Planning and Optimization

G3. Landmarks: Orderings & LM-Count Heuristic

Malte Helmert and Gabriele Röger

Universität Basel

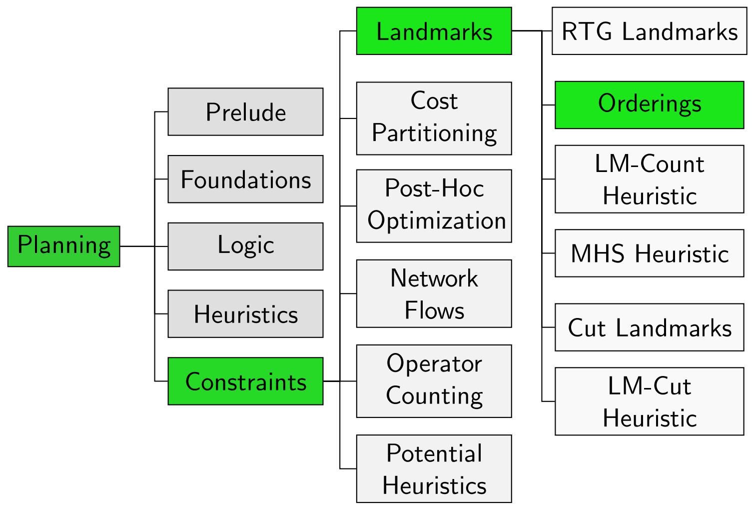
November 30, 2022

Planning and Optimization

November 30, 2022 — G3. Landmarks: Orderings & LM-Count Heuristic

G3.1 Landmark Orderings

Content of this Course



Why Landmark Orderings?

- To compute a landmark heuristic estimate for state s we need landmarks for s.
- We could invest the time to compute them for every state from scratch.
- Alternatively, we can compute landmarks once and propagate them over operator applications.
- Landmark orderings are used to detect landmarks that should be further considered because they (again) need to be satisfied later.
- (We will later see yet another approach, where heuristic computation and landmark computation are integrated ⇝ LM-Cut.)

Example

Consider task $\langle\{a,b,c,d\}, I, \{o_1, o_2, \dots, o_n\}, d\rangle$ with

- $I(v) = \bot$ for $v \in \{a,b,c,d\}$,
- $o_1 = \langle \top, a \wedge b\rangle$, and
- $o_2 = \langle a, c \wedge \neg a \wedge \neg b\rangle$.

You know that a, b, c and d are all fact landmarks for I.

- What landmarks are still required to be made true in state $I[\![\langle o_1, o_2\rangle]\!]$?
- You get the additional information that variable a must be true immediately before d is first made true. Any changes?

Terminology

Let $\pi = \langle o_1, \dots, o_n\rangle$ be a sequence of operators applicable in state I and let φ be a formula over the state variables.

- φ is true at time i if $I[\![\langle o_1, \dots, o_i\rangle]\!] \models \varphi$.
- Also special case $i = 0$: φ is true at time 0 if $I \models \varphi$.
- No formula is true at time $i < 0$.
- φ is added at time i if it is true at time i but not at time $i - 1$.
- φ is first added at time i if it is true at time i but not at any time $j < i$.
  We denote this i by $\mathit{first}(\varphi, \pi)$.
- $\mathit{last}(\varphi, \pi)$ denotes last time in which φ is added in π.

Landmark Orderings

Definition (Landmark Orderings)

Let φ and ψ be formula landmarks. There is

- a natural ordering between φ and ψ (written $\varphi \to \psi$) if in each plan π it holds that $\mathit{first}(\varphi, \pi) < \mathit{first}(\psi, \pi)$.
  "φ must be true some time strictly before ψ is first added'.'
- a greedy-necessary ordering between φ and ψ (written $\varphi \to_{gn} \psi$) if for every plan $\pi = \langle o_1, \dots, o_n\rangle$ it holds that $s[\![\langle o_1, \dots, o_{\mathit{first}(\psi,\pi)-1}\rangle]\!] \models \varphi$.
  "φ must be true immediately before ψ is first added'.'
- a reasonable ordering between φ and ψ (written $\varphi \to_r \psi$) if in each plan π it holds that $\mathit{first}(\varphi, \pi) \le \mathit{last}(\psi, \pi)$.
  "φ must be true some time before ψ is last added'.'

Natural Orderings

Definition

There is a natural ordering between φ and ψ (written $\varphi \to \psi$) if in each plan π it holds that $first(\varphi, \pi) < first(\psi, \pi)$.

- We can directly determine natural orderings from the LM sets computed from the simplified relaxed task graph.
- For fact landmarks v, v' with $v \neq v'$, if $n_{v'} \in LM(n_v)$ then $v' \to v$.

Greedy-necessary Orderings

Definition

There is a greedy-necessary ordering between φ and ψ (written $\varphi \to_{\mathrm{gn}} \psi$) if in each plan where ψ is first added at time i, φ is true at time $i - 1$.

- We can again determine such orderings from the sRTG.
- For an OR node n_v, we define the set of first achievers as $FA(n_v) = \{n_o \mid n_o \in succ(n_v) \text{ and } n_v \notin LM(n_o)\}$.
- Then $v' \to_{\mathrm{gn}} v$ if $n_{v'} \in succ(n_o)$ for all $n_o \in FA(n_v)$.

G3.2 Landmark Propagation

Example Revisited

Consider task $\langle \{a, b, c, d\}, I, \{o_1, o_2, \dots, o_n\}, d \rangle$ with

- $I(v) = \bot$ for $v \in \{a, b, c, d\}$,
- $o_1 = \langle \top, a \wedge b \rangle$ and $o_2 = \langle a, c \wedge \neg a \wedge \neg b \rangle$.

You know that a, b, c and d are all fact landmarks for I.

- What landmarks are still required to be made true in state $I[\![\langle o_1, o_2 \rangle]\!]$? All not achieved yet on the state path
- You get the additional information that variable a must be true immediately before d is first made true. Any changes? Exploit orderings to determine landmarks that are still required.
- There is another path to the same state where b was never true. What now? Exploit information from multiple paths.

Context in Search

A landmark graph captures all known landmark information for a current state.

LM-BFS algorithm

$graphs$[init()] := compute_landmark_graph(init())
$open$.insert(init())
while $open \neq \emptyset$ **do**
  $s = open$.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G} = graphs[s]$
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs($graphs[s'], \mathcal{G}'$)
    $graphs[s']$:= extend_landmark_graph($\mathcal{G}'', s'$)
    $open$.insert(s')

Landmark Graph

We combine all known landmark information for the current state in a landmark graph.

Definition (Landmark Graph)

Let Π be a planning task, s be a state of Π and $\mathcal{L}$ be a set of formula landmarks for the initial state with set of orderings $\mathcal{O}$.

A landmark graph for state s is a triple $\mathcal{G} = \langle \mathcal{L}^+, \mathcal{L}^-, \mathcal{O} \rangle$, where $\mathcal{L}^+, \mathcal{L}^- \subseteq \mathcal{L}$ and

- $\mathcal{L}^+$ contains landmarks that were already true in all considered paths to s and
- $\mathcal{L}^-$ contains landmarks for s that are not true in s.

Initial Landmark Graph

LM-BFS algorithm

$graphs$[init()] := compute_landmark_graph(init())
$open$.insert(init())
while $open \neq \emptyset$ **do**
  $s = open$.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G} = graphs[s]$
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs($graphs[s'], \mathcal{G}'$)
    $graphs[s']$:= extend_landmark_graph($\mathcal{G}'', s'$)
    $open$.insert(s')

Compute $\mathcal{L}$ and $\mathcal{O}$ and return
$\langle \{\lambda \in \mathcal{L} \mid \text{init}() \models \lambda\}, \{\lambda \in \mathcal{L} \mid \text{init}() \not\models \lambda\}, \mathcal{O} \rangle$

Progression for a Transition

LM-BFS algorithm

$graphs$[init()] := compute_landmark_graph(init())
$open$.insert(init())
while $open \neq \emptyset$ **do**
  $s = open$.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G} = graphs[s]$
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs($graphs[s'], \mathcal{G}'$)
    $graphs[s']$:= extend_landmark_graph($\mathcal{G}'', s'$)
    $open$.insert(s')

Progression for a Transition

progress_landmark_graph($\langle \mathcal{L}^+, \mathcal{L}^-, \mathcal{O} \rangle, a, s'$)

$accept := \{\varphi \in \mathcal{L}^- \mid s' \models \varphi\}$
$\mathcal{L}'^+ := \mathcal{L}^+ \cup accept$
$\mathcal{L}'^- := \mathcal{L}^- \setminus accept$
return $\langle \mathcal{L}'^+, \mathcal{L}'^-, \mathcal{O} \rangle$

Exploit Information from Multiple Paths

LM-BFS algorithm

graphs[init()] := compute_landmark_graph(init())
open.insert(init())
while *open* $\neq \emptyset$ **do**
  s = *open*.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G}$ = *graphs*[s]
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs(*graphs*[s'], $\mathcal{G}'$)
    graphs[s'] := extend_landmark_graph($\mathcal{G}'', s'$)
    open.insert(s')

Exploit Information from Multiple Paths

merge_landmark_graphs($\langle \mathcal{L}_1^+, \mathcal{L}_1^-, \mathcal{O} \rangle, \langle \mathcal{L}_2^+, \mathcal{L}_2^-, \mathcal{O} \rangle$)

$\mathcal{L}^+ := \mathcal{L}_1^+ \cap \mathcal{L}_2^+$
$\mathcal{L}^- := \mathcal{L}_1^- \cup \mathcal{L}_2^-$
return $\langle \mathcal{L}^+, \mathcal{L}^-, \mathcal{O} \rangle$

Exploit Ordering Information

LM-BFS algorithm

graphs[init()] := compute_landmark_graph(init())
open.insert(init())
while *open* $\neq \emptyset$ **do**
  s = *open*.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G}$ = *graphs*[s]
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs(*graphs*[s'], $\mathcal{G}'$)
    graphs[s'] := extend_landmark_graph($\mathcal{G}'', s'$)
    open.insert(s')

How do we define *graphs*[s'] if we encounter s' for the first time?

Exploit Ordering Information

LM-BFS algorithm

graphs[init()] := compute_landmark_graph(init())
open.insert(init())
while *open* $\neq \emptyset$ **do**
  $s = open$.pop()
  if is_goal(s) **then return** extract_plan(s);
  $\mathcal{G} = graphs[s]$
  foreach $\langle a, s' \rangle \in succ(s)$ **do**
    $\mathcal{G}'$:= progress_landmark_graph($\mathcal{G}, a, s'$)
    $\mathcal{G}''$:= merge_landmark_graphs($graphs[s'], \mathcal{G}'$)
    $graphs[s']$:= extend_landmark_graph($\mathcal{G}'', s'$)
    open.insert(s')

Exploit Ordering Information

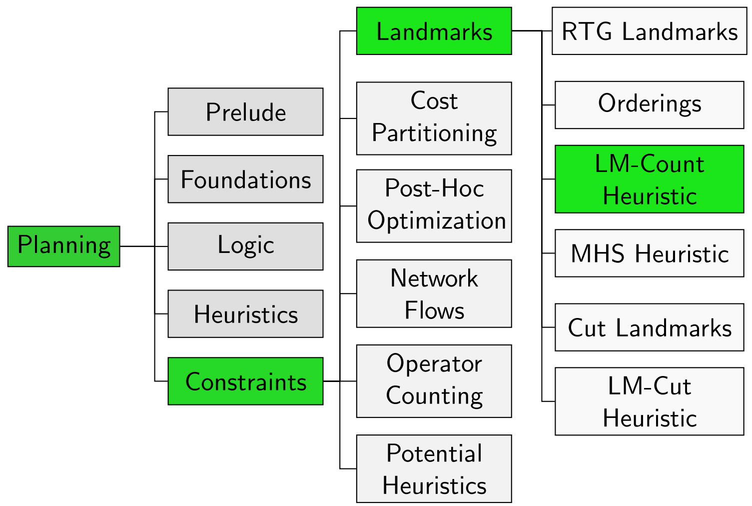
extend_landmark_graph($\langle \mathcal{L}^+, \mathcal{L}^-, \mathcal{O} \rangle, s'$)

$\mathcal{L}_G := \{\varphi \in \mathcal{L}^+ \mid s' \not\models \varphi$ and φ is implied by goal$\}$
$\mathcal{L}_{\mathrm{gn}} := \{\varphi \in \mathcal{L}^+ \mid s' \not\models \varphi$ and $\exists \varphi \rightarrow_{\mathrm{gn}} \psi \in \mathcal{O} : \psi \notin \mathcal{L}^+\}$
$\mathcal{L}_{\mathrm{r}} := \{\varphi \in \mathcal{L}^+ \mid \exists \psi \rightarrow_{\mathrm{r}} \varphi \in \mathcal{O}$ with $\psi \notin \mathcal{L}^+\}$
$\mathcal{L}'^- := \mathcal{L}^- \cup \mathcal{L}_G \cup \mathcal{L}_{\mathrm{gn}} \cup \mathcal{L}_{\mathrm{r}}$
return $\langle \mathcal{L}^+, \mathcal{L}'^-, \mathcal{O} \rangle$

- $\mathcal{L}_G$: "currently false but must be true in the goal"
- $\mathcal{L}_{\mathrm{gn}}$: "currently false but must be true immediately before some unachieved landmark becomes true"
- $\mathcal{L}_{\mathrm{r}}$: "must again become true after some unachieved landmark became true"

G3.3 Landmark-count Heuristic

Content of this Course



Landmark-count Heuristic

The landmark-count heuristic counts the landmarks that still have to be achieved.

Definition (LM-count Heuristic)

Let Π be a planning task, s be a state and $\mathcal{G} = \langle \mathcal{L}^+, \mathcal{L}^-, \mathcal{O} \rangle$ be a landmark graph for s.

The LM-count heuristic for s and $\mathcal{G}$ is

$$h^{\text{LM-count}}_{\mathcal{L}}(\mathcal{G}) = |\mathcal{L}^-|.$$

In the original work, the set $\mathcal{L}^-$ was determined without considering information from multiple paths.

LM-count Heuristic is Path-dependent

- LM-count heuristic gives estimates for landmark graphs, which depend on the considered paths.
- Search algorithms need estimates for states.
- $\rightsquigarrow$ we use estimate from the current landmark graph.
- $\rightsquigarrow$ heuristic estimate for a state is not well-defined.

LM-count Heuristic is Inadmissible

Example

Consider STRIPS planning task $\Pi = \langle \{a, b\}, \emptyset, \{o\}, \{a, b\} \rangle$ with $o = \langle \emptyset, \{a, b\}, \emptyset, 1 \rangle$. Let $\mathcal{L} = \{a, b\}$ and $\mathcal{O} = \emptyset$.

The estimate for the initial state is $h^{\text{LM-count}}(\langle \emptyset, \{a, b\}, \emptyset \rangle) = 2$ while $h^*(I) = 1$.

$\rightsquigarrow$ $h^{\text{LM-count}}$ is inadmissible.

LM-count Heuristic: Comments

- LM-Count alone is not a particularily informative heuristic.
- On the positive side, it complements h^{FF} very well.
- For example, the LAMA planning system alternates between expanding a state with minimal h^{FF} and minimal $h^{\text{LM-count}}$ estimate.
- There is an admissible variant of the heuristic based on operator cost partitioning.

G3.4 Summary

Summary

- We can propagate landmark sets over action applications.
- Landmark orderings can be useful for detecting when a landmark that has already been achieved should be further considered.
- We can combine the landmark information from several paths to the same state.
- The LM-count heuristic counts how many landmarks still need to be satisfied.
- The LM-count heuristic is inadmissible (but there is an admissible variant).