What is planning?

Planning is the art and practice of thinking before acting: of reviewing the courses of action one has available and predicting their expected (and unexpected) results to be able to choose the course of action most beneficial with respect to one's goals.

Patrik Haslum

What is planning?

Planning is the art and practice of thinking before acting: of reviewing the courses of action one has available and predicting their expected (and unexpected) results to be able to choose the course of action most beneficial with respect to one's goals.

Patrik Haslum

Problem of Action Selection

Approaches in AI to Problem of Action Selection

1. Programming: specify control by hand
2. Learning: learn control from experience
3. Planning: derive control automatically from model

Approaches in AI to Problem of Action Selection

1. Programming: specify control by hand
2. Learning: learn control from experience
3. Planning: derive control automatically from model

Planning is the model-based approach to action selection: produces the behavior from the model (solves the model)

Approaches in AI to Problem of Action Selection

1. Programming: specify control by hand
2. Learning: learn control from experience
3. Planning: derive control automatically from model

Planning is the model-based approach to action selection: produces the behavior from the model (solves the model)

Other famous model-based techniques: SAT, CSP/COP, MILP

Classical Planning Model

Classical planning model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where

- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function

$$s' = f(a, s) \text{ for } a \in A(s)$$

- Non-negative action costs $c(a, s)$

Classical Planning Model

Classical planning model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where

- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function
$$s' = f(a, s) \text{ for } a \in A(s)$$
- Non-negative action costs $c(a, s)$

Solution: sequence of applicable actions that maps s_0 into S_G

Classical Planning Model

Classical planning model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$, where

- Finite and discrete state space S
- A known initial state $s_0 \in S$
- A set $S_G \subseteq S$ of goal states
- Actions $A(s) \subseteq A$ applicable in each $s \in S$
- A deterministic transition function

$$s' = f(a, s) \text{ for } a \in A(s)$$

- Non-negative action costs $c(a, s)$

Solution: sequence of applicable actions that maps s_0 into S_G

Different models obtained by relaxing assumptions in blue: planning with preferences, conformant planning, contingent planning, FOND, MDPs, POMDPs, ...

Language for Classical Planning: STRIPS

A STRIPS Planning task is 5-tuple $\Pi = \langle F, O, c, I, G \rangle$:

- F: finite set of atoms (boolean variables)
- O: finite set of operators (actions) of form $\langle Add, Del, Pre \rangle$
  (Add/Delete/Preconditions; subsets of atoms)
- $c : O \mapsto \mathbb{R}^{0+}$ captures operator cost
- I: initial state (subset of atoms)
- G: goal description (subset of atoms)

Language for Classical Planning: SAS$^+$

A SAS$^+$ Planning task is 5-tuple $\Pi = \langle V, O, c, I, G \rangle$:

- V: finite set of finite-domain multi-valued variables
- O: finite set of operators (actions) of form $\langle pre, eff \rangle$ (Preconditions/Effects; partial variable assignments)
- $c : O \mapsto \mathbb{R}^{0+}$ captures operator cost
- I: initial state (variable assignment)
- G: goal description (partial variable assignment)

Plan: sequence of applicable actions that maps I into a state consistent with G

From Language to Models

A STRIPS Planning task $\Pi = \langle F, O, c, I, G \rangle$ determines state model $\mathcal{S}(\Pi)$ where

- the states $s \in S$ are collections of atoms from F
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Pre(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s) = c(a)$

♠ Solutions of $\mathcal{S}(\Pi)$ are plans of Π

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \to s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \to s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise
- Backward/reverse model: $s_{i+1} \to (a, s_i)$ where $Del(a) \cap s_{i+1} = \emptyset$ and
  $s_i = s_{i+1} - Add(a) + Pre(a)$

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \rightarrow s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise
- Backward/reverse model: $s_{i+1} \rightarrow (a, s_i)$ where $Del(a) \cap s_{i+1} = \emptyset$ and
  $s_i = s_{i+1} - Add(a) + Pre(a)$
- Inverse model: $(s_{i+1}, s_i) \rightarrow a$, where $Pre(a) \subseteq s_i$ and
  $s_{i+1} = s_i - Del(a) + Add(a)$

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \rightarrow s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise
- Backward/reverse model: $s_{i+1} \rightarrow (a, s_i)$ where $Del(a) \cap s_{i+1} = \emptyset$ and
  $s_i = s_{i+1} - Add(a) + Pre(a)$
- Inverse model: $(s_{i+1}, s_i) \rightarrow a$, where $Pre(a) \subseteq s_i$ and
  $s_{i+1} = s_i - Del(a) + Add(a)$
- Rewards approximate $-c^*(s_{i+1})$, the negative true cost of reaching the goal (reward obtainable) from s_{i+1}: $(s_i, a, s_{i+1}) \rightarrow h(s_{i+1}) - h(s_i)$

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \to s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise
- Backward/reverse model: $s_{i+1} \to (a, s_i)$ where $Del(a) \cap s_{i+1} = \emptyset$ and
  $s_i = s_{i+1} - Add(a) + Pre(a)$
- Inverse model: $(s_{i+1}, s_i) \to a$, where $Pre(a) \subseteq s_i$ and
  $s_{i+1} = s_i - Del(a) + Add(a)$
- Rewards approximate $-c^*(s_{i+1})$, the negative true cost of reaching the goal (reward obtainable) from s_{i+1}: $(s_i, a, s_{i+1}) \to h(s_{i+1}) - h(s_i)$
- Non-deterministic setting is (slightly) more complicated

Planning and Model-based Reinforcement Learning

- Forward model: $(a, s_i) \rightarrow s_{i+1} = s_i - Del(a) + Add(a)$ if $Pre(a) \subseteq s_i$,
  $(a, s_i) \mapsto s_i$ otherwise
- Backward/reverse model: $s_{i+1} \rightarrow (a, s_i)$ where $Del(a) \cap s_{i+1} = \emptyset$ and
  $s_i = s_{i+1} - Add(a) + Pre(a)$
- Inverse model: $(s_{i+1}, s_i) \rightarrow a$, where $Pre(a) \subseteq s_i$ and
  $s_{i+1} = s_i - Del(a) + Add(a)$
- Rewards approximate $-c^*(s_{i+1})$, the negative true cost of reaching the goal (reward obtainable) from s_{i+1}: $(s_i, a, s_{i+1}) \rightarrow h(s_{i+1}) - h(s_i)$
- Non-deterministic setting is (slightly) more complicated
- In what follows: the benefit of the more informative model

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better

Classical Planning: Computational problems

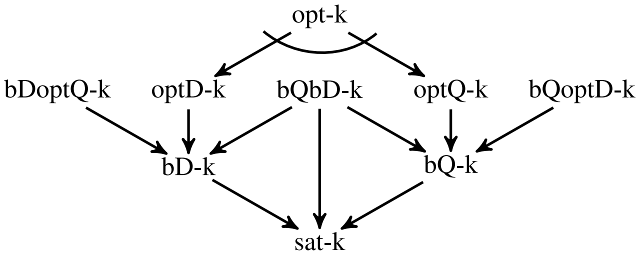
- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist
- Top-quality planning: find all plans up to a certain cost

Classical Planning: Computational problems

- Cost-optimal planning: find a plan that minimizes summed operator cost
- Satisficing planning: find a plan, cheaper plans a better
- Agile planning: find a plan, quicker is better
- Top-k planning: find k plans such that no cheaper plans exist
- Top-quality planning: find all plans up to a certain cost
- Diverse planning: variety of problems, aiming at obtaining diverse set of plans, considering plan quality as well



Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)

Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)
- Oversubscription planning: find a plan of bounded cost that maximizes utility

Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)
- Oversubscription planning: find a plan of bounded cost that maximizes utility
- Conformant planning: find a plan that works for all possible initial states

Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)
- Oversubscription planning: find a plan of bounded cost that maximizes utility
- Conformant planning: find a plan that works for all possible initial states
- Contingent planning: find a branching plan that works for all possible initial states

Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)
- Oversubscription planning: find a plan of bounded cost that maximizes utility
- Conformant planning: find a plan that works for all possible initial states
- Contingent planning: find a branching plan that works for all possible initial states
- Temporal planning: find which actions to execute and their consistent schedule

Beyond Classical Planning: Some computational problems

- Net-benefit planning: find a plan maximizes (utility - cost)
- Oversubscription planning: find a plan of bounded cost that maximizes utility
- Conformant planning: find a plan that works for all possible initial states
- Contingent planning: find a branching plan that works for all possible initial states
- Temporal planning: find which actions to execute and their consistent schedule
- Probabilistic planning: find a policy, mapping of states to actions, optimizing, e.g.
  - expected total rewards over a finite horizon
  - expected average rewards over an infinite horizon
  - expected discounted rewards over an infinite horizon

Why is planning difficult?

- Solutions to classical planning problems are paths from an initial state to a goal state in the transition graph
  - Efficiently solvable by Dijkstra's algorithm in $O(|V| \log |V| + |E|)$ time
  - Why don't we solve all planning problems this way?
- State spaces may be huge: 10^{100} states is not uncommon
  - Constructing the transition graph is infeasible!
  - Planning algorithms try to avoid constructing whole graph, use concise representation for many transitions as a single action
  - Complexity measured in terms of (concise) input size: simplest case is PSPACE-complete
- Planning algorithms often are more efficient than obvious solution methods constructing the transition graph and using e.g. Dijkstra's algorithm (mostly infeasible in practice)

Why is planning difficult?

- Solutions to classical planning problems are paths from an initial state to a goal state in the transition graph
  - Efficiently solvable by Dijkstra's algorithm in $O(|V| \log |V| + |E|)$ time
  - Why don't we solve all planning problems this way?
- State spaces may be huge: 10^{100} states is not uncommon
  - Constructing the transition graph is infeasible!
  - Planning algorithms try to avoid constructing whole graph, use concise representation for many transitions as a single action
  - Complexity measured in terms of (concise) input size: simplest case is PSPACE-complete
- Planning algorithms often are more efficient than obvious solution methods constructing the transition graph and using e.g. Dijkstra's algorithm (mostly infeasible in practice)

Why is planning difficult?

- Solutions to classical planning problems are paths from an initial state to a goal state in the transition graph
  - Efficiently solvable by Dijkstra's algorithm in $O(|V| \log |V| + |E|)$ time
  - Why don't we solve all planning problems this way?
- State spaces may be huge: 10^{100} states is not uncommon
  - Constructing the transition graph is infeasible!
  - Planning algorithms try to avoid constructing whole graph, use concise representation for many transitions as a single action
  - Complexity measured in terms of (concise) input size: simplest case is PSPACE-complete
- Planning algorithms often are more efficient than obvious solution methods constructing the transition graph and using e.g. Dijkstra's algorithm (mostly infeasible in practice)

Why is planning difficult?

- Solutions to classical planning problems are paths from an initial state to a goal state in the transition graph
  - Efficiently solvable by Dijkstra's algorithm in $O(|V| \log |V| + |E|)$ time
  - Why don't we solve all planning problems this way?
- State spaces may be huge: 10^{100} states is not uncommon
  - Constructing the transition graph is infeasible!
  - Planning algorithms try to avoid constructing whole graph, use concise representation for many transitions as a single action
  - Complexity measured in terms of (concise) input size: simplest case is PSPACE-complete
- Planning algorithms often are more efficient than obvious solution methods constructing the transition graph and using e.g. Dijkstra's algorithm (mostly infeasible in practice)

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest
- Partial Order (POCL) Planning (80's): work on any subgoal, resolve threats; UCPOP 1992

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest
- Partial Order (POCL) Planning (80's): work on any subgoal, resolve threats; UCPOP 1992
- Graphplan (1995 - . . .): build graph containing all possible parallel plans up to certain length; then extract plan by searching the graph backward from G

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest
- Partial Order (POCL) Planning (80's): work on any subgoal, resolve threats; UCPOP 1992
- Graphplan (1995 - . . .): build graph containing all possible parallel plans up to certain length; then extract plan by searching the graph backward from G
- SatPlan (1996 - . . .): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest
- Partial Order (POCL) Planning (80's): work on any subgoal, resolve threats; UCPOP 1992
- Graphplan (1995 - ...): build graph containing all possible parallel plans up to certain length; then extract plan by searching the graph backward from G
- SatPlan (1996 - ...): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- Heuristic Search Planning (1996 - ...): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P

Computational Approaches to Classical Planning

- STRIPS algorithm (70's): Total ordering planning backward from G; work always on top subgoal in stack, delay rest
- Partial Order (POCL) Planning (80's): work on any subgoal, resolve threats; UCPOP 1992
- Graphplan (1995 - ...): build graph containing all possible parallel plans up to certain length; then extract plan by searching the graph backward from G
- SatPlan (1996 - ...): map planning problem given horizon into SAT problem; use state-of-the-art SAT solver
- Heuristic Search Planning (1996 - ...): search state space $\mathcal{S}(P)$ with heuristic function h extracted from problem P
- Model Checking Planning (1998 - ...): search state space $\mathcal{S}(P)$ with 'symbolic' BrFS where sets of states represented by formulas implemented by BDDs

Heuristic Search Planning

- Key development in planning in the 90's, is automatic extraction of heuristic functions to guide search for plans

Heuristic Search Planning

- Key development in planning in the 90's, is automatic extraction of heuristic functions to guide search for plans
- General idea: heuristics often explained as optimal cost functions of relaxed (simplified) problems (Pearl 83), approximating c^* the optimal cost function in Π

Heuristic Search Planning

- Key development in planning in the 90's, is automatic extraction of heuristic functions to guide search for plans
- General idea: heuristics often explained as optimal cost functions of relaxed (simplified) problems (Pearl 83), approximating c^* the optimal cost function in Π
- Most common relaxation in planning, Π^+, obtained by dropping delete-lists from ops in Π. If $c^*(\Pi)$ is optimal cost of Π, then

$$h^+(\Pi) \stackrel{\text{def}}{=} c^*(\Pi^+)$$

Heuristic Search Planning

- Key development in planning in the 90's, is automatic extraction of heuristic functions to guide search for plans
- General idea: heuristics often explained as optimal cost functions of relaxed (simplified) problems (Pearl 83), approximating c^* the optimal cost function in Π
- Most common relaxation in planning, Π^+, obtained by dropping delete-lists from ops in Π. If $c^*(\Pi)$ is optimal cost of Π, then

$$h^+(\Pi) \stackrel{\text{def}}{=} c^*(\Pi^+)$$

- Heuristic h^+ intractable but easy to approximate; i.e.
  - computing optimal plan for Π^+ is intractable, but
  - computing a non-optimal plan for Π^+ (relaxed plan) easy

Heuristic Search Planning

- Key development in planning in the 90's, is automatic extraction of heuristic functions to guide search for plans
- General idea: heuristics often explained as optimal cost functions of relaxed (simplified) problems (Pearl 83), approximating c^* the optimal cost function in Π
- Most common relaxation in planning, Π^+, obtained by dropping delete-lists from ops in Π. If $c^*(\Pi)$ is optimal cost of Π, then

$$h^+(\Pi) \stackrel{\text{def}}{=} c^*(\Pi^+)$$

- Heuristic h^+ intractable but easy to approximate; i.e.
  - computing optimal plan for Π^+ is intractable, but
  - computing a non-optimal plan for Π^+ (relaxed plan) easy
- Some state-of-the-art planners still rely on Π^+ ...

Heuristics for Classical Planning – Overview

Delete-relaxation

- h^+ (Hoffmann & Nebel, '01)
- h^{max} and h^{add} (Bonet & Geffner, '01)
- h^{FF} (Hoffmann & Nebel, '01)
- h^{pmax} (Mirkis & Domshlak, '07)
- h^{sa} (Keyder & Geffner, '08)
- Semi-Relaxed Plan Heuristics (Keyder et al., '12,'14; Haslum '13; Hoffman et al., '14)
- Red-black Planning Heuristics (Katz et al., '13a,b; Katz & Hoffman '14; Domshlak et al., '15; Gnad & Hoffman, '15; Speicher et al., '17; Katz '19; Fiser et al, '21)

Heuristics for Classical Planning – Overview

Delete-relaxation

- h^+ (Hoffmann & Nebel, '01)
- h^{max} and h^{add} (Bonet & Geffner, '01)
- h^{FF} (Hoffmann & Nebel, '01)
- h^{pmax} (Mirkis & Domshlak, '07)
- h^{sa} (Keyder & Geffner, '08)
- Semi-Relaxed Plan Heuristics (Keyder et al., '12,'14; Haslum '13; Hoffman et al., '14)
- Red-black Planning Heuristics (Katz et al., '13a,b; Katz & Hoffman '14; Domshlak et al., '15; Gnad & Hoffman, '15; Speicher et al., '17; Katz '19; Fiser et al, '21)

Critical path

- h^m (Haslum & Geffner, '00) with $h^1 \equiv h^{max}$

Heuristics for Classical Planning – Overview

Abstractions

- PDBs (Edelkamp, '01; Haslum et al., '05, '07)
- Merge & Shrink (Helmert et al., '07,'14; Katz et al, '12; Sievers et al., '14)
- Implicit Abstractions (Katz & Domshlak, '08, '10)
- Counterexample-guided Abstraction Refinement (CEGAR) (Seipp & Helmert, '13, '14, '18)

Heuristics for Classical Planning – Overview

Abstractions

- PDBs (Edelkamp, '01; Haslum et al., '05, '07)
- Merge & Shrink (Helmert et al., '07,'14; Katz et al, '12; Sievers et al., '14)
- Implicit Abstractions (Katz & Domshlak, '08, '10)
- Counterexample-guided Abstraction Refinement (CEGAR) (Seipp & Helmert, '13, '14, '18)

Landmarks

- Landmark count (Hoffmann et al., '04)
- h^L and h^{LA} (Karpas & Domshlak, '09)
- *LM-cut* (Helmert & Domshlak, '10)

Heuristics for Classical Planning – Overview

Abstractions

- PDBs (Edelkamp, '01; Haslum et al., '05, '07)
- Merge & Shrink (Helmert et al., '07,'14; Katz et al, '12; Sievers et al., '14)
- Implicit Abstractions (Katz & Domshlak, '08, '10)
- Counterexample-guided Abstraction Refinement (CEGAR) (Seipp & Helmert, '13, '14, '18)

Landmarks

- Landmark count (Hoffmann et al., '04)
- h^L and h^{LA} (Karpas & Domshlak, '09)
- *LM-cut* (Helmert & Domshlak, '10)

Potential Heuristics

- h^{pot} (Pommerening et al., '15; Seipp et al., '15)

Search Pruning Techniques

Partial Order Reduction (Alkhazraji et al., ’12; Wehrle et al., ’13; Wehrle & Helmert, ’14; Roeger et al., ’20)

- Exploit independence of operators
- Preserve at least one permutation of every plan

Search Pruning Techniques

Partial Order Reduction (Alkhazraji et al., '12; Wehrle et al., '13; Wehrle & Helmert, '14; Roeger et al., '20)

- Exploit independence of operators
- Preserve at least one permutation of every plan

Structural Symmetries (Pochter et al., '11; Domshlak et al., '12; Sievers et al., '19)

- Exploit search space automorphisms
- Find path in quotient system, translate to plan

Search Pruning Techniques

Partial Order Reduction (Alkhazraji et al., '12; Wehrle et al., '13; Wehrle & Helmert, '14; Roeger et al., '20)

- Exploit independence of operators
- Preserve at least one permutation of every plan

Structural Symmetries (Pochter et al., '11; Domshlak et al., '12; Sievers et al., '19)

- Exploit search space automorphisms
- Find path in quotient system, translate to plan

Novelty Pruning (Lipovetzky & Geffner, '12, '14, '17; Katz et al., '17; Tuisov & Katz, '21)

- Define a series of increasing size search spaces
- Search for plan in each, until one found

Search Pruning Techniques

Partial Order Reduction (Alkhazraji et al., '12; Wehrle et al., '13; Wehrle & Helmert, '14; Roeger et al., '20)

- Exploit independence of operators
- Preserve at least one permutation of every plan

Structural Symmetries (Pochter et al., '11; Domshlak et al., '12; Sievers et al., '19)

- Exploit search space automorphisms
- Find path in quotient system, translate to plan

Novelty Pruning (Lipovetzky & Geffner, '12, '14, '17; Katz et al., '17; Tuisov & Katz, '21)

- Define a series of increasing size search spaces
- Search for plan in each, until one found

Each of these techniques can lead to exponential reduction of the search space

International Planning Competition (IPC)

- How do you compare so many planners?

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008,

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL
- Multiple tracks: Deterministic, Uncertainty/Probabilistic, Temporal, Learning, Unsolvability, Hierarchical, ...

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL
- Multiple tracks: Deterministic, Uncertainty/Probabilistic, Temporal, Learning, Unsolvability, Hierarchical, . . .
- Multiple sub-tracks: cost-optimal, cost-bounded, satisficing, agile, . . .

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL
- Multiple tracks: Deterministic, Uncertainty/Probabilistic, Temporal, Learning, Unsolvability, Hierarchical, . . .
- Multiple sub-tracks: cost-optimal, cost-bounded, satisficing, agile, . . .
- Rules: (most tracks) participants submit their planners, then organizers choose the domains/instances and run all the submitted planners.

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL
- Multiple tracks: Deterministic, Uncertainty/Probabilistic, Temporal, Learning, Unsolvability, Hierarchical, ...
- Multiple sub-tracks: cost-optimal, cost-bounded, satisficing, agile, ...
- Rules: (most tracks) participants submit their planners, then organizers choose the domains/instances and run all the submitted planners.
- Winners get all the glory! (and some cash prizes)

International Planning Competition (IPC)

- How do you compare so many planners? Competitions!
- Run every ≈2 years: 1998, 2000, 2002, 2004, 2006, 2008, 2011, 2014, 2018, (possibly 2022)
- Unified language: PDDL
- Multiple tracks: Deterministic, Uncertainty/Probabilistic, Temporal, Learning, Unsolvability, Hierarchical, . . .
- Multiple sub-tracks: cost-optimal, cost-bounded, satisficing, agile, . . .
- Rules: (most tracks) participants submit their planners, then organizers choose the domains/instances and run all the submitted planners.
- Winners get all the glory! (and some cash prizes)
- Huge driver for research in planning for the last 20+ years.

Famous Planners: IPC Top Performers

- Cost-optimal:
  - FDSS (IPC'11)
  - SymBA* (IPC'14)
  - Delfi (IPC'18)
- Satisficing:
  - LAMA (IPC'08,11)
  - IBaCoP (IPC'14)
  - Mercury (IPC'14)
  - FDSS (IPC'18)
  - LAPKT-DUAL-BFWS (IPC'18)
- Probabilistic:
  - Prost (Since 2014)
  - Prost-DD (IPC'18)
  - Random-Bandit (IPC'18)
- Temporal:
  - YAHSP3-MT (IPC'14)
  - Temporal-FD (IPC'14)

Major Planning Toolkits/Systems/Families

- Fast-Forward (better known as FF): Classical satisficing, numeric, conformant, contingent planners (Hoffmann & Nebel, '01)
- Fast Downward: Classical cost-optimal, satisficing, agile, cost-bounded. Variants built on top of Fast Downward include: OSP, FOND, probabilistic, temporal, . . . (Helmert, '06)
- Lightweight Automated Planning ToolKiT (LAPKT): Classical cost-optimal, satisficing, agile (Ramirez, Lipovetzky, and Muise, '15)
- LPG: Classical satisficing, numeric, temporal, diverse, . . . (Gerevini & Serina, '02)
- SHOP2: HTN planning (Nau et al., '03)
- OPTIC: Temporal planning (Benton et al., '12)
- Many many more . . .

Non-IPC Planners and Tools

- Planning service (and docker) for cost-optimal, agile, satisficing, top-k, top-quality, diverse planning (by Katz et al.)
- Planner in the cloud, collection of tools and APIs (by Muise et al.)
- Forbid-Iterative Collection of planners for top-k, top-quality, diverse planning (Katz & Sohrabi, '20; Katz et al., '20a,'20b,'22)
- Top-k planners: K^* (Katz et al., '18) and SymK (Speck et al., '20)
- OSP planners (Katz et al., '19; Katz & Keyder, '19; Speck & Katz, '21)
- FOND planner PRP (Muise et al., '12,'14a,b)
- Pyperplan: Lightweight python-based planner developed for educational purposes (Alkhazraji et al., '20)

Other Major Community Efforts

- Slack workspace
- ICAPS web site
- Community GitHub
- PlanUtils: General library for setting up linux-based environments for developing, running, and evaluating planners
- Planning Wiki (initial effort), including a list of planners

All links