Potential Heuristics in Satisficing Planning

Alexander Rovner

University of Basel

February 12, 2020

Classical Planning

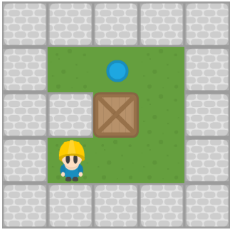
SAS$^+$ Planning Task $\Pi = \langle V, I, \gamma, O \rangle$:

Classical Planning

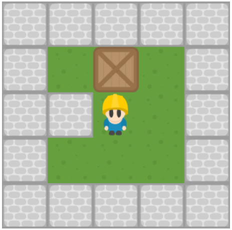
SAS$^+$ Planning Task $\Pi = \langle V, I, \gamma, O \rangle$:
state variables $V = \{\texttt{player-pos}, \texttt{box-pos}\}$

Classical Planning

SAS$^+$ Planning Task $\Pi = \langle V, I, \gamma, O \rangle$:
state variables $V = \{\texttt{player-pos}, \texttt{box-pos}\}$



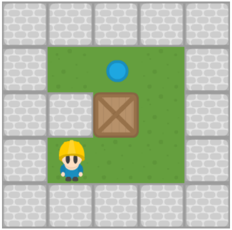
initial state I



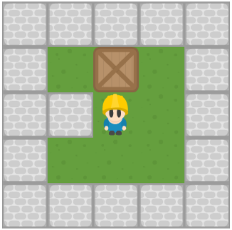
goal state $s_\star \supseteq \gamma$

Classical Planning

SAS$^+$ Planning Task $\Pi = \langle V, I, \gamma, O \rangle$:
state variables $V = \{\texttt{player-pos}, \texttt{box-pos}\}$



initial state I

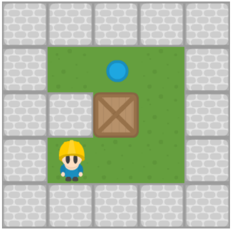


goal state $s_\star \supseteq \gamma$

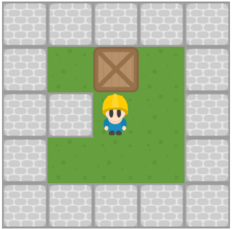
set of operators O, where each $o \in O$ has a *precondition*, *effect*, and a *cost*

Classical Planning

SAS$^+$ Planning Task $\Pi = \langle V, I, \gamma, O \rangle$:
state variables $V = \{\texttt{player-pos}, \texttt{box-pos}\}$



initial state I



goal state $s_\star \supseteq \gamma$

set of operators O, where each $o \in O$ has a *precondition*, *effect*, and a *cost*

Goal: find a sequence of actions that transforms I into a goal state

Potential Heuristics

- Task induces a graph called *transition system/state space*.
- Use search algorithm (e.g. A*, GBFS) to find a path from the initial state to some goal state.
- Search algorithms are guided towards the goal by *heuristic functions*.

Potential Heuristics

- Task induces a graph called *transition system/state space*.
- Use search algorithm (e.g. A*, GBFS) to find a path from the initial state to some goal state.
- Search algorithms are guided towards the goal by *heuristic functions*.
- In this thesis: **potential heuristics**.

Potential Heuristics

Definition: Potential Heuristics

Linear combination of features $F \in \mathcal{F}$ that are present in the given state s:

$$h^{\text{pot}}(s) := \sum_{F \in \mathcal{F}} w(F)[F \subseteq s]$$

where $w(F)$ is the weight of feature F and F is a set of facts.

Potential Heuristics

Definition: Potential Heuristics

Linear combination of features $F \in \mathcal{F}$ that are present in the given state s:

$$h^{\text{pot}}(s) := \sum_{F \in \mathcal{F}} w(F)[F \subseteq s]$$

where $w(F)$ is the weight of feature F and F is a set of facts.

Central Question: how to select weights $w(F)$ for each $F \in \mathcal{F}$?

Potential Heuristics

Definition: Potential Heuristics

Linear combination of features $F \in \mathcal{F}$ that are present in the given state s:

$$h^{\mathrm{pot}}(s) := \sum_{F \in \mathcal{F}} w(F)[F \subseteq s]$$

where $w(F)$ is the weight of feature F and F is a set of facts.

Central Question: how to select weights $w(F)$ for each $F \in \mathcal{F}$?

- In Optimal Planning: choose $w(F)$ such that h^{pot} is admissible

Potential Heuristics

Definition: Potential Heuristics

Linear combination of features $F \in \mathcal{F}$ that are present in the given state s:

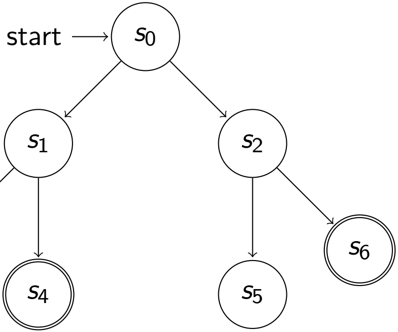
$$h^{\text{pot}}(s) := \sum_{F \in \mathcal{F}} w(F)[F \subseteq s]$$

where $w(F)$ is the weight of feature F and F is a set of facts.

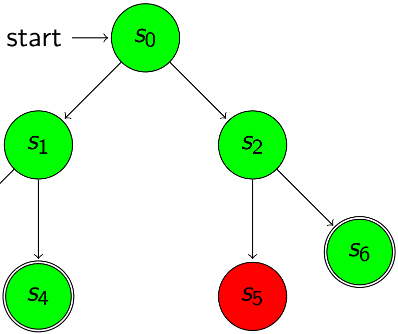
Central Question: how to select weights $w(F)$ for each $F \in \mathcal{F}$?

- In Optimal Planning: choose $w(F)$ such that h^{pot} is admissible
- In Satisficing Planning: we focus on heuristics that are **descending** and **dead-end avoiding** (DDA)

DDA Heuristics

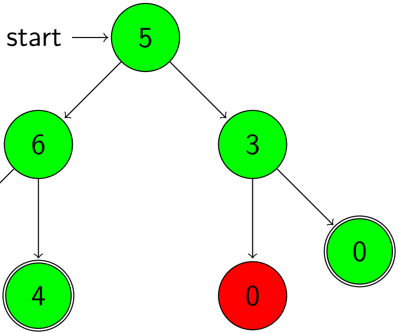


DDA Heuristics



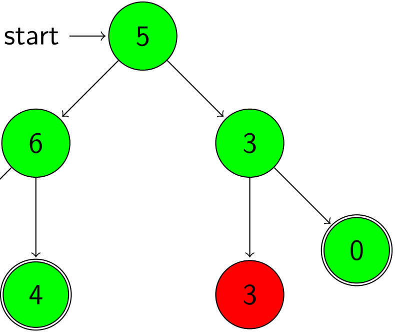
States that are **reachable** and **solvable** are called **alive**.

DDA Heuristics



A heuristic is **descending** if every alive non-goal state has an improving successor.

DDA Heuristics



A heuristic is **dead-end avoiding** if only alive successors are improving.

Complexity of Computing DDA Heuristics

Central Question: How hard is it to come up with a DDA heuristic?

Complexity of Computing DDA Heuristics

Central Question: How hard is it to come up with a DDA heuristic?

Definition: IsDDA decision problem

GIVEN: heuristic h and task Π

QUESTION: is h DDA in task Π?

Complexity of Computing DDA Heuristics

Central Question: How hard is it to come up with a DDA heuristic?

Definition: IsDDA decision problem

GIVEN: heuristic h and task Π

QUESTION: is h DDA in task Π?

Claim

IsDDA is a PSPACE-complete problem.

Complexity of Computing DDA Heuristics

Central Question: How hard is it to come up with a DDA heuristic?

Definition: IsDDA decision problem

Given: heuristic h and task Π
Question: is h DDA in task Π?

Claim

IsDDA is a PSPACE-complete problem.

Proof idea: show that NotDDA (complement of IsDDA) is PSPACE-complete and use the fact that PSPACE=coPSPACE.

PSPACE-hardness of NotDDA

Key Observations

1. If task Π is unsolvable then it has no alive states.
2. In tasks without alive states, any heuristic is DDA.

Proof: NotDDA is PSPACE-hard

Reduction from PlanEx: given task Π...

PSPACE-hardness of NotDDA

Key Observations

1. If task Π is unsolvable then it has no alive states.
2. In tasks without alive states, any heuristic is DDA.

Proof: NotDDA is PSPACE-hard

Reduction from PlanEx: given task Π...

- construct a heuristic that is never DDA (e.g. $\hat{h}(s) = 0 \ \forall s$)

PSPACE-hardness of NotDDA

Key Observations

1. If task Π is unsolvable then it has no alive states.
2. In tasks without alive states, any heuristic is DDA.

Proof: NotDDA is PSPACE-hard

Reduction from PlanEx: given task Π...

- construct a heuristic that is never DDA (e.g. $\hat{h}(s) = 0\ \forall s$)
- $\Pi \in$ PlanEx iff $\langle \Pi, \hat{h} \rangle \in$ NotDDA.

PSPACE-hardness of NotDDA

Key Observations

1. If task Π is unsolvable then it has no alive states.
2. In tasks without alive states, any heuristic is DDA.

Proof: NotDDA is PSPACE-hard

Reduction from PlanEx: given task Π...

- construct a heuristic that is never DDA (e.g. $\hat{h}(s) = 0 \ \forall s$)
- $\Pi \in$ PlanEx iff $\langle \Pi, \hat{h} \rangle \in$ NotDDA.
- $\Pi \notin$ PlanEx iff $\langle \Pi, \hat{h} \rangle \notin$ NotDDA.

PSPACE-membership of NotDDA

PSPACE algorithm sketch

For each state s of the planning task:

1. if s is not alive $\Rightarrow$ **continue**
2. for all successors s' of s:
   1. if s' is not alive and $h(s') < h(s) \Rightarrow$ **accept**
3. if there exists no s' with $h(s') < h(s) \Rightarrow$ **accept**

otherwise **fail**

PSPACE-membership of NotDDA

PSPACE algorithm sketch

For each state s of the planning task:

1. if s is not alive $\Rightarrow$ **continue**
2. for all successors s' of s:
   1. if s' is not alive and $h(s') < h(s) \Rightarrow$ **accept**
3. if there exists no s' with $h(s') < h(s) \Rightarrow$ **accept**

otherwise **fail**

DDA computation is as hard as planning itself!

$\Rightarrow$ Need approximation algorithms.

Naive Approach

Naive Approach: compute weights by solving a MIP model.

Naive Approach

Naive Approach: compute weights by solving a MIP model.

$$\begin{aligned} \min \quad & 0 && (1) \\ \text{s.t.} \quad & \bigvee_{s' \in succ(s)} h(s') + 1 \leq h(s) \quad \text{for } s \in S_A && (2) \\ & h(s') \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D && (3) \end{aligned}$$

S_A: set of all alive states
T_D: set of all transitions from an alive state to an unsolvable one

Naive Approach

Naive Approach: compute weights by solving a MIP model.

$$\min \quad 0 \tag{1}$$

$$\text{s.t.} \quad \bigvee_{s' \in succ(s)} h(s') + 1 \leq h(s) \quad \text{for } s \in S_A \tag{2}$$

$$h(s') \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D \tag{3}$$

S_A: set of all alive states

T_D: set of all transitions from an alive state to an unsolvable one

Problem: Solver usually fails to find an initial solution.

$\Rightarrow$ Add slack variables to the model.

Naive Approach

MIP model with slack variables:

$$\min \quad \sum_{s \in S_A} \alpha_s + \sum_{\langle s,s' \rangle \in T_D} \beta_{(s,s')} \tag{4}$$

$$\text{s.t.} \quad \bigvee_{s' \in succ(s)} h(s') + 1 - \alpha_s \leq h(s) \quad \text{for } s \in S_A \tag{5}$$

$$h(s') + \beta_{\langle s,s' \rangle} \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D \tag{6}$$

$$\alpha_s \geq 0 \quad \text{for } s \in S_A \tag{7}$$

$$\beta_{\langle s,s' \rangle} \geq 0 \quad \text{for } \langle s, s' \rangle \in T_D \tag{8}$$

Naive Approach

MIP model with slack variables:

$$\min \sum_{s \in S_A} \alpha_s + \sum_{\langle s,s' \rangle \in T_D} \beta_{(s,s')} \tag{4}$$

$$\text{s.t.} \bigvee_{s' \in succ(s)} h(s') + 1 - \alpha_s \leq h(s) \quad \text{for } s \in S_A \tag{5}$$

$$h(s') + \beta_{\langle s,s' \rangle} \geq h(s) \quad \text{for } \langle s, s' \rangle \in T_D \tag{6}$$

$$\alpha_s \geq 0 \quad \text{for } s \in S_A \tag{7}$$

$$\beta_{\langle s,s' \rangle} \geq 0 \quad \text{for } \langle s, s' \rangle \in T_D \tag{8}$$

- Simple first solution: assign large values to all α and β
- Can stop MIP solver early and work with an approximation.
- Problem: this does not scale!

Forward-Sampling

- Simple Alternative: construct the same MIP over a random *subset* of all states.
- Main Question: how to generate the subset?
  $\Rightarrow$ perform a random walk starting in the initial state
- The sample will only contain reachable states
  $\Rightarrow$ can only *assume* that they are also solvable

Backward-Sampling

- Can also generate the sample by walking backwards from some goal
- This also gives us the goal-distance of each state
- Idea: sample a pair of states where one is closer to the goal than the other

Backward-Sampling

- Can also generate the sample by walking backwards from some goal
- This also gives us the goal-distance of each state
- Idea: sample a pair of states where one is closer to the goal than the other
  $\Rightarrow$ can formulate an LP instead of a MIP

Backward-Sampling

- Can also generate the sample by walking backwards from some goal
- This also gives us the goal-distance of each state
- Idea: sample a pair of states where one is closer to the goal than the other
  $\Rightarrow$ can formulate an LP instead of a MIP

$$\min \sum_{(s,s') \in S_{sample}} \alpha_{(s,s')} \tag{9}$$

$$\text{s.t.} \quad h(s) - h(s') + \alpha_{(s,s')} \geq 1 \tag{10}$$

$$\alpha_{(s,s')} \geq 0 \quad \text{for } (s, s') \in S_{sample} \tag{11}$$

Abstract DDA Potential Heuristics

- Naive algorithm does not scale due to the large state space

Abstract DDA Potential Heuristics

- Naive algorithm does not scale due to the large state space
- Idea: use abstractions to obtain a smaller state space

Abstract DDA Potential Heuristics

- Naive algorithm does not scale due to the large state space
- Idea: use abstractions to obtain a smaller state space
- Abstract DDA Potential Heuristics:
  1. use pattern selection algorithm to select an abstraction P
  2. create corresponding abstract task Π^P
  3. use exact algorithm to compute DDA heuristic h_P^{DDA} for Π^P
  4. use h_P^{DDA} for searching the original state space

Abstract DDA Potential Heuristics

- Naive algorithm does not scale due to the large state space
- Idea: use abstractions to obtain a smaller state space
- Abstract DDA Potential Heuristics:
  1. use pattern selection algorithm to select an abstraction P
  2. create corresponding abstract task Π^P
  3. use exact algorithm to compute DDA heuristic h_P^{DDA} for Π^P
  4. use h_P^{DDA} for searching the original state space

we can combine multiple such heuristics by summation

Experimental Setup

Setup:

- 1816 planning tasks
- 8 GB memory limit
- 30 min time limit
- systematically generate all features up to dimension 2

Coverage: Naive Approach

- 157 out of 1816 tasks solved

Coverage: Naive Approach

- 157 out of 1816 tasks solved
- Scalability issues:
  - too many constraints
  - too many features
  - MIP hardness

Coverage: Forward-Sampling

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate MIP over a sample ($sz \in \{125, 250, 500, 1000\}$)
- too many features
  $\Rightarrow$ use all features vs. use only 1000 randomly selected ones
- MIP hardness $\Rightarrow$ unaddressed

Coverage: Forward-Sampling

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate MIP over a sample ($sz \in \{125, 250, 500, 1000\}$)
- too many features
  $\Rightarrow$ use all features vs. use only 1000 randomly selected ones
- MIP hardness $\Rightarrow$ unaddressed

	all features	**1000 features**
$sz = 125$	442	521
$sz = 250$	431	512
$sz = 500$	409	493
$sz = 1000$	381	490

Coverage: Backward-Sampling

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate LP over a sample ($sz \in \{125, 250, 500, 1000\}$)
- too many features
  $\Rightarrow$ use all features vs. use only 1000 randomly selected ones
- MIP hardness $\Rightarrow$ use an LP model

Coverage: Backward-Sampling

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate LP over a sample ($sz \in \{125, 250, 500, 1000\}$)
- too many features
  $\Rightarrow$ use all features vs. use only 1000 randomly selected ones
- MIP hardness $\Rightarrow$ use an LP model

	all features	**1000 features**
$sz = 125$	469	538
$sz = 250$	477	560
$sz = 500$	479	575
$sz = 1000$	487	575

Coverage: Single Abstract DDA Heuristic

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate MIP for an abstraction
  $(sz \in \{256, 512, 1024, 2048\})$
- too many features $\Rightarrow$ resolved due to abstraction
- MIP hardness $\Rightarrow$ unaddressed

Coverage: Single Abstract DDA Heuristic

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate MIP for an abstraction
  ($sz \in \{256, 512, 1024, 2048\}$)
- too many features $\Rightarrow$ resolved due to abstraction
- MIP hardness $\Rightarrow$ unaddressed

	single abs-DDA	**single PDB**
$sz = 256$	581	732
$sz = 512$	561	747
$sz = 1024$	513	758
$sz = 2048$	455	768

Coverage: Multiple Abstract DDA Heuristics

Scalability issues:

- too many constraints
  $\Rightarrow$ formulate MIP for an abstraction
  ($sz \in \{128, 256, 512, 1024\}$) and atomic abstractions
- too many features $\Rightarrow$ resolved due to abstraction
- MIP hardness $\Rightarrow$ unaddressed

Coverage: Multiple Abstract DDA Heuristics

Scalability issues:

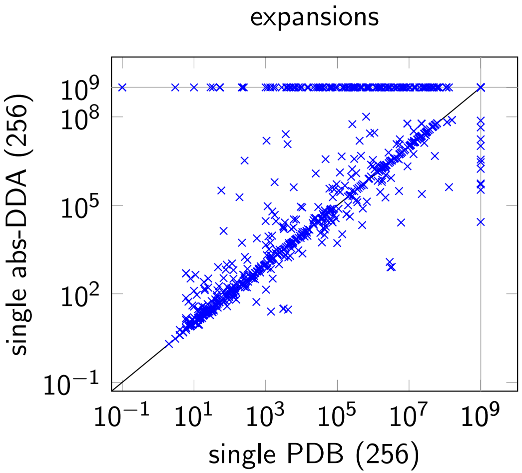
- too many constraints
  $\Rightarrow$ formulate MIP for an abstraction
  ($sz \in \{128, 256, 512, 1024\}$) and atomic abstractions
- too many features $\Rightarrow$ resolved due to abstraction
- MIP hardness $\Rightarrow$ unaddressed

	multiple abs-DDA	**multiple PDB**
atomic	1028	1107
$sz = 128$	1005	1121
$sz = 256$	1005	1130
$sz = 512$	1005	1128
$sz = 1024$	999	1130

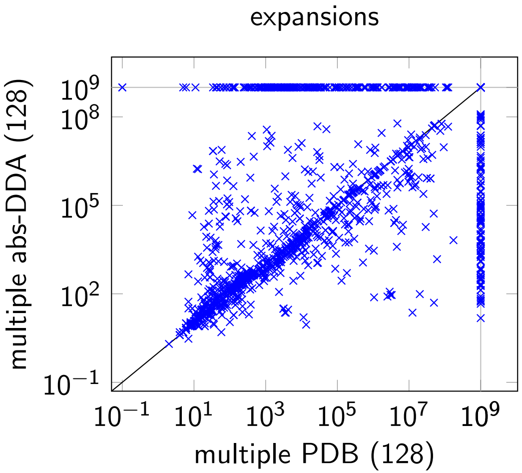
Coverage

	bw-sampling	multiple abs-DDA	multiple PDBs
logistics98	3	8	**35**
visitall14	0	0	**20**
openstacks08	8	**30**	6
parcprinter11	0	**12**	0
tpp	8	**29**	9
snake18	**18**	5	7

Heuristic Quality



Heuristic Quality



Conclusion

- DDA heuristics are PSPACE-hard to compute
- approximation algorithms are necessary
  $\Rightarrow$ most promising approach: abs-DDA potential heuristics
- outscaled by PDBs (PDB computation is more efficient)
- Heuristic quality is comparable to PDBs