Non-classical Heuristics for Classical Planning

Erez Karpas

Advisors:
Carmel Domshlak Shaul Markovitch

Faculty of Industrial Engineering and Management,
Technion — Israel Institute of Technology

April 17, 2012

Outline

Domain Independent Planning

> "Planning is the art and practice of thinking before acting"
>
> Patrik Haslum

> "Planning is the model based approach to autonomous behavior"
>
> Hector Geffner

Domain Independent Planning

"Planning is the art and practice of thinking before acting"

Patrik Haslum

"Planning is the model based approach to autonomous behavior"

Hector Geffner

Domain Independent Planning

> “Planning is the art and practice of thinking **before acting**”
>
> Patrik Haslum

> “Planning is the **model based** approach to autonomous behavior”
>
> Hector Geffner

Domain Independent Planning

"Planning is the art and practice of thinking before acting"

Patrik Haslum

"Planning is the model based approach to autonomous behavior"

Hector Geffner

"Planning is just a way of avoiding figuring out what to do next"

Rodney Brooks

Domain Independent Planning Problems

- A domain independent planning problem contains:
  - Initial world state
  - Desired goal condition
  - Set of deterministic actions
- A solution is a sequence of actions:
  - Transforms the initial world state into a goal state
- We are interested in optimal planning:
  - Find (one of) the cheapest possible plans

STRIPS

- A STRIPS planning problem with action costs is a 5-tuple $\Pi = \langle P, s_0, G, A, C \rangle$
  - P is a set of boolean propositions
  - $s_0 \subseteq P$ is the initial state
  - $G \subseteq P$ is the goal
  - A is a set of actions.
  - Each action is a triple $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$
  - $C : A \rightarrow \mathbb{R}^{0+}$ assigns a cost to each action
- Applying action sequence $\overline{\rho} = \langle a_0, a_1, \ldots, a_n \rangle$ at state s leads to $s[[\overline{\rho}]]$
- The cost of action sequence $\overline{\rho}$ is $\sum_{i=0}^{n} C(a_i)$

Solving Planning Problems

- Several methods for solving planning problems exist:
  - Compilation into SAT or CP
  - Symbolic search
  - Bidirectional search
  - Heuristic forward search
- We focus on heuristic forward search
- We need heuristics, because the state space of a planning problem is huge

Search Problems

- A search problem contains:
  - Initial world state
  - Set of goal states
  - Set of deterministic actions
- A solution is a sequence of actions:
  - Transforms the initial world state into a goal state
- We are interested in optimal search:
  - Find (one of) the cheapest possible plans

Heuristic Forward Search

- It is easy to see that planning $\Rightarrow$ search
- Heuristic forward search:
  1. Maintains a list of candidate states (open list)
  2. At each iteration, a state is removed from the list
  3. If it is not a goal state, all of its successors are added to the list
- The choice of which state to remove usually involves a heuristic evaluation function
  - Evaluates the merit of each state

Outline

Heuristic Evaluation Functions

- A heuristic evaluation function estimates the distance from states to the goal
- Heuristics are sometimes defined as functions from states to non-negative numbers

Heuristic Evaluation Functions

- A heuristic evaluation function estimates the distance from states to the goal
- Heuristics are sometimes defined as functions from states to non-negative numbers. This is not general enough!

Heuristic Evaluation Functions

- A heuristic evaluation function estimates the distance from states to the goal
- Heuristics are sometimes defined as functions from states to non-negative numbers. This is not general enough!

"the promise of a node is estimated numerically by a **heuristic evaluation function** $f(n)$ which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

Judea Pearl, *Heuristics*, 1984

Information Sources for Heuristics

1. The description of the state
2. The description of the goal and any extra knowledge about the problem domain
3. The information gathered by the search up to that point

Information Sources for Heuristics

1. The description of the state
2. The description of the goal and any extra knowledge about the problem domain
   - In domain independent planning, this is the problem description in STRIPS
3. The information gathered by the search up to that point

Information Sources for Heuristics

1. The description of the state
2. The description of the goal and any extra knowledge about the problem domain
   - In domain independent planning, this is the problem description in STRIPS
3. The information gathered by the search up to that point
   - This is where the "usual" definition fails
   - We will focus on heuristics which exploit this information

Formal Framework for Heuristics

Search History

A sequence of states $\omega = \langle s_0, s_1, \ldots s_n \rangle$ is a possible search history of search problem ρ iff:

1. s_0 is the initial state of ρ
2. Every other state in the sequence is a successor of one of the previous states

The set of all possible search histories of ρ is denoted by $\mathscr{H}_\rho$
The set of all possible paths from the initial state is denoted by Γ

Heuristic Evaluation Function

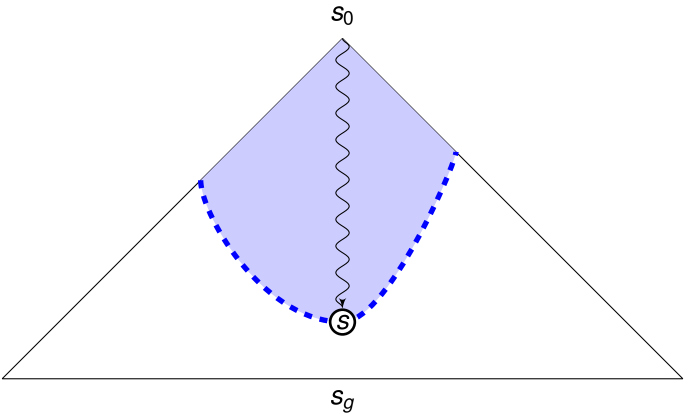
A heuristic evaluation function for search problem ρ is a function $h : \mathscr{H}_\rho \times \Gamma \rightarrow \mathbb{R}^{0+}$

Properties of Heuristics

Using our formal framework, we can discuss properties of heuristics:

1. Which information gathered by the search they use?
2. Are they admissible?

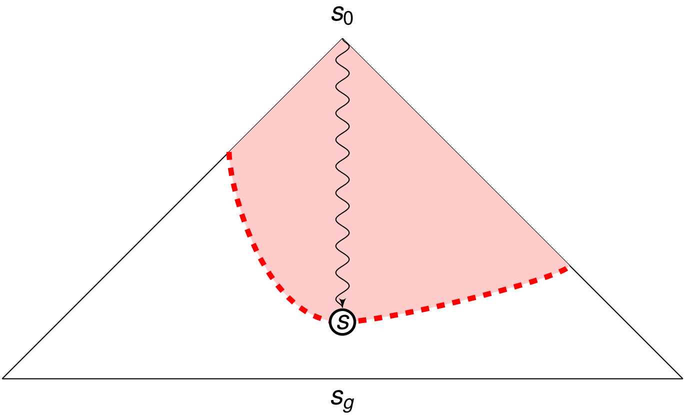
Taxonomy of Heuristics: Information Dependence



History Independent

A heuristic is history independent iff $h(\omega_1, \pi) = h(\omega_2, \pi)$ for any two search histories ω_1, ω_2 and any path π

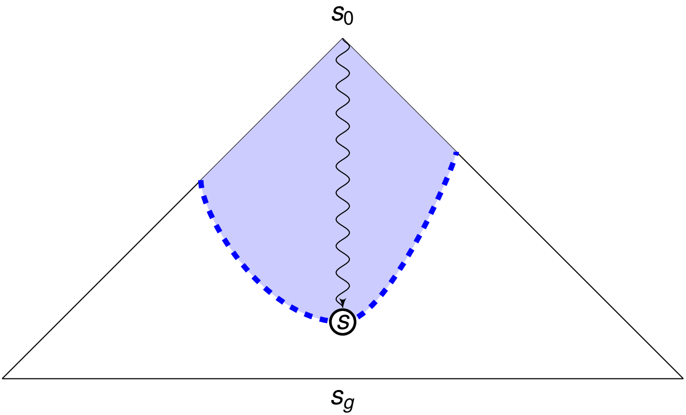
Taxonomy of Heuristics: Information Dependence



History Independent

A heuristic is history independent iff $h(\omega_1, \pi) = h(\omega_2, \pi)$ for any two search histories ω_1, ω_2 and any path π

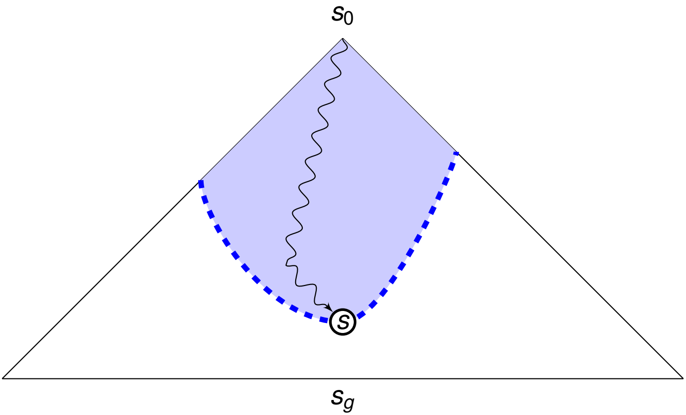
Taxonomy of Heuristics: Information Dependence



Path Independent

A heuristic is path independent iff $h(\omega, \pi_1) = h(\omega, \pi_2)$ for any two paths π_1, π_2 reaching the same state, and for any search history ω

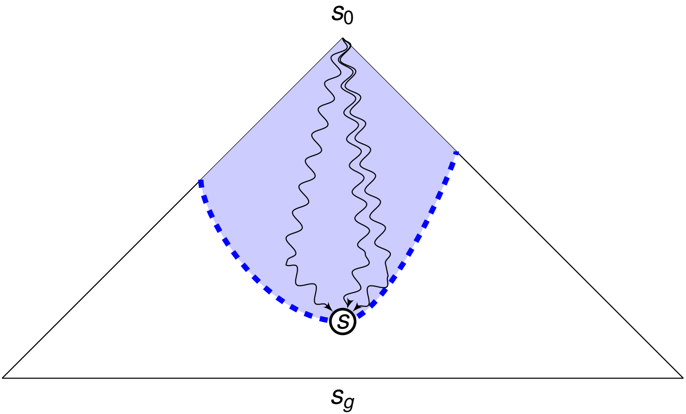
Taxonomy of Heuristics: Information Dependence



Path Independent

A heuristic is path independent iff $h(\omega, \pi_1) = h(\omega, \pi_2)$ for any two paths π_1, π_2 reaching the same state, and for any search history ω

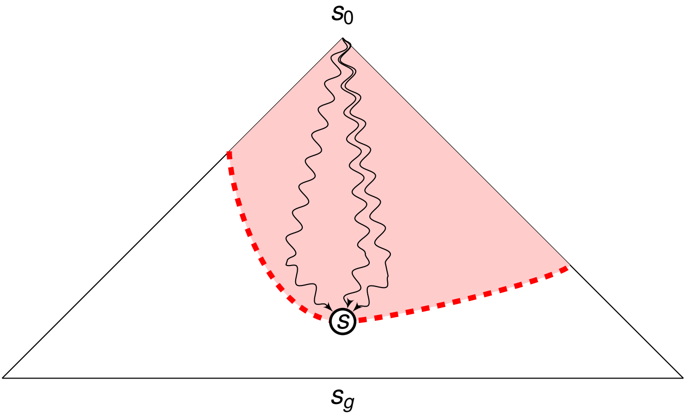
Taxonomy of Heuristics: Information Dependence



Special Case: Multi Path Dependent

A a path independent heuristic is multi path dependent iff $h(\omega_1, \pi) = h(\omega_2, \pi)$ for any two search histories ω_1, ω_2 such that the set of explored paths leading to s_0 $[[\pi]]$ is the same in ω_1 and ω_2

Taxonomy of Heuristics: Information Dependence



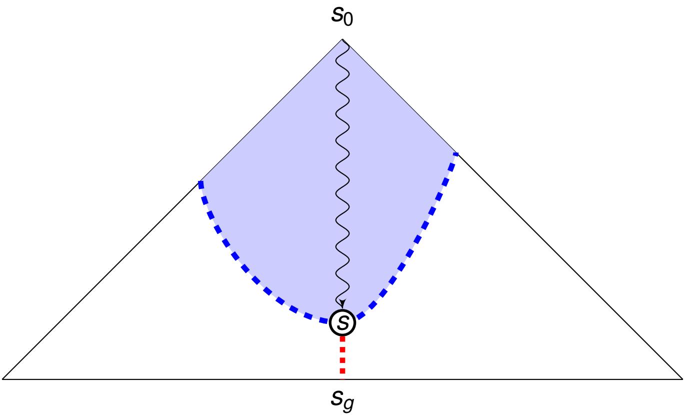
Special Case: Multi Path Dependent

A a path independent heuristic is multi path dependent iff $h(\omega_1, \pi) = h(\omega_2, \pi)$ for any two search histories ω_1, ω_2 such that the set of explored paths leading to s_0 $[[\pi]]$ is the same in ω_1 and ω_2

Information Dependence: Examples

		History	
		Independent	Dependent
Path	Independent	Classical	Selective Max, h_{LA} (multi-path)
	Dependent	$\exists$-opt landmarks h^{LM} (Richter et al. 2008)	Future work

Taxonomy of Heuristics: Admissibility



Admissible

A heuristic is admissible iff $h(\omega, \pi) \leq h^*(s_0[[\pi]])$ for any search history ω and any path π.

Optimality and Admissibility

- We know that A^* search with an admissible heuristic guarantees an optimal solution
- Is this a necessary condition?

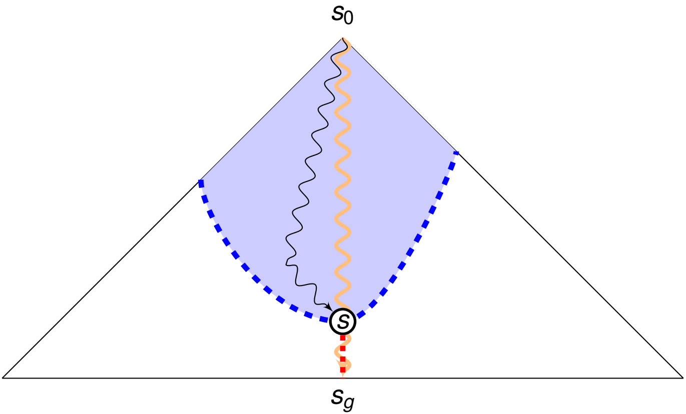
Optimality and Admissibility

- We know that A^* search with an admissible heuristic guarantees an optimal solution
- Is this a necessary condition?

Optimality and Admissibility

- We know that A^* search with an admissible heuristic guarantees an optimal solution
- Is this a necessary condition? No

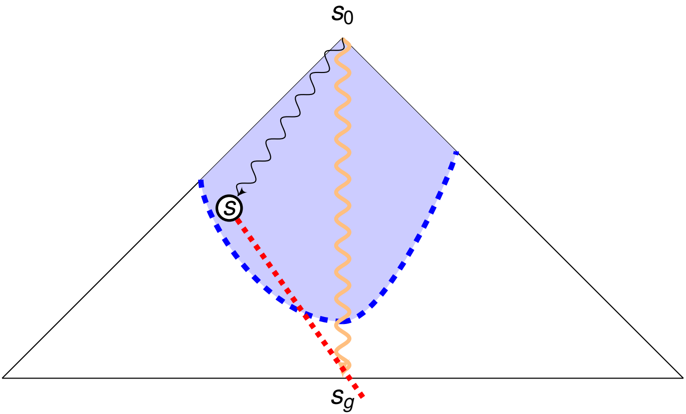
Global Admissibility



Globally Admissible

A heuristic is globally admissible iff there exists some optimal solution $\overline{\rho}$ such that for any state s along $\overline{\rho}$ any search history ω, and any path π to s: $h(\omega, \pi) \leq h^*(s)$.

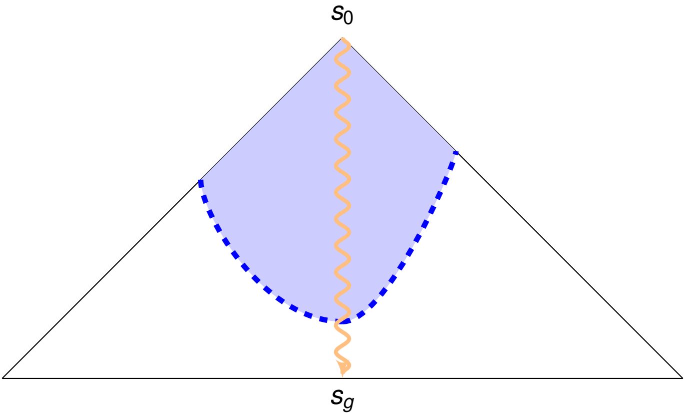
Global Admissibility



Globally Admissible

A heuristic is globally admissible iff there exists some optimal solution $\overline{\rho}$ such that for any state s along $\overline{\rho}$ any search history ω, and any path π to s: $h(\omega,\pi) \leq h^*(s)$.

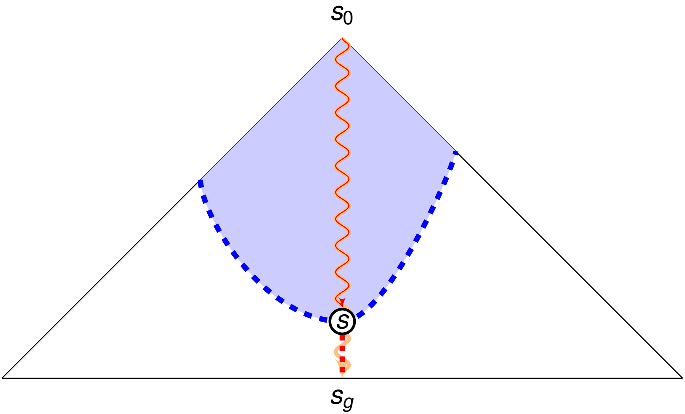
Global Path Admissibility



Globally Path Admissible

A heuristic is $\{\overline{\rho}\}$-admissible iff any search history ω and for any prefix π of $\overline{\rho}$: $h(\omega, \pi) \leq h^*(s_0[[\pi]])$

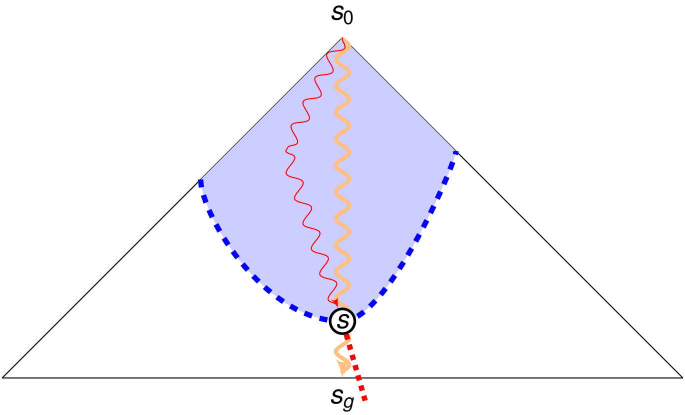
Global Path Admissibility



Globally Path Admissible

A heuristic is $\{\overline{\rho}\}$-admissible iff any search history ω and for any prefix π of $\overline{\rho}$: $h(\omega, \pi) \leq h^*(s_0[[\pi]])$

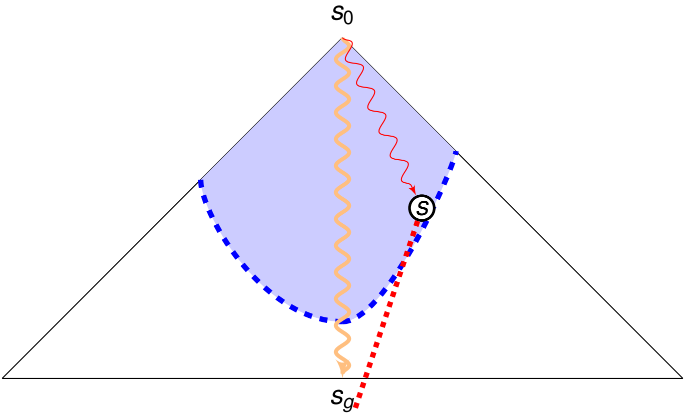
Global Path Admissibility



Globally Path Admissible

A heuristic is $\{\overline{\rho}\}$-admissible iff any search history ω and for any prefix π of $\overline{\rho}$: $h(\omega, \pi) \leq h^*(s_0[[\pi]])$

Global Path Admissibility



Globally Path Admissible

A heuristic is $\{\overline{\rho}\}$-admissible iff any search history ω and for any prefix π of $\overline{\rho}$: $h(\omega, \pi) \leq h^*(s_0[[\pi]])$

Search with Path-admissible Heuristics

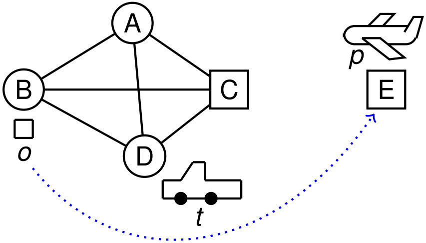
- Path-admissibility be generalized to a set of solutions χ
- If χ is the set of all optimal solutions, we call h path-admissible
- Using a path-admissible heuristic with A^* does not guarantee admissibility
- However, other search algorithms can guarantee an optimal solution is found with a path-admissible heuristic

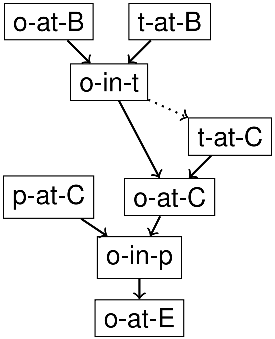
Outline

Landmarks

- A landmark is a formula that must be true at some point in every plan (Hoffmann, Porteous & Sebastia 2004)
- Landmarks can be (partially) ordered according to the order in which they must be achieved
- Some landmarks and orderings can be discovered automatically

Example Planning Problem - Logistics





Partial landmarks graph

(Example due to Silvia Richter)

Outline

Using Landmarks for Heuristic Estimates

- The number of landmarks that still need to be achieved is an (inadmissible) heuristic estimate (Richter, Helmert and Westphal 2008)
- Used by *LAMA* - winner of the IPC-2008 and IPC-2011 sequential satisficing track

- We assume that landmarks and orderings are discovered in a pre-processing phase, and the same landmark graph is used throughout the planning phase

Path-dependent Heuristics

- Suppose we are in state s. Did we achieve landmark ϕ yet?
- There is no way to tell just by looking at s
- Achieved landmarks are a function of path, not state
- The landmarks that still need to be achieved are path-dependent

The Landmark Heuristic

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = (L \setminus \text{Accepted}(s,\pi)) \cup \text{ReqAgain}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks — the landmarks which were achieved along π
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks — landmarks that must be achieved again according to a set of easy to check rules

The Landmark Heuristic

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = (L \setminus \mathrm{Accepted}(s,\pi)) \cup \mathrm{ReqAgain}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\mathrm{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks — the landmarks which were achieved along π
- $\mathrm{ReqAgain}(s,\pi) \subseteq \mathrm{Accepted}(s,\pi)$ is the set of *required again* landmarks — landmarks that must be achieved again according to a set of easy to check rules

The Landmark Heuristic

- The landmarks that still need to be achieved after reaching state s via path π are

$$L(s,\pi) = (L \setminus \text{Accepted}(s,\pi)) \cup \text{ReqAgain}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks — the landmarks which were achieved along π
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks — landmarks that must be achieved again according to a set of easy to check rules

The Landmark Heuristic

- The landmarks that still need to be achieved after reaching state s via path π are

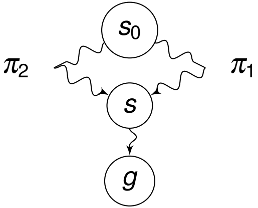
$$L(s,\pi) = (L \setminus \text{Accepted}(s,\pi)) \cup \text{ReqAgain}(s,\pi)$$

- L is the set of all (discovered) landmarks
- $\text{Accepted}(s,\pi) \subset L$ is the set of *accepted* landmarks — the landmarks which were achieved along π
- $\text{ReqAgain}(s,\pi) \subseteq \text{Accepted}(s,\pi)$ is the set of *required again* landmarks — landmarks that must be achieved again according to a set of easy to check rules

Admissible Landmark Heuristic

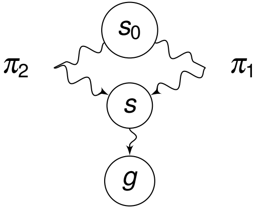
- Suppose we have a set of landmarks that need to be achieved $L(s, \pi)$
- We get an admissible heuristic by performing an action cost partitioning
    1. Partition the cost of each action between the landmarks it achieves
    2. Assign an admissible estimate (cost) for each landmark
    3. Sum over the costs of landmarks
- Admissibility follows from Katz and Domshlak (2010)

Multi-path Dependence



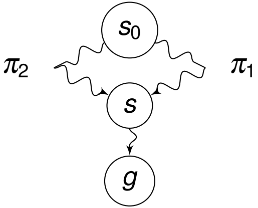
- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

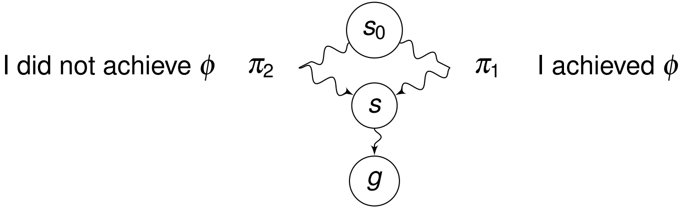
Multi-path Dependence



I achieved ϕ

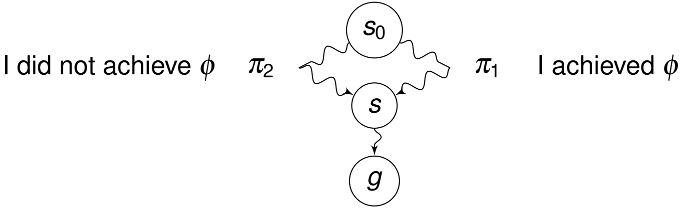
- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Multi-path Dependence



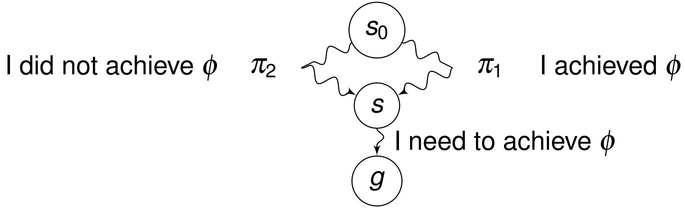
- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Multi-path Dependence



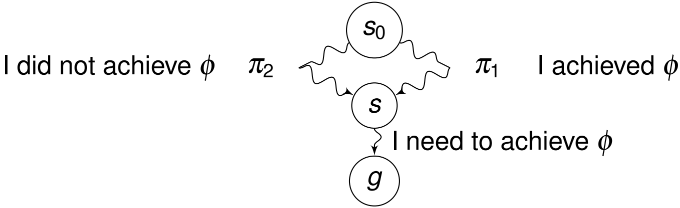
- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Multi-path Dependence



- Suppose state s was reached by paths π_1, π_2
- Suppose π_1 achieved landmark ϕ and π_2 did not
- Then ϕ needs to be achieved after state s
- Proof: ϕ is a landmark, therefore it needs to be true in all valid plans, including valid plans that start with π_2

Fusing Data from Multiple Paths

- Suppose $\mathscr{P}$ is a set of paths from s_0 to a state s. Define

$$L(s, \mathscr{P}) = (L \setminus \text{Accepted}(s, \mathscr{P})) \cup \text{ReqAgain}(s, \mathscr{P})$$

where

  - $\text{Accepted}(s, \mathscr{P}) = \bigcap_{\pi \in \mathscr{P}} \text{Accepted}(s, \pi)$
  - $\text{ReqAgain}(s, \mathscr{P}) \subseteq \text{Accepted}(s, \mathscr{P})$ is specified as before by s and the various rules

- $L(s, \mathscr{P})$ is the set of landmarks that we know still needs to be achieved after reaching state s via the paths in $\mathscr{P}$ (Karpas and Domshlak, 2009)

Outline

Intended Effects

Motivation

Why did the chicken cross the road?

To get to the other side

Intended Effects

Motivation

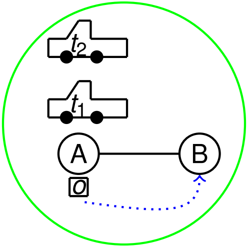
Why did the chicken cross the road?
To get to the other side

Observation

Every along action an optimal plan is there for a reason

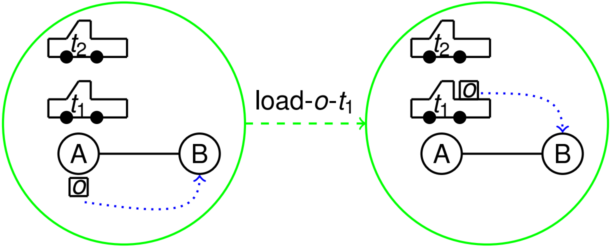
- Achieve a precondition for another action
- Achieve a goal

Intended Effects — Example



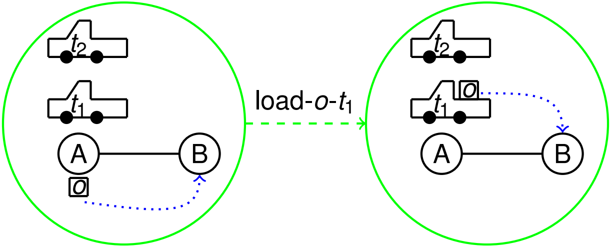
- There must be a reason for applying load-o-t_1
- load-o-t_1 achieves o-in-t_1
- Any continuation of this path to an optimal plan must use some action which requires o-in-t_1

Intended Effects — Example



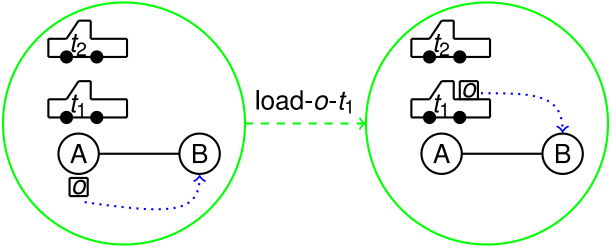
- There must be a reason for applying load-o-t_1
- load-o-t_1 achieves o-in-t_1
- Any continuation of this path to an optimal plan must use some action which requires o-in-t_1

Intended Effects — Example



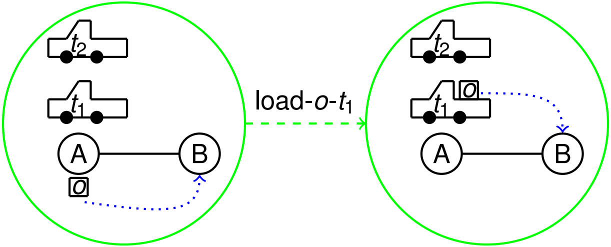
- There must be a reason for applying load-o-t_1
- load-o-t_1 achieves o-in-t_1
- Any continuation of this path to an optimal plan must use some action which requires o-in-t_1

Intended Effects — Example



- There must be a reason for applying load-o-t_1
- load-o-t_1 achieves o-in-t_1
- Any continuation of this path to an optimal plan must use some action which requires o-in-t_1

Intended Effects — Example



- There must be a reason for applying load-o-t_1
- load-o-t_1 achieves o-in-t_1
- Any continuation of this path to an optimal plan must use some action which requires o-in-t_1

Intended Effects — Intuition

- We formalize chicken logic using the notion of Intended Effects
- A set of propositions $X \subseteq s_0[[\pi]]$ is an intended effect of path π, if we can use X to continue π into an optimal plan
- Using X refers to the presence of causal links in the optimal plan

Causal Link

Let $\pi = \langle a_0, a_1, \ldots a_n \rangle$ be some path. The triple $\langle a_i, p, a_j \rangle$ forms a *causal link* in π if a_i is the actual provider of precondition p for a_j.

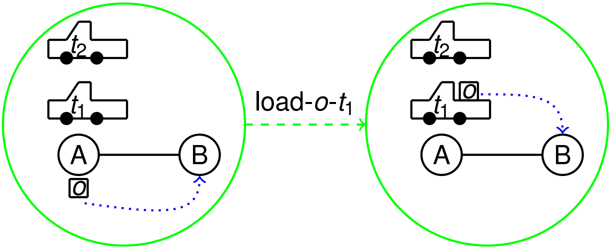
Intended Effects — Formal Definition

Intended Effects

Let $\textsc{opt}$ be a set of optimal plans for planning task Π. Given a path $\pi = \langle a_0, a_1, \ldots a_n \rangle$ a set of propositions $X \subseteq s_0[[\pi]]$ is an $\textsc{opt}$-intended effect of π iff there exists a path π' such that $\pi \cdot \pi' \in \textsc{opt}$ and π' consumes exactly X ($p \in X$ iff there is a causal link $\langle a_i, p, a_j \rangle$ in $\pi \cdot \pi'$, with $a_i \in \pi$ and $a_j \in \pi'$).

- $\mathrm{IE}(\pi|\textsc{opt})$ — the set of all $\textsc{opt}$-intended effect of π
- $\mathrm{IE}(\pi) = \mathrm{IE}(\pi|\textsc{opt})$ when $\textsc{opt}$ is the set of all optimal plans

Intended Effects — Set Example



The Intended Effects of $\pi = \langle \text{load-}o\text{-}t_1 \rangle$ are $\{\{o\text{-in-}t_1\}\}$

Intended Effects — It's Logical

- Working directly with the set of subsets $\mathrm{IE}(\pi|\mathrm{OPT})$ is difficult
- We can interpret $\mathrm{IE}(\pi|\mathrm{OPT})$ as a boolean formula ϕ

$$X \in \mathrm{IE}(\pi|\mathrm{OPT}) \iff X \models \phi$$

- We can also interpret any path π' from s_0 $[[\pi]]$ as a boolean valuation over propositions P

$$p = \mathrm{TRUE} \iff \text{there is a causal link } \langle a_i, p, a_j \rangle \text{ with } a_i \in \pi \text{ and } a_j \in \pi'$$

- Thus we can check if path $\pi' \models \phi$

Intended Effects — It's Logical

- Working directly with the set of subsets $\mathrm{IE}(\pi|\mathrm{OPT})$ is difficult
- We can interpret $\mathrm{IE}(\pi|\mathrm{OPT})$ as a boolean formula ϕ

$$X \in \mathrm{IE}(\pi|\mathrm{OPT}) \iff X \models \phi$$

- We can also interpret any path π' from s_0 $[[\pi]]$ as a boolean valuation over propositions P

$$p = \mathrm{TRUE} \iff \text{there is a causal link } \langle a_i, p, a_j \rangle \text{ with } a_i \in \pi \text{ and } a_j \in \pi'$$

- Thus we can check if path $\pi' \models \phi$

Intended Effects — It's Logical

- Working directly with the set of subsets $\text{IE}(\pi|\text{OPT})$ is difficult
- We can interpret $\text{IE}(\pi|\text{OPT})$ as a boolean formula ϕ

$$X \in \text{IE}(\pi|\text{OPT}) \iff X \models \phi$$

- We can also interpret any path π' from s_0 $[[\pi]]$ as a boolean valuation over propositions P

$$p = \text{TRUE} \iff \text{there is a causal link } \langle a_i, p, a_j \rangle \text{ with } a_i \in \pi \text{ and } a_j \in \pi'$$

- Thus we can check if path $\pi' \models \phi$

Intended Effects — It's Logical

- Working directly with the set of subsets $\mathrm{IE}(\pi|\textsc{opt})$ is difficult
- We can interpret $\mathrm{IE}(\pi|\textsc{opt})$ as a boolean formula ϕ

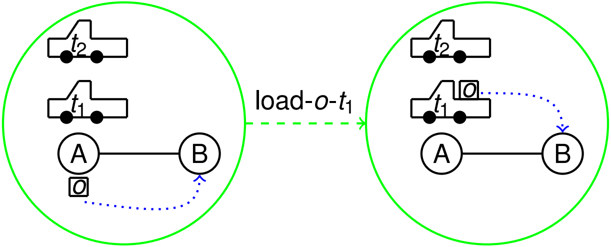
$$X \in \mathrm{IE}(\pi|\textsc{opt}) \Longleftrightarrow X \models \phi$$

- We can also interpret any path π' from s_0 $[[\pi]]$ as a boolean valuation over propositions P

$$p = \textsc{true} \Longleftrightarrow \text{there is a causal link } \langle a_i, p, a_j \rangle \text{ with } a_i \in \pi \text{ and } a_j \in \pi'$$

- Thus we can check if path $\pi' \models \phi$

Intended Effects — Formula Example



The Intended Effects of $\pi = \langle \text{load-}o\text{-}t_1 \rangle$ are described by the formula $\phi = o\text{-in-}t_1$

Intended Effects — What Are They Good For?

We can use a logical formula describing $\mathrm{IE}(\pi|\text{OPT})$ to derive constraints about what must happen in any continuation of π to a plan in OPT.

Theorem 1

Let OPT be a set of optimal plans for a planning task Π, π be a path, and ϕ be a propositional logic formula describing $\mathrm{IE}(\pi|\text{OPT})$. Then, for any $s_0[[\pi]]$-plan π', $\pi \cdot \pi' \in \text{OPT}$ implies $\pi' \models \phi$.

Intended Effects — The Bad News

It's P-SPACE Hard to find the intended effects of path π.

Theorem 2

Let INTENDED be the following decision problem: Given a planning task Π, a path π, and a set of propositions $X \subseteq P$, is $X \in \mathrm{IE}(\pi)$? Deciding INTENDED is P-SPACE Complete.

Approximate Intended Effects — The Good News

We can use supersets of $\mathrm{IE}(\pi|\textsc{opt})$ to derive constraints about any continuation of π.

Theorem 3

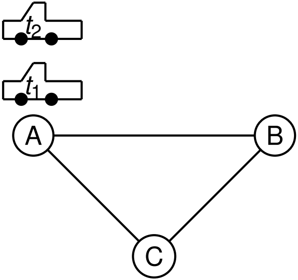
Let $\textsc{opt}$ be a set of optimal plans for a planning task Π, π be a path, $\mathrm{PIE}(\pi|\textsc{opt}) \supseteq \mathrm{IE}(\pi|\textsc{opt})$ be a set of possible $\textsc{opt}$-intended effects of π, and ϕ be a logical formula describing $\mathrm{PIE}(\pi|\textsc{opt})$. Then, for any path π' from $s_0[[\pi]]$, $\pi \cdot \pi' \in \textsc{opt}$ implies $\pi' \models \phi$.

Finding Approximate Intended Effects — Shortcuts

- Intuition: X can not be an intended effect of π if there is a cheaper way to achieve X

- Assume we have some library $\mathscr{L}$ of "shortcut" paths
- $X \subseteq s_0[[\pi]]$ can not be an intended effect of π if there exists some $\pi' \in \mathscr{L}$ such that:
  1. $C(\pi') < C(\pi)$
  2. $X \subseteq s_0[[\pi']]$

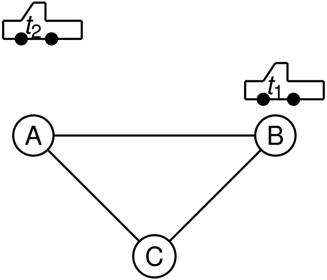
Shortcuts Example



Causal Structure

$\pi = \langle \qquad \rangle$

Shortcuts Example

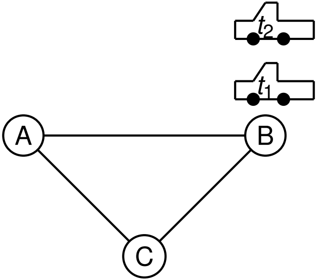


Causal Structure

drive-t_1-A-B

$$\pi = \langle \text{ drive-}t_1\text{-}A\text{-}B \qquad\qquad\qquad\qquad \rangle$$

Shortcuts Example

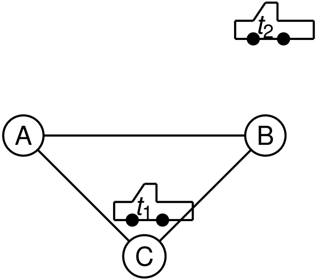


Causal Structure

drive-t_1-A-B drive-t_2-A-B

$\pi = \langle$ drive-t_1-A-B ,drive-t_2-A-B $\rangle$

Shortcuts Example



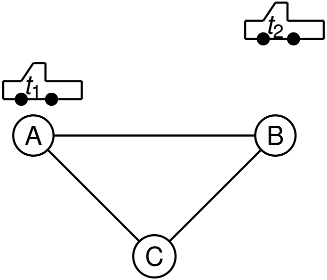
Causal Structure

drive-t_1-A-B → drive-t_1-B-C

drive-t_2-A-B

$$\pi = \langle \text{ drive-}t_1\text{-}A\text{-}B\text{ ,drive-}t_2\text{-}A\text{-}B\text{ ,drive-}t_1\text{-}B\text{-}C \qquad\qquad \rangle$$

Shortcuts Example



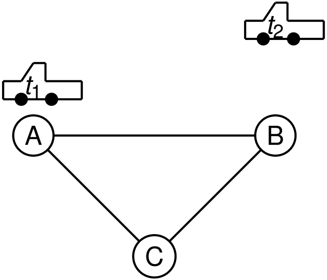
Causal Structure

drive-t_1-A-B → drive-t_1-B-C → drive-t_1-C-A

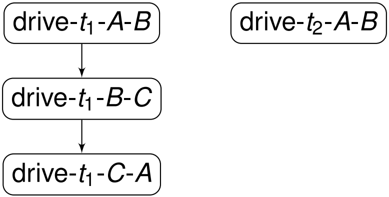
drive-t_2-A-B

$$\pi = \langle \text{ drive-}t_1\text{-}A\text{-}B\text{ ,drive-}t_2\text{-}A\text{-}B\text{ ,drive-}t_1\text{-}B\text{-}C\text{ ,drive-}t_1\text{-}C\text{-}A \rangle$$

Shortcuts Example



Causal Structure



$\pi = \langle$ drive-t_1-A-B ,drive-t_2-A-B ,drive-t_1-B-C ,drive-t_1-C-A $\rangle$

$\pi' = \langle$drive-t_2-A-$B\rangle$

Shortcuts in Logic Form

- For $X \subseteq s_0[[\pi]]$ to be an intended effect of π, it must achieve something that no shortcut does
- Expressed as a CNF formula:

$$\phi_{\mathscr{L}}(\pi) = \bigwedge_{\pi' \in \mathscr{L} : C(\pi') < C(\pi)} \bigvee_{p \in s_0[[\pi]] \setminus s_0[[\pi']]} p$$

- Each clause of this formula stands for an existential optimal disjunctive action landmark: There must exist some action in some optimal continuation that consumes one of its propositions

Finding Shortcuts

- Where does the shortcut library $\mathscr{L}$ come from?
- It does not need to be static — it can be dynamically generated for each path
- We use the causal structure of the current path — a graph whose nodes are actions, with an edge from a_i to a_j if there is a causal link where a_i provides some proposition for a_j
- We attempt to remove parts of the causal structure, to obtain a "shortcut"

Shortcuts as Landmarks

- The formula $\phi_{\mathscr{L}}(\pi)$ describes $\exists$-opt landmarks — landmarks which occur in some optimal plan
- We can incorporate those landmarks with “regular” landmarks, and derive a heuristic using the cost partitioning method
- The resulting heuristic is path admissible
- To guarantee optimality, we modify A^* to reevaluate $h(s)$ every time a cheaper path to s is found

Outline

Motivation

- We want to do domain independent optimal planning, in a time-bounded setting
- Use A^*

Motivation

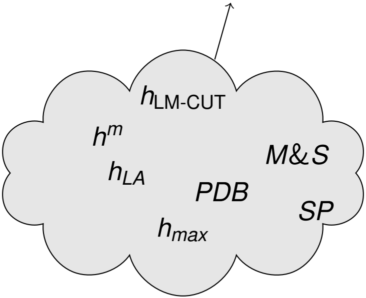
- We want to do domain independent optimal planning, in a time-bounded setting
- Use A^*

$$f = g + h$$

Motivation

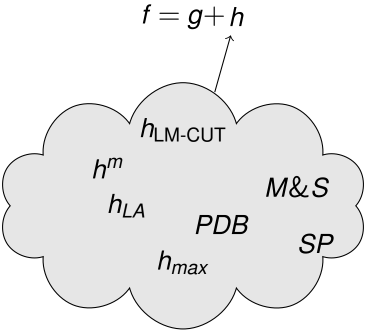
- We want to do domain independent optimal planning, in a time-bounded setting
- Use A^*

$$f = g + h$$



Motivation

- We want to do domain independent optimal planning, in a time-bounded setting
- Use A^*



Which heuristic is the best?

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic
- Sample results:

Domain	h_{LA}	$h_{\text{LM-CUT}}$	$\max_h$
airport	25	**38**	36
freecell	**28**	15	22

Number of problems solved in 30 minutes

Why Settle for One?

- There is no single best heuristic, so why settle only for one?
- We can use the maximum of several heuristics to get a more informative heuristic
- Sample results:

Domain	h_{LA}	$h_{\text{LM-CUT}}$	$\max_h$
airport	25	**38**	36
freecell	**28**	15	22

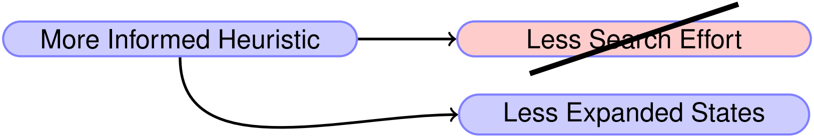
Number of problems solved in 30 minutes

- A more informed heuristic solves less problems — something is rotten in the kingdom of A^*

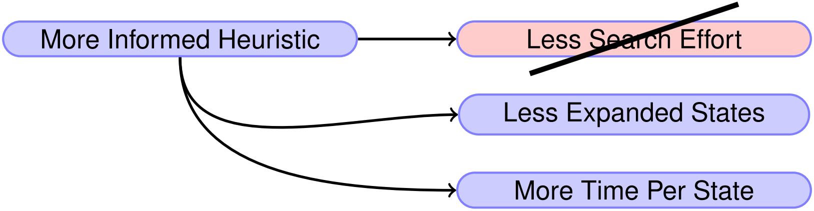
The Accuracy / Computation Time Tradeoff

More Informed Heuristic $\rightarrow$ Less Search Effort

The Accuracy / Computation Time Tradeoff



The Accuracy / Computation Time Tradeoff



The Accuracy / Computation Time Tradeoff

More Informed Heuristic → ~~Less Search Effort~~

More Informed Heuristic → Less Expanded States

$$t_{max_h} = t_{h_{LA}} + t_{h_{\text{LM-CUT}}}$$ → More Time Per State

The Accuracy / Computation Time Tradeoff

More Informed Heuristic

~~Less Search Effort~~

Less Expanded States

$t_{max_h} = t_{h_{LA}} + t_{h_{\text{LM-CUT}}}$

More Time Per State

Conclusion

A more informed heuristic is not necessarily better

A Simple Observation

- So how can we benefit from multiple heuristics?
- Simple observation: the maximum of several heuristics — is simply the value of one of those heuristics
- This leads to the following idea:
  - Given state s, and heuristics $\{h_1, \ldots, h_n\}$
  - Choose $h_i = \text{ORACLE}(s, \{h_1, \ldots, h_n\})$
  - Compute only $h_i(s)$

A Simple Observation

- So how can we benefit from multiple heuristics?
- Simple observation: the maximum of several heuristics — is simply the value of one of those heuristics
- This leads to the following idea:
  - Given state s, and heuristics $\{h_1, \ldots, h_n\}$
  - Choose $h_i = \text{ORACLE}(s, \{h_1, \ldots, h_n\})$
  - Compute only $h_i(s)$

A Simple Observation

- So how can we benefit from multiple heuristics?
- Simple observation: the maximum of several heuristics — is simply the value of one of those heuristics
- This leads to the following idea:
  - Given state s, and heuristics $\{h_1, \ldots, h_n\}$
  - Choose $h_i = \text{ORACLE}(s, \{h_1, \ldots, h_n\})$
  - Compute only $h_i(s)$

The Oracle

- How do we define ORACLE?
  - Naive answer: use the heuristic which gives the maximum value

$$\text{ORACLE}(s, \{h_1, \ldots, h_n\}) = \operatorname*{argmax}_i h_i(s)$$

  - Why is this naive?
  - Because sometimes the extra time to compute the most informed heuristic is not worth it
  - Example: $h_{\text{LM-CUT}}$ is about 9.4 times slower than h_{LA}

The Oracle

- How do we define ORACLE?
  - Naive answer: use the heuristic which gives the maximum value

$$\text{ORACLE}(s, \{h_1, \ldots, h_n\}) = \operatorname*{argmax}_i h_i(s)$$

  - Why is this naive?
  - Because sometimes the extra time to compute the most informed heuristic is not worth it
  - Example: $h_{\text{LM-CUT}}$ is about 9.4 times slower than h_{LA}

The Oracle

- How do we define ORACLE?
  - Naive answer: use the heuristic which gives the maximum value

$$\text{ORACLE}(s, \{h_1, \ldots, h_n\}) = \operatorname*{argmax}_i h_i(s)$$

  - Why is this naive?

  - Because sometimes the extra time to compute the most informed heuristic is not worth it
  - Example: $h_{\text{LM-CUT}}$ is about 9.4 times slower than h_{LA}

The Oracle

- How do we define ORACLE?
  - Naive answer: use the heuristic which gives the maximum value

$$\text{ORACLE}(s, \{h_1, \ldots, h_n\}) = \operatorname*{argmax}_i h_i(s)$$

  - Why is this naive?
  - Because sometimes the extra time to compute the most informed heuristic is not worth it
  - Example: $h_{\text{LM-CUT}}$ is about 9.4 times slower than h_{LA}

The Oracle

- How do we define ORACLE?
  - Naive answer: use the heuristic which gives the maximum value

$$\text{ORACLE}(s, \{h_1, \dots, h_n\}) = \operatorname*{argmax}_i h_i(s)$$

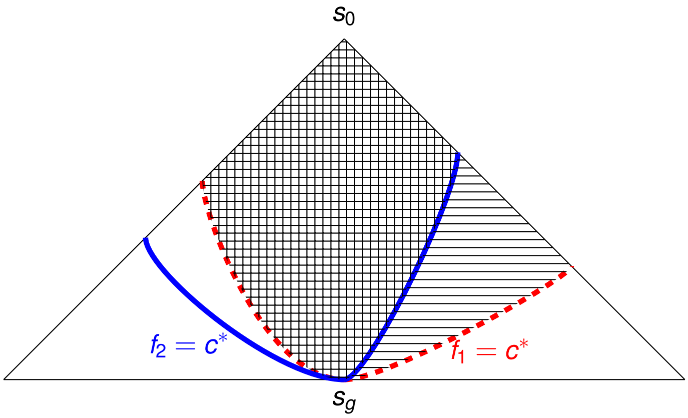
  - Why is this naive?
  - Because sometimes the extra time to compute the most informed heuristic is not worth it
  - Example: $h_{\text{LM-CUT}}$ is about 9.4 times slower than h_{LA}

Selective Max

- Develop a theoretical model for determining which heuristic is best to compute at each state, in order to minimize search time
- Derive a decision rule from the model, which is used as a target concept for a classifier
- Describe an online learning scheme which uses this classifier during search

Theoretical Model

- We will not go into the details



Decision Rule

- From our theoretical model, we get the following decision rule:

Decision Rule

$$\text{Compute } h_2 \iff h_2 - h_1 > \alpha \log_b(t_2/t_1)$$

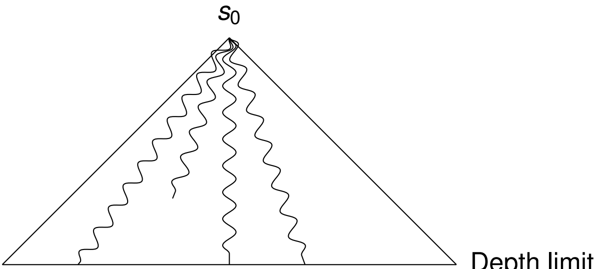
- h_1, h_2 are the heuristics
- t_1, t_2 are their respective computation times
- WLOG $t_2 \geq t_1$
- b is the branching factor
- α is a hyper parameter

Learning

- Pre-search:
  - Collecting training examples
  - Labeling training examples
  - Generating features
  - Building a classifier
- During search:
  - Classification
  - Active learning

Collecting Training Examples

- State space is sampled by performing random walks
- Several sampling procedures available
- The exact details are not important



Labeling Training Examples

- b, t_1, t_2 are estimated from the collected examples
- $h_2 - h_1$ is calculated for each state
- Each example is labeled by h_2 iff $h_2 - h_1 > \alpha \log_b(t_2/t_1)$

Generating Features

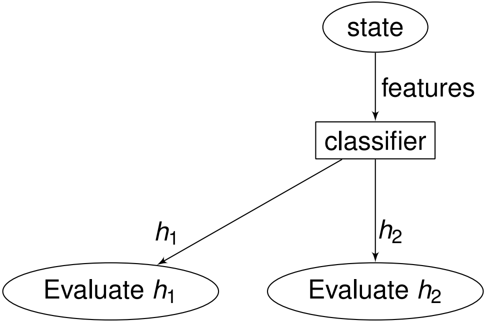
- We perform online learning, for a specific problem, so we do not need to generalize across problems
- This allows us to use features which fully describe each state
- We use the simplest features — just values of state variables

Building a Classifier

- We use the Naive Bayes classifier
  - Very fast
  - Incremental — can be updated quickly on the fly
  - Provides probability distribution for classification

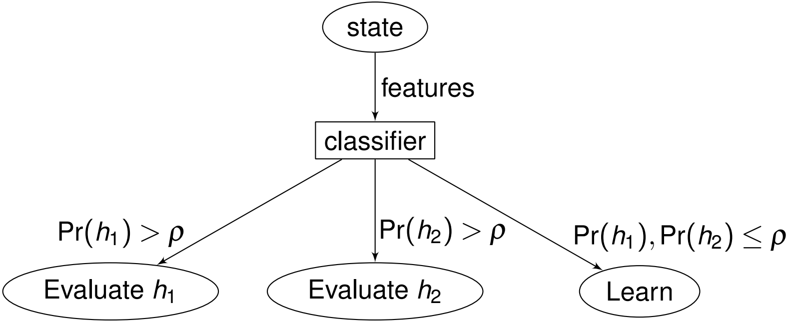
Using the classifier

State Evaluation



Using the classifier

State Evaluation



Selective Max Conclusion

- This is an active online learning scheme
- This approach can be easily extended to multiple heuristics
  - Learn a classifier for each pair
  - Decide which heuristic to use by voting
- The resulting heuristic is history dependent — the order in which all previous states are encountered matters

Outline

Conclusion

- Presented a formal framework for defining
  - State–, path–, multi path–, and history-dependent heuristics
  - Consistent, admissible, globally admissible heuristics
  - Path-admissible heuristics
- Presented path– and multi path-dependent landmark heuristics
- Presented path-admissible $\exists$-opt landmark heuristic
- Presented history-dependent heuristic combination selective max

Bottom Line

Even if you're doing classical planning, you're not limited to classical heuristics