Speech about Dipoma Thesis

Solving Non-Deterministic Planning Problems with Pattern Database Heuristics

(Anwendung von Pattern-Database-Heuristiken zum Lösen nichtdeterministischer Planungsprobleme)

Author: Pascal Bercher
Supervisor: Robert Mattmüller

Problem Formulation (informal)

given: Non-deterministic planning problem $\mathcal{P}$.
desired: Strong plan.

chosen procedure: Heuristic search (via AO* algorithm).
chosen heuristic: Pattern database heuristics (abstraction heuristics).

Problem Formulation (informal)

given: Non-deterministic planning problem $\mathcal{P}$.
desired: Strong plan.

chosen procedure: Heuristic search (via AO* algorithm).
chosen heuristic: Pattern database heuristics (abstraction heuristics).

given:
Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \ldots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:
Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- *Var*, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \ldots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:

Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var$ and $i \in \{1, \dots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\dots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:
Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- *Var*, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- *A*, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var$ and $i \in \{1, \dots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\dots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:

Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \dots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\dots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:

Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \ldots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:

Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \dots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\dots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

Problem Formulation (formal)

given:
Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var$ and $i \in \{1, \ldots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

given:

Non-deterministic planning problem $\mathcal{P} = (Var, A, s_0, G)$ with:

- Var, finite set of state variables.
  $S = 2^{Var}$ is the state space.
- A, finite set of actions $a = \langle pre(a), eff(a) \rangle$ and:
  - $pre(a) \subseteq Var$ and
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ and } i \in \{1, \ldots, n\} \}$.
  - Its application (if $pre(a) \subseteq s$) leads to:
    $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - The effect variables of a are $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, the initial state.
- $G \subseteq S$, non-empty set of goal states.

desired:
strong plan, i.e. a strategy $\pi : S \to A$, which transforms $s_0 \in S$ in states of $G \subseteq S$. (Can be represented by means of solution graphs.)

Problem Formulation (formal)

desired:
strong plan, i.e. a strategy $\pi : S \to A$, which transforms $s_0 \in S$ in states of $G \subseteq S$. (Can be represented by means of solution graphs.)

AND/OR graphs:
$\mathcal{P}$ induces an AND/OR graph (hypergraph) $\mathcal{G} = (V, C)$ with:

- $V = S \; (= 2^{Var})$, set of nodes and
- C, set of connectors.
  For each $v \in V$ is $c = (v, app(v, a)) \in C$, if $pre(a) \subseteq v$.
  We call $v = pred(c)$ and $app(v, a) = succ(c)$.

Solution graphs:
Restriction to those AND/OR graphs, which represent a strong plan.

AND/OR graphs:
$\mathcal{P}$ induces an AND/OR graph (hypergraph) $\mathcal{G} = (V, C)$ with:

- $V = S$ $(= 2^{Var})$, set of nodes and
- C, set of connectors.
  For each $v \in V$ is $c = (v, app(v, a)) \in C$, if $pre(a) \subseteq v$.
  We call $v = pred(c)$ and $app(v, a) = succ(c)$.

Solution graphs:
Restriction to those AND/OR graphs, which represent a strong plan.

AND/OR graphs:
$\mathcal{P}$ induces an AND/OR graph (hypergraph) $\mathcal{G} = (V, C)$ with:

- $V = S \ (= 2^{Var})$, set of nodes and
- C, set of connectors.
  For each $v \in V$ is $c = (v, app(v, a)) \in C$, if $pre(a) \subseteq v$.
  We call $v = pred(c)$ and $app(v, a) = succ(c)$.

Solution graphs:
Restriction to those AND/OR graphs, which represent a strong plan.

AND/OR graphs:
$\mathcal{P}$ induces an AND/OR graph (hypergraph) $\mathcal{G} = (V, C)$ with:

- $V = S\ (= 2^{Var})$, set of nodes and
- C, set of connectors.
  For each $v \in V$ is $c = (v, app(v, a)) \in C$, if $pre(a) \subseteq v$.
  We call $v = pred(c)$ and $app(v, a) = succ(c)$.

Solution graphs:
Restriction to those AND/OR graphs, which represent a strong plan.

Solution graphs:
A solution graph $\mathcal{G} = (V, C)$ of $\mathcal{P} = (Var, A, s_0, G)$ is a connected, *cycle-free* AND/OR graph with:

- $s_0 \in V$,
- for all $v \in V$ holds:
  either it holds $v \in G$ or there is exactly one $c \in C$ with $pred(c) = v$.

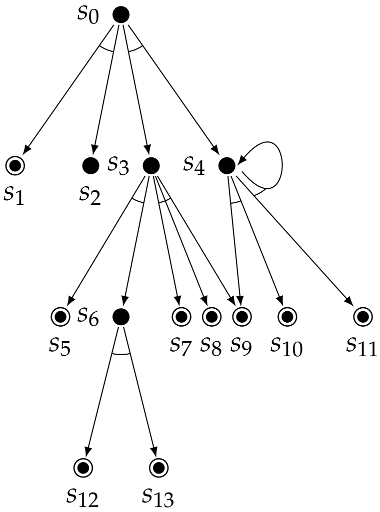
Solution graphs:
A solution graph $\mathcal{G} = (V, C)$ of $\mathcal{P} = (Var, A, s_0, G)$ is a connected, *cycle-free* AND/OR graph with:

- $s_0 \in V$,
- for all $v \in V$ holds:
  either it holds $v \in G$ or there is exactly one $c \in C$ with $pred(c) = v$.

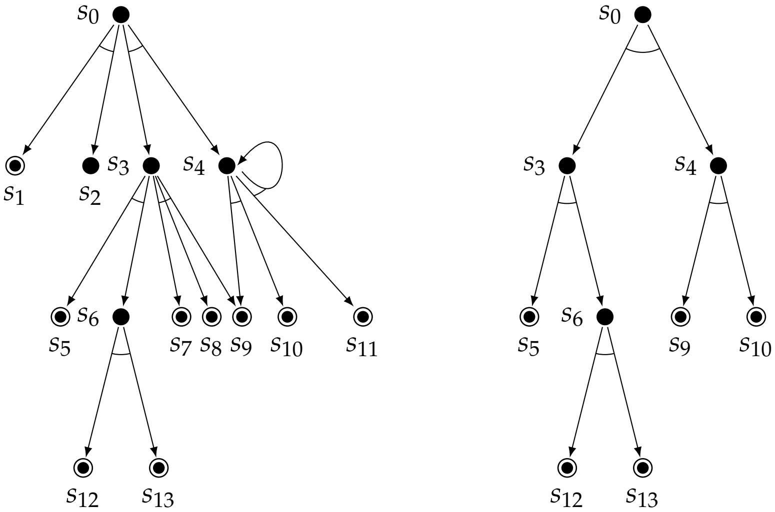
Solution graphs:
A solution graph $\mathcal{G} = (V, C)$ of $\mathcal{P} = (Var, A, s_0, G)$ is a connected, *cycle-free* AND/OR graph with:

- $s_0 \in V$,
- for all $v \in V$ holds:
  either it holds $v \in G$ or there is exactly one $c \in C$ with $pred(c) = v$.

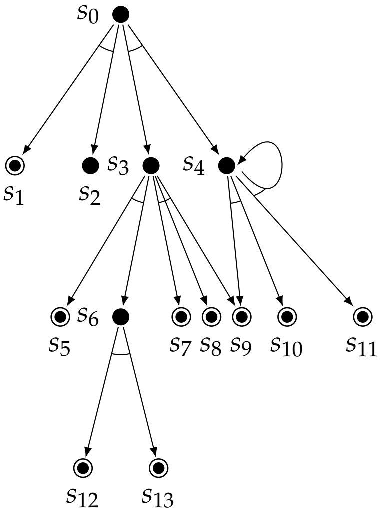
AND/OR Graphs, Solution Graphs, (Non-)Examples

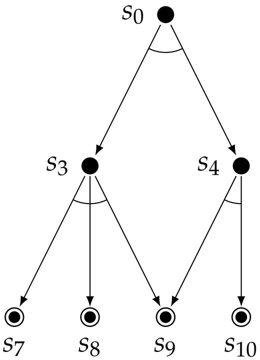


AND/OR Graphs, Solution Graphs, (Non-)Examples

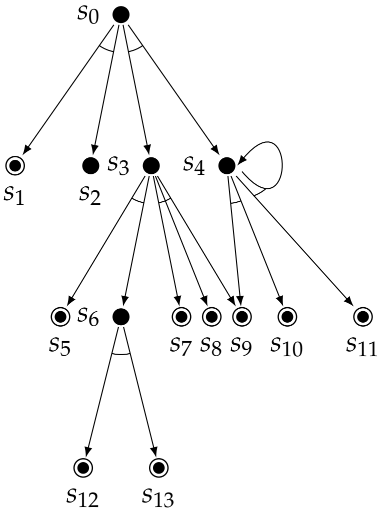


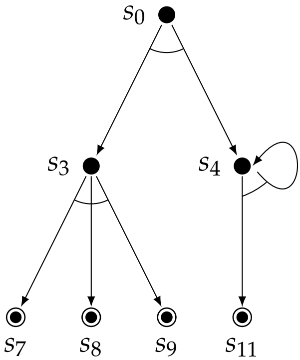
AND/OR Graphs, Solution Graphs, (Non-)Examples





AND/OR Graphs, Solution Graphs, (Non-)Examples





AO* Algorithm, informal

Forward step:

- Traverse graph using only marked connectors and find non-expanded leef with maximal heuristic value (fail first).
- Expand this leef.

Backward step:

- Update cost value of nodes and update markings.
- Optimal cost value for $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, where the $\mathcal{G}$'s are solution graphs for v.

AO* Algorithm, informal

Forward step:

- Traverse graph using only marked connectors and find non-expanded leef with maximal heuristic value (fail first).
- Expand this leef.

Backward step:

- Update cost value of nodes and update markings.
- Optimal cost value for $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, where the $\mathcal{G}$'s are solution graphs for v.

Forward step:

- Traverse graph using only marked connectors and find non-expanded leef with maximal heuristic value (fail first).
- Expand this leef.

Backward step:

- Update cost value of nodes and update markings.
- Optimal cost value for $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, where the $\mathcal{G}$'s are solution graphs for v.

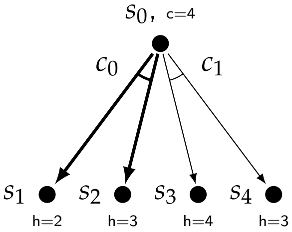
Forward step:

- Traverse graph using only marked connectors and find non-expanded leef with maximal heuristic value (fail first).
- Expand this leef.

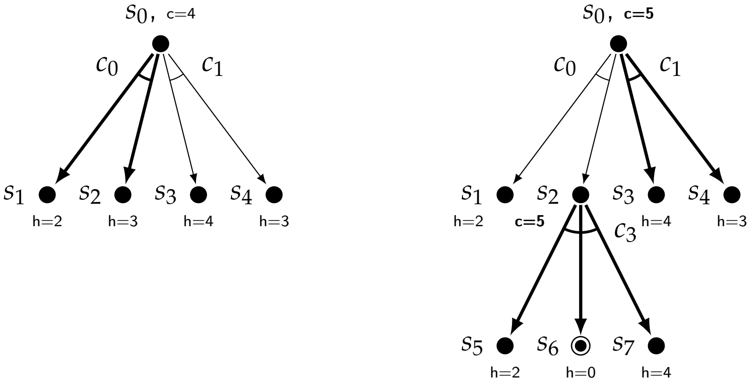
Backward step:

- Update cost value of nodes and update markings.
- Optimal cost value for $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, where the $\mathcal{G}$'s are solution graphs for v.

AO* Algorithm, Example



AO* Algorithm, Example



Preprocessing:

- Simplify problem via abstraction.
- For each abstraction: Solve the problem completely.
- For each abstract state: Save its cost in Pattern Database.

During the Search:

- Access the abstract states to use their cost to calculate heuristic.

Preprocessing:

- Simplify problem via abstraction.
- For each abstraction: Solve the problem completely.
- For each abstract state: Save its cost in Pattern Database.

During the Search:

- Access the abstract states to use their cost to calculate heuristic.

Preprocessing:

- Simplify problem via abstraction.
- For each abstraction: Solve the problem completely.
- For each abstract state: Save its cost in Pattern Database.

During the Search:

- Access the abstract states to use their cost to calculate heuristic.

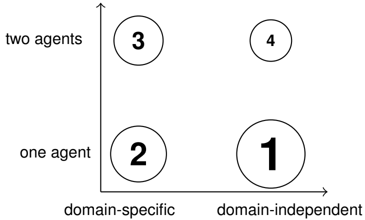
Preprocessing:

- Simplify problem via abstraction.
- For each abstraction: Solve the problem completely.
- For each abstract state: Save its cost in Pattern Database.

During the Search:

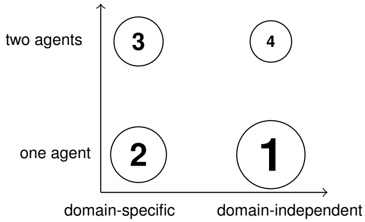
- Access the abstract states to use their cost to calculate heuristic.

Existing Work



(1) Prieditis (1993), Edelkamp (2001,2002,2006), Felner et al. (2004), Haslum et al. (2005, 2007)
(2) Culberson & Schaeffer (1996,1998), Korf & Felnder (2002), Felner et al. (2004)
(3) Samadi et al. (2008)
(4) none known

Existing Work



(1) Prieditis (1993), Edelkamp (2001,2002,2006), Felner et al. (2004), Haslum et al. (2005, 2007)
(2) Culberson & Schaeffer (1996,1998), Korf & Felnder (2002), Felner et al. (2004)
(3) Samadi et al. (2008)
(4) none known

Definition:
The abstraction $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ is equal to the planning problem $\mathcal{P}$, restricted to the pattern $P_i \subseteq Var$.

It holds:

- $Var^i := P_i$,
- For $s \in S$ is $s^i := s \cap P_i$ and for $var \subseteq Var$ is $var^i := var \cap P_i$, respectively.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ for $a \in A$. Then, $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
The abstraction $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ is equal to the planning problem $\mathcal{P}$, restricted to the pattern $P_i \subseteq Var$.

It holds:

- $Var^i := P_i$,
- For $s \in S$ is $s^i := s \cap P_i$ and for $var \subseteq Var$ is $var^i := var \cap P_i$, respectively.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ for $a \in A$. Then, $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
The abstraction $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ is equal to the planning problem $\mathcal{P}$, restricted to the pattern $P_i \subseteq Var$.

It holds:

- $Var^i := P_i$,
- For $s \in S$ is $s^i := s \cap P_i$ and for $var \subseteq Var$ is $var^i := var \cap P_i$, respectively.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ for $a \in A$. Then, $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
The abstraction $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ is equal to the planning problem $\mathcal{P}$, restricted to the pattern $P_i \subseteq Var$.

It holds:

- $Var^i := P_i$,
- For $s \in S$ is $s^i := s \cap P_i$ and for $var \subseteq Var$ is $var^i := var \cap P_i$, respectively.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ for $a \in A$. Then, $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
The abstraction $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ is equal to the planning problem $\mathcal{P}$, restricted to the pattern $P_i \subseteq Var$.

It holds:

- $Var^i := P_i$,
- For $s \in S$ is $s^i := s \cap P_i$ and for $var \subseteq Var$ is $var^i := var \cap P_i$, respectively.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ for $a \in A$. Then, $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Calculation of the Pattern Database

For a state s^i and $s \in S$, its heuristic h^i is

$$h^i(s^i) := cost^*(s^i) := \min_{\mathcal{G}} height(\mathcal{G}),$$

where the $\mathcal{G}$'s are solution graphs for s^i.

Calculation process:

- Calculate the complete AND/OR graph of $\mathcal{P}^i$.
- Solve it by Value Iteration.

Question:
How to calculate $h(s)$, if there are multiple patterns?

For a state s^i and $s \in S$, its heuristic h^i is

$$h^i(s^i) := cost^*(s^i) := \min_{\mathcal{G}} height(\mathcal{G}),$$

where the $\mathcal{G}$'s are solution graphs for s^i.

Calculation process:

- Calculate the complete AND/OR graph of $\mathcal{P}^i$.
- Solve it by Value Iteration.

Question:
How to calculate $h(s)$, if there are multiple patterns?

For a state s^i and $s \in S$, its heuristic h^i is

$$h^i(s^i) := cost^*(s^i) := \min_{\mathcal{G}} height(\mathcal{G}),$$

where the $\mathcal{G}$'s are solution graphs for s^i.

Calculation process:

- Calculate the complete AND/OR graph of $\mathcal{P}^i$.
- Solve it by Value Iteration.

Question:
How to calculate $h(s)$, if there are multiple patterns?

Calculation of the heuristic value (1)

given:
Set of patterns $P := \{P_1 \ldots, P_m\}$.

It holds: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ is admissible.

Question:
How to generate a heuristic that is more informed?

Answer:
By using Additivity.

given:
Set of patterns $P := \{P_1 \ldots, P_m\}$.

It holds: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ is admissible.

Question:
How to generate a heuristic that is more informed?

Answer:
By using Additivity.

given:
Set of patterns $P := \{P_1 \ldots, P_m\}$.

It holds: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ is admissible.

Question:
How to generate a heuristic that is more informed?

Answer:
By using Additivity.

Definition:
The set $P := \{P_1, \ldots, P_k\}$ with $k \in \mathbb{N}$ is additive, if for all states $s \in S$ holds $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Theorem, known from classical planning by Edelkamp (2001):

Theorem:
If for all $a \in A$ and for all patterns $P_i \in P$ holds:

If $P_i \cap effvar(a) \neq \emptyset$, then it holds for all $P_j \in P$ with $P_j \neq P_i$, that $P_j \cap effvar(a) = \emptyset$.

Then the patterns of P are additive.

This theorem also applies to non-deterministic planning problems. (Proof by induction, omitted)

Definition:
The set $P := \{P_1, \ldots, P_k\}$ with $k \in \mathbb{N}$ is additive, if for all states $s \in S$ holds $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Theorem, known from classical planning by Edelkamp (2001):

Theorem:
If for all $a \in A$ and for all patterns $P_i \in P$ holds:

If $P_i \cap effvar(a) \neq \emptyset$, then it holds for all $P_j \in P$ with $P_j \neq P_i$, that $P_j \cap effvar(a) = \emptyset$.

Then the patterns of P are additive.

This theorem also applies to non-deterministic planning problems. (Proof by induction, omitted)

Definition:
The set $P := \{P_1, \ldots, P_k\}$ with $k \in \mathbb{N}$ is additive, if for all states $s \in S$ holds $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Theorem, known from classical planning by Edelkamp (2001):

Theorem:
If for all $a \in A$ and for all patterns $P_i \in P$ holds:

If $P_i \cap effvar(a) \neq \emptyset$, then it holds for all $P_j \in P$ with $P_j \neq P_i$, that $P_j \cap effvar(a) = \emptyset$.

Then the patterns of P are additive.

This theorem also applies to non-deterministic planning problems. (Proof by induction, omitted)

Definition:
The set $P := \{P_1, \ldots, P_k\}$ with $k \in \mathbb{N}$ is additive, if for all states $s \in S$ holds $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Theorem, known from classical planning by Edelkamp (2001):

Theorem:
If for all $a \in A$ and for all patterns $P_i \in P$ holds:

If $P_i \cap effvar(a) \neq \emptyset$, then it holds for all $P_j \in P$ with $P_j \neq P_i$, that $P_j \cap effvar(a) = \emptyset$.

Then the patterns of P are additive.

This theorem also applies to non-deterministic planning problems. (Proof by induction, omitted)

Calculation of the heuristic value (2)

given:
Set P of patterns $P := \{P_1 \ldots, P_m\}$ and partition M in additive patterns, i.e. $M := \{M_1, \ldots, M_m\}$ sets of patterns with $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ and the M_i are additive.

(Now: Identification of s^{P_i} and h^{P_i} with s^i and h^i.)

It holds:

$$h(s) := \max_{M_i \in M} \sum_{P \in M_i} h^P(s^P) \text{ is admissible.}$$

Calculation of the heuristic value (2)

given:
Set P of patterns $P := \{P_1 \ldots, P_m\}$ and partition M in additive patterns, i.e. $M := \{M_1, \ldots, M_m\}$ sets of patterns with $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ and the M_i are additive.

(Now: Identification of s^{P_i} and h^{P_i} with s^i and h^i.)

It holds:

$$h(s) := \max_{M_i \in M} \sum_{P \in M_i} h^P(s^P) \text{ is admissible.}$$

given:
Set P of patterns $P := \{P_1 \ldots, P_m\}$ and partition M in additive patterns, i.e. $M := \{M_1, \ldots, M_m\}$ sets of patterns with $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ and the M_i are additive.

(Now: Identification of s^{P_i} and h^{P_i} with s^i and h^i.)

It holds:

$$h(s) := \max_{M_i \in M} \sum_{P \in M_i} h^P(s^P) \text{ is admissible.}$$

Open Question:

How to find the set P of pattens, or rather a suitably partition M of P?

$\longrightarrow$

(Still) manually. Automation via local search in principle possible. Cf. Haslum et al. (2005,2007), Edelkamp (2006).

How to find the set P of pattens, or rather a suitably partition M of P?
$\longrightarrow$
(Still) manually. Automation via local search in principle possible. Cf. Haslum et al. (2005,2007), Edelkamp (2006).

- Tireworld
  (no additivity)
- Chain of Rooms
  (additivity: sequential subproblems)
- Coin Flip
  (additivity: parallel subproblems)

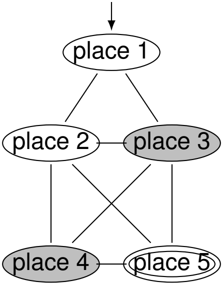
Domains

- Tireworld
  (no additivity)
- Chain of Rooms
  (additivity: sequential subproblems)
- Coin Flip
  (additivity: parallel subproblems)

Domains

- Tireworld
  (no additivity)
- Chain of Rooms
  (additivity: sequential subproblems)
- Coin Flip
  (additivity: parallel subproblems)

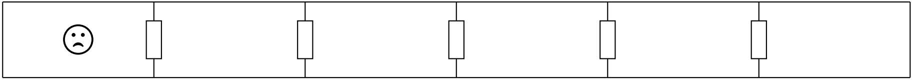
Tireworld, problem description



Actions:

- Use street (non-deterministically: flat tire).
- Take spare tire (if present).
- Change spare (if flat tire and spare present).

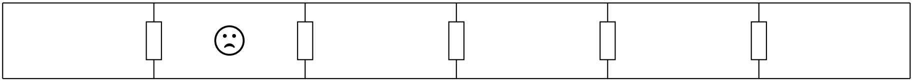
Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

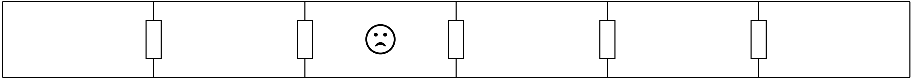
Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

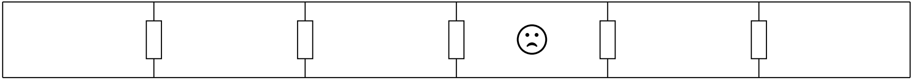
Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

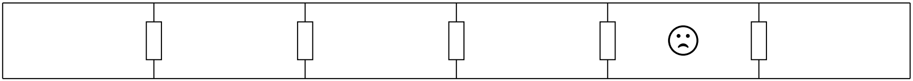
Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

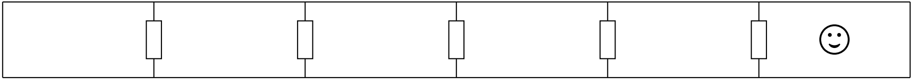
Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

Chain of Rooms, problem description



actions:

- Go into neighboring room, if door is open.
- Turn light on, if it is off
  (non-deterministically: door open/closed).
- Open door.

Coin Flip, problem description

given:

- *n* coins, all initially in a bag.
- Actions:
  - Flip coin, if not flipped before.
  - Reverse coin, if flipped before.

desired:

Strategy, that all coins show heads.

given:

- *n* coins, all initially in a bag.
- Actions:
  - Flip coin, if not flipped before.
  - Reverse coin, if flipped before.

desired:

Strategy, that all coins show heads.

given:

- *n* coins, all initially in a bag.
- Actions:
  - Flip coin, if not flipped before.
  - Reverse coin, if flipped before.

desired:

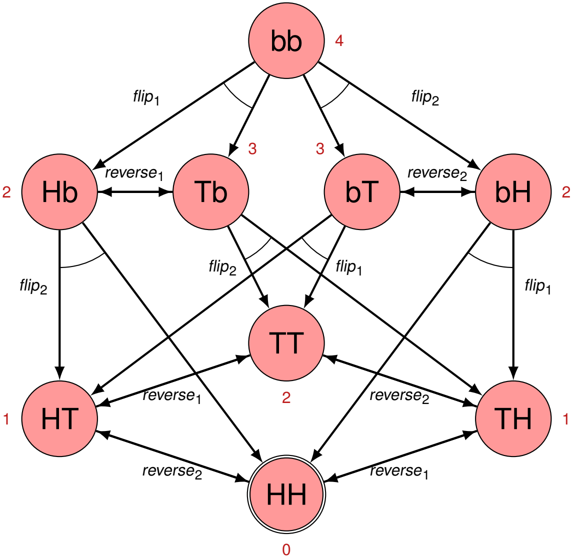
Strategy, that all coins show heads.

given:

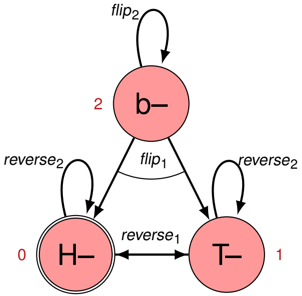
- *n* coins, all initially in a bag.
- Actions:
  - Flip coin, if not flipped before.
  - Reverse coin, if flipped before.

desired:
Strategy, that all coins show heads.

Coin Flip (two coins), example for additive patterns

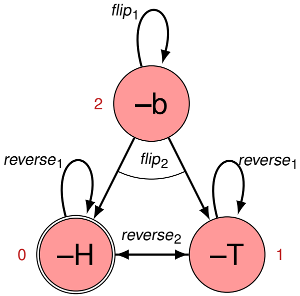


Coin Flip (two coins), example for additive patterns



$$h^{P_1}(bb^{P_1}) = h^{P_1}(b-) = 2$$

Coin Flip (two coins), example for additive patterns



$$h^{P_2}(bb^{P_2}) = h^{P_2}(-b) = 2 \qquad \Rightarrow$$

$$h(bb) = h(bb^{P_1}) + h(bb^{P_2}) = h^{P_2}(b-) + h^{P_2}(-b) = 2 + 2 = 4$$

Comparison between:

- AO* algorithm and *no heuristik*,
- AO* algorithm and *adversarial FF heuristic*,
- AO* algorithm and *pattern database heuristic*,

Comparison possible to:

- POND: Partially Observable Nondeterministic Planner (Bryce et al., 2006)
- MBP: Model Based Planner (Cimatti et al., 2003)
- Gamer (Edelkamp and Kissmann)

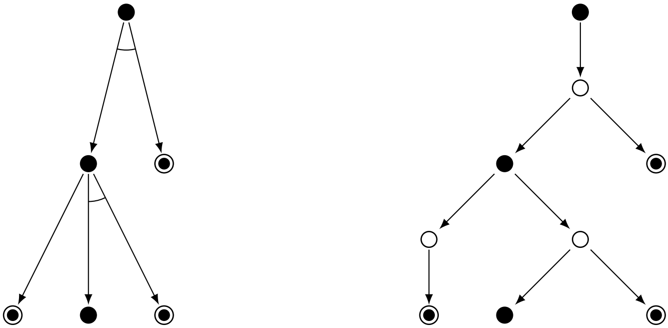
Comparison between:

- AO* algorithm and *no heuristik*,
- AO* algorithm and *adversarial FF heuristic*,
- AO* algorithm and *pattern database heuristic*,

Comparison possible to:

- POND: Partially Observable Nondeterministic Planner (Bryce et al., 2006)
- MBP: Model Based Planner (Cimatti et al., 2003)
- Gamer (Edelkamp and Kissmann)

The planner uses explicite AND/OR nodes and edges instead of only one sort of nodes and connectors:



Results: Tireworld

#places	uninformed			adv. FF heuristic			PDB heuristic				
	time	RAM	nodes	time	RAM	nodes	time pre	time search	time	RAM	nodes
4	**0**	**0**	85	**0**	**0**	62	0	0	**0**	**0**	**46**
6	**0**	**0**	84	**0**	**0**	47	2	0	2	8	**37**
8	**0**	1	153	**0**	**0**	**43**	1	0	1	6	**43**
10	**0**	1	293	**0**	**0**	96	2	0	2	7	**67**
20	**0**	4	504	**0**	**1**	**126**	2	0	2	8	128
40	93	2097	115541	**6**	**117**	**6451**	3	6	9	243	12827
60	2	196	9088	**1**	**24**	**1086**	2	0	2	41	1565
80	1	163	5924	**0**	**26**	915	1	0	2	30	**746**
100	17	745	20249	**1**	**42**	**1101**	1	0	2	50	1154
120	–	–	–	17	464	10739	1	1	**3**	**81**	**1632**
140	23	1584	33418	5	90	1849	2	0	**2**	**66**	**1202**
160	–	–	–	8	205	3914	2	0	**3**	**72**	**1175**
180	11	552	9599	**2**	**62**	**1055**	2	2	4	136	2239
200	41	1622	22175	5	130	1746	2	1	**3**	**93**	**1081**

– := out of memory (4 GB)

Ten patterns: Each pattern contains the goal place, the spare, the flat tire and seven random places.

Results: Chain of Rooms

#rooms	uninformed			adv. FF heuristic			PDB heuristic				
	time	RAM	nodes	time	RAM	nodes	time pre	time search	time	RAM	nodes
4	**0**	**0**	38	**0**	**0**	**30**	0	0	**0**	**0**	**30**
6	**0**	**0**	91	**0**	**0**	62	0	0	**0**	**0**	**57**
8	**0**	1	170	**0**	**0**	109	0	0	**0**	**0**	**81**
10	**0**	**1**	272	**0**	**1**	169	0	0	**0**	**1**	**147**
20	**0**	6	1142	**0**	**3**	643	1	0	1	**3**	**464**
40	**1**	40	4682	8	20	2494	3	0	4	**18**	**2043**
60	**5**	121	10621	40	59	5543	9	1	10	**52**	**4827**
80	**17**	278	18962	122	131	9793	16	3	20	**121**	**8804**
100	43	533	29702	300	251	15244	30	9	**39**	**230**	**13983**
120	92	909	42841	609	443	21893	43	19	**62**	**394**	**20367**
140	162	1390	58382	1164	686	29743	63	37	**101**	**617**	**27944**
160	257	2090	76322	2054	994	38794	94	63	**157**	**902**	**36723**
180	468	2986	96661	3369	1457	49043	127	98	**225**	**1283**	**46707**
200	–	–	–	5855	1961	60493	164	154	**319**	**1763**	**57884**

– := out of memory (4 GB)

Patterns: Four neighboring rooms per pattern.

Results: Coin Flip

#coins	uninformed			adv. FF heuristic			PDB heuristic				
	time	RAM	nodes	time	RAM	nodes	time pre	time search	time	RAM	nodes
4	**0**	1	349	**0**	1	313	0	0	**0**	**0**	**69**
6	**0**	11	4809	**0**	8	3374	0	0	**0**	**0**	**161**
8	18	145	58377	3	46	18641	0	0	**0**	**1**	**293**
10	1102	1542	588813	77	483	186221	0	0	**0**	**1**	**465**
20	–	–	–	–	–	–	0	0	**0**	**7**	**1925**
40	–	–	–	–	–	–	1	1	**2**	**49**	**7845**
60	–	–	–	–	–	–	3	6	**9**	**161**	**17765**
80	–	–	–	–	–	–	6	18	**24**	**384**	**31685**
100	–	–	–	–	–	–	12	45	**57**	**714**	**49605**
120	–	–	–	–	–	–	18	96	**114**	**1176**	**71525**
140	–	–	–	–	–	–	28	163	**192**	**1784**	**97445**
160	–	–	–	–	–	–	39	289	**329**	**2589**	**127365**
180	–	–	–	–	–	–	58	–	–	–	–
200	–	–	–	–	–	–	74	–	–	–	–

– := out of memory (4 GB)

Patterns: Two coins per pattern.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Summary:

- Additivity criterion also applies to non-deterministic planning.
- Pattern database heuristics (therefore) can achieve good results (depending on decomposition).

Outlook:

- Finding strong *cyclic* plans.
- Automated pattern selection.
- Making use of multi-valued variables.
- More efficiency by using one sort of nodes and connectors.

Thanks for your attention!

References

Joseph C. Culberson and Jonathan Schaeffer.
Searching with pattern databases.
In *Advances in Artificial Intelligence (Lecture Notes in Artificial Intelligence 1081)*, pages 402–416. Springer-Verlag, 1996.

Joseph C. Culberson and Jonathan Schaeffer.
Pattern databases.
Computational Intelligence, 14(3):318–334, 1998.

Stefan Edelkamp.
Planning with pattern databases.
In *Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 13–24, 2001.

Stefan Edelkamp.
Symbolic pattern databases in heuristic search planning.
In *Proceedings of AIPS-02*, pages 274–283, 2002.

References

Stefan Edelkamp.
Automated creation of pattern database search heuristics.
In *MoChArt*, pages 35–50, 2006.

Ariel Felner, Richard Korf, and Sarit Hanan.
Additive pattern database heuristics.
Journal of Artificial Intelligence Research, 22:279–318, 2004.

Patrick Haslum, Blai Bonet, and Héctor Geffner.
New admissible heuristics for domain-independent planning.
In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, volume 20, pages 1163–1168, 2005.

- Patrik Haslum, Adi Botea, Blai Bonet, Malte Helmert, and Sven Koenig.
  Domain-independent construction of pattern database heuristics for cost-optimal planning.
  In *In Proc. AAAI 2007*, pages 1007–1012, 2007.
- Richard E. Korf and Ariel Felner.
  Disjoint pattern database heuristics.
  Artificial Intelligence Journal (AIJ), 134:9–22, 2002.
- Armand E. Prieditis.
  Machine discovery of effective admissible heuristics.
  In *Machine Learning*, pages 117–141, 1993.

Mehdi Samadi, Jonathan Schaeffer, Fatemeh Torabi Asr, Majid Samar, and Zohreh Azimifar.
Using abstraction in two-player games.
In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, pages 545–549, 2008.