Agenda

1. Basics
2. Blind Systematic Search Algorithms
3. Heuristic Functions
4. Informed Systematic Search Algorithms
5. Local Search Algorithms
6. Conclusion

Hill-Climbing

Hill-Climbing

$\sigma :=$ make-root-node(init())
forever:
  if is-goal(state(σ)):
    return extract-solution(σ)
  $\Sigma' := \{$ make-node$(\sigma, a, s') \mid (a, s') \in$ succ(state(σ)) $\}$
  $\sigma :=$ an element of Σ' minimizing h /* (random tie breaking) */

Hill-Climbing

Hill-Climbing

$\sigma :=$ make-root-node(init())
forever:
  if is-goal(state(σ)):
    return extract-solution(σ)
  $\Sigma' := \{$ make-node$(\sigma, a, s') \mid (a, s') \in$ succ(state(σ)) $\}$
  $\sigma :=$ an element of Σ' minimizing h /* (random tie breaking) */

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this complete or optimal?

Hill-Climbing

Hill-Climbing

$\sigma :=$ make-root-node(init())
forever:
  if is-goal(state(σ)):
    return extract-solution(σ)
  $\Sigma' := \{$ make-node$(\sigma, a, s') \mid (a, s') \in$ succ(state(σ)) $\}$
  $\sigma :=$ an element of Σ' minimizing h /* (random tie breaking) */

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this complete or optimal? No.
- Can easily get stuck in local minima where immediate improvements of $h(\sigma)$ are not possible.
- Many variations: tie-breaking strategies, restarts, . . .

Enforced Hill-Climbing

Enforced Hill-Climbing: Procedure *improve*

def *improve*(σ_0):
    queue := **new** fifo queue
    queue.push-back(σ_0)
    closed $:= \emptyset$
    while not *queue*.empty():
        $\sigma =$ *queue*.pop-front()
        if *state*(σ) $\notin$ *closed*:
          closed := *closed* $\cup$ {*state*(σ)}
          if h(*state*(σ)) $<$ h(*state*(σ_0)): **return** σ
          for each $(a, s') \in$ succ(*state*(σ)):
              $\sigma' :=$ make-node(σ, a, s')
              queue.push-back(σ')
    fail

$\rightsquigarrow$ Breadth-first search for state with strictly smaller h-value.

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

σ := make-root-node(init())
while not is-goal(state(σ)):
    σ := improve(σ)
return extract-solution(σ)

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal?

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.
- Is this complete?

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.
- Is this complete? In general, no. Under particular circumstances, yes. Assume that h is goal-aware.

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.
- Is this complete? In general, no. Under particular circumstances, yes. Assume that h is goal-aware.
  $\rightarrow$ Procedure *improve* fails: no state with strictly smaller h-value reachable from s, thus (with assumption) goal not reachable from s.

Enforced Hill-Climbing, ctd.

Enforced Hill-Climbing

$\sigma :=$ make-root-node(init())
while not is-goal(state(σ)):
    $\sigma :=$ improve(σ)
return extract-solution(σ)

Remarks:

- Makes sense only if $h(s) > 0$ for $s \notin S^G$.
- Is this optimal? No.
- Is this complete? In general, no. Under particular circumstances, yes. Assume that h is goal-aware.
  $\rightarrow$ Procedure *improve* fails: no state with strictly smaller h-value reachable from s, thus (with assumption) goal not reachable from s.
  $\rightarrow$ This can, for example, not happen if the state space is undirected, i.e., if for all transitions $s \rightarrow s'$ in Θ_Π there is a transition $s' \rightarrow s$.

Properties of Search Algorithms

	DFS	BrFS	ID	A*	HC	IDA*
Complete	No	Yes	Yes	Yes	No	Yes
Optimal	No	Yes*	Yes	Yes	No	Yes
Time	∞	b^d	b^d	b^d	∞	b^d
Space	$b \cdot d$	b^d	$b \cdot d$	b^d	b	$b \cdot d$

- Parameters: d is solution depth; b is branching factor
- Breadth First Search (BrFS) optimal when costs are uniform
- A*/IDA* optimal when h is **admissible**; $h \leq h^*$

Agenda

1 Models

2 Languages

3 Complexity

4 Computational Approaches

5 IPC

6 Conclusion

Models, Languages, and Solvers

- A planner is a solver over a class of models; it takes a model description, and computes the corresponding controller

$$\textit{Model} \Longrightarrow \boxed{\textit{Planner}} \Longrightarrow \textit{Controller}$$

- Many models, many solution forms: uncertainty, feedback, costs, . . .
- Models described in suitable planning languages (Strips, PDDL, PPDDL, . . .) where states represent interpretations over the language.

A Basic Language for Classical Planning: Strips

- A **problem** in STRIPS is a tuple $P = \langle F, O, I, G \rangle$:
  - F stands for set of all atoms (boolean vars)
  - O stands for set of all operators (actions)
  - $I \subseteq F$ stands for initial situation
  - $G \subseteq F$ stands for goal situation
- Operators $o \in O$ **represented** by
  - the Add list $Add(o) \subseteq F$
  - the Delete list $Del(o) \subseteq F$
  - the Precondition list $Pre(o) \subseteq F$

From Language to Models (STRIPS Semantics)

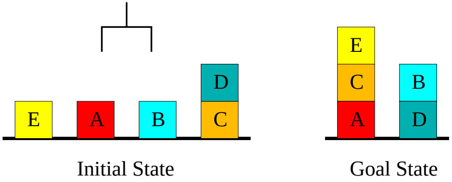
A STRIPS problem $P = \langle F, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are collections of atoms from F. $S = 2^F$
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are ops in O s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

$\rightarrow$ (Optimal) **Solution** of P is (optimal) **solution** of $\mathcal{S}(P)$

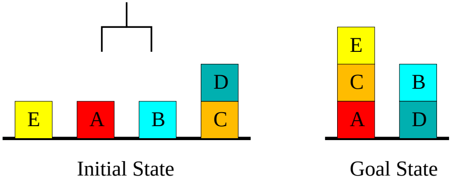
$\rightarrow$ Slight language extensions often convenient: **negation**, **conditional effects**, **non-boolean variables**; some required for describing richer models (costs, probabilities, ...).

(Oh no it's) The Blocksworld



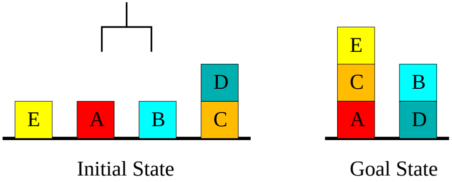
- Propositions: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \dots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- Goal: $\{on(E, C), on(C, A), on(B, D)\}$.
- Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$?

(Oh no it's) The Blocksworld



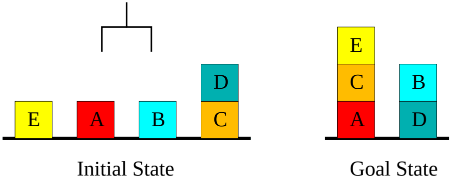
- **Propositions**: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- **Initial state**: $\{onTable(E), clear(E), \ldots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- **Goal**: $\{on(E, C), on(C, A), on(B, D)\}$.
- **Actions**: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$? $pre : \{holding(x), clear(y)\}$

(Oh no it's) The Blocksworld



- Propositions: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E), clear(E), \dots, onTable(C), on(D, C), clear(D), armEmpty()\}$.
- Goal: $\{on(E, C), on(C, A), on(B, D)\}$.
- Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$? $pre : \{holding(x), clear(y)\}$
  $add : \{on(x, y), armEmpty(), clear(x)\}\}$

(Oh no it's) The Blocksworld



- **Propositions**: $on(x,y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- **Initial state**: $\{onTable(E), clear(E), \dots, onTable(C), on(D,C), clear(D), armEmpty()\}$.
- **Goal**: $\{on(E,C), on(C,A), on(B,D)\}$.
- **Actions**: $stack(x,y)$, $unstack(x,y)$, $putdown(x)$, $pickup(x)$.
- $stack(x,y)$? $pre : \{holding(x), clear(y)\}$
  $add : \{on(x,y), armEmpty(), clear(x)\}\}$
  $del : \{holding(x), clear(y)\}$.

PDDL Quick Facts

PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

PDDL Quick Facts

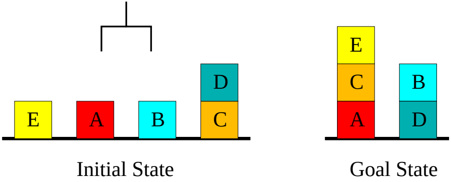
PDDL is not a propositional language:

- Representation is lifted, using object variables to be instantiated from a finite set of objects. (Similar to predicate logic)
- Action schemas parameterized by objects.
- Predicates to be instantiated with objects.

A PDDL planning task comes in two pieces:

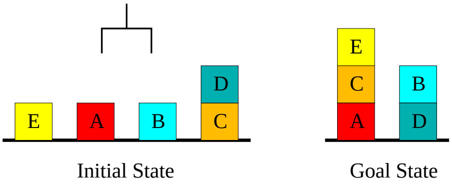
- The domain file and the problem file.
- The problem file gives the objects, the initial state, and the goal state.
- The domain file gives the predicates and the operators; each benchmark domain has *one* domain file.

The Blocksworld in PDDL: Domain File




```
(define (domain blocksworld)
(:predicates (clear ?x) (holding ?x) (on ?x ?y)
             (on-table ?x) (arm-empty))
(:action stack
 :parameters (?x ?y)
 :precondition (and (clear ?y) (holding ?x))
 :effect (and (arm-empty) (on ?x ?y)
              (not (clear ?y)) (not (holding ?x)))
)
...
```

The Blocksworld in PDDL: Problem File




```
(define (problem bw-abcde)
(:domain blocksworld)
(:objects a b c d e)
(:init (on-table a) (clear a)
       (on-table b) (clear b)
       (on-table e) (clear e)
       (on-table c) (on d c) (clear d)
       (arm-empty))
(:goal (and (on e c) (on c a) (on b d))))
```

Example: Logistics in Strips PDDL

```
(define (domain logistics)
  (:requirements :strips :typing :equality)
  (:types airport - location  truck airplane - vehicle  vehicle packet -
  (:predicates (loc-at ?x - location ?y - city) (at ?x - thing ?y - locat
  (:action load
    :parameters (?x - packet ?y - vehicle ?z - location)
    :precondition (and (at ?x ?z) (at ?y ?z))
    :effect (and (not (at ?x ?z)) (in ?x ?y)))
  (:action unload ..)
  (:action drive
    :parameters (?x - truck ?y - location ?z - location ?c - city)
    :precondition (and (loc-at ?z ?c) (loc-at ?y ?c) (not (= ?z ?y)) (at
    :effect (and (not (at ?x ?z)) (at ?x ?y)))
...
(define (problem log3_2)
  (:domain logistics)
  (:objects packet1 packet2 - packet  truck1 truck2 truck3 - truck  airpl
  (:init (at packet1 office1) (at packet2 office3) ...)
  (:goal (and (at packet1 office2) (at packet2 office2))))
```