Vortrag zur Diplomarbeit

Anwendung von Pattern-Database-Heuristiken zum Lösen nichtdeterministischer Planungsprobleme

Autor: Pascal Bercher
Betreuer: Robert Mattmüller

Problemstellung (informell)

gegeben: Nichtdeterministisches Planungsproblem $\mathcal{P}$.
gesucht: Starker Plan.

gewählter Ansatz: Heuristische Suche (mit AO*-Algorithmus).
gewählte Heuristik: Pattern-Database-Heuristiken (Abstraktionsheuristiken).

gegeben: Nichtdeterministisches Planungsproblem $\mathcal{P}$.
gesucht: Starker Plan.

gewählter Ansatz: Heuristische Suche (mit AO*-Algorithmus).
gewählte Heuristik: Pattern-Database-Heuristiken (Abstraktionsheuristiken).

gegeben:

Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:
Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:
Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:
Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:

Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \setminus del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:

Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:

Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:

Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gegeben:
Nichtdeterministisches Planungsproblem $\mathcal{P} = (Var, A, s_0, G)$ mit:

- Var, endliche Menge von Zustandsvariablen.
  $S = 2^{Var}$ ist der Zustandsraum.
- A, endliche Menge von Aktionen $a = \langle pre(a), eff(a) \rangle$ und:
  - $pre(a) \subseteq Var$ und
  - $eff(a) = \{ \langle add_i, del_i \rangle \mid add_i, del_i \subseteq Var \text{ und } i \in \{1, \ldots, n\} \}$.
  - Es gilt: $app(s, a) = \{ (s \cup add) \backslash del \mid \langle add, del \rangle \in eff(a) \}$
  - Die Effektvariablen von a sind $effvar(a) = \bigcup_{i \in \{1,\ldots,n\}} add_i \cup del_i$
- $s_0 \in S$, der Initialzustand.
- $G \subseteq S$, nichtleere Menge von Zielzuständen.

gesucht:
Starker Plan, d. h. eine Strategie $\pi : S \to A$, die bei Anwendung $s_0 \in S$ in Zustände aus $G \subseteq S$ überführt. (Kann durch Lösungsgraph repräsentiert werden.)

gesucht:
Starker Plan, d. h. eine Strategie $\pi : S \to A$, die bei Anwendung $s_0 \in S$ in Zustände aus $G \subseteq S$ überführt. (Kann durch Lösungsgraph repräsentiert werden.)

UND/ODER-Graphen:
$\mathcal{P}$ induziert einen UND/ODER-Graphen (Hypergraph) $\mathcal{G} = (V, C)$ mit:

- $V = S$ $(= 2^{Var})$, Menge der Knoten und
- C, Menge der Konnektoren.
  Für jedes $v \in V$ ist $c = (v, app(v, a)) \in C$, falls $pre(a) \subseteq v$.
  Wir nennen $v = pred(c)$ und $app(v, a) = succ(c)$.

Lösungsgraphen:
Einschränkung auf UND/ODER-Graphen, die einen starken Plan repräsentieren!

UND/ODER-Graphen:
$\mathcal{P}$ induziert einen UND/ODER-Graphen (Hypergraph) $\mathcal{G} = (V, C)$ mit:

- $V = S$ $(= 2^{Var})$, Menge der Knoten und
- C, Menge der Konnektoren.
  Für jedes $v \in V$ ist $c = (v, app(v, a)) \in C$, falls $pre(a) \subseteq v$.
  Wir nennen $v = pred(c)$ und $app(v, a) = succ(c)$.

Lösungsgraphen:
Einschränkung auf UND/ODER-Graphen, die einen starken Plan repräsentieren!

UND/ODER-Graphen:

$\mathcal{P}$ induziert einen UND/ODER-Graphen (Hypergraph) $\mathcal{G} = (V, C)$ mit:

- $V = S$ $(= 2^{Var})$, Menge der Knoten und
- C, Menge der Konnektoren.
  Für jedes $v \in V$ ist $c = (v, app(v, a)) \in C$, falls $pre(a) \subseteq v$.
  Wir nennen $v = pred(c)$ und $app(v, a) = succ(c)$.

Lösungsgraphen:

Einschränkung auf UND/ODER-Graphen, die einen starken Plan repräsentieren!

UND/ODER-Graphen:

$\mathcal{P}$ induziert einen UND/ODER-Graphen (Hypergraph) $\mathcal{G} = (V, C)$ mit:

- $V = S$ $(= 2^{Var})$, Menge der Knoten und
- C, Menge der Konnektoren.
  Für jedes $v \in V$ ist $c = (v, app(v, a)) \in C$, falls $pre(a) \subseteq v$.
  Wir nennen $v = pred(c)$ und $app(v, a) = succ(c)$.

Lösungsgraphen:

Einschränkung auf UND/ODER-Graphen, die einen starken Plan repräsentieren!

Lösungsgraphen:
Ein Lösungsgraph $\mathcal{G} = (V, C)$ von $\mathcal{P} = (Var, A, s_0, G)$ ist ein zusammenhängender, *zyklenfreier* UND/ODER-Graph mit:

- $s_0 \in V$,
- für alle $v \in V$ gilt:
  entweder ist $v \in G$ oder es existiert genau ein $c \in C$ mit $pred(c) = v$.

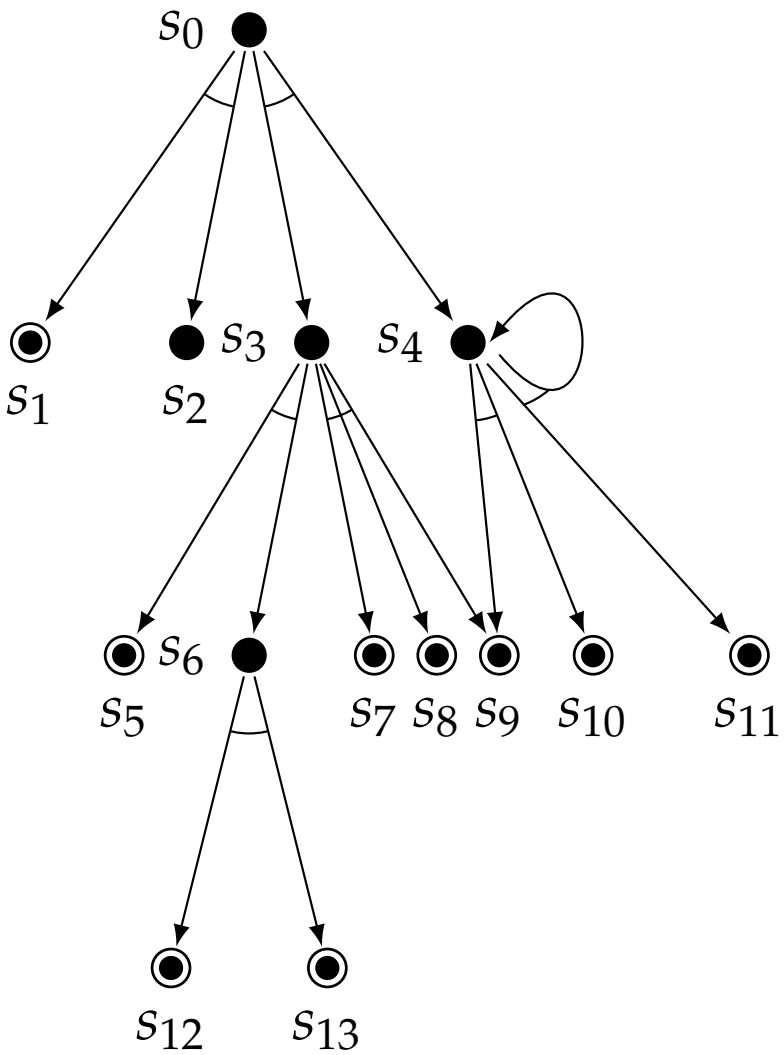
Lösungsgraphen:
Ein Lösungsgraph $\mathcal{G} = (V, C)$ von $\mathcal{P} = (Var, A, s_0, G)$ ist ein zusammenhängender, *zyklenfreier* UND/ODER-Graph mit:

- $s_0 \in V$,
- für alle $v \in V$ gilt:
  entweder ist $v \in G$ oder es existiert genau ein $c \in C$ mit $pred(c) = v$.

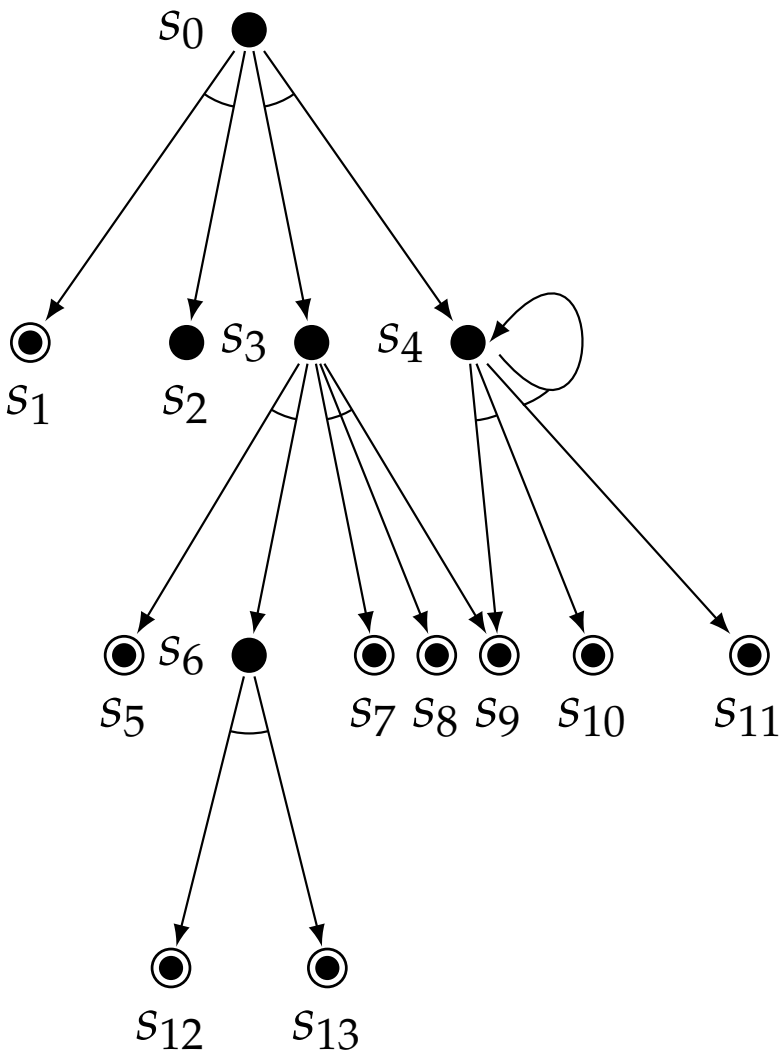
Lösungsgraphen:
Ein Lösungsgraph $\mathcal{G} = (V, C)$ von $\mathcal{P} = (Var, A, s_0, G)$ ist ein zusammenhängender, *zyklenfreier* UND/ODER-Graph mit:

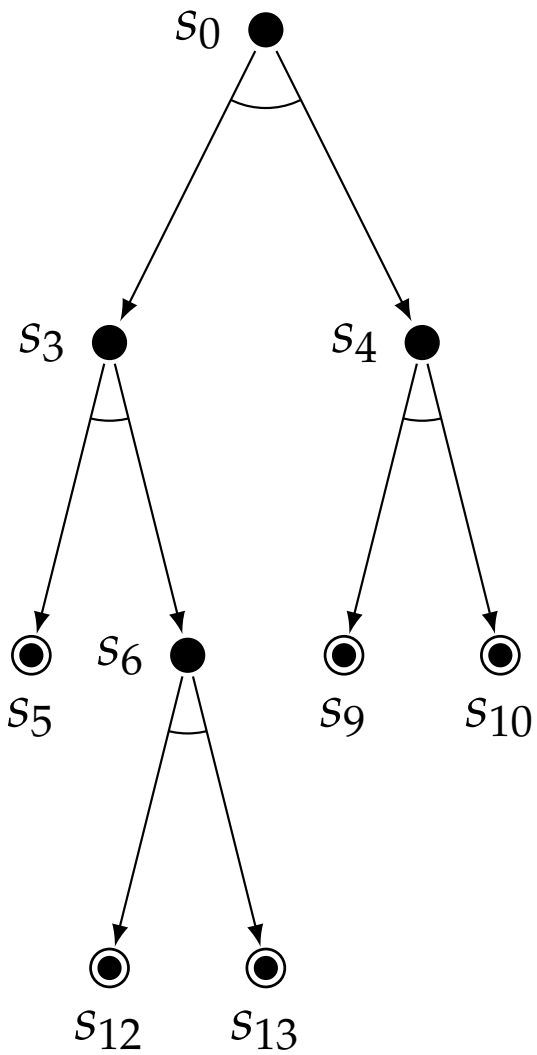
- $s_0 \in V$,
- für alle $v \in V$ gilt:
  entweder ist $v \in G$ oder es existiert genau ein $c \in C$ mit $pred(c) = v$.

UND/ODER-Graphen, Lösungsgraphen, Beispiele

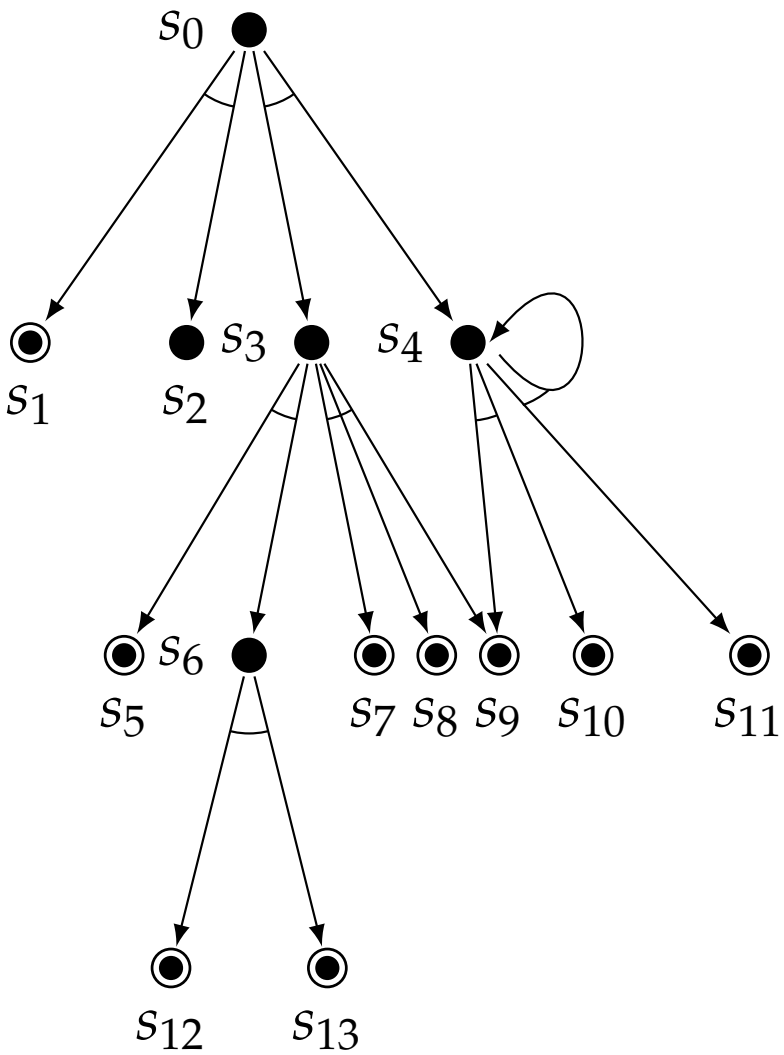


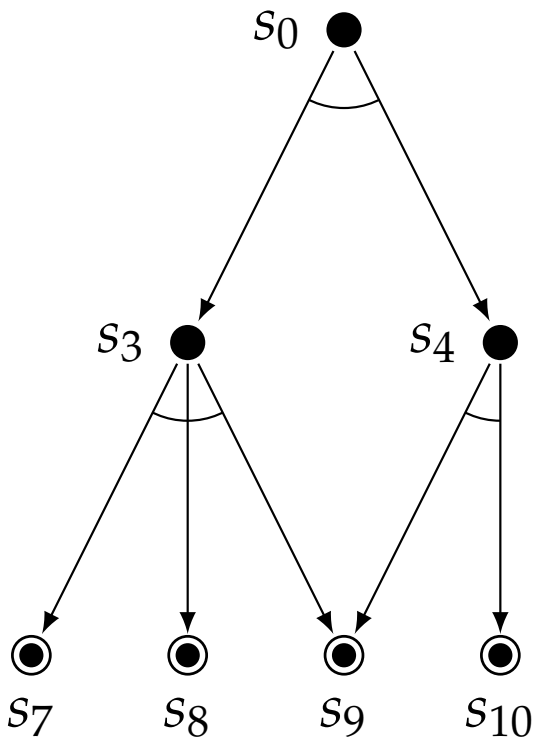
UND/ODER-Graphen, Lösungsgraphen, Beispiele



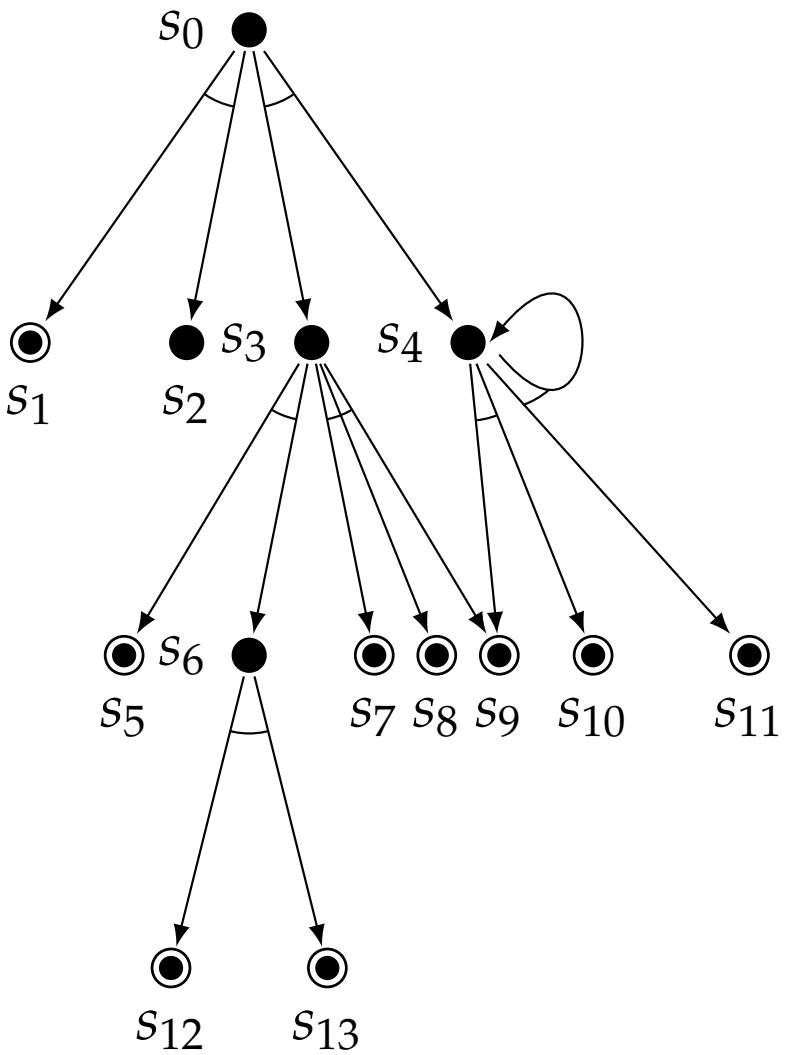


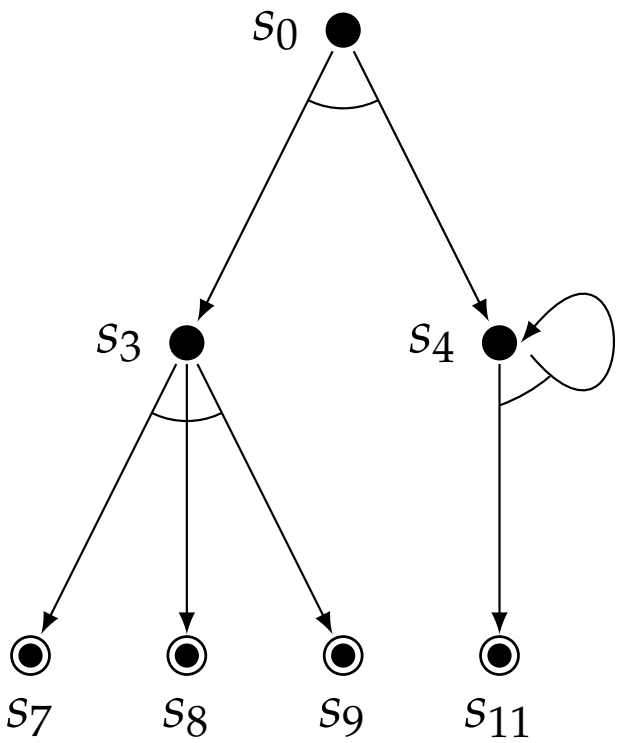
UND/ODER-Graphen, Lösungsgraphen, Beispiele





UND/ODER-Graphen, Lösungsgraphen, Beispiele





Vorwärtsschritt:

- Traversiere Graphen entlang markierter Konnektoren und finde nichtexpandiertes Blatt mit maximalem Heuristikwert (fail first).
- Expandiere dieses Blatt.

Rückwärtsschritt:

- Aktualisiere Kostenwerte der Knoten und setzte Markierungen um.
- Optimaler Kostenwert für $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, wobei die $\mathcal{G}$ Lösungsgraphen für v sind.

Vorwärtsschritt:

- Traversiere Graphen entlang markierter Konnektoren und finde nichtexpandiertes Blatt mit maximalem Heuristikwert (fail first).
- Expandiere dieses Blatt.

Rückwärtsschritt:

- Aktualisiere Kostenwerte der Knoten und setzte Markierungen um.
- Optimaler Kostenwert für $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, wobei die $\mathcal{G}$ Lösungsgraphen für v sind.

Vorwärtsschritt:

- Traversiere Graphen entlang markierter Konnektoren und finde nichtexpandiertes Blatt mit maximalem Heuristikwert (fail first).
- Expandiere dieses Blatt.

Rückwärtsschritt:

- Aktualisiere Kostenwerte der Knoten und setzte Markierungen um.
- Optimaler Kostenwert für $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, wobei die $\mathcal{G}$ Lösungsgraphen für v sind.

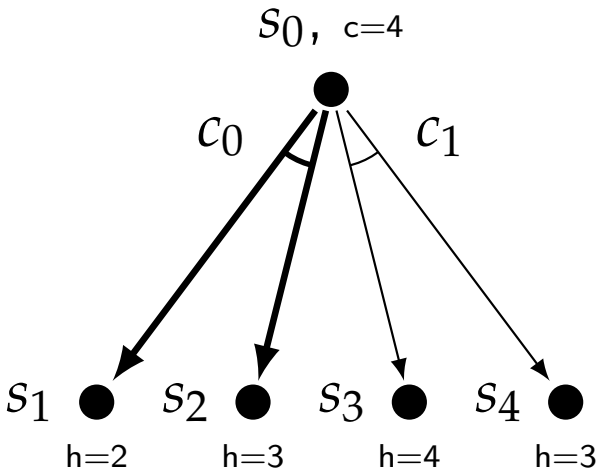
Vorwärtsschritt:

- Traversiere Graphen entlang markierter Konnektoren und finde nichtexpandiertes Blatt mit maximalem Heuristikwert (fail first).
- Expandiere dieses Blatt.

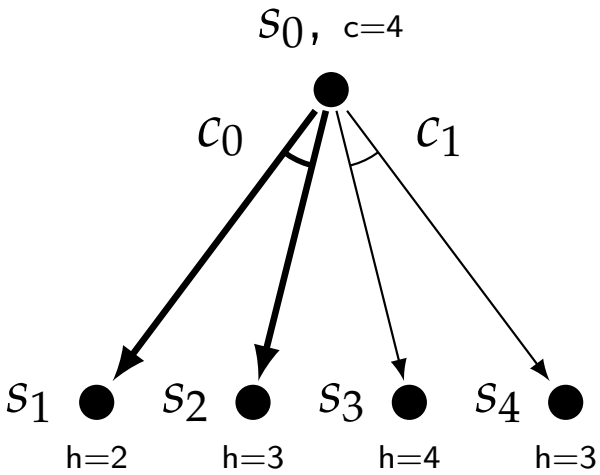
Rückwärtsschritt:

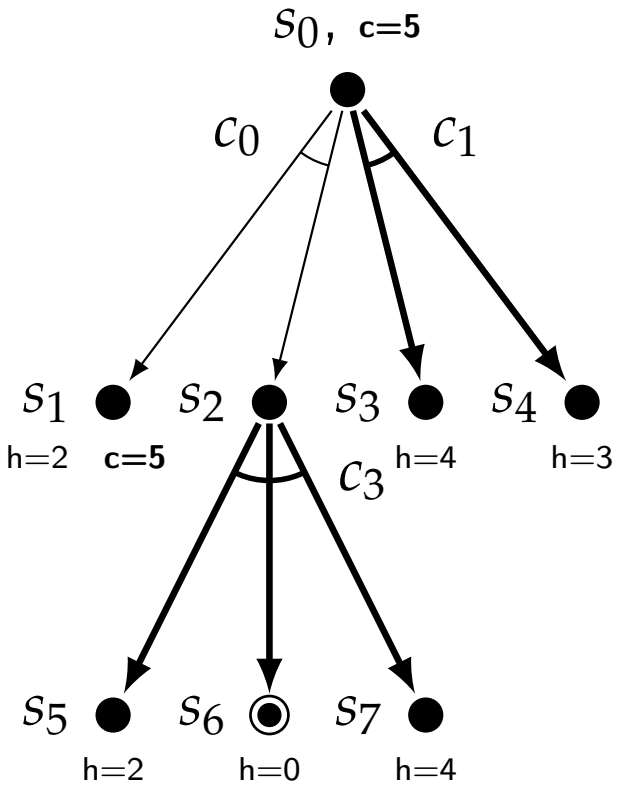
- Aktualisiere Kostenwerte der Knoten und setzte Markierungen um.
- Optimaler Kostenwert für $v \in V$: $cost^*(v) := \min_{\mathcal{G}} height(\mathcal{G})$, wobei die $\mathcal{G}$ Lösungsgraphen für v sind.

AO*-Algorithmus, Beispiel



AO*-Algorithmus, Beispiel





Vorverarbeitung:

- Problem durch Abstraktion vereinfachen.
- Für jede Abstraktion: Problem vollständig lösen.
- Für jeden abstrakten Zustand: Kosten in Pattern-Database ablegen.

Während der Suche:

- Zugriff auf die abstrakten Zustände, um Kosten für Heuristik zu verwenden.

Vorverarbeitung:

- Problem durch Abstraktion vereinfachen.
- Für jede Abstraktion: Problem vollständig lösen.
- Für jeden abstrakten Zustand: Kosten in Pattern-Database ablegen.

Während der Suche:

- Zugriff auf die abstrakten Zustände, um Kosten für Heuristik zu verwenden.

Vorverarbeitung:

- Problem durch Abstraktion vereinfachen.
- Für jede Abstraktion: Problem vollständig lösen.
- Für jeden abstrakten Zustand: Kosten in Pattern-Database ablegen.

Während der Suche:

- Zugriff auf die abstrakten Zustände, um Kosten für Heuristik zu verwenden.

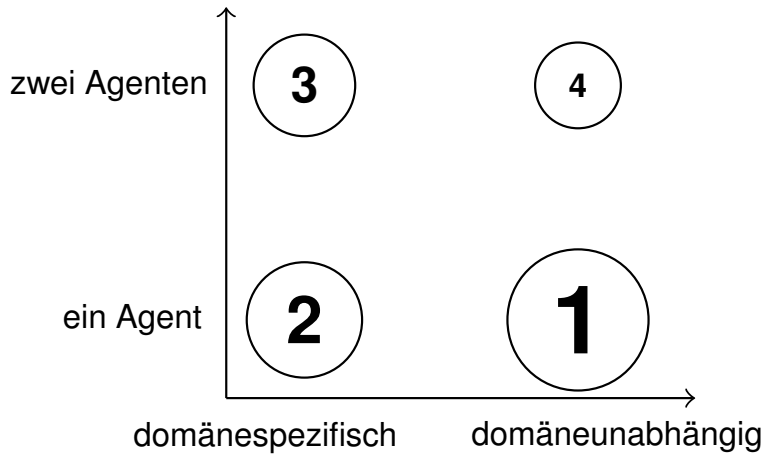
Vorverarbeitung:

- Problem durch Abstraktion vereinfachen.
- Für jede Abstraktion: Problem vollständig lösen.
- Für jeden abstrakten Zustand: Kosten in Pattern-Database ablegen.

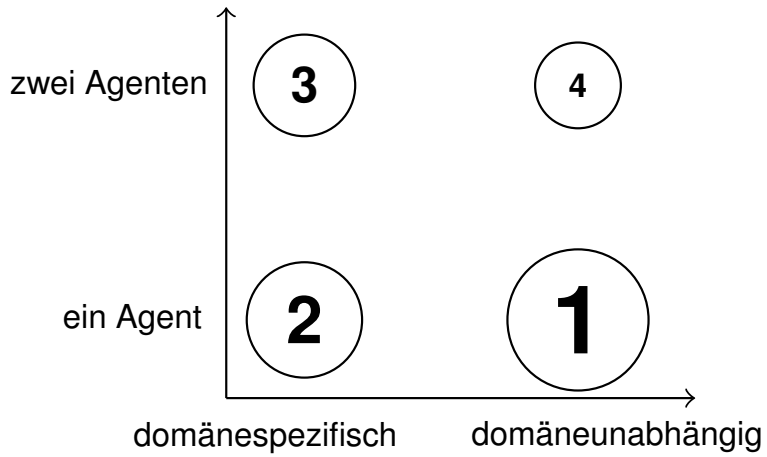
Während der Suche:

- Zugriff auf die abstrakten Zustände, um Kosten für Heuristik zu verwenden.

Existierende Arbeiten



(1) Prieditis (1993), Edelkamp (2001,2002,2006), Felner u.a. (2004), Haslum u.a. (2005, 2007)
(2) Culberson & Schaeffer (1996,1998), Korf & Felnder (2002), Felner u.a. (2004)
(3) Samadi u.a. (2008)
(4) keine bekannt



(1) Prieditis (1993), Edelkamp (2001,2002,2006), Felner u.a. (2004), Haslum u.a. (2005, 2007)
(2) Culberson & Schaeffer (1996,1998), Korf & Felnder (2002), Felner u.a. (2004)
(3) Samadi u.a. (2008)
(4) keine bekannt

Definition:
Die Abstraktion $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ ist das Planungsproblem $\mathcal{P}$, eingeschränkt auf das Pattern $P_i \subseteq Var$.

Es gilt:

- $Var^i := P_i$,
- Für $s \in S$ ist $s^i := s \cap P_i$, bzw. für $var \subseteq Var$ ist $var^i := var \cap P_i$.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ für $a \in A$. Dann ist $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
Die Abstraktion $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ ist das Planungsproblem $\mathcal{P}$, eingeschränkt auf das Pattern $P_i \subseteq Var$.

Es gilt:

- $Var^i := P_i$,
- Für $s \in S$ ist $s^i := s \cap P_i$, bzw. für $var \subseteq Var$ ist $var^i := var \cap P_i$.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ für $a \in A$. Dann ist $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
Die Abstraktion $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ ist das Planungsproblem $\mathcal{P}$, eingeschränkt auf das Pattern $P_i \subseteq Var$.

Es gilt:

- $Var^i := P_i$,
- Für $s \in S$ ist $s^i := s \cap P_i$, bzw. für $var \subseteq Var$ ist $var^i := var \cap P_i$.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ für $a \in A$. Dann ist $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
Die Abstraktion $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ ist das Planungsproblem $\mathcal{P}$, eingeschränkt auf das Pattern $P_i \subseteq Var$.

Es gilt:

- $Var^i := P_i$,
- Für $s \in S$ ist $s^i := s \cap P_i$, bzw. für $var \subseteq Var$ ist $var^i := var \cap P_i$.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ für $a \in A$. Dann ist $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Definition:
Die Abstraktion $\mathcal{P}^i = (Var^i, A^i, s_0^i, G^i)$ ist das Planungsproblem $\mathcal{P}$, eingeschränkt auf das Pattern $P_i \subseteq Var$.

Es gilt:

- $Var^i := P_i$,
- Für $s \in S$ ist $s^i := s \cap P_i$, bzw. für $var \subseteq Var$ ist $var^i := var \cap P_i$.
- $a^i := \langle pre(a)^i, \{ \langle add^i, del^i \rangle \mid \langle add, del \rangle \in eff(a) \} \rangle$ für $a \in A$. Dann ist $A^i := \{ a^i \mid a \in A \}$.
- $G^i := \{ g^i \mid g \in G \}$.

Für einen Zustand s^i und $s \in S$ ergibt sich die Heuristik h^i für s^i durch

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

wobei die $\mathcal{G}$ Lösungsgraphen für s^i sind.

Berechnung durch:

- Erstellung des vollständigen UND/ODER-Graphen von $\mathcal{P}^i$.
- Lösung durch Value-Iteration.

Frage:
Auf welche Weise Heuristik $h(s)$ berechnen, falls mehrere Patterns gegeben sind?

Für einen Zustand s^i und $s \in S$ ergibt sich die Heuristik h^i für s^i durch

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

wobei die $\mathcal{G}$ Lösungsgraphen für s^i sind.

Berechnung durch:

- Erstellung des vollständigen UND/ODER-Graphen von $\mathcal{P}^i$.
- Lösung durch Value-Iteration.

Frage:
Auf welche Weise Heuristik $h(s)$ berechnen, falls mehrere Patterns gegeben sind?

Für einen Zustand s^i und $s \in S$ ergibt sich die Heuristik h^i für s^i durch

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

wobei die $\mathcal{G}$ Lösungsgraphen für s^i sind.

Berechnung durch:

- Erstellung des vollständigen UND/ODER-Graphen von $\mathcal{P}^i$.
- Lösung durch Value-Iteration.

Frage:
Auf welche Weise Heuristik $h(s)$ berechnen, falls mehrere Patterns gegeben sind?

Bestimmung des Heuristikwerts (1)

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$.

Es gilt: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ ist zulässig.

Frage:
Wie kann eine informiertere Heuristik gewonnen werden?

Antwort:
Durch Ausnutzung von Additivität.

Bestimmung des Heuristikwerts (1)

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$.

Es gilt: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ ist zulässig.

Frage:
Wie kann eine informiertere Heuristik gewonnen werden?

Antwort:
Durch Ausnutzung von Additivität.

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$.

Es gilt: $h_1(s) := \max_{i \in \{1,\ldots,m\}} h^i(s^i)$ ist zulässig.

Frage:
Wie kann eine informiertere Heuristik gewonnen werden?

Antwort:
Durch Ausnutzung von Additivität.

Definition:
Die Menge $\{P_1, \ldots, P_k\}$ mit $k \in \mathbb{N}$ heißt additiv, falls für alle Zustände $s \in S$ gilt $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Im klassischen Planen gilt nach Korf & Felner (2002) und nach Edelkamp (2001):

Satz:
Falls für alle $a \in A$ und für alle Patterns $P_i \in P$ gilt: *Wenn* $P_i \cap effvar(a) \neq \emptyset$, *dann gilt für alle* $P_j \in P$ *mit* $P_j \neq P_i$, *dass* $P_j \cap effvar(a) = \emptyset$, dann sind die Patterns aus P additiv.

Dieser Satz gilt auch für nichtdeterministische Planungsprobleme.

Definition:
Die Menge $\{P_1, \ldots, P_k\}$ mit $k \in \mathbb{N}$ heißt additiv, falls für alle Zustände $s \in S$ gilt $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Im klassischen Planen gilt nach Korf & Felner (2002) und nach Edelkamp (2001):

Satz:
Falls für alle $a \in A$ und für alle Patterns $P_i \in P$ gilt: *Wenn* $P_i \cap effvar(a) \neq \emptyset$, *dann gilt für alle* $P_j \in P$ *mit* $P_j \neq P_i$, *dass* $P_j \cap effvar(a) = \emptyset$, dann sind die Patterns aus P additiv.

Dieser Satz gilt auch für nichtdeterministische Planungsprobleme.

Definition:
Die Menge $\{P_1, \ldots, P_k\}$ mit $k \in \mathbb{N}$ heißt additiv, falls für alle Zustände $s \in S$ gilt $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Im klassischen Planen gilt nach Korf & Felner (2002) und nach Edelkamp (2001):

Satz:
Falls für alle $a \in A$ und für alle Patterns $P_i \in P$ gilt: *Wenn* $P_i \cap effvar(a) \neq \emptyset$, *dann gilt für alle* $P_j \in P$ *mit* $P_j \neq P_i$, *dass* $P_j \cap effvar(a) = \emptyset$, dann sind die Patterns aus P additiv.

Dieser Satz gilt auch für nichtdeterministische Planungsprobleme.

Definition:
Die Menge $\{P_1, \ldots, P_k\}$ mit $k \in \mathbb{N}$ heißt additiv, falls für alle Zustände $s \in S$ gilt $\sum_{i=1}^{k} h^i(s^i) \leq cost^*(s)$.

Im klassischen Planen gilt nach Korf & Felner (2002) und nach Edelkamp (2001):

Satz:
Falls für alle $a \in A$ und für alle Patterns $P_i \in P$ gilt: *Wenn* $P_i \cap effvar(a) \neq \emptyset$, *dann gilt für alle* $P_j \in P$ *mit* $P_j \neq P_i$, *dass* $P_j \cap effvar(a) = \emptyset$, dann sind die Patterns aus P additiv.

Dieser Satz gilt auch für nichtdeterministische Planungsprobleme.

Bestimmung des Heuristikwerts (2)

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$ und Partition M in additive Patterns, d.h. $M := \{M_1, \ldots, M_m\}$ Mengen von Patterns mit $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ und die M_i sind additiv.

(Nun: Identifikation von s^{P_i} und h^{P_i} mit s^i und h^i.)

Es gilt:

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

Bestimmung des Heuristikwerts (2)

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$ und Partition M in additive Patterns, d.h. $M := \{M_1, \ldots, M_m\}$ Mengen von Patterns mit $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ und die M_i sind additiv.

(Nun: Identifikation von s^{P_i} und h^{P_i} mit s^i und h^i.)

Es gilt:

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

gegeben:
Menge P von Patterns $P := \{P_1 \ldots, P_m\}$ und Partition M in additive Patterns, d.h. $M := \{M_1, \ldots, M_m\}$ Mengen von Patterns mit $\bigcup_{i \in \{1,\ldots,m\}} M_i = P$ und die M_i sind additiv.

(Nun: Identifikation von s^{P_i} und h^{P_i} mit s^i und h^i.)

Es gilt:

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

Offene Frage:

Wie wird die Menge P der Patterns gefunden, bzw. eine geeignete Partition M von P?

$\rightarrow$

(Noch) manuell. Automatisierung mittels lokaler Suche prinzipiell möglich. Vgl. Haslum u.a. (2005,2007), Edelkamp (2006).

Offene Frage:

Wie wird die Menge P der Patterns gefunden, bzw. eine geeignete Partition M von P?

$\rightarrow$

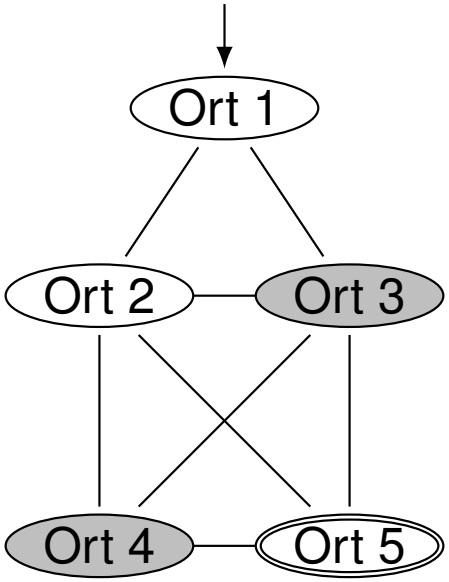
(Noch) manuell. Automatisierung mittels lokaler Suche prinzipiell möglich. Vgl. Haslum u.a. (2005,2007), Edelkamp (2006).

- Tireworld
  (keine Additivität)
- Chain-of-Rooms
  (Additivität: sequentielle Subprobleme)
- Coin-Flip
  (Additivität: parallele Subprobleme)

- Tireworld
  (keine Additivität)
- Chain-of-Rooms
  (Additivität: sequentielle Subprobleme)
- Coin-Flip
  (Additivität: parallele Subprobleme)

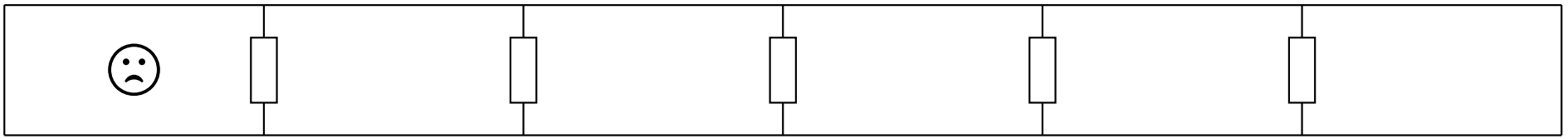
- Tireworld
  (keine Additivität)
- Chain-of-Rooms
  (Additivität: sequentielle Subprobleme)
- Coin-Flip
  (Additivität: parallele Subprobleme)

Tireworld, Problembeschreibung



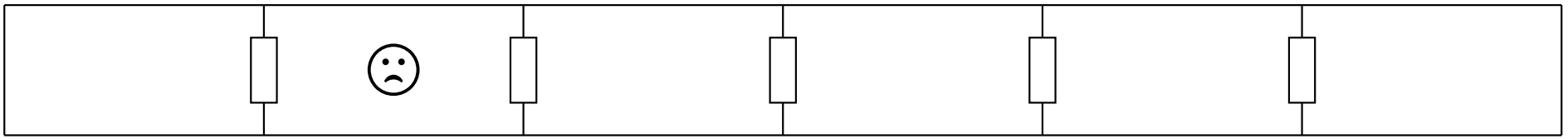
Aktionen:

- Straße benutzen (nichtdeterministisch: Platten).
- Ersatzrad aufnehmen (falls vorhanden).
- Rad austauschen (falls Platten und Ersatzrad).



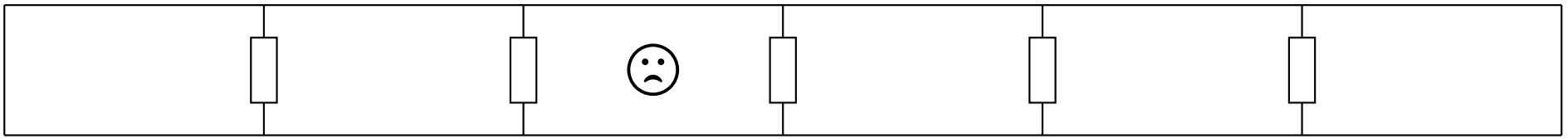
Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.



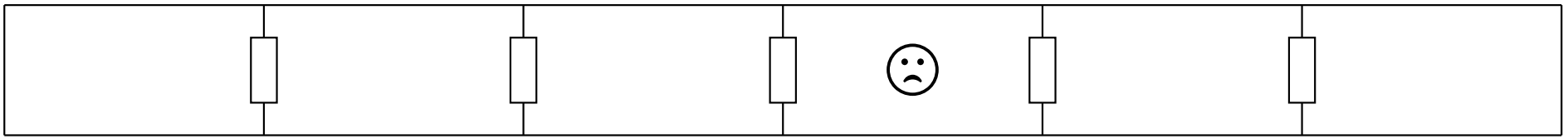
Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.



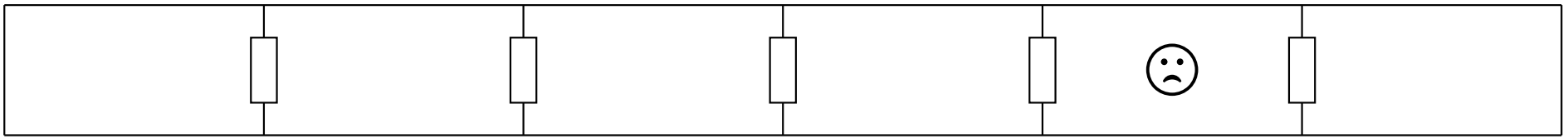
Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.



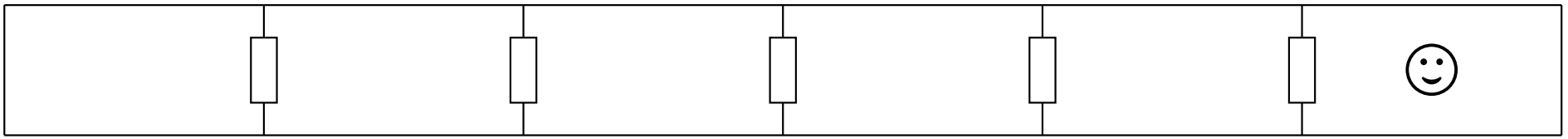
Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.



Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.



Aktionen:

- Gehe in benachbarten Raum, falls Tür offen.
- Schalte das Licht ein, falls es aus ist (nichtdeterministisch: Tür offen/geschlossen).
- Öffne Tür.

gegeben:

- *n* Münzen, die sich initial in einem Beutel befinden.
- Aktionen:
  - Münze werfen, falls sie noch nicht geworfen wurde.
  - Münze umdrehen, falls sie bereits geworfen wurde.

gesucht:

Strategie, damit alle Münzen Kopf zeigen.

gegeben:

- *n* Münzen, die sich initial in einem Beutel befinden.
- Aktionen:
  - Münze werfen, falls sie noch nicht geworfen wurde.
  - Münze umdrehen, falls sie bereits geworfen wurde.

gesucht:

Strategie, damit alle Münzen Kopf zeigen.

gegeben:

- *n* Münzen, die sich initial in einem Beutel befinden.
- Aktionen:
  - Münze werfen, falls sie noch nicht geworfen wurde.
  - Münze umdrehen, falls sie bereits geworfen wurde.

gesucht:

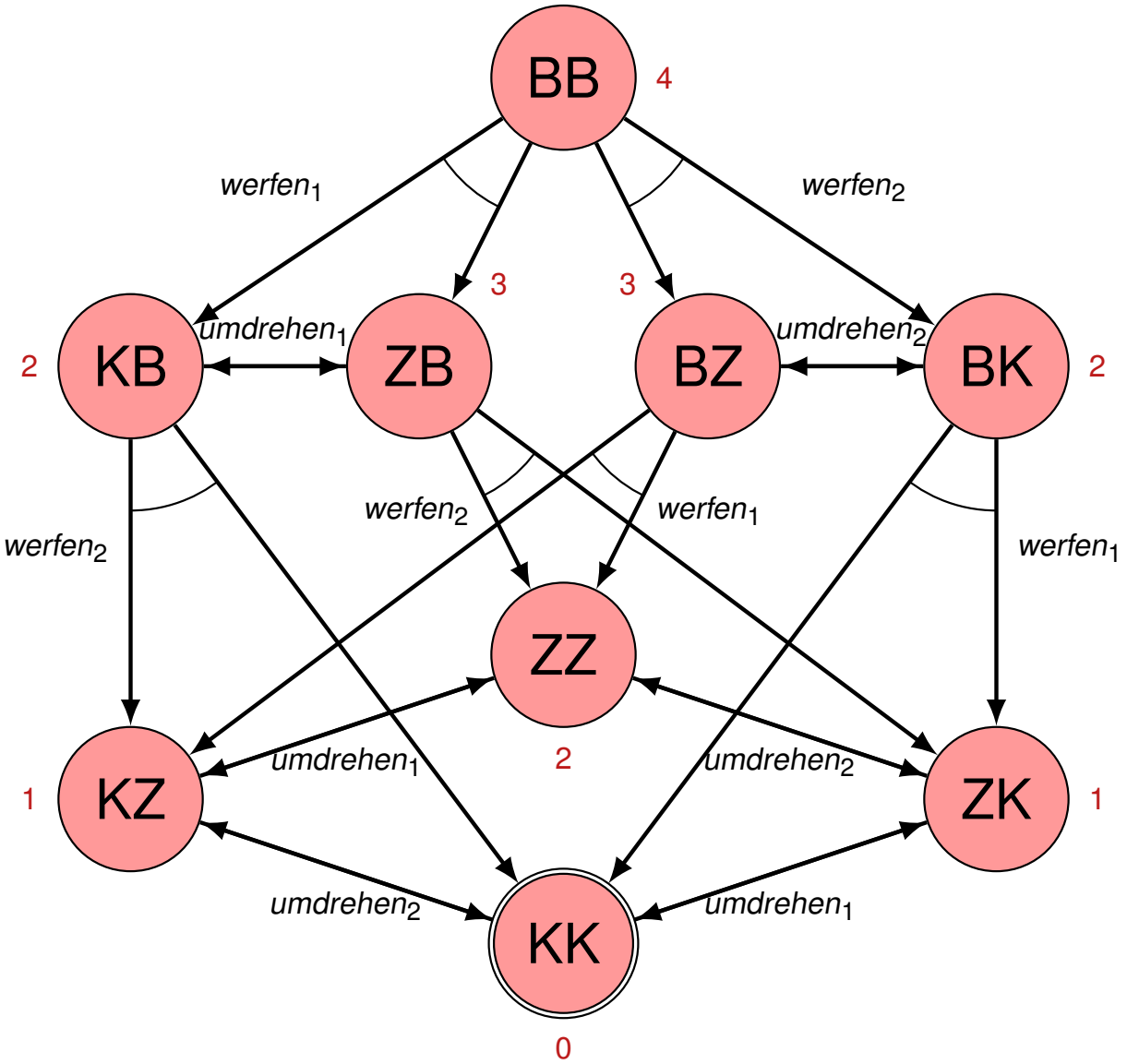
Strategie, damit alle Münzen Kopf zeigen.

gegeben:

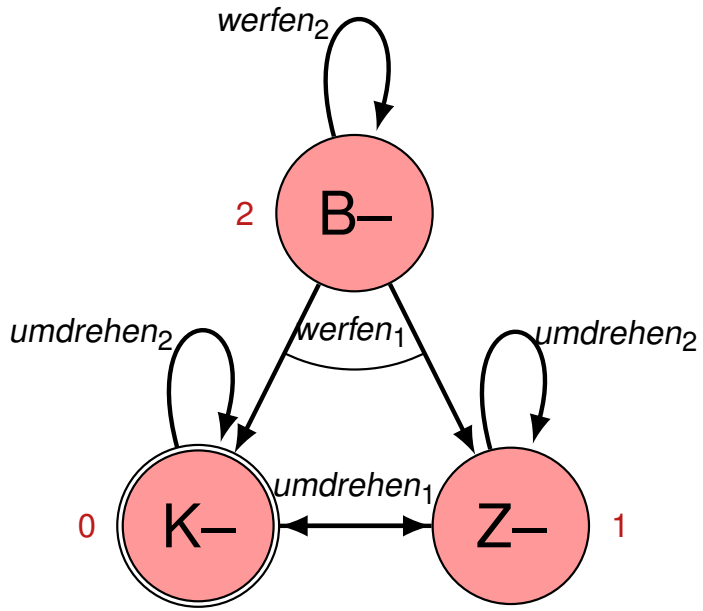
- *n* Münzen, die sich initial in einem Beutel befinden.
- Aktionen:
  - Münze werfen, falls sie noch nicht geworfen wurde.
  - Münze umdrehen, falls sie bereits geworfen wurde.

gesucht:
Strategie, damit alle Münzen Kopf zeigen.

Coin-Flip (zwei Münzen), Beispiel für additive Patterns

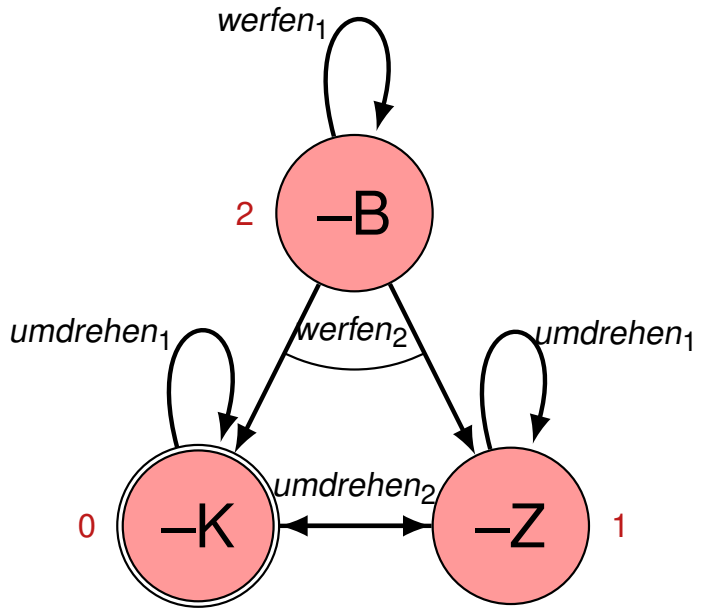


Coin-Flip (zwei Münzen), Beispiel für additive Patterns



$$h^{P_1}(BB^{P_1}) = h^{P_1}(B-) = 2$$

Coin-Flip (zwei Münzen), Beispiel für additive Patterns



$$h^{P_2}(BB^{P_2}) = h^{P_2}(-B) = 2 \qquad \Rightarrow$$

$$h(BB) = h(BB^{P_1}) + h(BB^{P_2}) = h^{P_2}(B-) + h^{P_2}(-B) = 2 + 2 = 4$$

Vergleich zwischen:

- AO*-Algorithmus und *keiner Heuristik*,
- AO*-Algorithmus und *adversariale FF-Heuristik*,
- AO*-Algorithmus und *Pattern-Database-Heuristik*,

Vergleiche möglich mit:

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

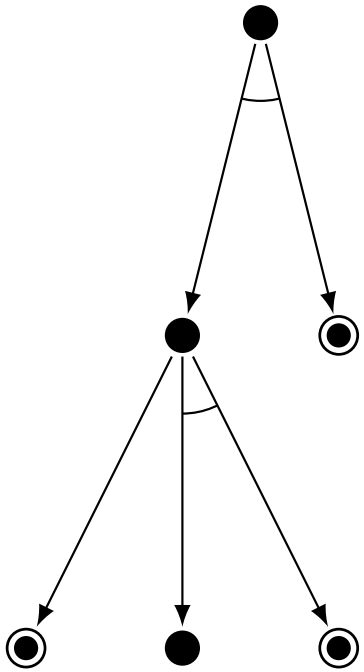
Vergleich zwischen:

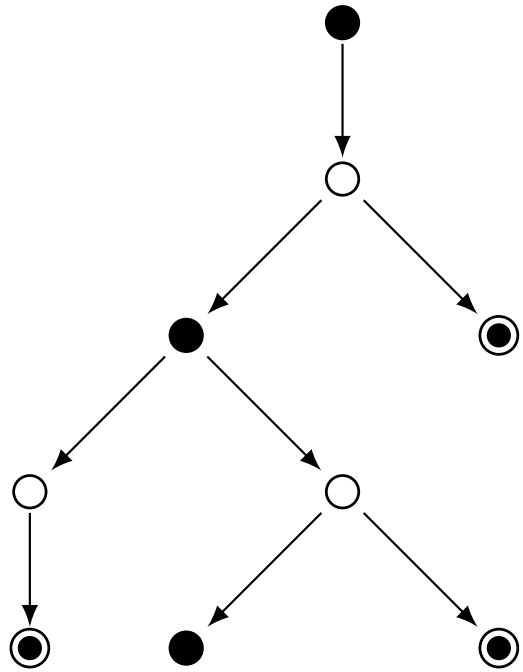
- AO*-Algorithmus und *keiner Heuristik*,
- AO*-Algorithmus und *adversariale FF-Heuristik*,
- AO*-Algorithmus und *Pattern-Database-Heuristik*,

Vergleiche möglich mit:

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

Der verwendete Planer nutzt explizite UND- und ODER-Knoten, statt einer Sorte Knoten und Konnektoren.





Ergebnisse: Tireworld

#Orte	uninformiert			adv. FF-Heuristik			PDB-Heuristik				
	Zeit	RAM	Knoten	Zeit	RAM	Knoten	Zeit pre	Zeit Suche	Zeit	RAM	Knoten
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**

– := ungenügend Arbeitsspeicher (4 GB)

Zehn Patterns: Jedes Pattern enthält den Zielort, das Ersatzrad, den Platten und sieben zufällige Orte.

Ergebnisse: Chain-of-Rooms

#Räume	uninformiert			adv. FF-Heuristik			PDB-Heuristik				
	Zeit	RAM	Knoten	Zeit	RAM	Knoten	Zeit pre	Zeit Suche	Zeit	RAM	Knoten
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**

– := ungenügend Arbeitsspeicher (4 GB)

Patterns: Vier benachbarte Räume pro Pattern.

Ergebnisse: Coin-Flip

#Münzen	uninformiert			adv. FF-Heuristik			PDB-Heuristik				
	Zeit	RAM	Knoten	Zeit	RAM	Knoten	Zeit pre	Zeit Suche	Zeit	RAM	Knoten
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	–	–	–	–

– := ungenügend Arbeitsspeicher (4 GB)

Patterns: Zwei Münzen pro Pattern.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Zusammenfassung:

- Additivitätskriterium auch auf nichtdeterministisches Planen übertragbar.
- Pattern-Database-Heuristiken können gute Ergebnisse zeigen. (Insbesondere bei Dekomposition.)

Ausblick:

- Finden von starken *zyklischen* Plänen.
- Automatisierte Pattern-Selektion
- Umstellung auf mehrwertige Variablen
- Effizienzsteigerung, z.B. durch Umstellung auf Konnektoren, statt UND/ODER-Zuständen.

Danke für Ihre Aufmerksamkeit!


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.


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.

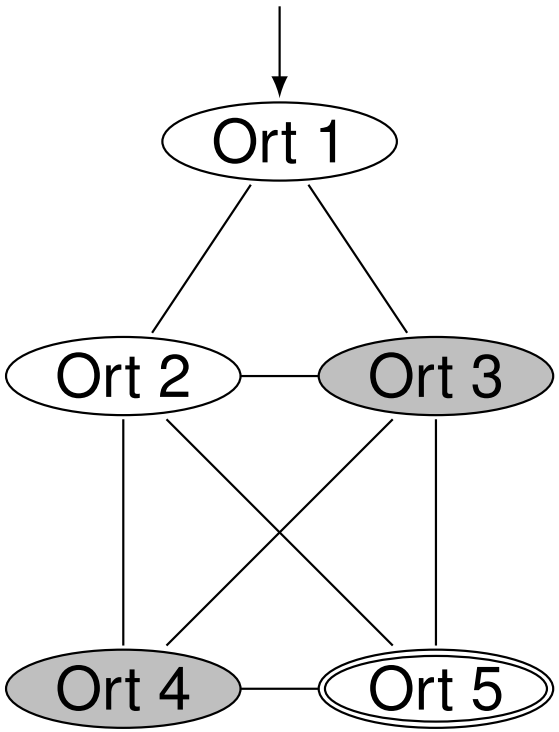
gegeben:

- Ungerichteter Graph (V, E) und
  - $v_0 \in V$ Startort,
  - $v_g \in V$ Zielort,
  - $V_E \subseteq V$ Orte mit Ersatzreifen.
- Aktionen:
  - Von v nach v' fahren, falls $(v, v') \in V$. (Nichtdeterministisch: Platten)
  - Ersatzrad aufnehmen (falls vorhanden).
  - Rad austauschen (falls Platten und Ersatzrad)

gesucht:

Strategie, um von v_0 nach v_g zu gelangen.

Beispiel:



gegeben:

- Ungerichteter Graph (V, E) mit
  - $V = \{v_0, \ldots, v_n\}$ (Räume) und
  - $E = \{ \{v_i, v_{i+1}\} \mid i \in \{0, \ldots, n-1\} \}$ (Türen)
  - $status(e) = unbekannt$ für alle $e \in E$.
- Aktionen:
  - Gehe von v nach v', falls $e = \{v, v'\} \in E$ und falls $status(e) = offen$.
  - Betrachte Tür $e \in E$, falls $status(e) = unbekannt$. Nichtdet. Effekt: $status(e) \in \{offen, geschlossen\}$.
  - Öffne Tür e, falls $status(e) \in \{offen, geschlossen\}$. Effekt: $status(e) = offen$.

gesucht:

Strategie, um von Raum v_0 zu Raum v_n zu gelangen.