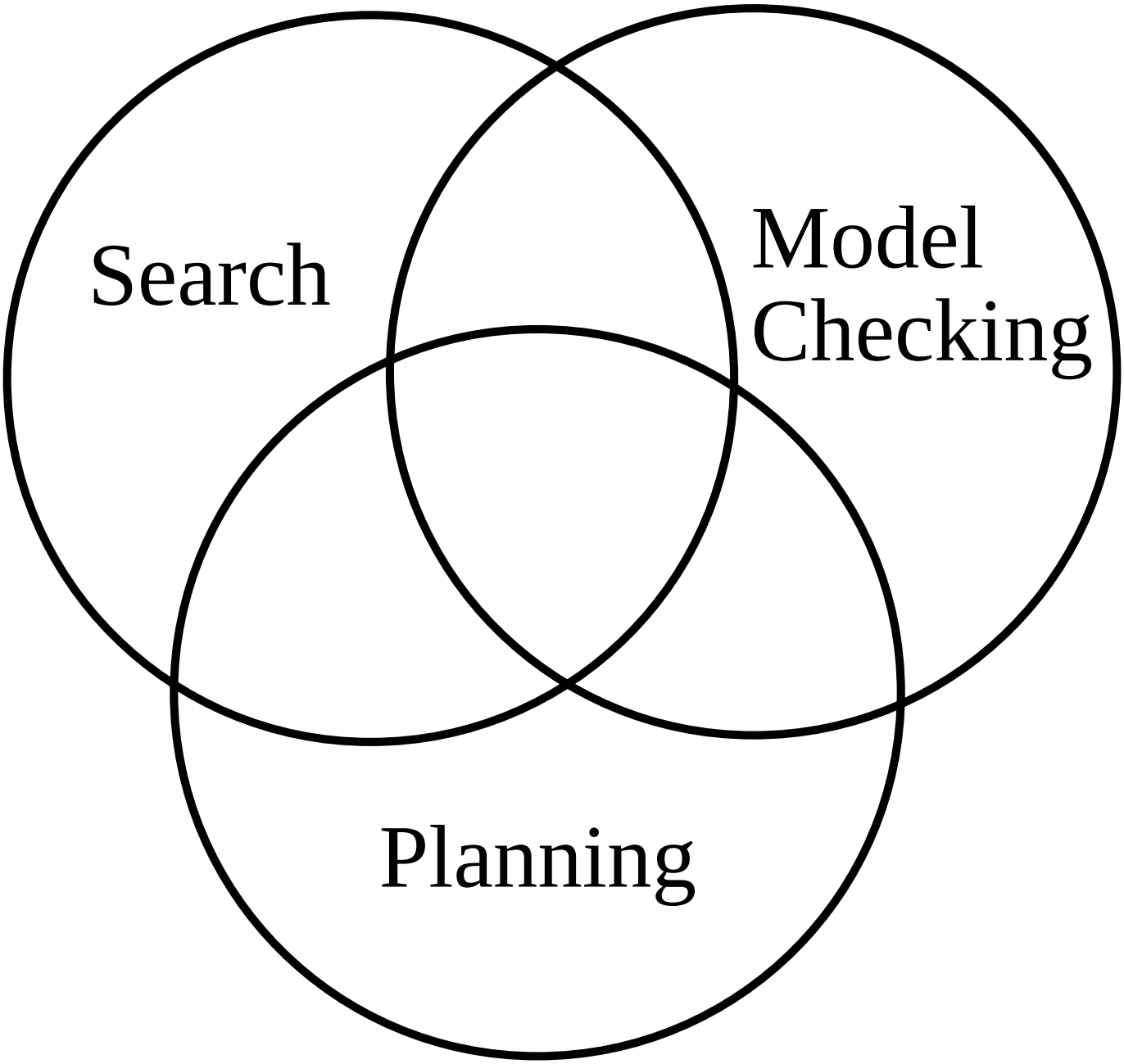
Dagstuhl Seminar

Common Exploration Techniques in Search, Planning and Model Checking

Stefan Edelkamp

Institut für Informatik,
Albert-Ludwigs-Universität,
Georges-Köhler-Allee, D-79110 Freiburg
eMail: edelkamp@informatik.uni-freiburg.de

1 Exploration of Large State Spaces



2 State Space Search

A state space problem $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ consists of

1. a set of states $\mathcal{S}$,
2. a set of operators $\mathcal{O} : \mathcal{S} \rightarrow \mathcal{S}$,
3. a cost function $c : \mathcal{O} \rightarrow \mathbf{R}$
4. an initial state $\mathcal{I} \subseteq S$
5. a set of goal states $\mathcal{G} \subseteq \mathcal{S}$

State spaces implicitly span weighted problem graphs

$$G_{\mathcal{P}} = (V, E, w)$$

In most cases: $w \equiv 1$

Sequential Solutions

A solution

$$\pi_s = (o_1, \ldots, o_k)$$

is an sequence of operators that transforms the initial state $\mathcal{I}$ into one of the goal states $G \in \mathcal{G}$

The cost $c(\pi_s)$ of a solution π_s is $c(o_1) + \ldots + c(o_k)$

Optimal solutions correspond to shortest paths in $G_{\mathcal{P}}$

Schedules

A schedule of $(o_1, \ldots, o_k)$ is vector

$$(t_1, \ldots, t_k) \in \mathbf{R}^k$$

Two operators o and o' in $\mathcal{O}$ are dependent, if the application of o affects the application of o'.

In $(o_1, \ldots, o_k)$ operator o_i precedes o_j, $o_i \leq_o o_j$ for short, if o and o' are dependent and $i \leq j$.

A concurrent solution

$$\pi_c = ((o_1, t_1), \ldots, (o_k, t_k))$$

is a schedule $(t_1, \ldots, t_k)$ of a sequential solution $\pi_s = (o_1, \ldots, o_k)$ with $o_i \leq_o o_j$, $1 \leq i < j \leq k$

Optimal Concurrent Solutions

Algorithm to compute critical path length for a sequence $(o_1, \ldots, o_k)$ of operators
$e(o_i)$ is the earliest end time of operator o_i
$d(o_i) = t_i - t_{i-1}$ is the duration of operator o_i

Procedure PERT

for all $i \in \{1, \ldots, k\}$
$e(o_i) = d(o_i)$
for all $j \in \{1, \ldots, k-1\}$
if $(o_i \leq_o o_j)$
if $e(o_i) < e(o_j) + d(o_j)$
$e(o_i) \leftarrow e(o_j) + d(o_j)$
return $e(o_k)$

Heuristics

A heuristic estimate h is a mapping from $\mathcal{S} \rightarrow \mathbf{R}^+$

Often h admissible, i.e. a lower bound

The sequential merit $f(n)$ of a state n with generating path $(o_1, \ldots, o_k)$ is the sum of

- generating path length $g(n) = c(o_1) + \ldots + c(o_n)$

- heuristic estimate $h(n)$ for the cost from n to achieve a goal state

The estimate is admissible, if $h(n) < \delta(\mathcal{I}, \mathcal{G})$ ($\delta(\mathcal{I}, \mathcal{G})$ min cost from $\mathcal{I}$ to $\mathcal{I}$)

Concurrent Merits

The concurrent merit $f(n)$ of a state n with generating path $(o_1, \ldots, o_k)$ is the critical path length of

- $(o_1, \ldots, o_k)$ concatenated with
- a relaxed solution $(o_{k+1}, \ldots, o_{k+h(n)})$

The latter (h_s) is a solution to a problem abstraction.

3 Search Algorithms

DFS: Searches solution in $G_{\mathcal{P}}$

Nested-DFS: Postorder search for cycles in $G_{\mathcal{P}}$

BFS: Searches shortest path in $G_{\mathcal{P}}$ with $w \equiv 1$

Dijkstra: Searches shortest path in $G_{\mathcal{P}}$

Heuristic Search Algorithms

A*: Searches shorstest path in $G'_{\mathcal{P}}$, where $G'_{\mathcal{P}}$ is obtained by re-weighting
$w'(u,v) = w(u,v) + h(v) - h(u)$ in $G_{\mathcal{P}}$

- $f(\mathcal{I}) = h(\mathcal{I})$
- $f(v) = \min\{f(v), f(u) + w(u,v) + h(v) - h(u)\}$

IDA*: Bounded DFS iteration with increasing threshold on f

DFBnB: Lower bound by heuristics, upper bounds constructivly

4 Common Techniques

Weighting: New merit $f(n) = g(n) + \gamma \cdot h(n)$
[Weighted-A*, Weighted-IDA*] c

Parallelism: Paging in hierarchical memory structures
[Localized-A*] c+o

Partial-Order: Select subset of operators
[PO-IDA*,PO-A*] c

Pruning: Automatically deadlocks and abbreviations
[Decompose-A*,On-Line-FSM-A*] c+o

State Compaction: Store visited list compact
[Suffix-Lists] c+o

In this Talk

Numerical Resources: Schedule sequential solutions [PERT-A*] c+o

Abstraction: Pattern datatabases improve the heuristic [PDB-A*,Disjoint-PDB-A*] c+o

Symbolic Representation: Find encoding $b_1, \ldots, b_k$ for states and $b_1, b'_1, \ldots, b_k, b'_k$ for operators, build transition relation $T(b, b') = \bigvee_{o \in O} o(b, b')$. [Symbolic-{BFS,Dijkstra,A*}, Symbolic-PDBs-A*] c+o

State Compaction: Store visited list partially [Partial-IDA*]

5 Own Research

Search	
(ID)A*+FSM+Suffix-Trees	[KI-1997]
(ID)A*+BDDs	[KI-1998]
(ID)A*+Dead-Ends	[Info-2000]
(ID)A*+Localizing	[AAAI-2000]
(ID)A*+Bit-State (Atomix)	[KI-2001]
(ID)A*+Suffix-Lists	[KI-2001]
Planning	
BDDs	[ECP-1999]
(ID)A*+BDDs	[AAAI-SS-2001]
(ID)A*+PDBs	[ECP-2001]
(ID)A*+BDDs+PDBs	[AIPS-2002]*
(ID)A*+Scheduling	[AIPS-2002]*
Model Checking	
(ID)A*+BDDs	[FM-1999]
(ID)A*+Safety	[AAAI-SS-2001]
(ID)A*+Liveness	[SPIN-2001]
(ID)A*+Trails	[ENTCS-2001]
(ID)A*+Partial Order	[TACAS-2002]*

Planning

Let A, B be sets of propositional atoms. A grounded PDDL+ planning problem is a state space problem $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$, with $\mathcal{S} \subseteq 2^A \times D^{|B|}$ being the set of states, 2^A being the power set notation of A, $\mathcal{I} \in \mathcal{S}$, $\mathcal{G} \subseteq \mathcal{S}$, and $\mathcal{O}$ being the set of operators that transform states into states.

An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ has propositional preconditions α, propositional effects β, numerical preconditions γ, and numerical effects δ.

STRIPS: $\alpha \subseteq A$ is the precondition list and $\beta = (\beta_a, \beta_d)$ with add list $\beta_a \subseteq A$, and the delete list $\beta_d \subseteq A$.

Numerical Part

A numerical constraint $c \in \gamma$ is a triple $c = (h, \otimes, t)$, where $h \in B$, $\otimes \in \{\leq, <, =, >, \geq\}$, and $t \in T_c$

A numerical modifier $c \in \delta$, is a triple $c = (h, \oplus, t)$ where $h \in B$, $\oplus \in \{\leftarrow, \uparrow, \downarrow\}$, and $t \in T_c$.

Semantics

An operator $o = (\alpha, \beta, \gamma, \delta) \in \mathcal{O}$ applied to a state $S = (S_p, S_n) \in 2^A \times D^{|B|}$, $S_p \in 2^A$ and $S_n \in D^{|B|}$, yields a successor state $S' = (S'_p, S'_n) \in 2^A \times D^{|B|}$ as follows.

If $S_p \subseteq \alpha$ and S_n satisfies all $c \in \gamma$ then $S'_p = S_p \cup \beta_a \setminus \beta_d$ and for all $c \in \gamma$ the vector S_n is updated by $c \in \delta$.

$S_n = (S_1, \ldots, S_{|B|})$ satisfies a numerical constraint $c = (h, \otimes, t) \in \gamma$ if $s_h \otimes \text{eval}(S_n, t)$ is true

$S_n = (S_1, \ldots, S_{|B|})$ is updated to $S'_n = (S'_1, \ldots, S'_{|B|})$ by $c = (h, \otimes, t) \in \delta$, if
$S'_h = \text{eval}(S_n, t)$ for $\oplus = \leftarrow$,
$S'_h = S_h + \text{eval}(S_n, t)$ for $\oplus = \uparrow$, and
$S'_h = S_h - \text{eval}(S_n, t)$ for $\oplus = \downarrow$.

Formulae Tree

The set of formula formula trees T_c is recursively defined by the following conditions.

1. For all $b \in B$ we have $b \in T_c$,
2. For all $d \in D$ we have $d \in T_c$,
3. If $t_1, t_2 \in T_c$, then $-t_1 \in T_c$ $(t_1 + t_2) \in T_c$, $(t_1 - t_2) \in T_c$, $(t_1 \cdot t_2) \in T_c$, and $(t_1 / t_2) \in T_c$.

Complexity

Theorem; Grounded PDDL+ is undecidable

[Extension to numerical variables]

Theorem; Grounded PDDL+ is semi-decidable

[Enumerate all path]

If the state space is finite then PDDL+ problems are trivially decidable. Since $|2^A|$ is already finite, the crucial part is to show that $D^{|B|}$ is finite, which is true if both D and B are finite.

Theorem: Bounded Benchmarks are decidable

[Deadlines yield finite graphs]

6 Benchmark Problems

```
(define (domain desert-rat)
(:requirements :typing :durative-actions)
(:types vehicle supply-tank)
(:predicates
    (empty ?t - vehicle)
    (onground ?f - supply-tank)
    (in ?f - supply-tank ?t - vehicle)
    (static ?t - vehicle))
(:functions
    (distance ?t - vehicle)
    (fuel ?t - vehicle) (at ?f - supply-tank)
    (content ?f - supply-tank)
    (capacity ?t - vehicle)
    (speed ?t - vehicle) (frate ?t - vehicle)
    (duration5) (duration10)
    (total-fuel-consumed))
(:durative-action drive-out
 :parameters (?t - vehicle)
 :duration (= ?duration duration5)
 :condition (at start
   (>= (fuel ?t) (* duration5 (frate ?t))))
 :effect (and
   (at end (increase (distance ?t)
                     (* duration5 (speed ?t))))
   (at end (decrease (fuel ?t)
                     (* duration5 (frate ?t))))
   (at end (increase total-fuel-consumed
                     (* duration5 (frate ?t))))))
(:durative-action drive-back ...)
(:action load ...)
(:action unload ...)
(:action fill-up ...)
(:action refuel ...)
```

Discretized Desert-Rats Instance

```
(define (problem cross-the-desert)
    (:domain desert-rat)
    (:objects truck - vehicle
              f1 f2 f3 f4 f5 f6 - supply-tank)
    (:init (= total-fuel-consumed 0)
           (= (distance truck) 0)
           (= (fuel truck) 10)
           (= (capacity truck) 10)
           (= (frate truck) 1)
           (= (speed truck) 20)
           (empty truck)
           (static truck)
           (= duration5 5)
           (= duration10 10)
           (onground f1)
           (= (content f1) 10)
           (= (at f1) 0)
           (onground f2)
           (= (content f2) 10)
           (= (at f2) 0)
           (onground f3)
           (= (content f3) 10)
           (= (at f3) 0)
           (onground f4)
           (= (content f4) 10)
           (= (at f4) 0)
           (onground f5)
           (= (content f5) 10)
           (= (at f5) 0)
           (onground f6)
           (= (content f6) 10)
           (= (at f6) 0))
     (:goal (= (distance truck) 300))
     (:metric minimize total-fuel-consumed))
```

Zeno-Travel

```
(define (domain zeno-travel)
(:requirements :durative-actions :ty-
ping :fluents)
(:types aircraft person city)
(:predicates (at ?x - (either person aircraft)
                 ?c - city)
             (in ?p - person ?a - aircraft))
(:functions (fuel ?a - aircraft)
            (distance ?c1 - city ?c2 - city)
            (slow-speed ?a - aircraft)
            (fast-speed ?a - aircraft)
            (slow-burn ?a - aircraft)
            (fast-burn ?a - aircraft)
            (capacity ?a - aircraft)
            (refuel-rate ?a - aircraft)
            (total-fuel-used)
            (boarding-time)
            (debarking-time))
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c -
city)
 :duration (= ?duration boarding-time)
 :condition (and (at start (at ?p ?c))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at end (in ?p ?a))))
(:durative-action debark ...)
(:durative-action fly ...)
(:durative-action zoom ...)
(:durative-action refuel ...)
```

Zeno-Travel Instance

```
(define (problem zeno-travel-3)
  (:domain zeno-travel)
  (:objects plane - aircraft
            ernie scott dan - person
            city-a city-b city-c city-d - city)
  (:init (= total-fuel-used 0)
         (= debarking-time 20)
         (= boarding-time 30)
         (= (distance city-a city-b) 600)
         (= (distance city-b city-a) 600)
         (= (distance city-b city-c) 800)
         (= (distance city-c city-b) 800)
         (= (distance city-a city-c) 1000)
         (= (distance city-c city-a) 1000)
         (= (distance city-c city-d) 1000)
         (= (distance city-d city-c) 1000)
         (= (fast-speed plane) (/ 600 60))
         (= (slow-speed plane) (/ 400 60))
         (= (fuel plane) 750)
         (= (capacity plane) 750)
         (= (fast-burn plane) (/ 1 2))
         (= (slow-burn plane) (/ 1 3))
         (= (refuel-rate plane) (/ 750 60))
         (at plane city-a)
         (at scott city-a)
         (at dan city-c)
         (at ernie city-c))
  (:goal (and (at ernie city-d)
              (at scott city-d)))
  (:metric minimize
         (+ total-time total-fuel-used)))
```

Jugs-And-Water

```
(define (problem jugs1)
    (:domain Jugs)
    (:objects jug1 jug2 - jug)
    (:init (= (total-time) 0)
           (= (capacity jug1) 5)
           (= (capacity jug2) 3)
           (= (contents jug1) 0)
           (= (contents jug2) 0))
    (:goal (and (= (contents jug1) 1)))
    (:metric minimize total-time))
(define (problem jugs1)
    (:domain Jugs)
    (:objects jug1 jug2 - jug)
    (:init (= (total-time) 0)
           (= (capacity jug1) 5)
           (= (capacity jug2) 3)
           (= (contents jug1) 0)
           (= (contents jug2) 0))
    (:goal (and (= (contents jug1) 1)))
    (:metric minimize total-time))
```

Elimination of Duplicates

Consider the two sequences

```
zoom plane city-a city-c,
board dan plane,
refuel plane,
zoom plane city-c city-a,
board scott,
debark dan,
refuel plane,
```

and

```
board scott,
zoom city-a city-c plane,
board dan plane,
refuel plane,
zoom city-c city-a plane,
debark dan,
refuel plane
```

Dependence

Two grounded operators $o = (\alpha, \beta, \gamma, \delta)$ and $o' = (\alpha', \beta', \gamma', \delta')$ in $\mathcal{O}$ to be *dependent*, if either

1. $\alpha \cap (\beta'_a \cup \beta'_b) \neq \emptyset$, $(\beta_a \cup \beta_b) \cap \alpha' \neq \emptyset$,
2. For one $c = (h, \otimes, t) \in \gamma$ and one $c' = (h', \oplus, t') \in \delta'$ $h \in \mathit{LeafVariables}(t')$ or $h' \in \mathit{LeafVariables}(t)$,
3. For one $c' = (h', \otimes, t') \in \gamma'$ and one $c = (h, \oplus, t) \in \delta$: $h \in \mathit{LeafVariables}(t')$ or $h' \in \mathit{LeafVariables}(t)$,

Heuristics

1) h_f

2) h_p

3) h_n

Bootstrap $[\max(r_1), \ldots, \max(r_k)]$,

Divide $\sum(r_i(G) - r_i(S))/\max(r_i)$

4) $h_s =$ PERT$(p_g(u) \circ p_h(u))$ - PERT$(p_g(u))$

Any-Time Heuristic Search

```
Procedure Any-Time
    G ← ∅; α ← ∞
    Open ← S
    while Open ≠ ∅
        S ← Open.Extract()
        for all S' ∈ expand(S)
            if (S' ∈ G)
                cp ← CriticalPath()
                if cp < α
                    α ← cp
                    G ← S'
            else
                Open.Change(S')
    return G
```

Pattern Database Heuristics

An abstract planning problem
$\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$ of $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ with respect to a set of propositions P is

1. $\mathcal{S}|_R = \{\phi_R(S) \mid S \in \mathcal{S}\}$,
2. $\mathcal{I}|_R = \phi_R(\mathcal{I})$,
3. $\mathcal{G}|_R = \{\phi_R(G) \mid G \in \mathcal{G}\}$,
4. $\mathcal{O}|_R = \{\phi_R(O) \mid O \in \mathcal{O}\}$, with $\phi_R(O)$ for $O = (\alpha, (\beta_a, \beta_d)) \in \mathcal{O}$ defined as $\phi_R(O) = (\alpha|_R, (\beta_a|_R, \beta_d|_R))$.

A planning pattern database $\mathcal{PDB}_R$ is defined as

$$\mathcal{PDB}_R = \{(\delta_R(S), S) \mid S \in \mathcal{S}_R\}.$$

Two planning pattern databases $\mathcal{PDB}_R$ and $\mathcal{PDB}_Q$ are disjoint, if for all $O' \in \mathcal{O}|_R$, $O'' \in \mathcal{O}|_Q$ we have $\phi_R^{-1}(O') \cap \phi_Q^{-1}(O'') = \emptyset$

Symbolic Pattern Databases

For $o = (\alpha, (\beta_a, \beta_d))$ we have $T_o(b, b') =$
$(\bigwedge_{a_i \in \alpha} b_i) \wedge (\bigwedge_{a_j \in \beta_a} b'_j) \wedge (\bigwedge_{a_k \in \beta_d} \neg b'_k) \wedge frame$,

where frame encodes that all other atoms are preserved, i.e. $frame = \bigwedge_{a_j \notin \alpha \cup \beta_a \cup \beta_b} (b_j \equiv b'_j)$.

The relaxed transition relation $T|_R$ is constructed with respect to $o|_R = (\alpha|_R, (\beta_a|_R, \beta_d|_R))$

Algorithm

Algorithm Construct Symbolic Pattern Database

Input: Planning space abstraction

$$\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$$

Output: Symbolic Pattern Database $\mathcal{PDB}|_R$

Reached $\leftarrow$ Open $\leftarrow \mathcal{G}|_R$

$i \leftarrow 0$

while (Open $\not\equiv \emptyset$)

Succ $\leftarrow \exists b$ Open $\wedge\, T|_R(b', b)$

Open $\leftarrow$ Succ $\wedge \neg$ Reached

$\mathcal{PDB}|_R \leftarrow \mathcal{PDB}|_R \vee (v = i \wedge$ Open$)$

Reached $\leftarrow$ Reached $\vee$ Open

$i \leftarrow i + 1$

return $\mathcal{PDB}|_R$

Explicit Search

Algorithm *Explicit Pattern Database Search*
Input: Planning space $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$,
Symbolic Pattern Database $\mathcal{PDB}|_R$
Output: Solution length $\delta(\mathcal{I})$

```
Insert(Open, (I, PDB|_R(I)))
while (Open ≠ ∅)
  S ← DeleteMin(Open)
  if (S ∈ G) return g(S)
  for all successors S' of S
    f'(s') ← f(S) + 1 + (PDB|_R(S') − PDB|_R(S))
    if (Search(Open, S))
      if (f'(S) < f(S))
        DecreaseKey(Open, (S, f'(S))
    else Insert(Open, (S, f'(S))
```

Symbolic Search

Algorithm *Symbolic Pattern Database Search*

Open$\leftarrow \mathcal{PDB}|_R \wedge \mathcal{I}$

do

$f_{\min} = \min\{f \mid f \wedge \text{Open} \neq \emptyset\}$

Min$\leftarrow \exists f\ (\text{Open} \wedge f = f_{\min})$

Rest$\leftarrow$ Open $\wedge \neg$ Min

Succ $\leftarrow$

$\exists b', h, h'$

$\{$ Min$(b')\ \wedge T(b', b) \wedge$

$\mathcal{PDB}|_R(h', b') \wedge \mathcal{PDB}|_R(h, b) \wedge$

$(f = f_{\min} + h' - h + 1)\ \}$

Open $\leftarrow$ Rest $\vee$ Succ

while (Open $\wedge\ \mathcal{G} \equiv 0$)

return $f_{\min}$

Zeno

	s_{seq}	s_{con}	e	s	d	t
h_p	803.33	733.33	234	1137	11	0.04s
	780	713.33	842	3960	12	0.16s
	743.33	673.33	876	4082	12	0.16s
	766.66	670	1172	5371	12	0.22s
	730	630	2549	11604	12	0.49s
	730	600	7712	36364	12	1.57s
	670	570	9423	44084	13	1.93s
	670	540	36894	167593	13	7.54s
h_f	780	683.33	987	5519	12	0.40s
	766.66	670	1074	5878	12	0.42s
	766.66	640	1179	6323	12	0.45s
	730	630	1345	7010	12	0.50s
	730	600	1450	7455	12	0.53s
	670	570	5971	29122	13	2.10s
	670	540	6367	31026	13	2.23s
h_s	710	540	1285	5596	14	4.31s

```
0: (zoom plane city-a city-c) [100]
100: (board dan plane city-c) [30]
100: (refuel plane city-c) [40]
100: (board ernie plane city-c) [30]
140: (zoom plane city-c city-a) [100]
240: (debark dan plane city-a) [20]
240: (board scott plane city-a) [30]
240: (refuel plane city-a) [40]
280: (zoom plane city-a city-c) [100]
380: (refuel plane city-c) [40]
420: (zoom plane city-c city-d) [100]
520: (debark scott plane city-d) [20]
520: (debark ernie plane city-d) [20]
```

Jugs

Jug-1 and Jug-2

	s_{seq}	s_{con}	e	s	d	t
(5,3)	0	0	11	13	6	0.00s
(1237,1721)	0	0	216	218	108	0.05s

Optimal Solution

```
0: (fill jug2) [0]
0: (pour jug2 jug1) [0]
0: (fill jug2) [0]
0: (pour jug2 jug1) [0]
0: (empty jug1) [0]
0: (pour jug2 jug1) [0]
```

Desert-Rats

	s_{seq}	s_{con}	e	s	d	t
(300)	15	15	20	60	5	0.01s
(500)	35	35	18413	47980	13	6.44s
(600)	60	60	436173	577233	24	153.14s

Optimal Solution to (600):

```
0: (load truck f5) [0]
0: (drive-out truck) [5]
5: (unload truck f5) [0]
5: (drive-back truck) [5]
10: (load truck f6) [0]
10: (refuel truck f2) [0]
10: (drive-out truck) [5]
15: (unload truck f6) [0]
15: (drive-back truck) [5]
20: (load truck f3) [0]
20: (refuel truck f1) [0]
20: (drive-out truck) [5]
25: (fill-up truck f5) [0]
25: (drive-out truck) [5]
30: (unload truck f3) [0]
30: (drive-back truck) [5]
35: (load truck f6) [0]
35: (refuel truck f5) [0]
35: (drive-out truck) [5]
40: (refuel truck f3) [0]
40: (drive-out truck) [10]
50: (unload truck f6) [0]
50: (refuel truck f6) [0]
50: (drive-out truck) [10]
```

Symbolic Pattern Databases, bound 2^{20}

p	s	$\overline{h}$
4	108	6.20
5	1,029	8.85
6	12,288	11.49
7	26,244/8	11.72+1.62
8	50,000/80	11.26+3.57
9	87,846/968	11.69+6.37
10	145,152/13,824	11.35+8.59
11	228,488/228,488	11.38+11.39
12	27,440/27,440/2,156	8.64+8.64+5.79
13	37,125/37,125/37,125	8.66+8.66+8.66
14	49,152/49,152/49,152/15	8.03+8.03+8.03+1.80
15	63,869/63,869/63,869/255	8.69+8.69+9.35+3.75

p	d	e	t_e^c	t_e^s	t_s^c	t_s^s
4	6	7	0.01s	0.00s	0.03s	0.00s
5	12	15	0.05s	0.00s	0.04s	0.00s
6	12	13	0.49s	0.00s	0.30s	0.00s
7	24	40	1.39s	0.00s	0.52s	0.01s
8	20	1,590	2.98s	0.10s	0.67s	0.40s
9	32	34,316	6.18s	3.75s	0.81s	13.92s
10	34	47,657	12.55s	5.72s	1.23s	16.35s
11	38	7,941	0.91s	3.09s	1.80s	3.04s
12	38	34,323	4.67s	5.31s	1.73s	13.24s
13	-	-	10.55s	-	2.45s	-
14	40	254,769	15.16s	58.23	3.32s	150.43s
15	-	-	21.88s	-	5.51s	-

Symbolic Pattern Databases, bound 2^{30}

p	s	$\overline{h}$
4	1,08	6.20
5	1,029	8.85
6	12,288	11.49
7	1,777,147	14.14
8	3e06	16.80
9	5.84e+07	19.47
10	1.43e+08	19.55+1.72
11	2.89e+07/1,690	17.05+6.47
12	5.27e+07/27,440	16.29+8.05
13	9.11e+07/506,250	16.89+10.93
14	1.50e+08/1.04e07	16.56+13.77
15	2.41e+08/2.41e08	16.59+16.56

p	d	i_f	i_b	t_s^c	t_s^s	t_b
4	6	6	0	0.02s	0.21s	0.17s
5	12	12	0	0.04s	0.30s	0.30s
6	12	12	0	0.30s	0.43s	1.09s
7	20	20	0	3.95s	0.76s	11.34s
8	18	9	11	0.67s	0.40s	2.80s
9	30	25	12	0.81s	13.92s	38.16s
10	34	60	12	66.16s	58.02s	297.51s
11	32	52	11	1,218s	261.76s	742.14s
12	34	142	15	38.57s	224.13s	1,059s
13	-	147	17	48.88s	time	memory
14	38	52	11	59.05s	150.92s	memory
15	-	-	-	time	-	memory

Model Checking

The state space is given by the synchonous product of Büchi automaton $\mathcal{B}$ and the Büchi automaton $\mathcal{A}$ for the property specification. It has to be shown that $L(\mathcal{B}) \cap \overline{L(\mathcal{A})} = \emptyset$. Any counterexample corresponds to a solution.

We concentrate on communicating FSMs and safety properties.

An (safety) model checking problem is a state space problem $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$, where

- $\mathcal{S} \subseteq F_1 \times \cdots \times F_k \times Q_1 \times \ldots \times Q_l$
- $\mathcal{O}$ is the set of transitions (manipulation of variables, queue access, etc.)
- $\mathcal{I}$ is the initial state
- $\mathcal{G}$ is the set of states defined by the error formula f

Error formula f can be inferred automatically from assertions, invariances and deadlocks

Heuristic

1) $h_f(S)$ number of transition to reach state with f

f	$H_f(S)$	$\overline{H}_f(S)$
$true$	0	∞
$false$	∞	0
a	if a then 0 else 1	if a then 1 else 0
$\neg g$	$\overline{H}_g(S)$	$H_g(S)$
$g \vee h$	$\min\{H_g(S), H_h(S)\}$	$\overline{H}_f(S) + \overline{H}_g(S)$
$g \wedge h$	$H_g(S) + H_h(S)$	$\min\{\overline{H}_g(S), \overline{H}_h(S)\}$
$full(q)$	$capacity(q) - \|q\|$	
$empty(q)$	$\|q\|$	

2) FSM-Distance $\sum_i D_i(u, v)$

3) Active Processes

4) Manhattan-Distance

7 Deadlock detection

GIOP	BFS	DFS	A*	BF	SPIN
s	40,847	218	31,066	117	326
e	37,266	218	27,061	65	326
t	151,671	327	108,971	126	364
l	58	134	58	65	134

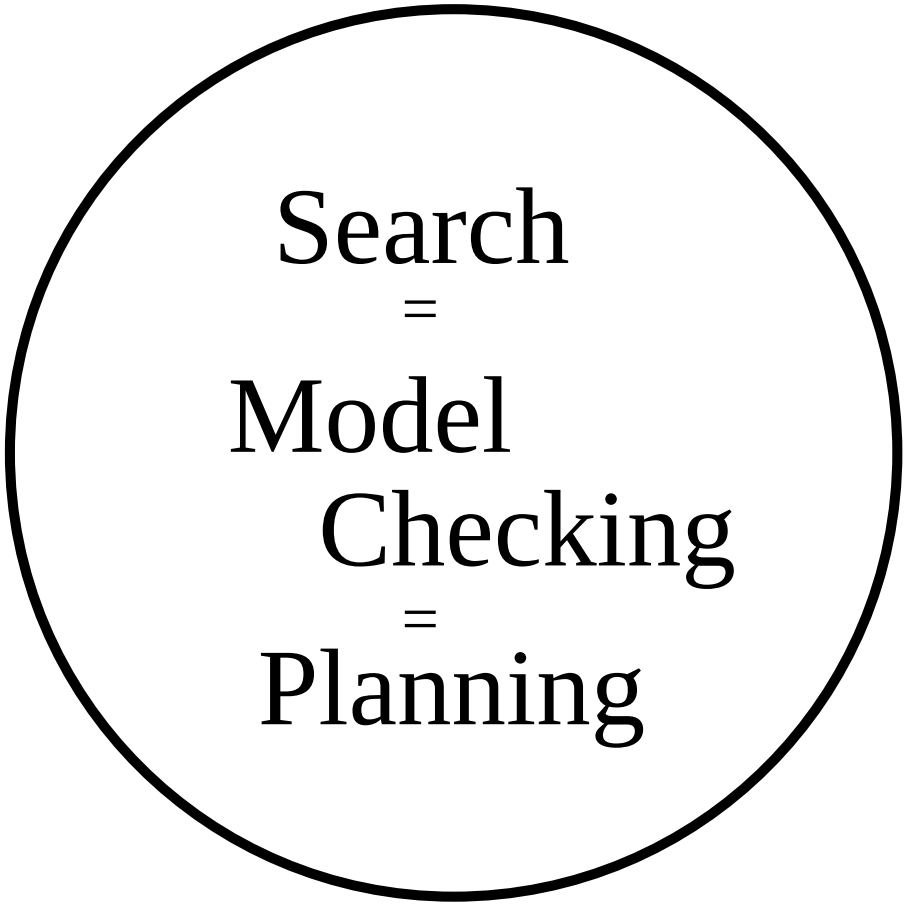
Phil	BFS	DFS	A*	BF	SPIN
s	3,678	1,341	67	493	1,341
e	2,875	1,341	17	225	1,341
t	15,775	1,772	73	622	1,772
l	18	1,362	34	66	1,362

Optical	BFS	DFS	A*	BF	SPIN
s	148,591	20	83	83	20
e	110,722	20	14	14	20
t	621,216	20	83	13	20
l	38	44	38	38	44

Partial IDA*

Depth	A*	IDA*	IDA*+bitstate
58	150,344	146,625	146,625
59	168,191	164,982	164,982
60	184,872	184,383	184,383
61	-	206,145	206,145
62	-	-	229,626
63	-	-	255,411
64	-	-	282,444
65	-	-	311,340
66	-	-	341,562
67	-	-	373,422
68	-	-	407,310
69	-	-	442,941
70	-	-	goal found

8 Conclusions



9 References: Heuristic Search

[KI-1997] S. Edelkamp: Suffix-Tree Automata in State-Space Search, German Conference on Artificial Intelligence, LNAI

[KI-1998] S. Edelkamp and F. Reffel: OBDDs in Heuristic Search, German Conference on Artificial Intelligence, LNAI

[Info-2000] S. Edelkamp: Neue Wege in der Exploration, Informatik-2000, LNCS

[AAAI-2000] S. Edelkamp: Localizing A*, National Conference on Artificial Intelligence, 2000

[KI-2001] S. Edelkamp and U. Meyer: Theory and Practice of Time-Space Trade-Offs in Memory-Limited Search, German Conference on Artificial Intelligence, LNAI

[KI-2001] F. Hüffer, S. Edelkamp, Henning Fernau and R. Niedermeier: Finding Optimal Solutions to Atomix, German Conference on Artificial Intelligence, LNAI

[AIJ-2001] R.Korf, M. Reid and S. Edelkamp: Time Complexity of Iterative-Deepening A*, AI-Journal

Planning

[ECP-1999] S. Edelkamp and M. Helmert: Exhibiting Knowledge in Planning Problems to Minimize State Encoding Length, European Conference on Planning

[AAAI-SS-2001] S. Edelkamp: Directed Symbolic Exploration in AI-Planning, AAAI-Spring Symposium

[ECP-2001] S. Edelkamp: Planning with Pattern Databases, European Conference on Planning

[AIPS-2002]* S. Edelkamp: Symbolic Pattern Databases, Submitted

[PlanSig-2001] S. Edelkamp: First Solutions to PDDL+ Planning Problems, Planning Special Interest Group, 2001

[AIPS-2002]* S. Edelkamp: Critical-Path Analysis for PDDL+- Planning, Submitted

Model Checking

[FM-1999]: F. Reffel and S. Edelkamp: Error Detection with Directed Model Checking, World Congress on Formal Methods

[AAAI-SS-2001] S. Edelkamp, A. Lluch-Lafuente and S. Leue: Protocol Verfication with Heuristic Search

[SPIN-2001] S. Edelkamp, A. Lluch-Lafuente and S. Leue: Directed Explicit State Model Checking with HSF-Spin

[ENTCS-2001]: S. Edelkamp, A. Lluch-Lafuente and S. Leue: Trail-Directed Model-Checking

[TACAS-2002]* S. Edelkamp, A. Lluch-Lafuente and S. Leue: Partial-Order Reduction in Directed Model Checking, Submitted