Let $\Sigma$ be the set of nodes of a given search space.

**Definition (optimal/perfect heuristic)**

The **optimal** or **perfect heuristic** of a search space is the heuristic $h^*$ which maps each search node $\sigma$ to the length of a shortest path from $\text{state}(\sigma)$ to any goal state.

**Note:** $h^*(\sigma) = \infty$ iff no goal state is reachable from $\sigma$. 
Properties of heuristics

A heuristic \( h \) is called

- **safe** if \( h^*(\sigma) = \infty \) for all \( \sigma \in \Sigma \) with \( h(\sigma) = \infty \)
- **goal-aware** if \( h(\sigma) = 0 \) for all goal nodes \( \sigma \in \Sigma \)
- **admissible** if \( h(\sigma) \leq h^*(\sigma) \) for all nodes \( \sigma \in \Sigma \)
- **consistent** if \( h(\sigma) \leq h(\sigma') + 1 \) for all nodes \( \sigma, \sigma' \in \Sigma \) such that \( \sigma' \) is a successor of \( \sigma \)

Relationships?
Greedy best-first search (with duplicate detection)

\[
\text{open} := \textbf{new} \text{ min-heap ordered by } (\sigma \mapsto h(\sigma)) \\
\text{open.insert} \left( \text{make-root-node} \left( \text{init}() \right) \right) \\
\text{closed} := \emptyset \\
\textbf{while not} \text{ open.empty():} \\
\quad \sigma = \text{open.pop-min()} \\
\quad \textbf{if} \ \text{state}(\sigma) \notin \text{closed}: \\
\quad \quad \text{closed} := \text{closed} \cup \{ \text{state}(\sigma) \} \\
\quad \quad \textbf{if} \ \text{is-goal} \left( \text{state}(\sigma) \right): \\
\quad \quad \quad \text{return} \ \text{extract-solution} \left( \sigma \right) \\
\quad \quad \quad \textbf{for each} \ (o, s) \in \text{succ} \left( \text{state}(\sigma) \right): \\
\quad \quad \quad \quad \sigma' := \text{make-node} \left( \sigma, o, s \right) \\
\quad \quad \quad \quad \textbf{if} \ \ h(\sigma') < \infty: \\
\quad \quad \quad \quad \quad \text{open.insert} \left( \sigma' \right) \\
\text{return} \ \text{unsolvable}
\]
Properties of greedy best-first search

- one of the three most commonly used algorithms for satisficing planning
- complete for safe heuristics (due to duplicate detection)
- suboptimal unless $h$ satisfies some very strong assumptions (similar to being perfect)
- invariant under all strictly monotonic transformations of $h$ (e.g., scaling with a positive constant or adding a constant)
A* (with duplicate detection and reopening)

\[
\begin{align*}
\text{open} & := \textbf{new} \text{ min-heap ordered by } (\sigma \mapsto g(\sigma) + h(\sigma)) \\
\text{open}.\text{insert}(\text{make-root-node}(\text{init}())) \\
\text{closed} & := \emptyset \\
\text{distance} & := \emptyset \\
\text{while not} \ \text{open}.\text{empty}(): \\
& \quad \sigma = \text{open}.\text{pop-min}() \\
& \quad \textbf{if} \ \text{state}(\sigma) \not\in \text{closed} \textbf{ or } g(\sigma) < \text{distance} (\text{state}(\sigma)):\ \\
& \quad \quad \text{closed} := \text{closed} \cup \{ \text{state}(\sigma) \} \\
& \quad \quad \text{distance}(\sigma) := g(\sigma) \\
& \quad \textbf{if} \ \text{is-goal}(\text{state}(\sigma)):
& \quad \quad \text{return} \ \text{extract-solution}(\sigma) \\
& \quad \quad \textbf{for each} \ (o, s) \in \text{succ}(\text{state}(\sigma)):\ \\
& \quad \quad \quad \sigma' := \text{make-node}(\sigma, o, s) \\
& \quad \quad \quad \textbf{if} \ h(\sigma') < \infty:\ \\
& \quad \quad \quad \quad \text{open}.\text{insert}(\sigma') \\
\text{return} \ \text{unsolvable}
\end{align*}
\]
Automated (AI) Planning

Deterministic planning tasks
Planning by state-space search
Progression
Regression
Search algorithms for planning
Uninformed search
Heuristic search
Heuristics
Systematic search
Local search

A* example

Example

0+3

I

3

G
Automated (AI) Planning

Deterministic planning tasks
Planning by state-space search
Progression
Regression
Search algorithms for planning
Uninformed search
Heuristic search
Heuristics
Systematic search
Local search

A* example

Example

I

0+3

1+2

G

2

3

A* example

Example
Automated (AI) Planning

Deterministic planning tasks
Planning by state-space search
Progression
Regression
Search algorithms for planning
Uninformed search
Heuristic search
Heuristics
Systematic search
Local search

A* example

Example

A* example diagram with nodes and edges labeled with numbers.
A* example

Example

Determined planning tasks
Planning by state-space search
Progression
Regression
Search algorithms for planning
Uninformed search
Heuristic search
Heuristics
Systematic search
Local search
Automated (AI) Planning

Deterministic planning tasks
Planning by state-space search
Progression
Regression
Search algorithms for planning
Uninformed search
Heuristic search
Heuristics
Systematic search
Local search

A* example

Example

A* example

Example
Terminology for A*

- **f value** of a node: defined by $f(\sigma) := g(\sigma) + h(\sigma)$
- **generated nodes**: nodes inserted into open at some point
- **expanded nodes**: nodes $\sigma$ popped from open for which the test against closed and distance succeeds
- **reexpanded nodes**: expanded nodes for which $state(\sigma) \in closed$ upon expansion (also called reopened nodes)
Properties of $A^*$

- the most commonly used algorithm for optimal planning
- rarely used for satisficing planning
- **complete** for safe heuristics (even without duplicate detection)
- **optimal** if $h$ is admissible and/or consistent (even without duplicate detection)
- never reopens nodes if $h$ is consistent

**Implementation notes:**

- in the heap-ordering procedure, it is considered a good idea to break ties in favour of lower $h$ values
- can simplify algorithm if we know that we only have to deal with consistent heuristics
- common, hard to spot bug: test membership in *closed* at the wrong time
Weighted A* (with duplicate detection and reopening)

\[
\text{Weighted A* (with duplicate detection and reopening)}
\]

\[
open := \textbf{new} \text{ min-heap ordered by } (\sigma \mapsto g(\sigma) + W \cdot h(\sigma))
\]

\[
\text{open.insert(make-root-node(init()))}
\]

\[
closed := \emptyset
\]

\[
distance := \emptyset
\]

\[
\textbf{while not} \ open.\text{empty()}:
\]

\[
\sigma = open.\text{pop-min()}
\]

\[
\textbf{if} \ \text{state}(\sigma) \notin \text{closed} \ \textbf{or} \ g(\sigma) < distance(\text{state}(\sigma)):
\]

\[
closed := \text{closed} \cup \{\text{state}(\sigma)\}
\]

\[
distance(\sigma) := g(\sigma)
\]

\[
\textbf{if} \ \text{is-goal}(\text{state}(\sigma)):
\]

\[
\text{return extract-solution(\sigma)}
\]

\[
\textbf{for each} \ (o, s) \in \text{succ}(\text{state}(\sigma)):
\]

\[
\sigma' := \text{make-node}(\sigma, o, s)
\]

\[
\textbf{if} \ h(\sigma') < \infty:
\]

\[
open.\text{insert}(\sigma')
\]

\[
\text{return unsolvable}
\]
Properties of weighted A*

The weight $W \in \mathbb{R}_0^+$ is a parameter of the algorithm.
- for $W = 0$, behaves like breadth-first search
- for $W = 1$, behaves like A*
- for $W \to \infty$, behaves like greedy best-first search

Properties:
- one of the three most commonly used algorithms for satisficing planning
- for $W > 1$, can prove similar properties to A*, replacing optimal with bounded suboptimal: generated solutions are at most a factor $W$ as long as optimal ones
Hill-climbing

\[ \sigma := \text{make-root-node} \left( \text{init}() \right) \]

\[ \text{forever:} \]
\[ \text{if is-goal} \left( \text{state} (\sigma) \right): \]
\[ \text{return extract-solution} (\sigma) \]
\[ \Sigma' := \{ \text{make-node} (\sigma, o, s) \mid \langle o, s \rangle \in \text{succ} (\text{state} (\sigma)) \} \]
\[ \sigma := \text{an element of } \Sigma' \text{ minimizing } h \text{ (random tie breaking)} \]

- 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: procedure improve

```python
def improve(σ₀):
    queue := new fifo-queue
    queue.push-back(σ₀)
    closed := ∅
    while not queue.empty():
        σ = queue.pop-front()
        if state(σ) ∉ closed:
            closed := closed ∪ {state(σ)}
            if h(σ) < h(σ₀):
                return σ
            for each ⟨o, s⟩ ∈ succ(state(σ)):
                σ' := make-node(σ, o, s)
                queue.push-back(σ')
    fail
```


→ breadth-first search for more promising node than σ₀
Enforced hill-climbing (ctd.)

Enforced hill-climbing

\[
\sigma := \text{make-root-node}(\text{init}())
\]

while not is-goal(state(\sigma)):
\[
\sigma := \text{improve}(\sigma)
\]

return extract-solution(\sigma)

- one of the three most commonly used algorithms for satisficing planning
- can fail if procedure improve fails (when the goal is unreachable from \(\sigma_0\))
- complete for undirected search spaces (where the successor relation is symmetric) if \(h(\sigma) = 0\) for all goal nodes and only for goal nodes