

Seminar AI Tools

WS 06/07

Thema:

„HTN Planning in JSHOP und JSHOP2“

Sandro Castronovo

16.11.2006

Einführung in HTN Planning

vom Task zum Plan

Beispiel

SHOP

Syntax und Semantik

SHOP Algorithmus

Beispiel mit konkreter Domäne

SHOP2

Warum SHOP2

Unterschiede zu SHOP

SHOP vs. SHOP2

Zusammenfassung

JSHOP2 Gui

Einführung in HTN Planning

vom Task zum Plan

Beispiel

SHOP

Syntax und Semantik

SHOP Algorithmus

Beispiel mit konkreter Domäne

SHOP2

Warum SHOP2

Unterschiede zu SHOP

SHOP vs. SHOP2

Zusammenfassung

JSHOP2 Gui

Einführung in HTN Planning

vom Task zum Plan

Beispiel

SHOP

Syntax und Semantik

SHOP Algorithmus

Beispiel mit konkreter Domäne

SHOP2

Warum SHOP2

Unterschiede zu SHOP

SHOP vs. SHOP2

Zusammenfassung

JSHOP2 Gui

Einführung in HTN Planning

vom Task zum Plan

Beispiel

SHOP

Syntax und Semantik

SHOP Algorithmus

Beispiel mit konkreter Domäne

SHOP2

Warum SHOP2

Unterschiede zu SHOP

SHOP vs. SHOP2

Zusammenfassung

JSHOP2 Gui

Einführung in HTN Planning

vom Task zum Plan

Beispiel

SHOP

Syntax und Semantik

SHOP Algorithmus

Beispiel mit konkreter Domäne

SHOP2

Warum SHOP2

Unterschiede zu SHOP

SHOP vs. SHOP2

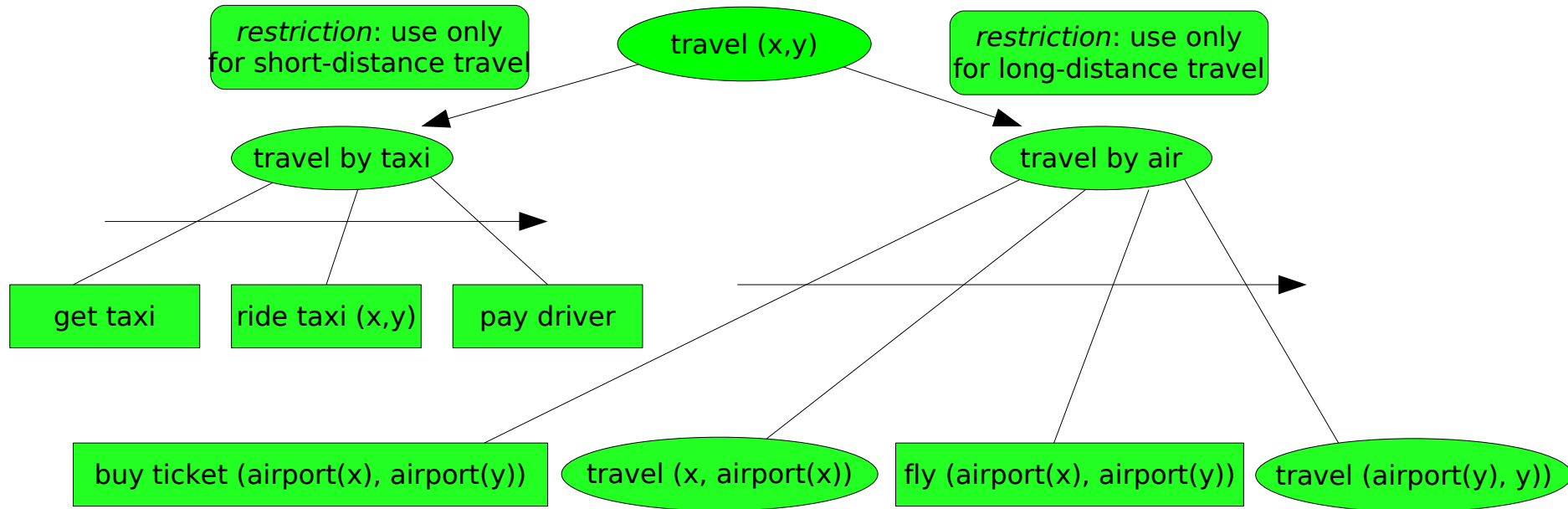
Zusammenfassung

JSHOP2 Gui

- HTN (Hierarchical Task Network)
- HTN Planning teilt Aufgaben in kleinere Aufgaben, bis primitive Elemente erreicht werden
- Jeder HTN Planer hat eine Wissensdatenbank mit Methoden, auf die er zurückgreift
- Jede Methode enthält eine Beschreibung, wie diese in kleinere Aufgaben unterteilt werden kann

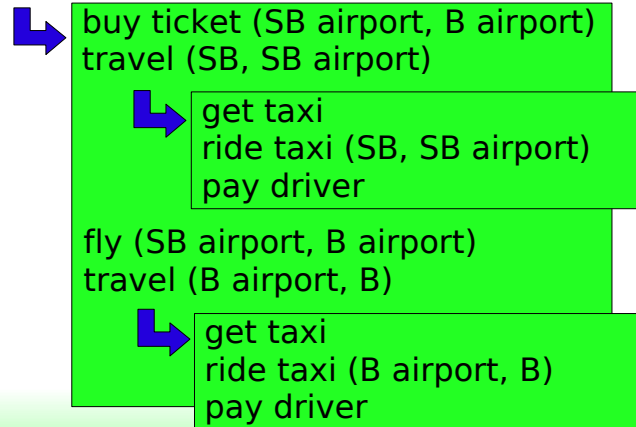
- backward-chaining und partial-order-planning nicht immer die beste Lösung für praktische Probleme
- HTN Planning benutzt eine total-order Strategie
- total-order Planning: Die einzelnen Schritte werden in der selben Reihenfolge geplant, wie sie später ausgeführt werden.

HTN Planning vom Task zum Plan



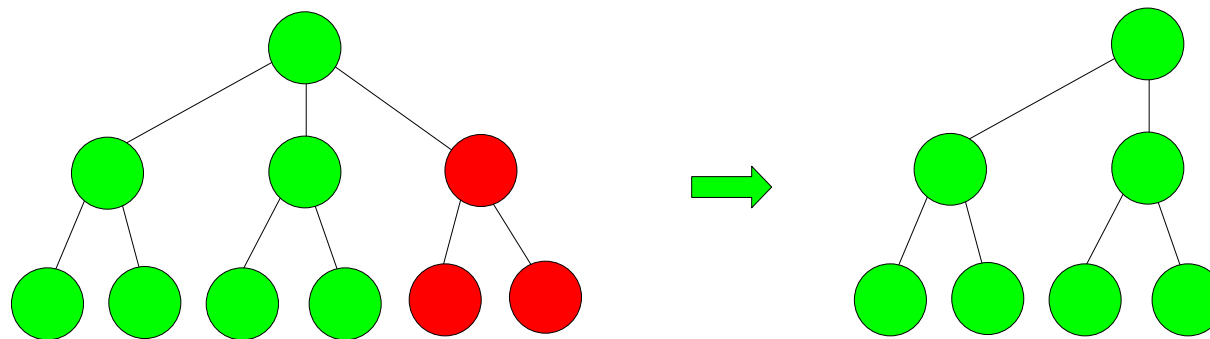
Task: travel(Saarbrücken, Berlin)

travel(SB,B)



Beispieldomäne für HTN: Das Kartenspiel Bridge

- Kartenspiel, das auf Strategien beruht (finessing, ruffing, etc.)
- Erstellen der Strategien über HTN Dekomposition (Bäume)
- Jeder Knoten im Baum repräsentiert einen möglichen Spielzug innerhalb der entsprechenden Strategie

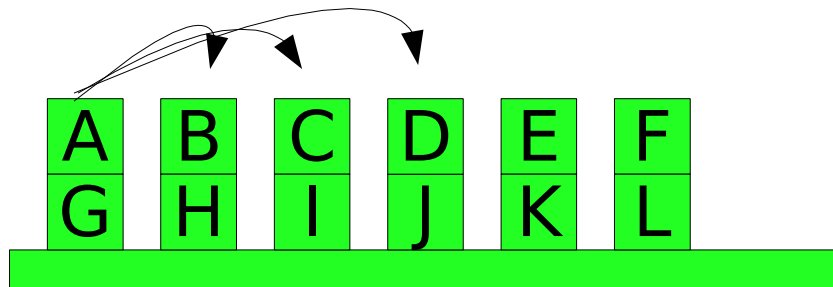


HTN: worst case: 305.000 Knoten im Baum

„Brute force“ Methode: worst case: $5.55 * 10^{24}$ Knoten

Aber: Abweichungen von der Strategie nicht
nicht berücksichtigt

backward/forward chaining

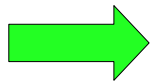


Startzustand



Zielzustand

- Vorwärtssuche sucht alle mögliche Zustände ab
- 36 Möglichkeiten => ineffizient
- HTN Planer suchen nicht alle Möglichkeiten ab. Nur die, die im HTN vorkommen



HTN Planer wissen beim Planen schon wie der Zustand der Welt sein wird, wenn der Task dann letztendlich ausgeführt wird

Eigenschaften von SHOP

- Plant seine Tasks in der selben Reihenfolge, in der sie später ausgeführt werden
- SHOP kennt zu jedem Zeitpunkt in der Planung den gesamten Zustand der Welt
- effiziente Domänenrepräsentation

SHOP

Syntax und Semantik

- › SHOP benutzt first-order-logic
- › Ein Zustand (*state*) ist eine Menge von Atomen (Fakten)
- › Ein *task* hat die allgemeine Form

$$(s \ t_1 \ t_2 \ t_3 \ \dots \ t_n)$$

wobei

s = Tasksymbol (Taskname)

$t_1 \ t_2 \ t_3 \ \dots \ t_n$ = Terme (Taskargumente)

- › primitiver Task: s beginnt mit „!“
- › zusammengesetzter Task: s beginnt nicht mit einem Sonderzeichen
- › eine Task Liste ist eine Menge von Tasks

Beispiel: (!walk ?here ?there)

- Es gibt *Axiome, Operatoren* und *Methoden*
- Ein *Axiom* ist eine Konjunktion von Hornklauseln

Notation:

$$\begin{aligned} & (:- (P) \\ & \quad ((H1) \\ & \quad \quad (H2) \\ & \quad \quad \cdot \cdot \\ & \quad \quad (Hn))) \end{aligned}$$

- P ist wahr, wenn H1 oder H2 oder ... Hn wahr

Operator

- › Mit einem *Operator* wird durch Manipulationen am Weltzustand der Zustand erreicht, den der Operator in seinem Kopf (head) definiert.

Syntax in SHOP: (`:operator h D A`)

wobei

`h` = head, primitiver Task

`D, A` = Deletions, Additions. Menge von Atomen, die keine anderen Variablen enthalten als `h`

```
(:operator (!putdown ?block)
  ((holding ?block))
  ((ontable ?block) (handempty)))
```

Methode

- Vorbedingungen einer *Methode* erfüllt => Zustand, der im Kopf der Methode steht, erreichbar durch Tasks, die die Methode enthält

Syntax in SHOP: (`:method h C T`)

wobei

h = head, zusammengesetzter Task

C = Vorbedingung, Menge von Konjunktionen

T = Tail, Taskliste

```
(:method (make-clear ?y) ((clear ?y)) nil
(:method (make-clear ?y)
  ((on ?x ?y))
  ((make-clear ?x)
  (!unstack ?x ?y) (!putdown ?x))
```

- Ein *Plan* ist eine Liste von Instanzen von Operatoren
- p *Plan* und S *Zustand*, dann ist $p(S)$ der Zustand, der von S aus mit p erreicht wird
- Ein *Planungsproblem* ist ein Tupel der Form
$$P = (S, T, D)$$

wobei

S = Zustand

T = Taskliste

D = Menge von Axiomen, Operatoren
und Methoden

$\prod (S, T, D)$ Menge aller Pläne für T von S mit D

ist rekursiv definiert als

T leer $\Rightarrow \prod (S, T, D)$ enthält nur den leeren Plan

sonst:

t := erster Task in T

R := verbleibende Tasks in T

unterscheide 3 Fälle

(1) Falls t primitiver task $\wedge \exists$ elementarer Plan q für t
 $\Rightarrow \prod (S, T, D) = \{ \text{append}(q, p) : p \in \prod (q(S), R, D) \}$

(2) Falls t primitiver Task $\wedge \neg \exists$ elementarer Plan p für t
 $\Rightarrow \prod (S, T, D) = \emptyset$

(3) Falls t zusammengesetzter Task:
 $\Rightarrow \prod (S, T, D) = \cup \{ \prod (S, \text{append}(r, R), D) : r \text{ einfache Reduktion von } t \}$

SHOP

Planungsalgorithmus

```
procedure find-plan (S,T,D)
  return seek-plan(S,T,D,nil)
end find-plan
```

```
procedure seek-plan (S,T,D,p)
  if T = nil then return the list (p)
  t = the first task in T; R = the remaining tasks
  if t is primitive then
    if there is a simple plan q for t then
      return seek-plan (q(S),R,D,append(p,q)) ①
    else return FAIL ②
  else
    for every simple reduction r for t in S
      result = seek-plan(S,append(r,R),D,p) ③
      if result != FAIL then return result
    end for
    return FAIL
  end if
end seek-plan
```

Transportproblem

- Tabelle 1: Domänenspezifikation. Axiome, Methoden und Operatoren
- Tabelle 2: Beispielproblem mit Startzustand und Taskliste

1. ((at downtown))
2. (weather-is good)
3. (have-cash 12)
4. (distance downtown park 2)
5. (distance downtown uptown 8)
6. (distance downtown suburb 12)
7. (at-taxi-stand taxi1 downtown)
8. (bus-route bus1 downtown park)
9. (bus-route bus2 downtown uptown)
10. (bus-route bus3 downtown suburb))

Taskliste: ((travel-to suburb))

gefundene Pläne

p = nil

Methode M2:

```
(:method (travel-to suburb)
  ((at downtown)
  (walking-distance downtown suburb))
```

Axiom A2:

```
(:- (walking-distance downtown suburb)
  ((weather-is good)
  (distance downtown suburb 12)
  (eval (<= 12 3))) => FAIL
```

1. ((at downtown)
2. (weather-is good)
3. (have-cash 12)
4. (distance downtown park 2)
5. (distance downtown uptown 8)
6. (distance downtown suburb 12)
7. (at-taxi-stand taxil downtown)
8. (bus-route bus1 downtown park)
9. (bus-route bus2 downtown uptown)
10. (bus-route bus3 downtown suburb))

Taskliste: ((travel-to suburb))

gefundene Pläne: p = nil

Methode M3:

```
(:method (travel-to suburb)
  ((at downtown)
   (at-taxi-stand taxil downtown)
   (distance downtown suburb 12)
   (have-taxi-fare 12)))
```

Axiom A1:

```
(:- (have-taxi-fare 12)
    ((have-cash 12)
     (eval (>= 12 (+ 1.5 12))))) => FAIL
```

SHOP

Beispieldomäne

1. ((at downtown)
2. (weather-is good)
3. (have-cash 12)
4. (distance downtown park 2)
5. (distance downtown uptown 8)
6. (distance downtown suburb 12)
7. (at-taxi-stand taxil1 downtown)
8. (bus-route bus1 downtown park)
9. (bus-route bus2 downtown uptown)
10. (bus-route bus3 downtown suburb))

Taskliste: ((travel-to suburb))

gefundene Pläne

p = ((~~((!wait-for bus3 downtown))~~)
 ((~~set-cash 11~~))
 (!ride-bus downtown suburb)))

Methode M3:

```
(:method (travel-to suburb)
  ((at downtown)(bus-route bus3 downtown suburb))
  ((!wait-for bus3 downtown)
    (pay-driver 1.00)
    (!ride-bus downtown suburb)))
```

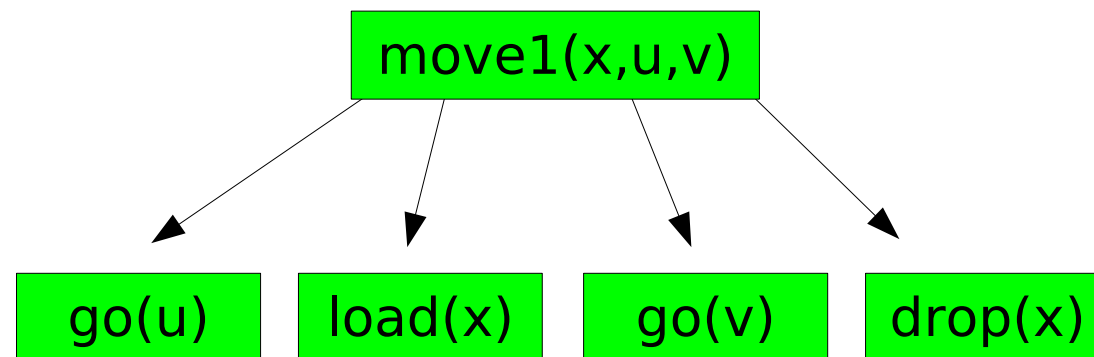
Methode M1:

```
(:method (pay-driver ?fare)
  ((have-cash 12)
    (eval (>= 12 1.00)))
  ((!set-cash 11,(- 12 1.00))))
```

- › aufwärtskompatibel zu SHOP
(mit geringen syntaktischen Änderungen)
- › Probleme bei SHOP: Sehr hoher Aufwand gute Wissensdatenbanken zu schreiben
- › SHOP wurde bei einem Wettbewerb disqualifiziert, weil das debugging der Wissensdatenbank zu lange gedauert hat
- › total-order-Restriktion von SHOP macht es unmöglich Unteraufgaben von Tasks zu verschachteln

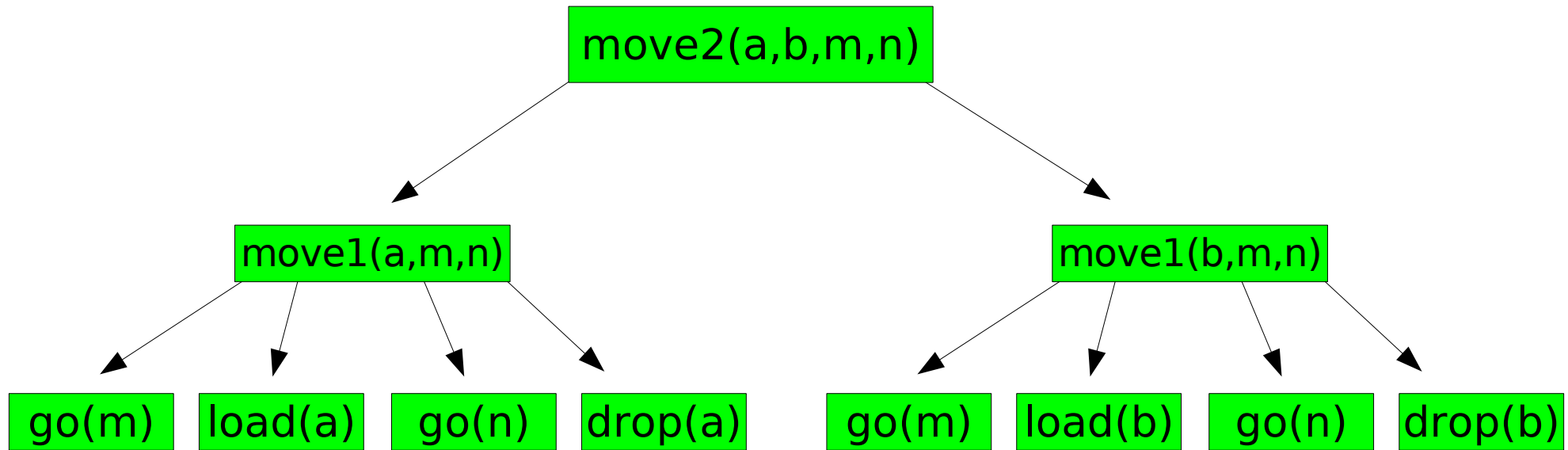
 Beispiel

- Ziel: zwei Pakete von a nach b zu transportieren
- SHOP braucht hierzu eine Methode:



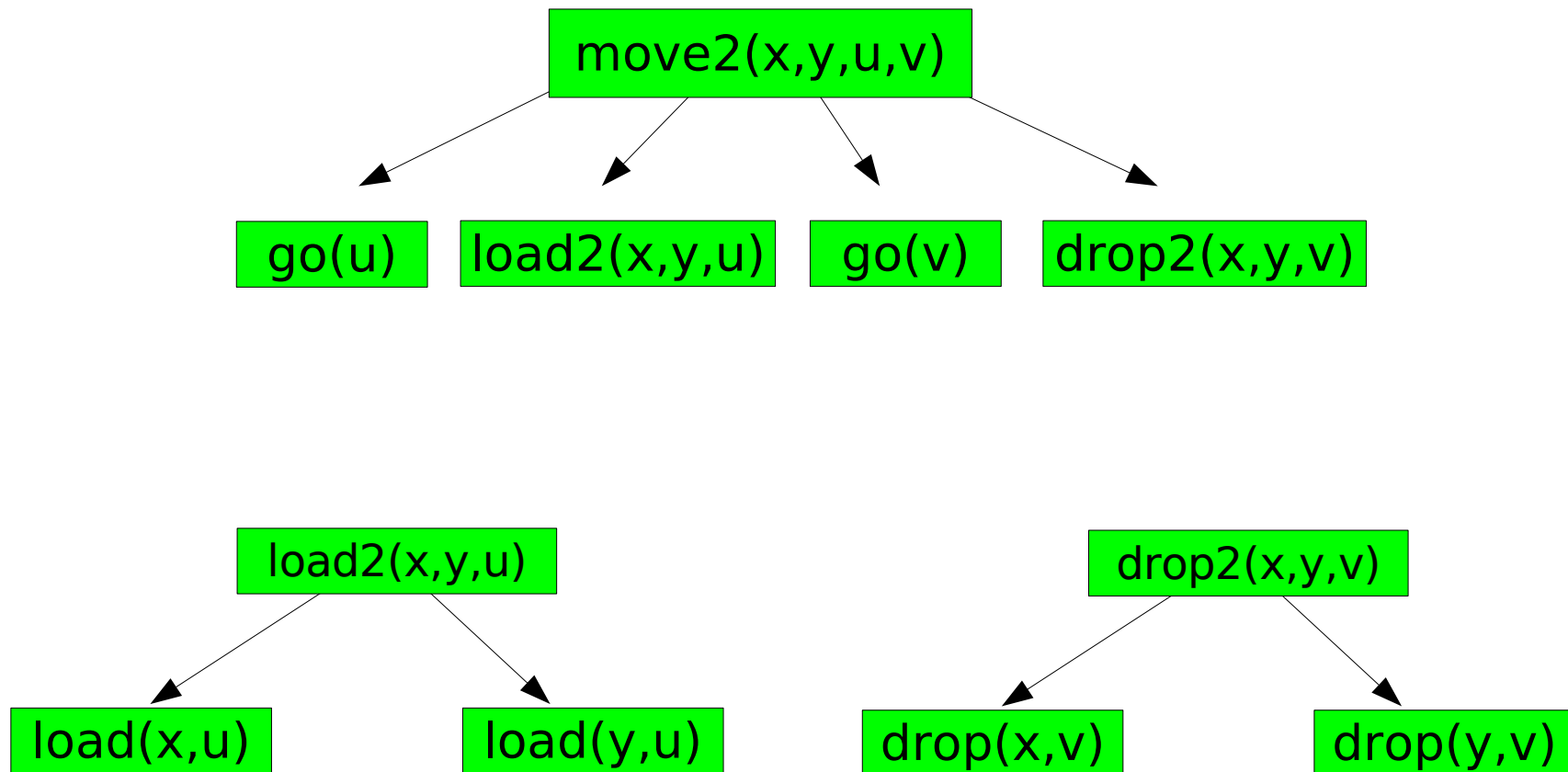
- Klappt gut, wenn ein Paket zu bewegen ist
- Was aber, wenn zwei Pakete von a nach b transportiert werden sollen?

Plan, den SHOP aus der Methode generiert

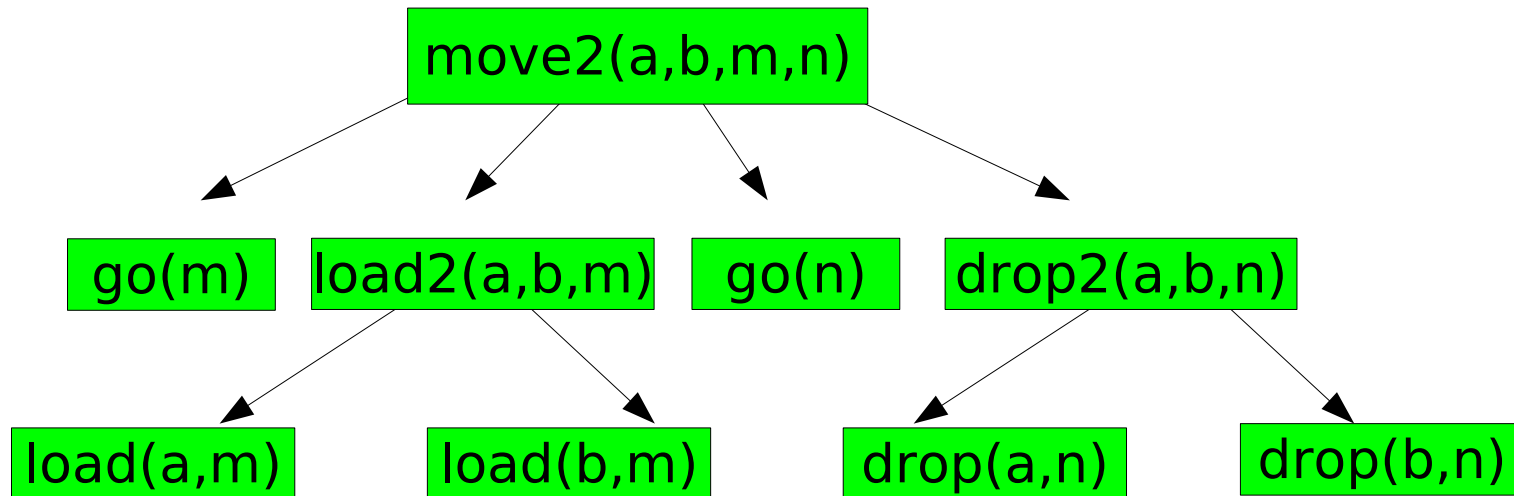


➔ ineffizient!

Besser wäre es, beide Pakete auf einmal zu transportieren:



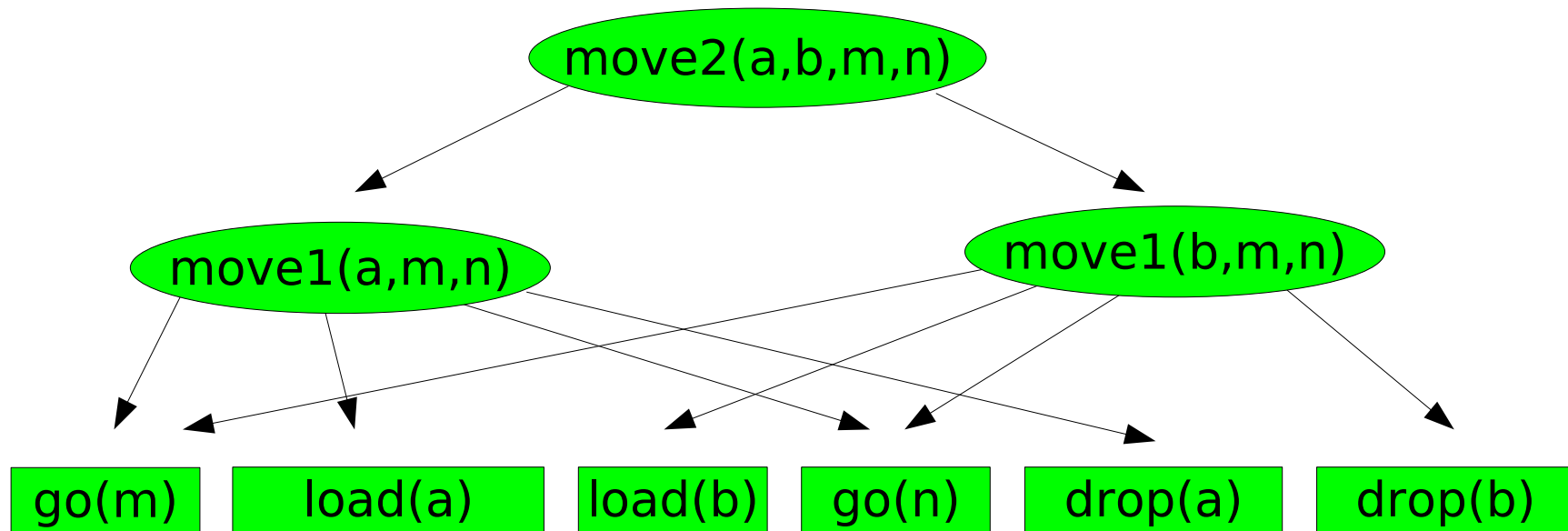
Dies resultiert im Plan



Aber: Es müssen explizit Methoden geschrieben werden, damit SHOP mit zwei Paketen umgehen kann

Und: Es handelt sich hier um ein einfaches Beispiel. Real treten komplexere Probleme auf

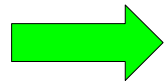
Effizienter wäre es, einen solchen Plan zu gestalten



 SHOP2

- › SHOP2 relaxiert die total-order-Strategie
- › relaxieren: lässt partial-order-planning für Subtasks der Operatoren zu
- › partial ordered: Auf den Subtasks herrscht partielle Ordnung
- › Reihenfolge der Ausführung nicht wie bei total-order für **alle** Subtasks festgelegt
- › Ansonsten die gleiche Syntax und Semantik wie SHOP

- › Erinnerung: SHOP wählt zu Beginn den ersten Task aus der Taskliste
- › SHOP2 wählt nichtdeterministisch einen Task aus der Taskliste, der keine Vorgänger hat
- › Durch partial-order-planning Verschachtelung von Methoden möglich



Es können sich Lösungen von Atomen überschneiden



Einführung eines Schutzmechanismus

Erinnerung: Operator in SHOP:

(:operator h D A)

wobei

h = head, primitiver Task

D, A = Deletions, Additions. Menge von Atomen,
die keine anderen Variablen enthalten als h

jetzt:

(:operator h D A P C)

wobei

P = Liste mit Atomen, die nicht gelöscht werden dürfen

C = Aufhebung des Schutzstatus von geschützten Atomen

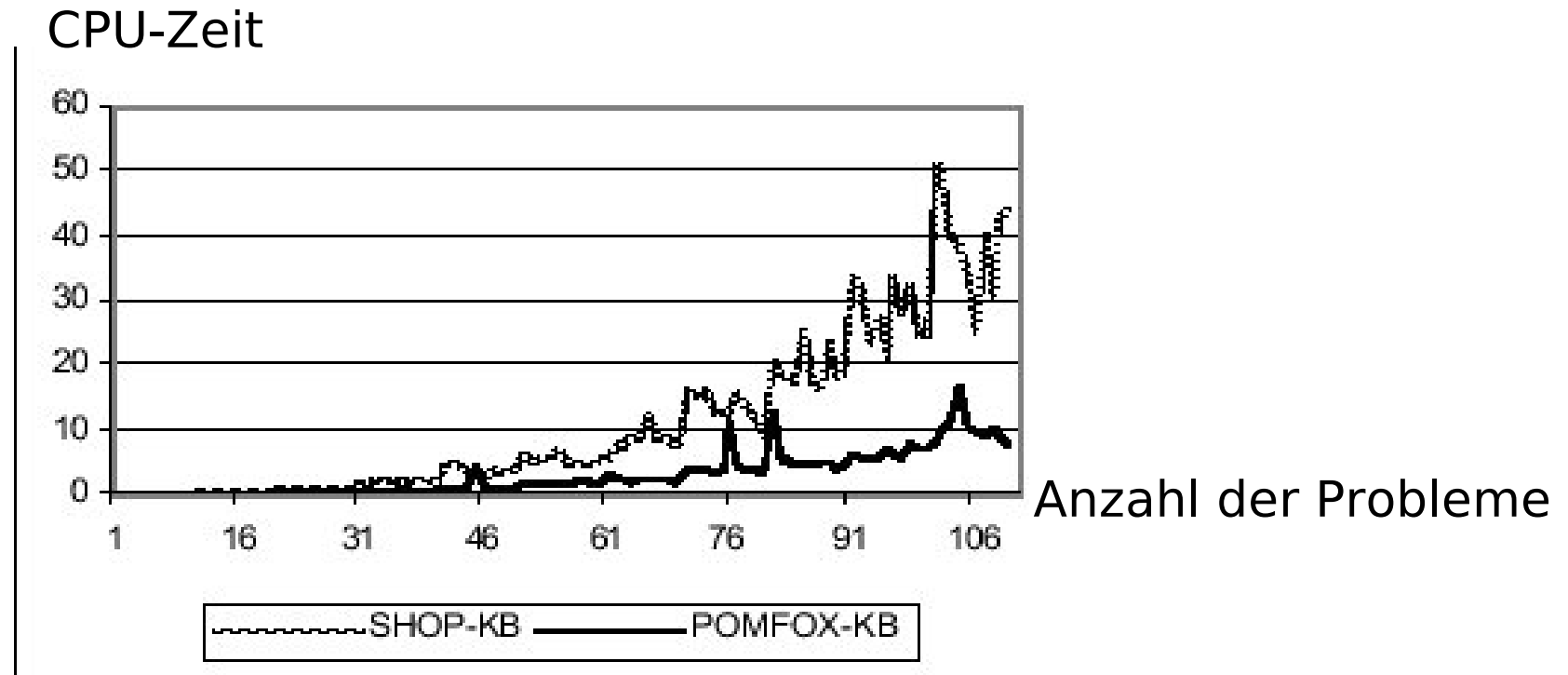
Logistikdomäne, Setup

- › SHOP knowledge base enthält komplexe Methoden für den Transport
- › SHOP2 knowledge base wesentlich einfacher zu schreiben
- › Vergleich über 110 zufällig generierte Probleme der Logistikdomäne
- › N Pakete mussten geliefert werden, $N = 10, 15, \dots, 60$
- › 10 Probleme für jedes $N \Rightarrow 110$ Probleme
- › Für jedes Paket: Anfangs- und Endposition zufällig und verschieden

Logistikdomäne, Ergebnisse

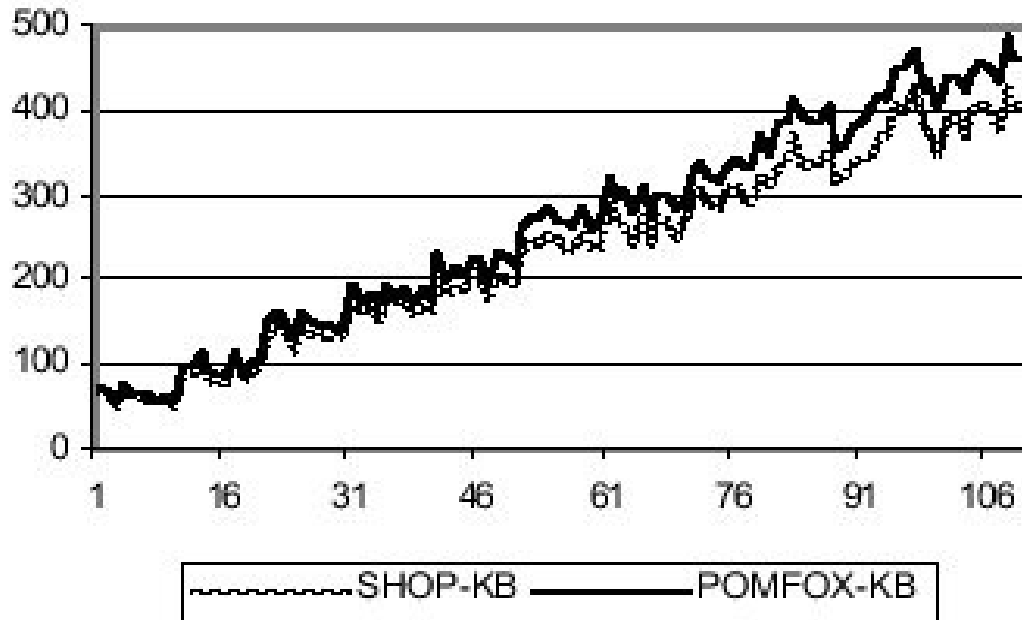
	SHOP knowledge base	SHOP2 knowledge base
Methods	50	10
Operators	7	7
Axioms	10	1

- SHOP knowledge base wesentlich größer als die von SHOP2
- SHOP KB nicht zu vereinfachen



SHOP2 mit SHOP2 KB im Schnitt 4-mal schneller

Plangröße



Anzahl der Probleme

Plangröße von SHOP2 mit SHOP2 KB leicht größer,
als SHOP2 mit SHOP KB

- HTN Planning teilt Aufgaben in kleinere Aufgaben, bis primitive Elemente erreicht werden
- Jeder HTN Planer hat eine Wissensdatenbank mit Methoden, auf die er zurückgreift
- Jede Methode enthält eine Beschreibung, wie diese in kleinere Aufgaben unterteilt werden kann

- HTN Planning benutzt eine total-order Strategie
- total-order Planning: Die einzelnen Schritte werden in der selben Reihenfolge geplant, wie sie später ausgeführt werden.
- SHOP und SHOP2 planen mit HTN
- Unterschied: SHOP2 relaxiert die total-order-Strategie und lässt partial-Order-planning zu

Fragen?

Blockwelt Domäne, Setup

- $N = 5, 10, \dots, 100$ Blöcke mussten versetzt werden
- fünf Probleme für jedes $N \Rightarrow 100$ Probleme
- ersten Block auf den Tisch legen, die folgenden zufällig entweder auf einen Turm oder auf den Tisch
- Zielzustand: Alle Blöcke geordnet aufeinander stapeln

SHOP vs. SHOP2

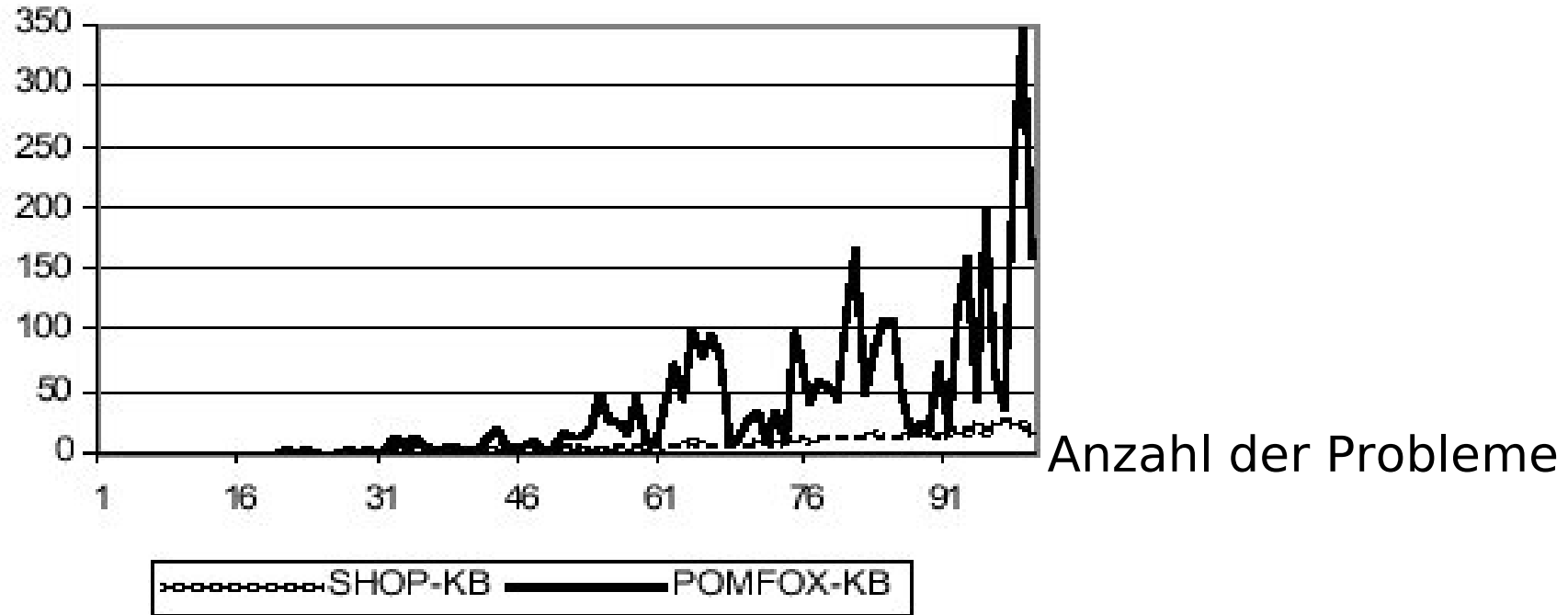
Blockwelt Domäne, Ergebnisse

	SHOP knowledge base	SHOP2 knowledge base
Methods	10	13
Operators	7	8
Axioms	1	5

- SHOP2 knowledge base größer als bei SHOP
- Grund: Schutzmechanismus erhöht in der Blockwelt den Aufwand

SHOP vs. SHOP2

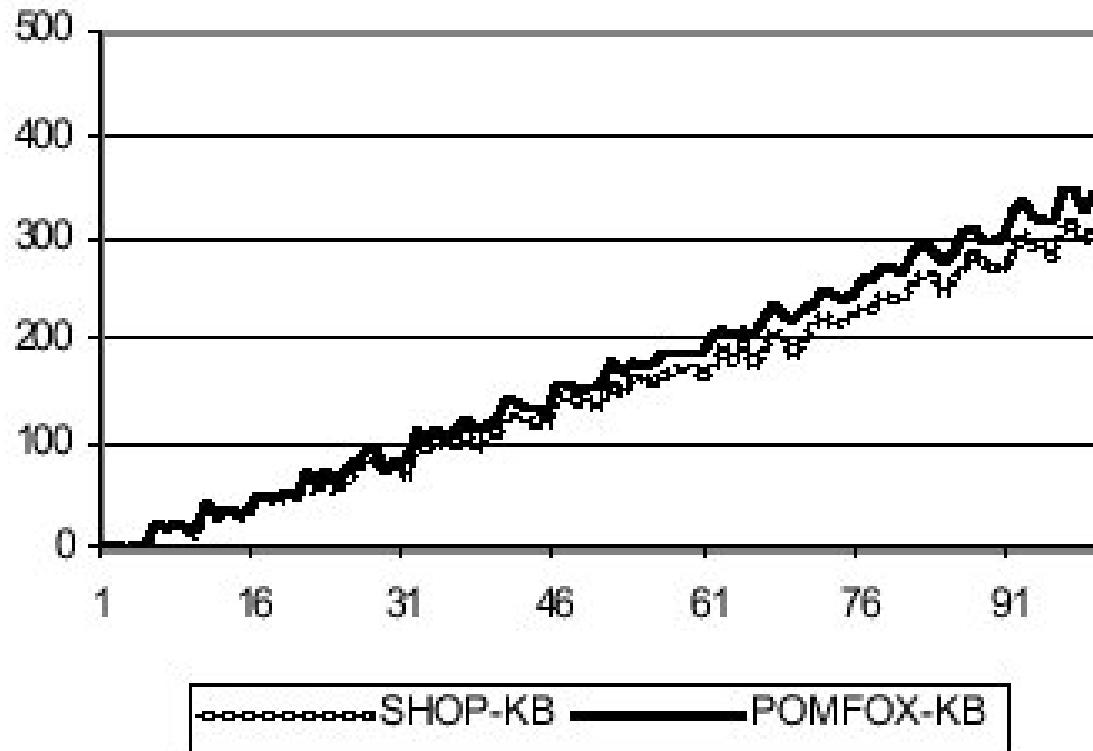
CPU-Zeit



In der Blockwelt SHOP2 mit SHOP2 KB langsamer als mit SHOP KB

SHOP vs. SHOP2

Plangröße



Anzahl der Probleme

Plangrößen mit SHOP KB leicht kleiner