properties of heuristics

heuristics = estimate cost of reaching the goal

aim: *guide* the search

exact value not necessary
fast calculation necessary

good heuristics = close to the actual cost

---

summry

- admissibility
- combining heuristics
- precomputing

admissibility

heuristics estimate the cost of reaching the goal

if always lower than the actual cost ⇒ admissible heuristics

why: A* finds the optimal solution before the others

---

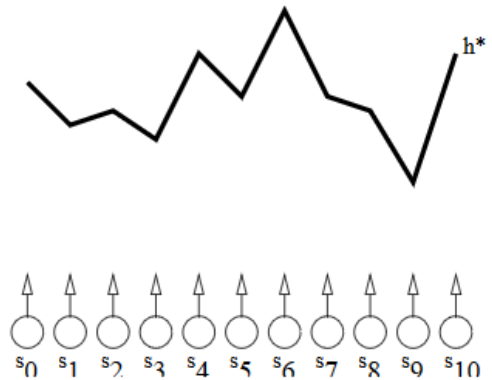
how good an heuristics is?

in general: good = closer to the actual cost

admissible = lower than the cost
therefore: good = as large as possible

[note] Read "lower" as "lower than or equal to".

heuristics: example



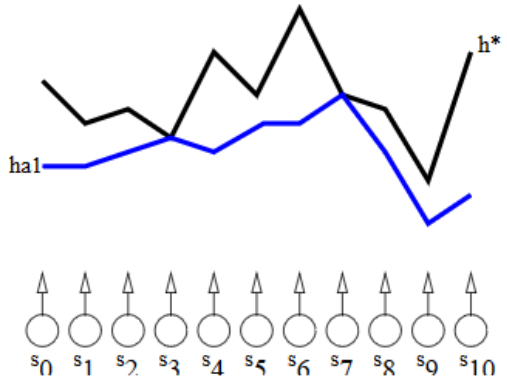
cost = vertical distance to the line

only vertical moves
each action moves up 1cm

now: example of admissible and non-admissible heuristics

---

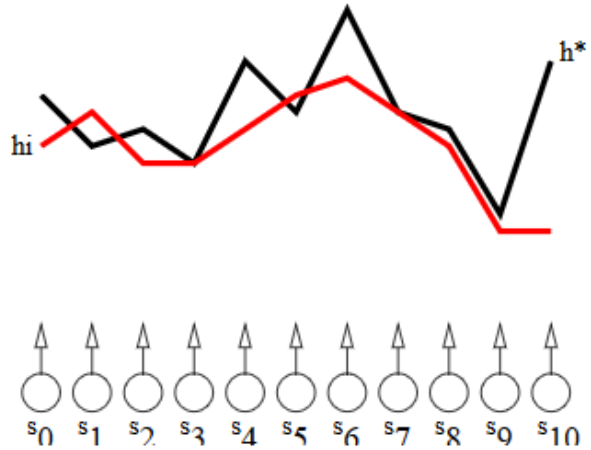
admissible heuristics



an admissible heuristics may *underestimate* the cost

can sometimes be exact (here: s_3 and s_7)
never overestimates

non-admissible heuristics



a non-admissible heuristics overestimates the cost
at least for some states
example: s_1 and s_5

may underestimate the cost for some states
or, may overestimate the cost for all states

---

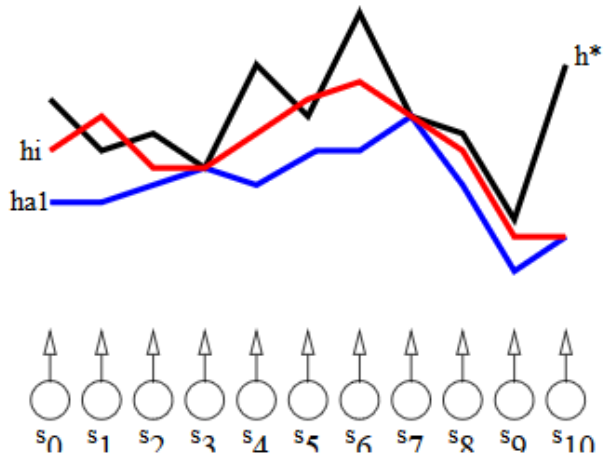
comparison of heuristics

which is better?

admissible: find optimal plans with A*

otherwise: as close to the goal as possible

admissible vs. non-admissible

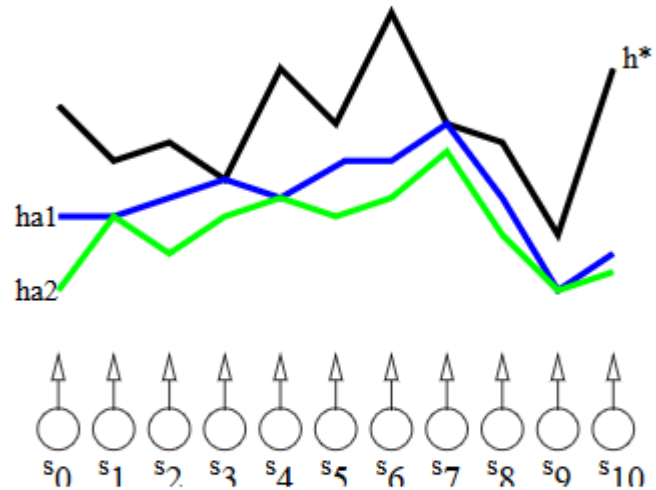


a non-admissible heuristics **may** be closer to reality than an admissible one

or it **may not**

if not interested in optimal plans:
the closest may be better regardless of admissibility

admissible vs. admissible



both admissible: lower than reality

the largest of the two is always closest to reality

in the example: `ha1` better than `ha2`

comparison of heuristics

if non-admissible heuristics can be used:
choose the one that is the closest to the goal

only admissible heuristics:
choose the largest

---

formal definition of admissibililty

$h^*(s)$ = actual cost of reaching the goal from state s
perfect heuristics

$h(s) \leq h^*(s)$ for all s: admissible

combining heuristics

if h and h' both admissible:
$h(s) \leq h^*(s)$
$h'(s) \leq h^*(s)$

consequence: $\max(h(s),h'(s)) \leq h^*(s)$
$\max(h,h')$ also admissible

lower than actual cost
better estimate of the actual cost

[note] The maximal of the two estimates is still lower than the actual cost, but is at least as large as the two; therefore, it is always as close to the actual cost as these.

multiple pattern databases

example heuristics:
fix a set of variables `P'`
delete all variables not in `P'`
calculate length of optimal plan `l`

can also be done for another set of variables `P''`
length of optimal plan `l'`

also admissible: `max(l,l')`

additivity

always: `max(l,l')` is lower than or equal the length of the actual plan

sometimes: also `l+l'` is!

if so: `l+l'` larger than `max(l,l')`
if also admissible: larger = better estimate

an example admissible heuristics

pattern database: remove some variables

a plan of the original instance can still be executed
variables removed from effects are also removed from preconditions

length of plan in the simplified problem is an admissible heuristics

---

an example additive heuristics

> "Theorem 1: If we partition a subset of the state variables in a problem instance into a collection of subsets, so that no operator function affects variables in more than one subset, then the sum of the optimal costs of solving the patterns corresponding to the initial values of the variables in each subset is a lower bound on the optimal cost of solving the original problem instance." (Felner, Korf and Hanan, 2004)

now: example

additive pattern databases: example

choose some subsets of variables
such that no action has effects in two or more more subsets

{x,y,z,w}	{y,z}	{w}
a: x → y,z b: y → -z c: x,y,-z → w goal: z,w	a: → y,z b: y → -z c: y,-z → goal: z	a: → b: → c: → w goal: w

correct partitioning: no action has effects in both {y,z} and w

calculate length of optimal plans in simplified problems
here: a and c, both of length 1

sum of lengths = 2
admissible heuristics

a general additive mechanism

simplify the problem in n different ways

distribute the cost of every action among the n simplifications

calculate heuristics in each of the n simplifications

heuristics can be added

how to put costs into actions

n simplifications

each action has cost 1/n in each

improvement:
if a has no effect in a simplification,
do not put 1/n cost in it
put 1/(n-1) in the others

cost partitioning: example

{x,y,z,w}	{x,y,z}	{y,w}
a: x ⇒ y,z b: y ⇒ -z c: x,y,-z ⇒ w goal: z,w	a (1/2): x ⇒ y,z b (1): y ⇒ -z c (0): x,y,-z ⇒ goal: y	a (1/2): ⇒ y b (0): y ⇒ c (1): y ⇒ w goal: w

all-or-nothing

the cost of each action is transferred to one simplfied problem
action has cost zero in the others

subcase of cost partitioning

additive heuristics

`a` has effects in two simplifications: cost is `1/2` and `1/2`
`b` has effects only in the first: cost is `1` and `0`
`c` has effects only in the second: cost is `0` and `1`

optimal plans: `a` and `a,c`
length is `1/2` and `1/2+1`

sum is `2`
still admissible

[note] In these examples, the simplified problems contain almost all variables. In practice, the simplified problems need to have few variables to be solved efficiently.

precomputing

use of heuristics: guide the search

example: greedy best-first
if in state `s`:
compute `h(s')` for all successors `s'` of `s`

useful: precompute common parts of calculation of `h`

[note] This is now shown for pattern database.

---

the database in pattern database

remove all variables but `xy`

several states becomes the same:
`xyzw`, `xy-z-w`, `xyz-w`, etc. all become `xy`

precompute all values of `h()`
store them in a table

to determine `h(xyz-w)`:

- remove variables, result `xy`
- lookup `h(xy)` in the table

generalized?

precomputing, in general

if computing `h(s)` and `h(s')` require some common operation:
do that only once, before starting the search

during the search, only do the computation specific to a state

possible for various heuristics

[note] For pattern database the idea of a database (table) is so obvious that it made into the name of the heuristics. Other heuristics may however benefit from a similar mechanism.