, from) \land Plane(p) \land Airport(from) \land Airport(to))$
- Computing the result: $Result(s, a) = (s - Del(a)) \cup Add(a)$ without explicit reference to time!
  (delete list contains all negative literals in $Effects(a)$, add list all positives)

Example: air cargo transport

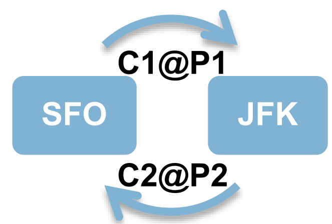
- A classical transportation problem: Loading / unloading cargo, flying between different airports
- Actions: $Load(cargo, plane, airport)$, $Unload(cargo, plane, airport)$, $Fly(plane, airport, airport)$
- Predicates: $In(cargo, plane)$, $At(cargo \vee plane, airport)$
- **Complete** PDDL **planning problem** description (with all variables existentially quantified [∃]):

Initial & goal state are given; $Action(s)$ and $Result(s, a)$ follow from action schemas.

$$
\begin{array}{l}
Init(At(C_1,\ SFO) \wedge At(C_2,\ JFK) \wedge At(P_1,\ SFO) \wedge At(P_2,\ JFK) \\
\quad \wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \\
\quad \wedge\ Airport(JFK) \wedge Airport(SFO)) \\
Goal(At(C_1,\ JFK) \wedge At(C_2,\ SFO)) \\
Action(Load(c,\ p,\ a), \\
\quad \text{PRECOND: } At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a) \\
\quad \text{EFFECT: } \neg\, At(c,\ a) \wedge In(c,\ p)) \\
Action(Unload(c,\ p,\ a), \\
\quad \text{PRECOND: } In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a) \\
\quad \text{EFFECT: } At(c,\ a) \wedge \neg\, In(c,\ p)) \\
Action(Fly(p,\ from,\ to), \\
\quad \text{PRECOND: } At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) \\
\quad \text{EFFECT: } \neg\, At(p,\ from) \wedge At(p,\ to))
\end{array}
$$

- **Plan**:

$$[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),$$
$$Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO).]$$



Example: blocks world

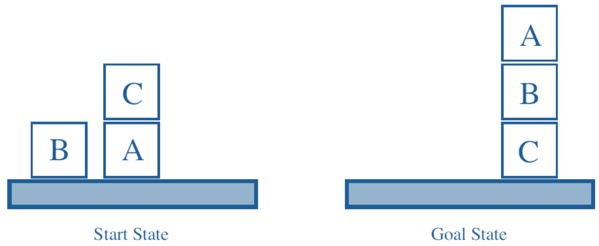
The blocks world

- A block is either on the table or on another block
- Blocks can be stacked (only if one fits directly on another)
- Goal: produce a given configuration of blocks on the table (specified as which is on top of what)
- Challenge: **No explicit quantifiers in PDDL** → need to introduce artificial predicates
  - Example: $Precondition$: $\neg\exists x\ On(x, A)$ not directly expressible → introduce predicate $Clear(A)$

The necessity for A having no block on top before becoming movable

Example

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$



→ A possible solution sequence: $[MoveToTable\ (C, A), Move(B, Table, C), Move(A, Table, B)]$

How difficult is planning?
Computational complexity of classical planning

Problem definition (see V06a for SAT)
- The PlanSAT problem: Does there exist a plan that achieves the goal?
- The bounded PlanSAT problem: Does there exist a solution of length at most k?
  → useful for optimal (i.e., shortest plan) planning

The PSPACE class contains problems **solvable** by a deterministic algorithm **with its memory constrained to be polynomial in the input** length
→ **larger** & **more difficult than NP** (but no constraint on time)

Complexity
- PlanSAT and bounded PlanSAT are PSPACE-complete
  →i.e., **difficult** (assumed to be not even in NP)!
- PlanSAT **without negative** preconditions and without negative effects is in P
  → i.e., **solvable**

Practice
- **Sub-optimal planning** is sometimes **easy**
- **PDDL has facilitated** the development **of very accurate** domain-independent **heuristics** making planning feasible (formalisms based on FOL have had less success)

2. ALGORITHMS FOR CLASSICAL PLANNING

Planning as state-space search
…approachable with any algorithm from V03 or local search

Two formulations
- Forward (progression): search considers actions that are **applicable**
- Backward (regression): search considers actions that are **relevant**
- **Neither** of them is **efficient without** good **heuristics**!

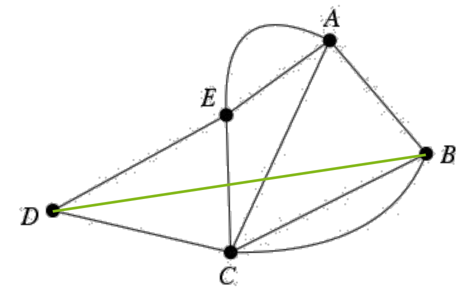
Futility of uninformed forward **search**
- Example 1: Buying a copy of AIMA
  - Tool: Action schema $Buy(isbn)$ with effect $Own(isbn)$
    → 10-digit ISBN leads to $10^{10} = \mathbf{10\ billion}$ **ground actions** to be enumerated
- Example 2: Moving all cargo from airport A to airport B
  - Setting: 10 airports with 5 planes and 20 pieces of cargo at each
  - Obvious solution: load all cargo at A in one of the planes, fly to B, unload everything (41 actions)
    → search graph has $\mathbf{2000^{41}}$ **nodes** up to this depth (assuming $\sim 2'000$ actions per state on average)

- 450 if all cargo is at airports without planes (10*5 airplanes can each fly to 9 other airports)
- 10450 if all cargo and planes are at the same airport (10*20 cargo packages can be each loaded into 10*5 planes, plus the 50 planes can fly to 9 other airports)

Heuristics for forward state-space search
Enabled by factored representations for states & actions

Possible domain-independent heuristics

- **Relaxing actions** (i.e., adding new links to the graph to ease the problem)
  - Ignore-preconditions heuristic: All actions are applicable anytime
    → leads e.g. easily to the 2 different heuristics for the n-puzzle of V03
  - Ignore-delete-lists heuristic: Removing all negative literals from effects
    → enables making monotonic progress towards goal, achievable e.g. with hill climbing



- **State abstractions** (i.e., collapsing multiple states into a single one to shrink the graph)
  - Reduce the state space by e.g. **ignoring** *some* fluents

Winners of the bi-annual ICAPS planning competition often used

- Heuristic search (→ see FastDownward system: Helmert et al. 2004, http://www.fast-downward.org/)
- Planning graphs (→ see appendix)
- SAT solvers (→ see V06a and below)

SATplan and CSP solvers
One option for planning

Translate PDDL description into a SAT problem or a CSP
- Goal state and all actions have to be propositionalized
- Action schemas have to be replaced by a set of ground actions (variables to be replaced by constants)
- Fluents need to be introduced for each time step
- …

➔ combinatorial explosion

Cost – benefit
- Remove a part of the benefits of the expressiveness of PDDL to…
- …gain access to efficient solution methods of SAT and CSP solvers

Hierarchical planning
A modern, more general alternative

The need for **abstraction**

How many willful decisions (= possible actions) in a 2-week vacation?

Technical solution sketch

- Hierarchical task networks (HTN): more **factored** representations for **actions** (besides states)
- Two kinds of actions:
  - Primitive actions: standard precondition-effect schemas
  - High level actions (HLA): e.g., *"go to ZRH"* → have one or more possible refinements
  - Refinement: a **sequence of HLAs or primitive** actions, maybe recursive
- Key benefits: Possibly huge **speed** improvements, possibility for **humans** to **define HLAs**

Hierarchical planning algorithms

HLA's generally have multiple possible implementations ... two approaches:

A. **Search for primitive solutions**
B. **Search for abstract solutions (reason about HLA's)**

A. Search for primitive solutions

- Algortithm **HIERARCHICAL-SEARCH**
  - **Recursively** chose an HLA in current plan
  - **Replace HLA** with one of its refinements, until plan **achieves its goal**
  - Can be implemented as **breadth-first search** (alternatives possible)

- **Computational advantage:**
  - **"Flat" problem** with d primitive actions (=depth), b allowable actions at each state (=branching): **O(b^d)**
  - **Hierarchical problem:** Assume each **HLA has r refinements into k actions** at next lower level: n=(d-1)/(k-1) refinement nodes, i.e., r^n decomposition trees
  - **Small r, large k: huge savings** (small number of refinements with long action sequence), equiv. to **k-th root of non-hierarchical cost**

Initial plan: [Act]
REFINEMENTS(): returns set of action sequences whose preconditions are satisfied

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
    if IS-EMPTY(frontier) then return failure
    plan ← POP(frontier)          // chooses the shallowest plan in frontier
    hla ← the first HLA in plan, or null if none
    prefix,suffix ← the action subsequences before and after hla in plan
    outcome ← RESULT(problem.INITIAL, prefix)
    if hla is null then          // so plan is primitive and outcome is its result
      if problem.IS-GOAL(outcome) then return plan
    else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
      add APPEND(prefix, sequence, suffix) to frontier
```

- **In practice:**
  - **Store** once-used implementations of complex HLA's, put in **library**, then **re-use for even more complex problems**
  - Ability to **generalize** from details specific to problem instance (name of worker in factory etc.)

Hierarchical planning algorithms (cont.)

$Refinement(Go(Home, SFO),$
$\quad \text{STEPS: } [Drive(Home, SFOLongTermParking),$
$\quad\quad Shuttle(SFOLongTermParking, SFO)]\ )$
$Refinement(Go(Home, SFO),$
$\quad \text{STEPS: } [Taxi(Home, SFO)]\ )$

Problem: HIERARCHICAL-SEARCH refines all HLA's to primitive action sequences to determine if plan is workable
➔ **Better: Reason about the HLA's directly**

B. Search for abstract solutions

- **Example:** Determine if the following 2-HLA high level plan
  `[Drive(Home, SFOLongTermParking), Shuttle(SFOLongTermParking,SFO)]`
  is a valid implementation of `Go(Home,SFO)`
- **Goal:** Derive high-level plan which provably achieves goal (results in exponential reduction of computational cost), i.e. has at least one valid implementation (downward refinement property)
- **Ansatz:** Consider precondition-effect descriptions of HLA's
- **Problem:** Describe effects of multiple-implementation-HLA's

- **First try:** positive (negative) effects resulting from every (any) implementation
  - too restrictive, e.g., if precondition for one implementation is not fulfilled
  - equivalent to someone else choosing implementation (i.e. nondeterministic), a.k.a. "demonic nondeterminisim"
- **Better:** Agent makes choices ("angelic nondetermininsm")

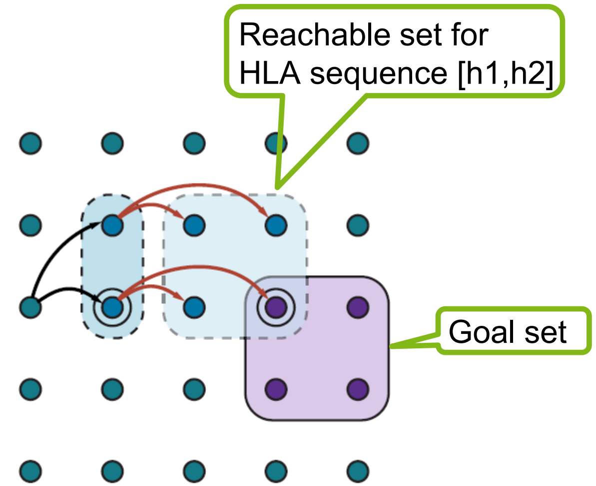
Hierarchical planning algorithms (cont.) Angelic Search

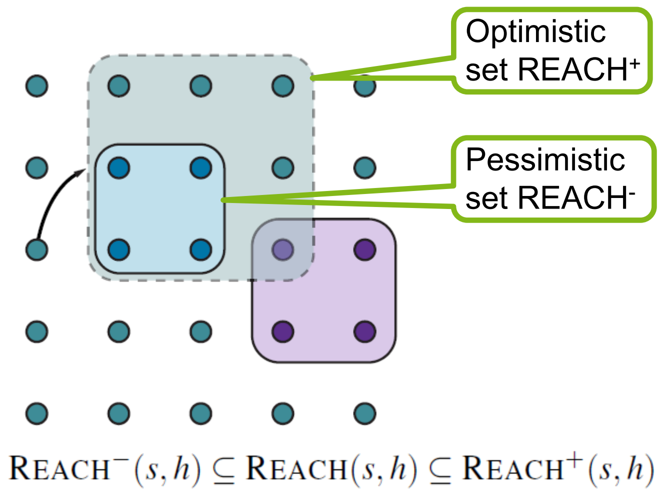
- **Reachable set:** set of states reachable by any of HLA h's implementation, given state s: `REACH(s,h)`
  - More rich/powerful for HLA's with multiple refinements
  - High-level plan achieves goal if reachable set intersects set of goal states
- **Algorithm:** Find high-level plan whose reachable set intersects goal, then refine

- **In practice:** HLA's may have infinitely many implementations → need **approximate descriptions of reachable set**: optimistic `REACH⁺(s,h)` / pessimistic `REACH⁻(s,h)`
  - If `REACH⁻` does intersect goal, plan does work
  - If `REACH⁺` does not intersect the goal, plan does not work
  - Else, undecidable: need to refine plan (see bottom figure)

- **Algorithm "ANGELIC-SEARCH"** (details see AIMA, sec. 11.4.3)
  - Computationally <u>much better</u> than hierarchical search

- **Example:** Clean your apartment (several rooms, connected by narrow corridors)
  - 5 Low-level actions (`u,d,l,r,suck`)
  - 2 HLA's: `Navigate, CleanRoom`
- Breadth-first: 5^d (d: length of shortest solution)
- Hierarchical search: still exponential (tries all ways consistent with hierarchy)
- Angelic search: approx. linear in number of squares





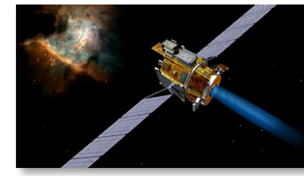
$$\text{REACH}^-(s,h) \subseteq \text{REACH}(s,h) \subseteq \text{REACH}^+(s,h)$$

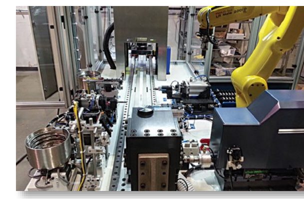
3. NEXT STEPS

Planning in the real world

Use cases

- **Spacecraft operation** in **real time** (1998)
  NASA's Deep Space 1 was controlled by a planning & scheduling system devising and carrying out plans like *«During the next week take pictures of the following asteroids and thrust 90% of the time»* (→ see [Nilsson, 2010] ch. 32.2.1)
- **Factory scheduling** (1985)
  4-week (3 shifts a day) production plan at Hitachi for assembly line of 350 products with 35 machines and >2000 different operations (→ see AIMA ch. 11.4.2, **HTN**)
- **Military operation planning** (1990)
  A scheduling program helped with the logistics of 1st gulf war and is said to have *«paid back all of DARPAs 30 years of investment in AI in a matter of a few months»* (→ see [Nilsson, 2010] ch. 23.3.3)





Challenges

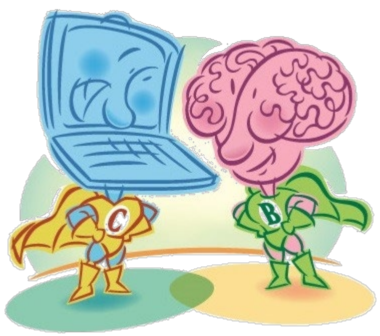
- Taking **resources** (incl. time) into account → scheduling
- Being **overwhelmed by** state space **size** → hierarchical planning
- Needing to **incorporate human** wisdom → hierarchical planning
- Coping with **uncertainty** → conformant / contingency / online planning (analog to AIMA ch. 4)
- Planning with **multiple agents** → planning with **cooperative and adversarial** multiagents is **unsolved**

Where's the intelligence?
Man vs. machine

- Planning is foremost an **exercise in controlling combinatorial** explosion
- It does so by **combining** efficient **search & logic**al reasoning
  → necessary speedups are achieved by **domain-independent heuristics** that exploit structure in the representation
  → this is **really smart**

- But: There is **no clear understanding** yet of **which** methods **work best on what** problems

- In contrast to popular opinion, AI planning is **widely applied in practice** today
  → Also, research is not "dead", but **less hyped** at the moment
  → Probably planning is the **best** that **symbolic AI** currently offers



Automating university timetabling by planning?
A search exercise

Automatic scheduling is a relevant subfield of AI planning. Likewise, automated timetable generation (often focused on university teaching timetables) is a vibrant field of study.

- Conduct a **quick literature research** on automated timetabling (e.g., https://scholar.google.ch/scholar?q=automated+timetabling)
- What **kind of approaches** are proposed? How do they **relate to AI planning** as you have heard of here?
- With your current understanding of AI – **how would you approach the problem**? What are your **options**?

Zeit	Montag 20.02.	Dienstag 21.02.	Mittwoch 22.02.	Donnerstag 23.02.	Freitag 24.02.
08:00 - 08:45	T_IT14a... Stdm TH 561 T_IT13t... Stdm TH 561 T_IT13t... Stdm TH 561 T_IT14a... Stdm TH 561 T_VSRep.BA Stdm TH 561 T_IT14t... Stdm TH 561 T_IT14t... Stdm TH 561				
08:50 - 09:35					
10:00 - 10:45	T_IT14t... Tugg TH 547 T_IT14t... Tugg TH 547 T_VSRep.BA Tugg TH 547 T_IT13t... Tugg TH 547 T_IT13t... Tugg TH 547 T_IT14a... Tugg TH 547 T_IT14a... Tugg TH 547				
10:50 - 11:35					
12:00 - 12:45					
12:50 - 13:35					T_IT14t... Stdm ZL O5.05 T_ITRep... Stdm ZL O5.05 T_IT14a... Stdm ZL O5.05 T_IT13t... Stdm ZL O5.05 T_IT13t... Stdm ZL O5.05 T_IT14a... Stdm ZL O5.05
14:00 - 14:45		T_IT14t... Tugg TH 544 T_IT14t... Tugg TH 544 T_VSRep.BA Tugg TH 544 T_IT13t... Tugg TH 544 T_IT13t... Tugg TH 544 T_IT14a... Tugg TH 544 T_IT14a... Tugg TH 544			
14:50 - 15:35					T_IT14a... Tugg ZL O5.16 T_IT13t... Tugg ZL O5.16 T_IT13t... Tugg ZL O5.16 T_ITRep... Tugg ZL O5.16 T_IT14t... Tugg ZL O5.16 T_IT14a... Tugg ZL O5.16
16:00 - 16:45					

Review

- **Planning** is **AI's main field**, due to success stories like remotely controlling a NASA spacecraft in real time
  - Planning refers to problem solving techniques (**i.e.**, **search**) on factored (i.e., **logic-based**) representations of states and actions, allowing for **fast algorithms**
  - **PDDL** describes the initial and goal **states as conjunctions** of literals; **actions** in terms of their **preconditions and effects**

- Effective domain-independent **heuristics** are derived **by subgoal independence** or **problem relaxation**

- Valid approaches include using **SAT or CSP solvers**
  - **FOL-based** planning has much-needed expressiveness for larger real-world problems, but yet **no efficient algorithms** (missing heuristics)

- **Hierarchical task networks (HTN)** allow the agent to take advice from the domain designer in the form of **high-level actions (HLAs)** that can be implemented in various ways by lower-level action sequences
  - Effects of HLAs can be defined with **angelic semantics**, allowing provably correct high-level plans to be derived without consideration of lower-level implementations
  - HTN methods can create **very large plans** required by many **real-world applications**

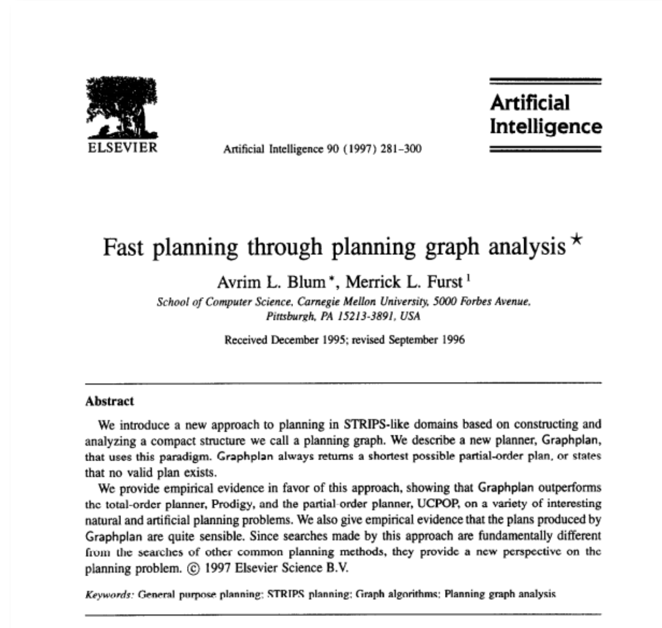
- It is yet **unknown which** approach **is best**

Zurich University
of Applied Sciences

APPENDIX

Planning graphs
An alternative to basic state-space search

Challenges so far
- **Exponential** size of the search **trees**
- **Not all heuristics** are **admissible** in general


ELSEVIER

Artificial Intelligence 90 (1997) 281–300

Artificial Intelligence

Fast planning through planning graph analysis ☆

Avrim L. Blum *, Merrick L. Furst [1]
School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3891, USA

Received December 1995; revised September 1996

Abstract

We introduce a new approach to planning in STRIPS-like domains based on constructing and analyzing a compact structure we call a planning graph. We describe a new planner, Graphplan, that uses this paradigm. Graphplan always returns a shortest possible partial-order plan, or states that no valid plan exists.

We provide empirical evidence in favor of this approach, showing that Graphplan outperforms the total-order planner, Prodigy, and the partial-order planner, UCPOP, on a variety of interesting natural and artificial planning problems. We also give empirical evidence that the plans produced by Graphplan are quite sensible. Since searches made by this approach are fundamentally different from the searches of other common planning methods, they provide a new perspective on the planning problem. © 1997 Elsevier Science B.V.

Keywords: General purpose planning; STRIPS planning; Graph algorithms; Planning graph analysis

Solution: the planning graph
1. **Propositionalize** the search tree:
   replace all action schemas by sets of ground actions (to remove variables etc.)
2. **Approximate** the complete propositionalized tree
   - **Polynomial size:** $O\big(n(a+l)^2\big)$ for a actions, l literals and n levels
   - Useful to **create admissible heuristics** like set-level heuristic (→ see appendix):
     cost of achieving $\wedge$ of goals = $\sum g_i$ (level cost) of goals in first level without mutual exclusivity

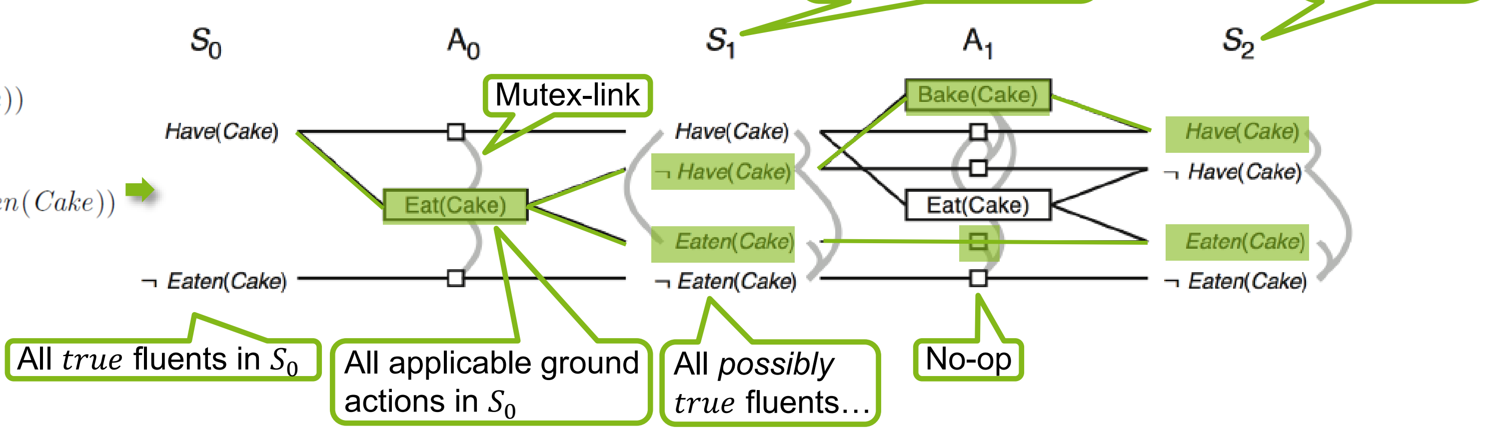
The planning graph

Organized in **alternating levels** of possible states S_i and applicable actions A_i

- S_i holds all **fluents that could be true** at that point
- A_i holds all **actions** that could have their preconditions satisfied
- Links *between* levels represent **preconditions** and **effects**
- Links *within* the levels express **conflicts** ("mutex"-links)

Example: the *«have cake and eat cake too»* problem

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake))$



The GraphPlan algorithm
Plan directly using the planning graph

```
function GraphPlan(problem) returns solution or failure
    graph ← Initial-Planning-Graph(problem) #i.e., create S_0
    goals ← Conjuncts(problem.GOAL) #AND of all goal literals
    nogoods ← an empty hash table #used for the same purpose as in constraint learning (→ see V05)
    for t = 0 to ∞ do
        if goals all non-mutex in S_t of graph then
            solution ← Extract-Solution(graph, goals, Numlevels(graph), nogoods) #e.g. CSP or backward search
            if solution ≠ failure then return solution
        if graph and nogoods have both leveled off then return failure
        graph ← Expand-Graph(graph, problem)
```

Description
- GraphPlan **expands** the graph with new levels $A_i \& S_{i+1}$ **until ∄ mutex** links **between goals**
- The `nogoods` list records `(level, goal)` where goal couldn't be satisfied at that level
  → prevents `ExtractSolution` from searching again if called with the same arguments
- **To extract** the actual **plan**, the algorithm **searches backwards** in the graph
  - Initial state is the last level S_n of the planning graph
  - Available actions at level S_i are conflict-free subsets of actions at A_{i-1} with effects realizing S_i's goals
  - Goal is to reach a state at S_0 such that all goals are satisfied

→ The plan **extraction** is the difficult part and is usually done with **greedy-like heuristics**

Using planning graphs to devise heuristics

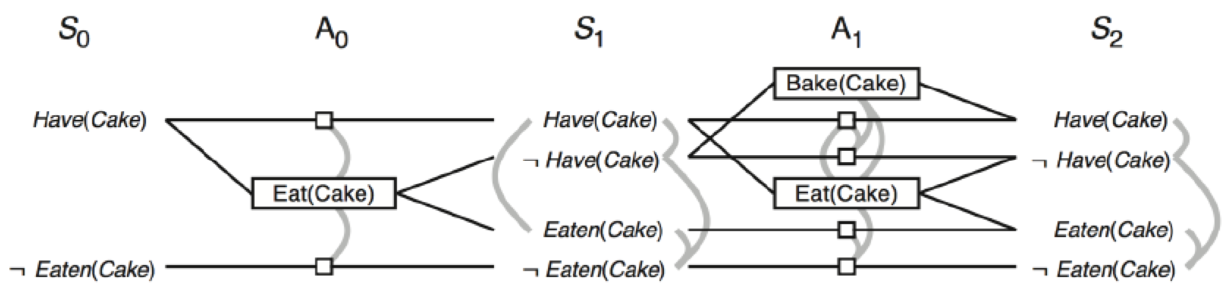
I.e., only one action per level / time step

A (serial) planning graph facilitates domain-independent heuristics

- Problem is unsolvable if any goal literal fails to appear in the final level
- Level cost of goal g_i: Level in planning graph at which g_i first appeared
- Heuristics for conjunctions of goals:
  - Max-level heuristic: **maximum** of the **level costs** of all subgoals
  - Level sum heuristic: **sum** of **level costs** of all subgoals (assuming independence → not necessarily admissible)
  - Set-level heuristic: **First level** at which all subgoals appear **without any pair being mutex**
- As with CSPs: **checking** for **pair-wise** consistency often **pays off**; higher order often doesn't

Heuristic values for $Have(Cake) \wedge Eaten(Cake)$:

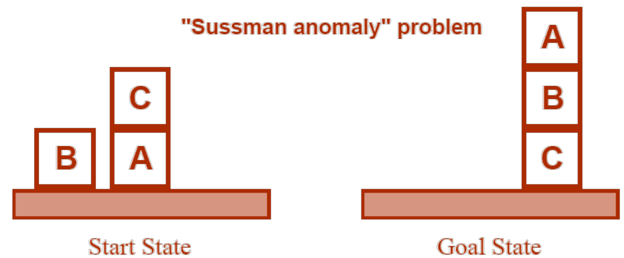
- Max-level: $\max(1,0) = 1$
- Level sum: $1 + 0 = 1$
- Set-level: 2 (**accurate!**)



Historical remark: Linear planning

Planners in the early 1970s considered **totally ordered** action **sequences**

- Problems were decomposed in subgoals
- Resulting subplans were stringed together in some order
  ➔This is called linear planning



But, linear planning **is incomplete!**

- There are some very simple problems it cannot handle
- E.g., the Sussman anomaly: Unsolvable by linear planner
  ➔ A complete planner must be able to interleave subplans

Enter partial-order planning, state-of-the-art during the 1980s and 90s

- Today mostly **used for specific tasks**, such as operations scheduling
- Also used **when** it is **important for humans to understand** the plans
  ➔ E.g., operational plans for spacecraft and Mars rovers are checked by human operators before uploaded to the vehicles