Plan search with heuristic search algorithms

- For forward and backward search (progression, regression) the search space consists of incomplete plans that are respectively prefixes of possible plans and suffixes of possible plans.
- Search starts from the empty plan.
- The neighbors/children of an incomplete plan in the search space are those that are obtained by
  1. adding an operator to the incomplete plan, or
  2. removing an operator from the incomplete plan.
- Systematic search algorithms (like A$*$) keep track of the incomplete plans generated so far, and therefore can go back to them.
  Hence removing operators from incomplete plans is only needed for local search algorithms which do not keep track of the history of the search process.

Plan search with heuristic search algorithms

- For forward and backward search (progression, regression) the search space consists of incomplete plans that are respectively prefixes of possible plans and suffixes of possible plans.
- Search starts from the empty plan.
- The neighbors/children of an incomplete plan in the search space are those that are obtained by
  1. adding an operator to the incomplete plan, or
  2. removing an operator from the incomplete plan.
- Systematic search algorithms (like A$*$) keep track of the incomplete plans generated so far, and therefore can go back to them.

  Hence removing operators from incomplete plans is only needed for local search algorithms which do not keep track of the history of the search process.

Plan search with heuristic search algorithms

- For forward and backward search (progression, regression) the search space consists of incomplete plans that are respectively prefixes of possible plans and suffixes of possible plans.
- Search starts from the empty plan.
- The neighbors/children of an incomplete plan in the search space are those that are obtained by
  1. adding an operator to the incomplete plan, or
  2. removing an operator from the incomplete plan.
- Systematic search algorithms (like A$*$) keep track of the incomplete plans generated so far, and therefore can go back to them.
  Hence removing operators from incomplete plans is only needed for local search algorithms which do not keep track of the history of the search process.

Plan search: incomplete plans for progression

For progression, the incomplete plans are prefixes $o_1, o_2, \ldots, o_n$ of potential plans.

An incomplete plan is extended by

1. adding an operator after the last operator, from $o_1, \ldots, o_n$ to $o_1, o_2, \ldots, o_n, o$ for some $o \in O$, or
2. removing one or more of the last operators, from $o_1, \ldots, o_n$ to $o_1, \ldots, o_i$ for some $i < n$. This is for local search algorithms only.

$o_1, o_2, \ldots, o_n$ is a plan if $app_{o_n}(app_{o_{n-}}(\cdots app_{o_1}(I) \cdots)) \models G$.

Plan search: incomplete plans for progression

For progression, the incomplete plans are prefixes $o_1, o_2, \ldots, o_n$ of potential plans.

An incomplete plan is extended by

1. adding an operator after the last operator, from $o_1, \ldots, o_n$ to $o_1, o_2, \ldots, o_n, o$ for some $o \in O$, or
2. removing one or more of the last operators, from $o_1, \ldots, o_n$ to $o_1, \ldots, o_i$ for some $i < n$. This is for local search algorithms only.

$o_1, o_2, \ldots, o_n$ is a plan if $app_{o_n}(app_{o_{n-}}(\cdots app_{o_1}(I)\cdots)) \models G$.

Plan search: incomplete plans for progression

For progression, the incomplete plans are prefixes $o_1, o_2, \ldots, o_n$ of potential plans.
An incomplete plan is extended by

1. adding an operator after the last operator,
   from $o_1, \ldots, o_n$ to $o_1, o_2, \ldots, o_n, o$ for some $o \in O$, or
2. removing one or more of the last operators,
   from $o_1, \ldots, o_n$ to $o_1, \ldots, o_i$ for some $i < n$.
   This is for local search algorithms only.

$o_1, o_2, \ldots, o_n$ is a plan if
$app_{o_n}(app_{o_{n-}}(\cdots app_{o_1}(I)\cdots)) \models G$.

Plan search: incomplete plans for progression

For progression, the incomplete plans are prefixes $o_1, o_2, \ldots, o_n$ of potential plans.
An incomplete plan is extended by

1. adding an operator after the last operator,
   from $o_1, \ldots, o_n$ to $o_1, o_2, \ldots, o_n, o$ for some $o \in O$, or
2. removing one or more of the last operators,
   from $o_1, \ldots, o_n$ to $o_1, \ldots, o_i$ for some $i < n$.
   This is for local search algorithms only.

$o_1, o_2, \ldots, o_n$ is a plan if
$app_{o_n}(app_{o_{n-}}(\cdots app_{o_1}(I)\cdots)) \models G$.

Plan search: incomplete plans for regression

For regression, the incomplete plans are suffixes $o_n, \ldots, o_1$ of potential plans.

An incomplete plan is extended by

1. adding an operator in front of the first operator, from $o_n, \ldots, o_1$ to $o, o_n, \ldots, o_1$ for $o \in O$, or
2. deleting one or more of the first operators, from $o_n, \ldots, o_1$ to $o_i, \ldots, o_1$ for some $i < n$. This is for local search algorithms only.

$o_n, \ldots, o_1$ is a plan if $I \models \mathit{regr}_{o_n}(\cdots \mathit{regr}_{o_2}(\mathit{regr}_{o_1}(G)) \cdots)$.

Remark

Above is for the simplest case when the formulae are not split. With splitting formalization is slightly trickier.

Plan search: incomplete plans for regression

For regression, the incomplete plans are suffixes $o_n, \ldots, o_1$ of potential plans.

An incomplete plan is extended by

1. adding an operator in front of the first operator, from $o_n, \ldots, o_1$ to $o, o_n, \ldots, o_1$ for $o \in O$, or
2. deleting one or more of the first operators, from $o_n, \ldots, o_1$ to $o_i, \ldots, o_1$ for some $i < n$. This is for local search algorithms only.

$o_n, \ldots, o_1$ is a plan if $I \models regr_{o_n}(\cdots regr_{o_2}(regr_{o_1}(G)) \cdots)$.

Remark

Above is for the simplest case when the formulae are not split. With splitting formalization is slightly trickier.

Plan search: incomplete plans for regression

For regression, the incomplete plans are suffixes $o_n, \ldots, o_1$ of potential plans.

An incomplete plan is extended by

1. adding an operator in front of the first operator, from $o_n, \ldots, o_1$ to $o, o_n, \ldots, o_1$ for $o \in O$, or
2. deleting one or more of the first operators, from $o_n, \ldots, o_1$ to $o_i, \ldots, o_1$ for some $i < n$. This is for local search algorithms only.

$o_n, \ldots, o_1$ is a plan if $I \models \mathit{regr}_{o_n}(\cdots \mathit{regr}_{o_2}(\mathit{regr}_{o_1}(G)) \cdots)$.

Remark

Above is for the simplest case when the formulae are not split. With splitting formalization is slightly trickier.

Plan search: incomplete plans for regression

For regression, the incomplete plans are suffixes $o_n, \ldots, o_1$ of potential plans.

An incomplete plan is extended by

1. adding an operator in front of the first operator, from $o_n, \ldots, o_1$ to $o, o_n, \ldots, o_1$ for $o \in O$, or
2. deleting one or more of the first operators, from $o_n, \ldots, o_1$ to $o_i, \ldots, o_1$ for some $i < n$. This is for local search algorithms only.

$o_n, \ldots, o_1$ is a plan if $I \models \mathit{regr}_{o_n}(\cdots \mathit{regr}_{o_2}(\mathit{regr}_{o_1}(G)) \cdots)$.

Remark

Above is for the simplest case when the formulae are not split. With splitting formalization is slightly trickier.

Plan search: incomplete plans for regression

For regression, the incomplete plans are suffixes $o_n, \dots, o_1$ of potential plans.

An incomplete plan is extended by

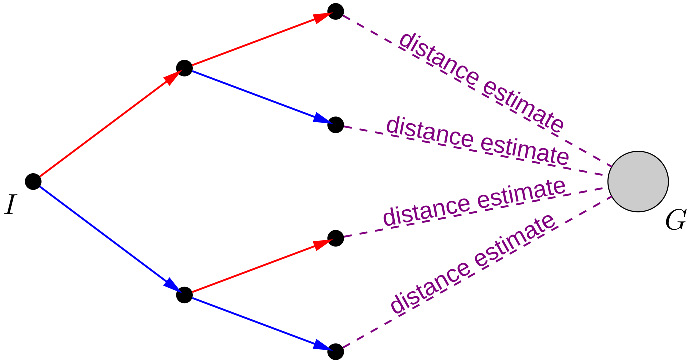
1. adding an operator in front of the first operator, from $o_n, \dots, o_1$ to $o, o_n, \dots, o_1$ for $o \in O$, or
2. deleting one or more of the first operators, from $o_n, \dots, o_1$ to $o_i, \dots, o_1$ for some $i < n$. This is for local search algorithms only.

$o_n, \dots, o_1$ is a plan if $I \models \mathit{regr}_{o_n}(\cdots \mathit{regr}_{o_2}(\mathit{regr}_{o_1}(G)) \cdots)$.

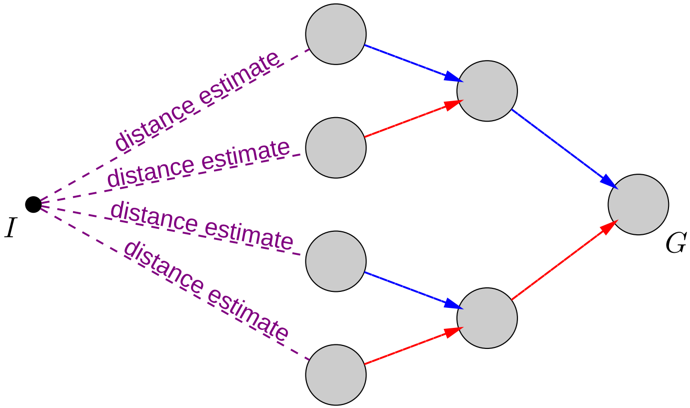
Remark

Above is for the simplest case when the formulae are not split. With splitting formalization is slightly trickier.

Planning by heuristic search
Forward search



Planning by heuristic search
Backward search



Select next operator $o \in O$ based on the estimated distance (number of operators) between

1. $app_o(app_{o_n}(app_{o_{n-1}}(\cdots app_{o_1}(I)\cdots)))$ and G, for forward search.
2. I and $regr_o(regr_{o_n}(\cdots regr_{o_2}(regr_{o_1}(G))\cdots))$, for backward search.

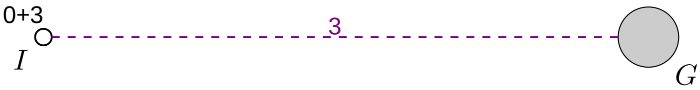
Search algorithms: A$*$

Search control of A$*$

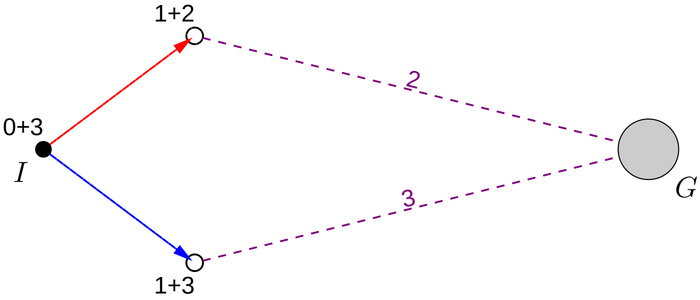
$A*$ uses the function $f(\sigma) = g(\sigma) + h(\sigma)$ to guide search:

- $g(\sigma)$ = cost so far i.e. number of operators in σ
- $h(\sigma)$ = estimated remaining cost (distance)
- admissibility: $h(\sigma)$ must be less than or equal the actual remaining cost $h*(\sigma)$ (distance), otherwise A$*$ is not guaranteed to find an optimal solution.

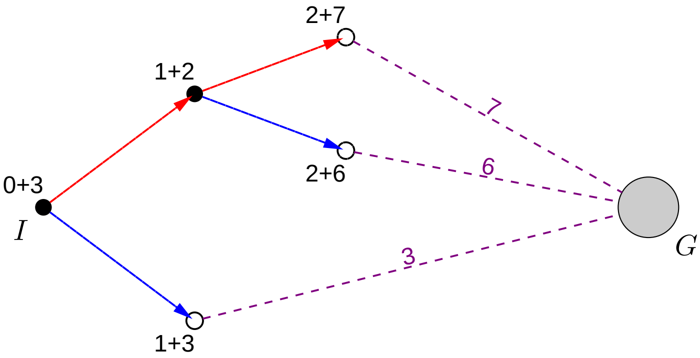
Search algorithms: A∗
Example



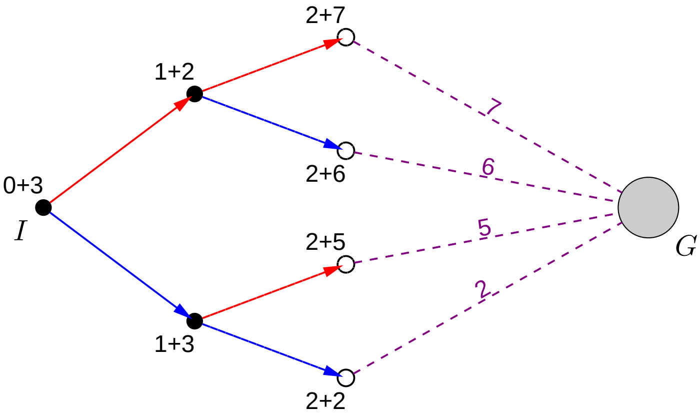
Search algorithms: A$*$
Example



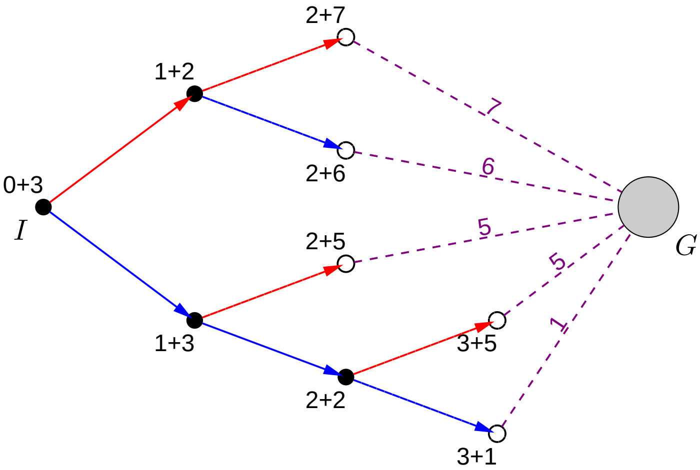
Search algorithms: A∗
Example



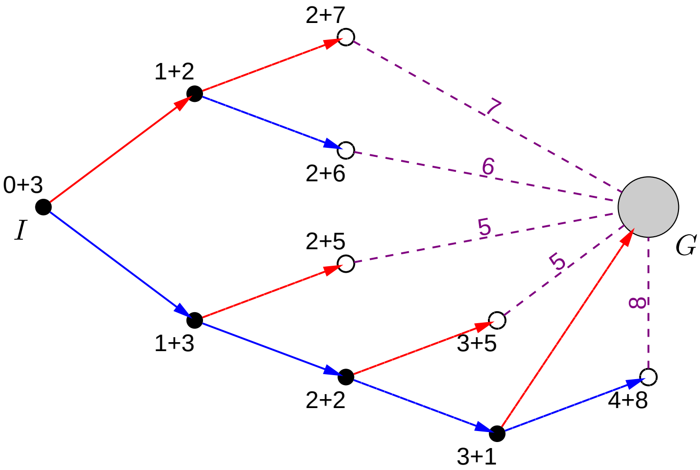
Search algorithms: A∗
Example



Search algorithms: A∗
Example



Search algorithms: A$*$
Example



Notation for operator sequences

$app_{o_1;o_2;\ldots;o_n}(s)$ denotes $app_{o_n}(\ldots app_{o_2}(app_{o_1}(s))\ldots)$ and ϵ denotes the empty sequence for which $app_\epsilon(s) = s$.

Algorithm A$*$

Forward search with A$*$ works as follows.

1. OPEN := $\{\epsilon\}$, CLOSED := $\emptyset$.
2. If OPEN = $\emptyset$, then stop: no solution.
3. Choose an element $\sigma \in$ OPEN with the least $f(\sigma)$.
4. If $app_\sigma(I) \models G$ then stop: solution found.
5. OPEN := OPEN$\setminus\{\sigma\}$; CLOSED := CLOSED$\cup\{\sigma\}$.
6. OPEN := OPEN $\cup(\{\sigma; o|o \in O\}\setminus$CLOSED).
7. Go to 2.

Local search: random walk

Random walk

1. $\sigma := \epsilon$
2. If $app_{\sigma}(I) \models G$, stop: σ is a plan.
3. Randomly choose a neighbor σ' of σ.
4. $\sigma := \sigma'$
5. Go to 2.

Remark

The algorithm usually does not find any solutions, unless almost every sequence of actions is a plan.

Local search: steepest descent hill-climbing

Hill-climbing

1. σ := ϵ
2. If $app_\sigma(I) \models G$, stop: σ is a plan.
3. Randomly choose neighbor σ' of σ with the least $h(\sigma')$.
4. σ := σ'
5. Go to 2.

Remark

The algorithm gets stuck in local minima: the 3rd step cannot be carried out because no neighbor is better than the current incomplete plan.

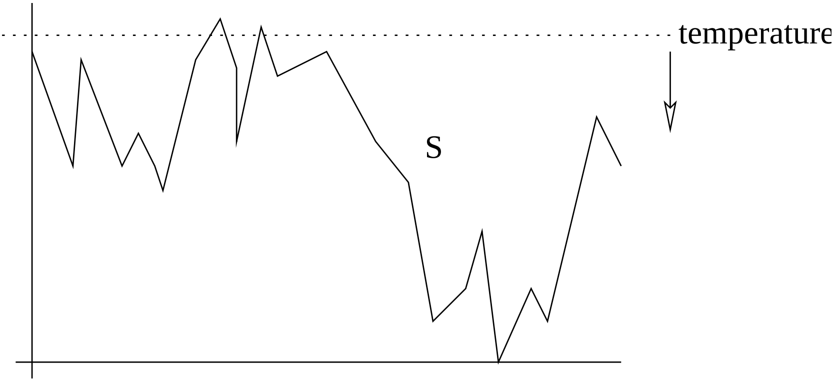
Local search: simulated annealing

Simulated annealing

1. $\sigma := \epsilon$
2. If $\mathit{app}_\sigma(I) \models G$, stop: σ is a plan.
3. Randomly choose a neighbor σ' of σ.
4. If $h(\sigma') < h(\sigma)$ go to 7.
5. With probability $\exp(-\frac{h(\sigma')-h(\sigma)}{T})$ go to 7.
6. Go to 3.
7. $\sigma := \sigma'$
8. Decrease T. (Different possible strategies!)
9. Go to 2.

The temperature T is initially high and then gradually decreased.

Local search: simulated annealing
Illustration



How to obtain heuristics?

General procedure for obtaining a heuristic

Solve a simplified / less restricted version of the problem.

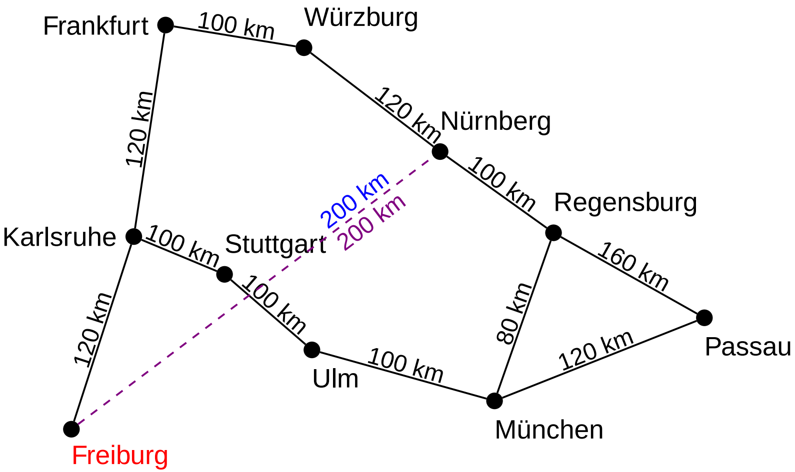
Example (Route-planning for the road network)

The road network is formalized as a weighted graph where the weight of an edge is the road distance between two locations.

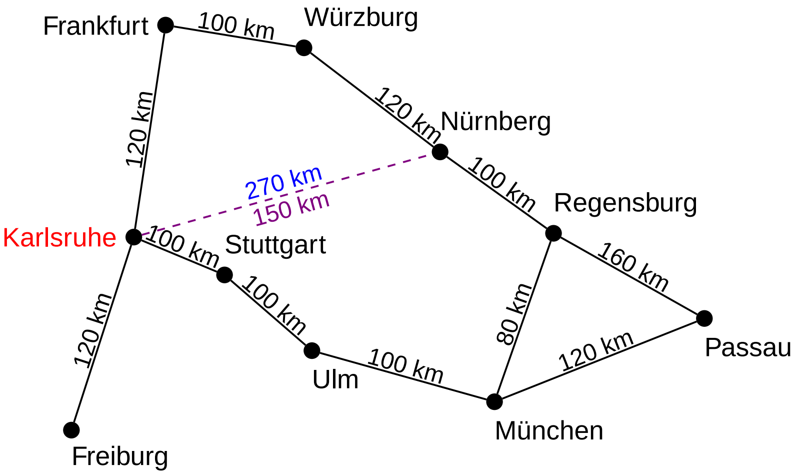
A heuristic is obtained from the Euclidean distance $\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$. It is a lower bound on the road distance between (x_1, y_1) and (x_2, y_2).

An admissible heuristic for route planning

Example

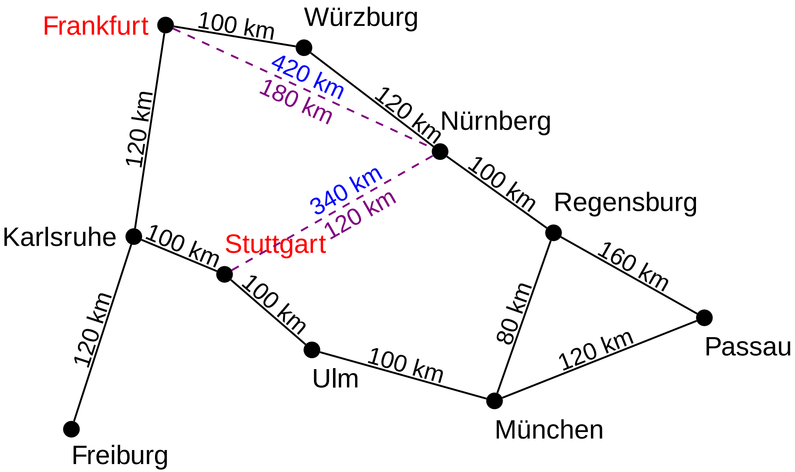


An admissible heuristic for route planning
Example

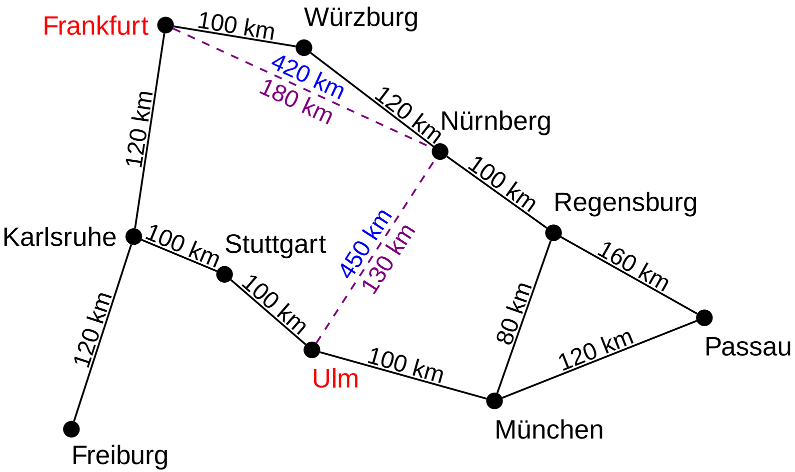


An admissible heuristic for route planning

Example

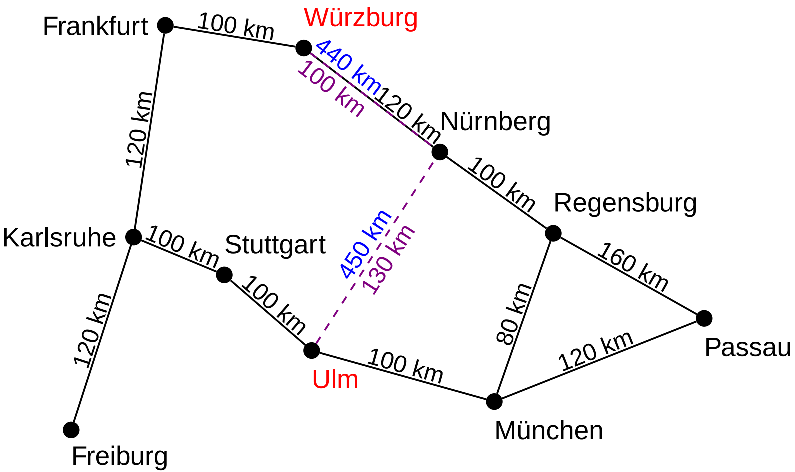


An admissible heuristic for route planning
Example



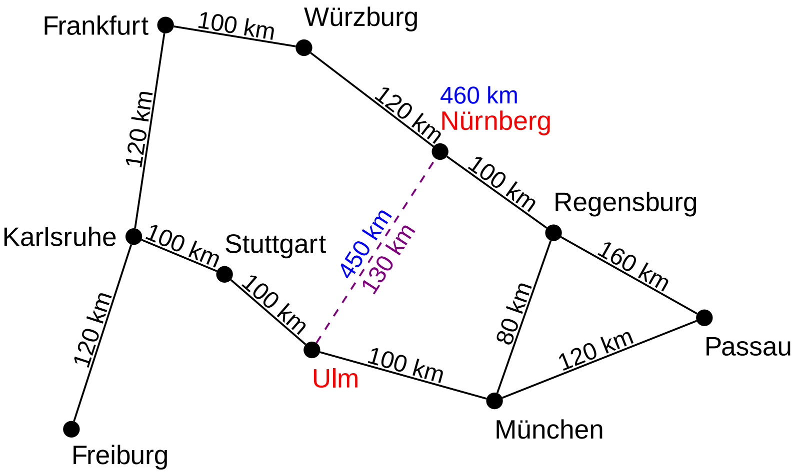
An admissible heuristic for route planning

Example

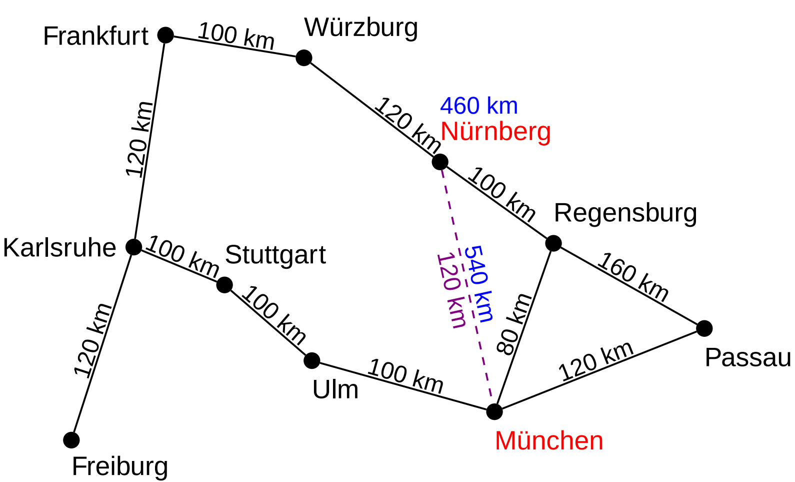


An admissible heuristic for route planning

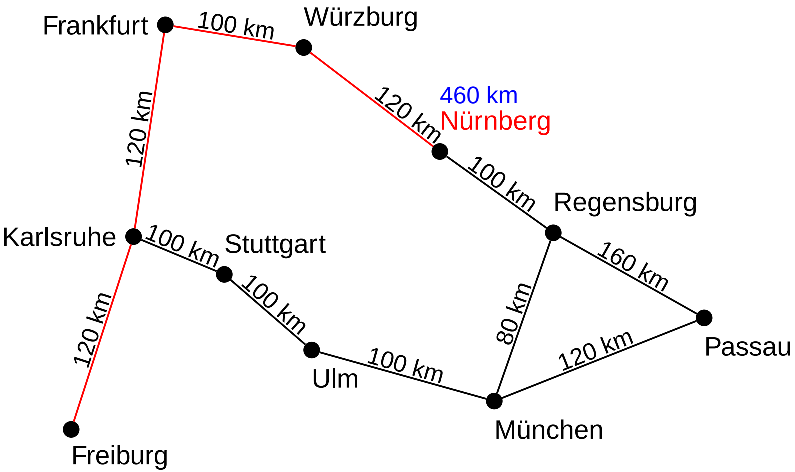
Example



An admissible heuristic for route planning
Example



An admissible heuristic for route planning
Example



Heuristics for deterministic planning
STRIPS

- STRIPS (Fikes & Nilsson, 1971) used the number of state variables that differ:

$$|\{a \in A | s(a) = s'(a)\}|.$$

  "The more goal literals an operator makes true, the more useful the operator is."

- The above heuristic is not admissible because one operator may reduce this measure by more than one. Instead,

$$\frac{|\{a \in A | s(a) = s'(a)\}|}{n}$$

  is admissible when no operator has $> n$ atomic effects.

Images

Definition

The image of a state s with respect to an action o is

$$\mathit{img}_o(s) = \{s' | sos'\}.$$

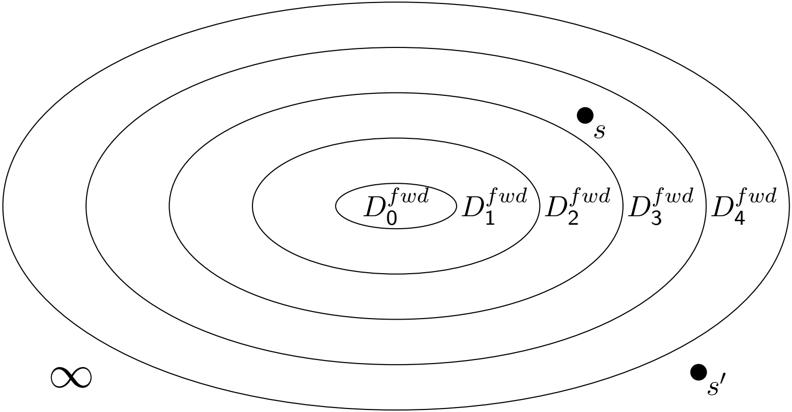
This can be generalized to sets T of states as follows.

$$\mathit{img}_o(T) = \bigcup_{s \in T} \mathit{img}_o(s)$$

We use these functions also for operators o: replace sos' by $sR(o)s'$ where $R(o)$ is the relation corresponding to o.

Distances
Illustration

Forward distance of state s is 3 because $s \in D_3^{fwd} \setminus D_2^{fwd}$.



As $D_i^{fwd} = D_4^{fwd}$ for all $i > 4$, forward distance of state s' is ∞.

Distances

Definition

Let I be a state and O a set of det. operators. Define the forward distance sets D_i^{fwd} for I, O by

$$\begin{aligned} D_0^{fwd} &= \{I\} \\ D_i^{fwd} &= D_{i-1}^{fwd} \cup \bigcup_{o \in O} \mathit{img}_o(D_{i-1}^{fwd}) \text{ for all } i \geq 1 \end{aligned}$$

Definition

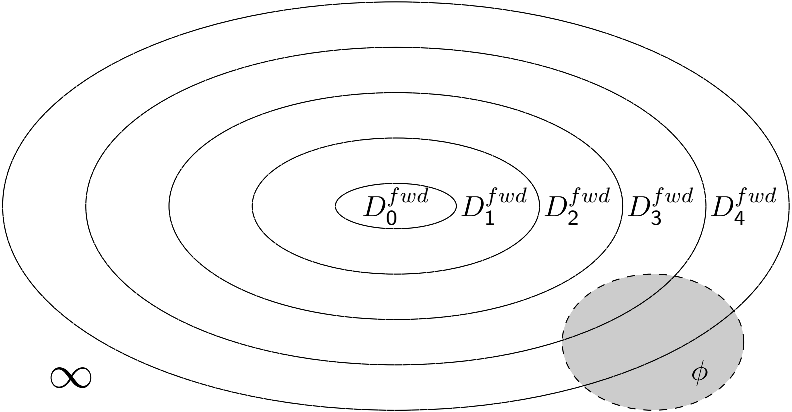
Let $D_0^{fwd}, D_1^{fwd}, \ldots$ be the forward distance sets for I, O. The forward distance of a state s from I is

$$\delta_I^{\mathit{fwd}}(s) = \begin{cases} 0 & \text{if } I = s, \\ i & \text{if } s \in D_i^{fwd} \backslash D_{i-1}^{fwd} \end{cases}$$

If $s \notin D_i^{fwd}$ for all $i \geq 0$ then the distance of s is ∞. States with a finite distance are reachable from I with O.

Distances
of formulae

$\delta_I^{\mathit{fwd}}(\phi) = 3$ since $s \models \phi$ for some $s \in D_3^{fwd}$ but for no $s \in D_2^{fwd}$.



Distances
of formulae

Theorem

$\delta_I^{fwd}(s)$ is the length n of a shortest sequence $o_1; \ldots; o_n$ of actions/operators for reaching s from I.

Definition

Let ϕ be a formula. The forward distance $\delta_I^{fwd}(\phi)$ of ϕ is i if there is state s such that $s \models \phi$ and $\delta_I^{fwd}(s) = i$ and there is no state s such that $s \models \phi$ and $\delta_I^{fwd}(s) < i$.

Heuristic: approximations of distances

- We define a relaxed/approximate notion of distances that is computable in polynomial time.
- Exact distances are as hard to compute as solving the planning problem: when the distances are known, a plan is obtained simply by repeatedly choosing an operator that reduces the distance to goals by one.
- The idea of our approximation is: instead of distances of states, consider distances of literals which are distances of states in which the literal is true.
- If there are n state variables, for exact distances we have to consider sets with up to 2^n states.
  For approximate distances sets with up to n literals suffice: polynomial time algorithms are possible.

Heuristic: approximations of distances

- We define a relaxed/approximate notion of distances that is computable in polynomial time.
- Exact distances are as hard to compute as solving the planning problem: when the distances are known, a plan is obtained simply by repeatedly choosing an operator that reduces the distance to goals by one.
- The idea of our approximation is: instead of distances of states, consider distances of literals which are distances of states in which the literal is true.
- If there are n state variables, for exact distances we have to consider sets with up to 2^n states.
  For approximate distances sets with up to n literals suffice: polynomial time algorithms are possible.

Heuristic: approximations of distances

- We define a relaxed/approximate notion of distances that is computable in polynomial time.
- Exact distances are as hard to compute as solving the planning problem: when the distances are known, a plan is obtained simply by repeatedly choosing an operator that reduces the distance to goals by one.
- The idea of our approximation is: instead of distances of states, consider distances of literals which are distances of states in which the literal is true.
- If there are n state variables, for exact distances we have to consider sets with up to 2^n states.
  For approximate distances sets with up to n literals suffice: polynomial time algorithms are possible.

Heuristic: approximations of distances

- We define a relaxed/approximate notion of distances that is computable in polynomial time.
- Exact distances are as hard to compute as solving the planning problem: when the distances are known, a plan is obtained simply by repeatedly choosing an operator that reduces the distance to goals by one.
- The idea of our approximation is: instead of distances of states, consider distances of literals which are distances of states in which the literal is true.
- If there are n state variables, for exact distances we have to consider sets with up to 2^n states.
  For approximate distances sets with up to n literals suffice: polynomial time algorithms are possible.

Sets of literals representing sets of states

Idea

Let S be the set of all states (valuations of state variables.) A set T of literals a and $\neg a$ represents the set $\{s \in S | s \models T\}$ of states.

Example

The following are equivalent.

1. $b \vee c$ is true in at least one state represented by $\{a, \neg c\}$.
2. $\{a, \neg c\} \cup \{b \vee c\}$ is satisfiable = *SAT*($\{a, \neg c\} \cup \{b \vee c\}$).

Distance estimation

Blocks world example

	D_0^{max}				
AonB	T	TF	TF	TF	TF
AonC	F	F	F	TF	TF
BonA	F	F	TF	TF	TF
BonC	T	T	T	TF	TF
ConA	F	F	F	TF	TF
ConB	F	F	F	TF	TF
AonT	F	TF	TF	TF	TF
BonT	F	F	TF	TF	TF
ConT	T	T	T	TF	TF
Aclear	T	T	TF	TF	TF
Bclear	F	TF	TF	TF	TF
Cclear	F	F	F	TF	TF

Distance estimation
Blocks world example

	D_0^{max}	D_1^{max}			
AonB	T	TF	TF	TF	TF
AonC	F	F	F	TF	TF
BonA	F	F	TF	TF	TF
BonC	T	T	T	TF	TF
ConA	F	F	F	TF	TF
ConB	F	F	F	TF	TF
AonT	F	TF	TF	TF	TF
BonT	F	F	TF	TF	TF
ConT	T	T	T	TF	TF
Aclear	T	T	TF	TF	TF
Bclear	F	TF	TF	TF	TF
Cclear	F	F	F	TF	TF

Distance estimation
Blocks world example

	D_0^{max}	D_1^{max}	D_2^{max}		
AonB	T	TF	TF	TF	TF
AonC	F	F	F	TF	TF
BonA	F	F	TF	TF	TF
BonC	T	T	T	TF	TF
ConA	F	F	F	TF	TF
ConB	F	F	F	TF	TF
AonT	F	TF	TF	TF	TF
BonT	F	F	TF	TF	TF
ConT	T	T	T	TF	TF
Aclear	T	T	TF	TF	TF
Bclear	F	TF	TF	TF	TF
Cclear	F	F	F	TF	TF

Distance estimation
Blocks world example

	D_0^{max}	D_1^{max}	D_2^{max}	D_3^{max}	D_4^{max}
AonB	T	TF	TF	TF	TF
AonC	F	F	F	TF	TF
BonA	F	F	TF	TF	TF
BonC	T	T	T	TF	TF
ConA	F	F	F	TF	TF
ConB	F	F	F	TF	TF
AonT	F	TF	TF	TF	TF
BonT	F	F	TF	TF	TF
ConT	T	T	T	TF	TF
Aclear	T	T	TF	TF	TF
Bclear	F	TF	TF	TF	TF
Cclear	F	F	F	TF	TF

Distance estimation
Blocks world example

Initially A is on B which is on C.

$$
\begin{aligned}
D_0^{max} &= \{Aclear, AonB, BonC, ConT, \neg AonC, \neg BonA,\\
&\quad \neg ConA, \neg ConB, \neg AonT, \neg BonT, \neg Bclear, \neg Cclear\}\\
D_1^{max} &= \{Aclear, BonC, ConT, \neg AonC, \neg BonA,\\
&\quad \neg ConA, \neg ConB, \neg BonT, \neg Cclear\}\\
D_2^{max} &= \{ConT, \neg AonC, \neg ConA, \neg ConB\}\\
D_3^{max} &= \emptyset
\end{aligned}
$$

New state variables values are possible at the given time points because of the following actions.

1. A onto table
2. B onto table, B onto A
3. C onto A, C onto B, A onto C

Distances of literals

Definition

$EPC_l(\langle c, e\rangle) = EPC_l(e) \wedge c \wedge \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$

Definition

Let $L = A \cup \{\neg a | a \in A\}$ be the set of literals on A. Let I be a state. Define the sets D_i^{max} for $i \geq 0$ as follows.

$$D_0^{max} = \{l \in L | I \models l\}$$
$$D_i^{max} = D_{i-1}^{max} \backslash \{l \in L | o \in O, SAT(D_{i-1}^{max} \cup \{EPC_{\bar{l}}(o)\})\}$$

Remark

Since we consider only finite sets A of state variables and $|D_0^{max}| = |A|$ and $D_{i+1}^{max} \subseteq D_i^{max}$ for all $i \geq 0$, necessarily $D_i^{max} = D_j^{max}$ for some $i \leq |A|$ and all $j > i$.

Max-distances of literals and states

Definition

The max-distance of a literal l (from I with O) is

$$\delta_I^{max}(l) = \begin{cases} 0 \text{ if } \bar{l} \notin D_0^{max} \\ d \text{ if } \bar{l} \in D_{d-1}^{max} \backslash D_d^{max} \text{ for } d \geq 1 \end{cases}$$

Definition

The max-distance of a state s (from I with O) is

$$\delta_I^{max}(s) = \begin{cases} 0 \text{ if } s \models D_0^{max} \\ d \text{ if } s \not\models D_{d-1}^{max} \text{ and } s \models D_d^{max} \text{ for } d \geq 1 \end{cases}$$

If $\delta_I^{max}(s) = n$ then $\delta_I^{max}(l) \leq n$ for all literals l such that $s \models l$, and $\delta_I^{max}(l) = n$ for at least one literal l.

Why are max-distances inaccurate?

Example

1. Consider the problem of switching on n lamps that are all switched off.
2. Each action switches on 1 lamp.
3. The distances of literal "lamp i is on" for every i is 1.
4. But the distance of the state with all lamps on is n.

The distance estimate of n goals in the above example is the maximum of the distances of individual goals, even though the sum of the distances in this case would be much more accurate. (See the lecture notes for further discussion of this topic.)

Distances of formulae

Based on the distances of literals we can define the distances of formulae. The distance of ϕ is n if $s \models \phi$ for at least one state having distance n and $s \not\models \phi$ for all states having distance $< n$.

Definition

The max-distance of a formula ϕ (from I with O) is

$$\delta_I^{max}(\phi) = \begin{cases} 0 \text{ if } SAT(D_0^{max} \cup \{\phi\}) \\ d \text{ if } SAT(D_d^{max} \cup \{\phi\}) \text{ and not } SAT(D_{d-1}^{max} \cup \{\phi\}) \text{ for } d \geq 1 \end{cases}$$

Relation between max-distances and distances

The sets D_i^{max} approximate the sets D_i^{fwd} upwards in the following way.

Theorem (A)

Let $D_i^{fwd}, i \geq 0$ be the forward distance sets and D_i^{max} the max-distance sets for I and O. Then for all $i \geq 0$, $D_i^{fwd} \subseteq \{s \in S | s \models D_i^{max}\}$ where S is the set of all states.

Proof.

By induction on i.

Base case $i = 0$: D_0^{fwd} consists of the unique initial state and D_0^{max} consists of exactly those literals that are true in the initial state, identifying the initial state uniquely. Hence $D_i^{fwd} = \{s \in S | s \models D_i^{max}\}$.

Relation between max-distances and distances

The sets D_i^{max} approximate the sets D_i^{fwd} upwards in the following way.

Theorem (A)

Let $D_i^{fwd}, i \geq 0$ be the forward distance sets and D_i^{max} the max-distance sets for I and O. Then for all $i \geq 0$, $D_i^{fwd} \subseteq \{s \in S | s \models D_i^{max}\}$ where S is the set of all states.

Proof.

By induction on i.

Base case $i = 0$: D_0^{fwd} consists of the unique initial state and D_0^{max} consists of exactly those literals that are true in the initial state, identifying the initial state uniquely. Hence $D_i^{fwd} = \{s \in S | s \models D_i^{max}\}$.

Relation between max-distances and distances
continued

proof continues.

Inductive case $i \geq 1$: Let s be any state in D_i^{fwd}. We show that $s \models D_i^{max}$. Let l be any literal in D_i^{max}.

1. Assume $s \in D_{i-1}^{fwd}$. As $D_i^{max} \subseteq D_{i-1}^{max}$ also $l \in D_{i-1}^{max}$. By the induction hypothesis $s \models l$.
2. Otherwise $s \in D_i^{fwd} \backslash D_{i-1}^{fwd}$.
   Hence there is $o \in O$ and $s_0 \in D_{i-1}^{fwd}$ with $s = app_o(s_0)$.
   By $D_i^{max} \subseteq D_{i-1}^{max}$ and the induction hypothesis $s_0 \models l$.
   As $l \in D_i^{max}$, not $SAT(D_{i-1}^{max} \cup \{EPC_{\bar{l}}(o)\})$ by def. of D_i^{max}.
   By $s_0 \in D_{i-1}^{fwd}$ and the induction hypothesis $s_0 \models D_{i-1}^{max}$.
   Hence $s_0 \not\models EPC_{\bar{l}}(o)$.
   By Lemma B applying o in s_0 does not make l false.
   Hence $s \models l$.

□

Properties of max-distances

Corollary

Let I be a state and ϕ a formula. Then for any sequence $o_1, \ldots, o_n$ of operators such that executing them in I results in state s such that $s \models \phi$, $n \geq \delta_I^{max}(\phi)$.

Hence we can use $\delta_I^{max}(\phi)$ for estimating the distance from I to ϕ. This never overestimates the actual distance (the heuristic is admissible) but may severely underestimate.

Distance estimation in polynomial time

- Computing max-distances takes polynomial time assuming the tests *SAT*$(D_i^{max} \cup \{\phi\})$ take polynomial time.
- However, performing these tests is of course in general NP-hard.
- Polynomial time special case: ϕ is a conjunction literals. Then *SAT*$(D_i^{max} \cup \{\phi\})$ if and only if $\bar{l} \notin D_i^{max}$ for all literals l in ϕ. You can verify that for STRIPS operators formulae ϕ always have this form after the obvious simplifications.
- Can we achieve polynomial runtime for arbitrary operators?

Distance estimation in polynomial time

- Computing max-distances takes polynomial time assuming the tests *SAT*$(D_i^{max} \cup \{\phi\})$ take polynomial time.
- However, performing these tests is of course in general NP-hard.
- Polynomial time special case: ϕ is a conjunction literals. Then *SAT*$(D_i^{max} \cup \{\phi\})$ if and only if $\bar{l} \notin D_i^{max}$ for all literals l in ϕ. You can verify that for STRIPS operators formulae ϕ always have this form after the obvious simplifications.
- Can we achieve polynomial runtime for arbitrary operators?

Distance estimation in polynomial time

- Computing max-distances takes polynomial time assuming the tests *SAT*$(D_i^{max} \cup \{\phi\})$ take polynomial time.
- However, performing these tests is of course in general NP-hard.
- Polynomial time special case: ϕ is a conjunction literals. Then *SAT*$(D_i^{max} \cup \{\phi\})$ if and only if $\bar{l} \notin D_i^{max}$ for all literals l in ϕ. You can verify that for STRIPS operators formulae ϕ always have this form after the obvious simplifications.
- Can we achieve polynomial runtime for arbitrary operators?

Distance estimation in polynomial time

- Computing max-distances takes polynomial time assuming the tests *SAT*$(D_i^{max} \cup \{\phi\})$ take polynomial time.
- However, performing these tests is of course in general NP-hard.
- Polynomial time special case: ϕ is a conjunction literals. Then *SAT*$(D_i^{max} \cup \{\phi\})$ if and only if $\bar{l} \notin D_i^{max}$ for all literals l in ϕ. You can verify that for STRIPS operators formulae ϕ always have this form after the obvious simplifications.
- Can we achieve polynomial runtime for arbitrary operators?

Distance estimation in polynomial time

- By approximating the satisfiability tests it becomes possible to compute max-distances in polynomial time.
- This is at the cost of a small further inaccuracy.
- Satisfiability tests *SAT*$(D \cup \{\phi\})$ are replaced by a weaker test asat(D, ϕ) such that

  $$\textit{if } \textit{SAT}(D \cup \{\phi\}) \textit{ then } \textit{asat}(D, \phi)$$

  (but not necessarily vice versa.)
- Max-distances remain admissible under such a weaker test.
- We next present procedure asat(D, ϕ) that is polynomial time computable.

The procedure asat(ϕ, D)

Our goal

Define procedure asat(ϕ, D) that is guaranteed to return true if $D \cup \{\phi\}$ is satisfiable, but may sometimes return true also when $D \cup \{\phi\}$ is unsatisfiable. Hence the procedure fails in one direction.

As a result, max-distance estimates

1. become slightly less accurate,
2. but remain admissible.

The procedure asat(ϕ, D)
Definition

Definition

Let D be a consistent set of literals. Then define

$$
\begin{aligned}
\text{asat}(D, \bot) &= \text{false} \\
\text{asat}(D, \top) &= \text{true} \\
\text{asat}(D, a) &= \text{true iff } \neg a \notin D \quad (\text{for } a \in A) \\
\text{asat}(D, \neg a) &= \text{true iff } a \notin D \quad (\text{for } a \in A) \\
\text{asat}(D, \neg\neg\phi) &= \text{asat}(D, \phi) \\
\text{asat}(D, \phi \vee \psi) &= \text{asat}(D, \phi) \text{ or asat}(D, \psi) \\
\text{asat}(D, \phi \wedge \psi) &= \text{asat}(D, \phi) \text{ and asat}(D, \psi) \\
\text{asat}(D, \neg(\phi \vee \psi)) &= \text{asat}(D, \neg\phi) \text{ and asat}(D, \neg\psi) \\
\text{asat}(D, \neg(\phi \wedge \psi)) &= \text{asat}(D, \neg\phi) \text{ or asat}(D, \neg\psi)
\end{aligned}
$$

The procedure asat(D, ϕ)

Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)
Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)

Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)

Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)
Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)
Examples

1. asat$(\emptyset, a)$ = true
2. asat$(\{\neg a\}, a)$ = false
3. asat$(\{\neg b\}, a)$ = true
4. asat$(\{\neg a, \neg b\}, a \wedge b)$ = false
5. asat$(\emptyset, a \wedge \neg a)$ = true but $a \wedge \neg a$ is not satisfiable!!!
6. asat$(\{\neg b, \neg c\}, a \wedge (b \vee c))$ = true

The procedure asat(D, ϕ)

Correctness

Lemma (ASAT)

Let ϕ be a formula and D a consistent set of literals (i.e. $\{a, \neg a\} \not\subseteq D$ for all $a \in A$.)
If $D \cup \{\phi\}$ is satisfiable then asat(D, ϕ) returns true.

Proof.

By induction on the structure of ϕ.

Base case 1 $\phi = \bot$: The set $D \cup \{\bot\}$ is not satisfiable, and hence the implication trivially holds.

Base case 2 $\phi = \top$: asat$(D, \top)$ always returns true, and hence the implication trivially holds.

Base case 3 $\phi = a$ for some $a \in A$: If $D \cup \{a\}$ is satisfiable, then $\neg a \notin D$, and hence asat(D, a) returns true.

The procedure asat(D, ϕ)
Correctness

Lemma (ASAT)

Let ϕ be a formula and D a consistent set of literals (i.e. $\{a, \neg a\} \not\subseteq D$ for all $a \in A$.)
If $D \cup \{\phi\}$ is satisfiable then asat(D, ϕ) returns true.

Proof.

By induction on the structure of ϕ.

Base case 1 $\phi = \bot$: The set $D \cup \{\bot\}$ is not satisfiable, and hence the implication trivially holds.

Base case 2 $\phi = \top$: asat$(D, \top)$ always returns true, and hence the implication trivially holds.

Base case 3 $\phi = a$ for some $a \in A$: If $D \cup \{a\}$ is satisfiable, then $\neg a \notin D$, and hence asat(D, a) returns true.

The procedure asat(D, ϕ)
Correctness

Lemma (ASAT)

Let ϕ be a formula and D a consistent set of literals (i.e. $\{a, \neg a\} \not\subseteq D$ for all $a \in A$.)
If $D \cup \{\phi\}$ is satisfiable then asat(D, ϕ) returns true.

Proof.

By induction on the structure of ϕ.

Base case 1 $\phi = \bot$: The set $D \cup \{\bot\}$ is not satisfiable, and hence the implication trivially holds.

Base case 2 $\phi = \top$: asat$(D, \top)$ always returns true, and hence the implication trivially holds.

Base case 3 $\phi = a$ for some $a \in A$: If $D \cup \{a\}$ is satisfiable, then $\neg a \notin D$, and hence asat(D, a) returns true.

proof continues.

Base case 4 $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable then $a \notin D$ and asat$(D, \neg a)$ returns true.

Inductive case 1 $\phi = \neg\neg\phi'$: ϕ and ϕ' are equivalent: claim follows from the induction hypothesis.

Inductive case 2 $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable, and by the induction hypothesis asat(D, ϕ_1) or asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3 $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable, and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5 $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

proof continues.

Base case 4 $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable then $a \notin D$ and asat$(D, \neg a)$ returns true.

Inductive case 1 $\phi = \neg\neg\phi'$: ϕ and ϕ' are equivalent: claim follows from the induction hypothesis.

Inductive case 2 $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable, and by the induction hypothesis asat(D, ϕ_1) or asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3 $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable, and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5 $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

proof continues.

Base case 4 $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable then $a \notin D$ and asat$(D, \neg a)$ returns true.

Inductive case 1 $\phi = \neg\neg\phi'$: ϕ and ϕ' are equivalent: claim follows from the induction hypothesis.

Inductive case 2 $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable, and by the induction hypothesis asat(D, ϕ_1) or asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3 $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable, and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5 $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

proof continues.

Base case 4 $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable then $a \notin D$ and asat$(D, \neg a)$ returns true.

Inductive case 1 $\phi = \neg\neg\phi'$: ϕ and ϕ' are equivalent: claim follows from the induction hypothesis.

Inductive case 2 $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable, and by the induction hypothesis asat(D, ϕ_1) or asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3 $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable, and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5 $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

The procedure asat(D, ϕ)
Correctness

proof continues.

Base case 4 $\phi = \neg a$ for some $a \in A$: If $D \cup \{\neg a\}$ is satisfiable then $a \notin D$ and asat$(D, \neg a)$ returns true.

Inductive case 1 $\phi = \neg\neg\phi'$: ϕ and ϕ' are equivalent: claim follows from the induction hypothesis.

Inductive case 2 $\phi = \phi_1 \vee \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then $D \cup \{\phi_1\}$ or $D \cup \{\phi_2\}$ is satisfiable, and by the induction hypothesis asat(D, ϕ_1) or asat(D, ϕ_2) returns true. Hence asat$(D, \phi_1 \vee \phi_2)$ returns true.

Inductive case 3 $\phi = \phi_1 \wedge \phi_2$: If $D \cup \{\phi\}$ is satisfiable, then both $D \cup \{\phi_1\}$ and $D \cup \{\phi_2\}$ are satisfiable, and by the induction hypothesis both asat(D, ϕ_1) and asat(D, ϕ_2) return true. Hence asat$(D, \phi_1 \wedge \phi_2)$ returns true.

Inductive cases 4 and 5 $\phi = \neg(\phi' \vee \psi')$ and $\phi = \neg(\phi' \wedge \psi')$: Like cases 2 and 3 by logical equivalence.

Relation between max-distances and distances

changes required for $\models^*$

proof continues.

Inductive case $i \geq 1$: Let s be any state in D_i^{fwd}. We show that $s \models D_i^{max}$. Let l be any literal in D_i^{max}.

1. Assume $s \in D_{i-1}^{fwd}$. As $D_i^{max} \subseteq D_{i-1}^{max}$ also $l \in D_{i-1}^{max}$. By the induction hypothesis $s \models l$.
2. Otherwise $s \in D_i^{fwd} \setminus D_{i-1}^{fwd}$.
   Hence there is $o \in O$ and $s_0 \in D_{i-1}^{fwd}$ with $s = app_o(s_0)$.
   By $D_i^{max} \subseteq D_{i-1}^{max}$ and the induction hypothesis $s_0 \models l$.
   As $l \in D_i^{max}$, not asat$(D_{i-1}^{max}, EPC_{\bar{l}}(o))$ by def. of D_i^{max}.
   Not asat$(D_{i-1}^{max}, EPC_{\bar{l}}(o))$ implies not $SAT(D_{i-1}^{max} \cup \{EPC_{\bar{l}}(o)\})$.
   By $s_0 \in D_{i-1}^{fwd}$ and the induction hypothesis $s_0 \models D_{i-1}^{max}$.
   Hence $s_0 \not\models EPC_{\bar{l}}(o)$.
   By Lemma B applying o in s_0 does not make l false.