Part VIII

Heuristics in Planning

Introduction

- Why heuristics are interested for planning ?
  - Although planning systems have become much more efficient, they still suffer from combinatorial complexity. Even restrited planning domains, the complexity can be intractable in the worst case
- Approach to study heuristics
  - Define a nondeterministic abstract search procedure in a space in which each node u, (i.e., structured collection of actions and constraints) represents a set of solution Π_u, (i.e., the set of all solution reachable from u), For instance, u is
    - in state-space planning, a simple sequence of actions
    - in plan-space planning, a set of actions, causal links, orderig constraints and bindings constraints
    - in graph based planning, a subgraph of the planning graph
    - etc.

Abstract Search Procedure (1/2)

- The abstract search procedure involves three main steps in addition to a terminaison step:
  1. A refinement step consists of modifying the collection of actions and/or constraints associated with a node u. In a refinement of u, the set of solution Π_u remains **u** nchanged
     - For instance, if we find out there is only one action a that meets a constraint in u, a is maked an explicit part of u and the constraints is removed
  2. A branching setp generates on or more children of u. These nodes will be the next candidates for the next node to visit
     - For instance, in forward state-space seach, each child corresponds to appending a different action to the end of a partial plan
  3. A pruning step consists of removing from the set of candidates nodes some nodes that appear to be unpromising for the search
     - For instance, a node migth be considered to be unpromising if we have a record of having already visited that node

Abstract Search Procedure (2/2)

Algorithm (Abstract-search(u))

```
if Terminal(u) then
    return u
else
    u ← Refine(u)
    B ← Branch(u)
    C ← Prune(B)
    if C = ∅ then
        return Failure
    else
        nondeterministically choose any v ∈ C
        return Abstract-search(v)
    end
end
```

Abstract Search Procedure for Plan-Space Planning

The different steps of the abstract search procedure for plan-space planning are the following:

1. **Branching** consists of selecting flaws and finding its resolvers
2. **Refinement** consists of applying a resolver to the current partial plan
3. **Pruning** : there is no pruning step
4. **Terminaison** occurs when no flaws are left in the partial plan

Note

Since paths in the plan space are likely to be infinite, a control strategy such as best-first search or iterative deepening should be used

Abstract Search Procedure for State-Space Planning

The different steps of the abstract search procedure for state-space planning are the following:

1. **Branching** are defined by actions
2. **Refinement** : there is no branching step
3. **Pruning** removes candidate nodes corresponding to cycle
4. **Terminaison** occurs when the plan goes all the way from the initial state to a goal

Note

A control strategy such as A*, branch-and-bound search or iterative deepening should be used

Abstract Search Procedure for Graph-Based Planning

The different steps of the abstract search procedure for graph-based planning are the following:

1. **Branching** idendifies possible actions that achieve subgoals
2. **Refinement** consists of propaging constraints for actions chosen in the branching step
3. **Pruning** uses the recorded nogood tuples of subgoals that failed in some layer
4. **Terminaison** occurs if the solution-extraction process succeeds

Note

Graph-based planning correspond to using abstract search procedure with iterative deepening control strategy.

Deterministic versus undeterministic search

- To implement a deterministic search procedure a node selection function (`Select(C)`) is needed to choose which node u to visite next from a set of candidates C
- Often the deterministic search is done in a depth-first manner

$\Longrightarrow$

Algorithm (Depth-first-search(u))

if `Terminal`(u) **then return** u
else
    $u \leftarrow$ `Refine`(u)
    $B \leftarrow$ `Branch`(u)
    $C \leftarrow$ `Prune`(B)
    while $C = \emptyset$ **do**
        $v \leftarrow$ `Select`(C)
        $C \leftarrow C - \{v\}$
        $\pi \leftarrow$ `Depth-first-search`(v)
        if $\pi \neq$ Failure **then return** π
        return Failure
    end
end

Design Principle for Heuristics : Relaxation

Node selection heuristic

Node selection heuristic

A node selection heuristic is any way of ranking a set of nodes in order of ther relative desirability. We will model this heuristic as function h that can be used to compute a numeric evaluation $h(u)$ for each candidates node $u \in C$, i.e.,

$$\texttt{Select}(C) = min\{h(u) \mid u \in C\}$$

Notes

1. Node selection heuristics are used for resolving nondeterministic choices
2. If there is a deterministic technique for choosing at each point the rigth node, this technique is not a heuristic
3. A node selection heuristic not always garantees to be the best choice but often lead to the best solution
4. A node selection heuristic must be easy to compute

Relaxation Principle

Node selection heuristics are often based on relaxation priciple:

Relaxation Principle

In order to assess how desirable a node u is, one considers a simpler problem that is obtained from the original one by making simplifying assumptions and by relaxing constraints

- One estimates how desirable u is by using u to solve the simpler relaxed problem and using that solution as an estimate of the solution one would get if one used u to solve the original problem
- On the other hand, the more simplified the relaxed problem is, the easier it will be to compute the heuristic

Admissible Node Selection Heuritic

Admissible Node Selection Heuristic

A node selection heuristic h is admissible if it is a lower bound estimate cost of a minimal solution reachable from u, i.e., $h(u) \leq h^*(u)$ with $h^*(u)$ the minimum cost of any solution reachable from u

- $h^*(u) = \infty$ if no solution is reachable from u

Notes

1. Admissible node selection heuristic is desirable if one seeks a optimal solution with respect to some cost criterion, e.g., path-finding A*
2. Heuristic search as iterative-deepening scheme, are usually able to garantee on optimal solution when guided with an admissible node selection heuristic

Heuristics for State-Space Planning

Reminder

- In state-space planning, each node u corresponds to a state s
- At some point the candicates nodes are the sucessor states of the current state s, for the actions applicable to s. For each action a to a state s:
  - In forward search the next state is given by the transition function:

$$\gamma(s,a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$$

  - In backward search the next state is given by the transition function:

$$\gamma(s,a)^{-1} = (s - \text{effects}^+(a)) \cup \text{precond}(a)$$

Relaxation principle

In order to choose the most preferable candidate state, we need to assess how close each action may bring us to the goal (forward search) or initial state s_0 (backward search).

A Simple Relaxation Heuristic (1/2)

- Simple relaxation heuristic idea
  - A very simple relaxation heuristic is to neglect $\text{effects}^-(a)$
- Consequences:
  - $\gamma(s,a)$ involves on a monotonic increase in the number of propositions of s
  - It is easier to compute distance goal with such simplified γ

Definition (Simple Relaxation Heuristic)

Let $s \in S$ be a state, p a proposition and g a set of propositions. The minimum distance from s to g, denoted $\Delta^*(s,g)$, is the minimum number of actions required to reach from s a state containing all proposition $p \in g$.

A Simple Relaxation Heuristic (2/2)

- Δ is given by the following equations:

$$\Delta(s,p) = 0 \quad \text{if } p \in s$$
$$\Delta(s,p) = \infty \quad \text{if } \forall a \in A, p \notin \text{effects}^+(a)$$
$$\Delta(s,g) = 0 \quad \text{if } g \subseteq s$$

otherwise :

$$\Delta(s,p) = \min_a\{1 + \Delta(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\}$$
$$\Delta(s,g) = \Sigma_{p \in g} \Delta(s,p)$$

Notes

1. These equations give the distance to g in the relaxed problem and
2. an estimate distance in the unrelaxed problem
3. The heuristic function can be define as $h(s) = \Delta(s,g)$

The Δ-Algorithm

- The Δ-algorithm is polynomial in time
- As minimum distance graph searching, the algorithm stops when a fixed point is reached

Algorithm (Delta(s))

foreach p **do**
  if $p \in s$ **then** $\Delta(s,p) \leftarrow 0$
  else $\Delta(s,p) \leftarrow \infty$
  $U \leftarrow \{s\}$
end
repeat
  foreach *a such that* $\exists u \in U$, *precond(a)* $\subseteq u$ **do**
    $U \leftarrow \{u\} \cup effects^+(a)$
    foreach $p \in effects^+(a)$ **do**
      $\Delta(s,p) \leftarrow min\{\Delta(s,p), 1 + \Sigma_{q \in precond(a)} \Delta(s,q)\}$
    end
  end
until *no change occurs in the above updates*

Heuristics Guided Forward Search

Algorithm (Heuristic-forward-Search(π, s, g, A))

if *s satisfies g* **then return** π

options $\leftarrow \{a \in \ \mid$ *a applicable to s*$\}$
foreach $a \in$ options **do** $\Delta(\gamma(s,a))$

while options $\neq \emptyset$ **do**
  $a \leftarrow min\{\Delta(\gamma(s,a), g) \mid a \in$ options $\}$
  options $\leftarrow$ options $-\{a\}$
  $\pi' \leftarrow$ `Heuristic-forward-Search`$(\pi, a, \gamma(s,a), g, A)$
  if $\pi' \neq$ Failure **then return** π'
end
return Failure

Heuristics Guided Backward Search

Algorithm (Heuristic-backward-Search(π, s_0, g, A))

if s_0 *satisfies g* **then return** π

options $\leftarrow \{a \in \ \mid$ *a revelant for g*$\}$
while options $\neq \emptyset$ **do**
  $a \leftarrow min\{\Delta(s, \gamma^{-1}(g,a)) \mid a \in$ options $\}$
  options $\leftarrow$ options $-\{a\}$
  $\pi' \leftarrow$ `Heuristic-backward-Search`$(a \cdot \pi, s_0, \gamma^{-1}(g,a), A)$
  if $\pi' \neq$ Failure **then return** π'
end
return Failure

Notes

1. We suppose that Δ-algorithm is run once initially
2. The backward search is more efficient than forward search because it has to be run less Δ-algorithm

Admissible State-Space Heuristics

- It can be desirable to use admissible heuristic function for two reasons:
  1. It may be interested in getting the shortest plan, e.g., cost may be associated to actions
  2. Admissible permit a safe pruning
     - If Y is the length of a plan and if $h(u) < Y$, h being admissible, then we are sure that non solution plan of length smaller that Y can be obtained from u.
       $\Rightarrow$ pruning does not affect completeness

Exercice

Is the simple heuristic h previouly introduced admissible ?
No, because $\Delta(s,g)$ is not a lower bound on the true minimal distance $\Delta^*(s,g)$. Assume a problem where there is an action a such that:

- *precond(a)* $\subseteq s_0$,
- *effects*$^+$*(a)* $= g$ *and*
- $s_0 \cap g = \emptyset$.

The distance to the goal is 1, but $\Delta(s_0, g) = \Sigma_{p \in g} \Delta(s_0, p) = |g|$

First Admissible heuristic

Idea

Instead of estimating the distance to a set of propositions g to be the sum of the distances to the elements of g, we estimate it to be the maximum distance to its propositions

- Now, Δ_1 is given by the following equations:

$$\Delta_1(s,p) = 0 \quad \text{if } p \in s$$
$$\Delta_1(s,p) = \infty \quad \text{if } \forall a \in A, p \notin \text{effects}^+(a)$$
$$\Delta_1(s,g) = 0 \quad \text{if } g \subseteq s$$

otherwise :

$$\Delta_1(s,p) = \min_a\{1 + \Delta_1(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\}$$
$$\Delta_1(s,g) = \max\{\Delta_1(s,p) \mid p \in g\}$$

- Experience shows that h_1 is not as informative as h even if h_1 is admissible

Second Admissible heuristic

Idea

Instead of considering that the distance to a set of propositions g is the maximum distance to propositions $p \in g$, we estimate it to be the maximum distance to a pair of propositions $\{p,q\}$

- Now, Δ_2 is given by the following recusive equations (terminaison cases remain unchanged):

$$\begin{aligned}
\Delta_2(s,p) &= \min_a\{1 + \Delta_2(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
\Delta_2(s,\{p,q\}) &= \min\{ \\
&\quad \min_a\{1 + \Delta_2(s, \text{precond}(a)) \mid \{p,q\} \in \text{effects}^+(a)\} \\
&\quad \min_a\{1 + \Delta_2(s, \{q\} \cup \text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
&\quad \min_a\{1 + \Delta_2(s, \{p\} \cup \text{precond}(a)) \mid q \in \text{effects}^+(a)\}\} \\
\Delta_2(s,g) &= \max_{p,q}\{\Delta_2(s,\{p,q\}) \mid \{p,q\} \subseteq g\}
\end{aligned}$$

Reminder : Graphplan Algorithm

Algorithm (GraphPlan(A, s_0, g))

$i \leftarrow 0, \nabla \leftarrow \emptyset, P_0 \leftarrow s_0$
repeat
  $i \leftarrow i+1$, $G \leftarrow$ `Expand`(G)
until $[g \subseteq P_i$ *and* $g \cap \mu P_i = \emptyset]$ *or* `Fixedpoint`(G)
if $g \not\subseteq P_i$ *or* $g \cap \mu P_i \neg \emptyset$ **then return** Failure
$\Pi \leftarrow$ `Extract`(G, g, i)
if `Fixedpoint`(G) **then return** $\eta \leftarrow |\nabla(\kappa)|$
else $\eta \leftarrow 0$
while $\Pi =$ Failure **do**
  $i \leftarrow i+1$, $G \leftarrow$ `Expand`(G), $\Pi \leftarrow$ `Extract`(G, g, i)
  if $\Pi =$ Failure *and* `Fixedpoint`(G) **then**
    if $\eta = |\nabla(\kappa)|$ **then return** Failure
    $\eta \leftarrow |\nabla(\kappa)|$
  end
end
return Π

Comments

- Graphplan looks like heuritic backward search procedure
  - Δ-procedure and `Expand` procedure in graphplan perform a reachability analysis
  - The main difference :
    - `Expand` builds a data stucture, the planning graph, which provides more information attached to propositions not just distance to s_0
- The planning graph approximate the distance $\Delta^*(s_0, g)$, that is the level of the first layer of the graph that $g \subseteq P_i$ and no pair of g is in μP_i
- Graphplan can be viewed as a heuristic search planner that first computes the distance estimates in a forward propagation manner and then searches backward from the goal using a iterative-deepening strategy augmented with a learning mechanisms (nogoods hashtable)

Heuristics for Plan-Space Planning

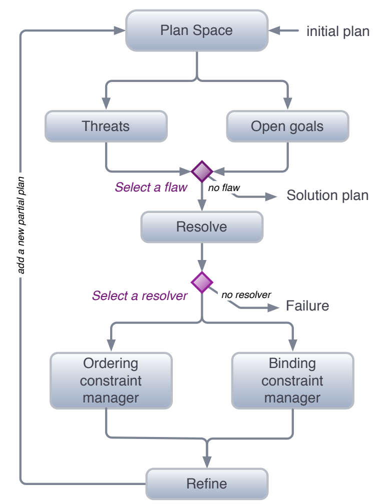
Reminder: PSP Procedure

Algorithm (PSP(π))

flaws $\leftarrow$ `OpenGoals`(π) $\cup$
`Threat`(π)
if flaws $= \emptyset$ **then return** π

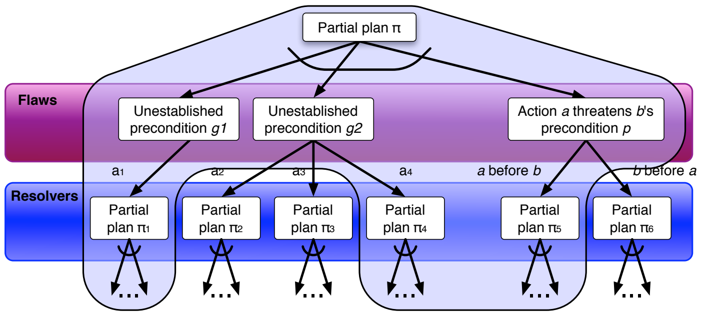
select any flaw sigma $\in$ flaws
resolvers $\leftarrow$ `Resolve`(σ,π)
if resolvers $= \emptyset$ **then return** Failure

nondeterministically choose a resolver $\rho \in$ resolvers
$\pi' \leftarrow$ `Refine`(ρ, π)
return `PSP`(π')



Reminder: Plan-Space

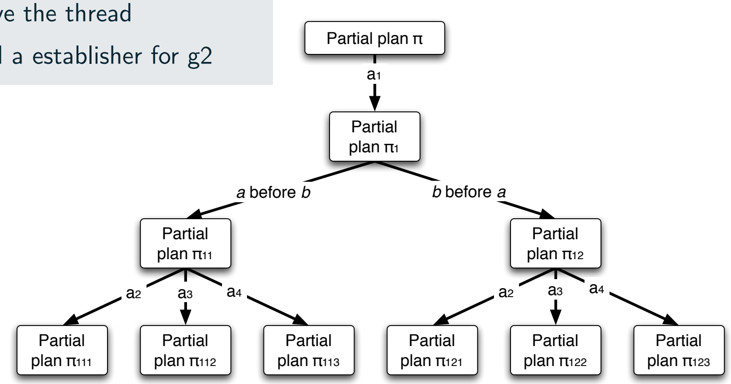
- Plan space can be viewed as AND/OR tree
- The flaw correspond to the AND branches
  - each flaw must be resolved in order to find a solution plan
- The resolver correspond to the OR branches
  - only one resolver is needed in order to a solution plan



Serialization tree example (1/3)

PSP choices

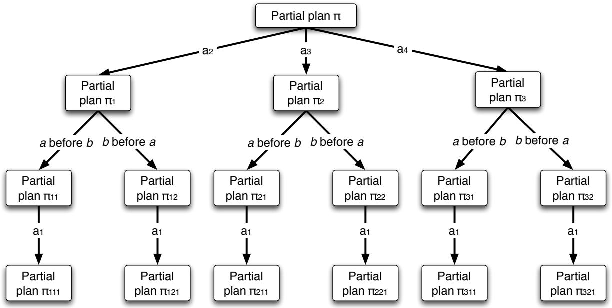
1. find an establisher for g1
2. solve the thread
3. find a establisher for g2



Serialization tree example (2/3)

PSP choices

1. find a establisher for g2
2. solve the thread
3. find an establisher for g1



Serialization tree example (3/3)

- All serialization trees lead to exactly the same set of solutions
- All serialization trees do not contain the same number of nodes
- The speed of PSP varies significantly depending on the number of node explore. Thus PSP speeds depends on the order in which its selects flaws to resolve

Question

How to choose the flaw to resolve to reduce the number of nodes to explored ?

The FAF-Heuristic

Idea

The fewest alternatives first (FAF) is to choose the flaw having the smallest branching factor as early as possible in oder to limit the cost of eventual backtracks.

- The FAF-heuristic is easy to compute $\Theta(n)$ where n is the number of flaws in a partial plan
- The FAF-heuristic works relatively well compared with other flaw selection heuristics

Other Flaw-Selection Heuristics

- Zero-commitment: chooses flaw that has not already been choosen in order to cut as soon as possible unachievable branches (low overhead)
- Least-commitment: always selects a open goal which generates the fewest refined plans (higth overhead)
- Least-cost-flaw-repair : same as "Least-commitment" applied to the threat too (higth overhead)
- LIFO: Last in last out choice of the flaw (low overhead)
- ZLIFO: Threat are selected depending "LIFO" strategy and open goal depending "Zero-commitment" (low overhead)

Resolver-Selection Heuristics

- The technics presented for state space planning cannot be applied
  - because they rely on relaxed distances between states, while states are not explicit in the plan space
- Hence, we have to come up with other means to rank the candidate nodes, i.e., partial plan, at a search point

Simple Heuristics (1/2)

Idea

The choice of the resolver is based on an A* best-first search strategy with a heuristic

$$f(\pi) = g(\pi) + h(\pi)$$

where

- $g(\pi)$ the cost of the partial plan π and
- $h(\pi)$ estimate of the additionnal cost of the best complete solution that extends π

Simple Heuristics (2/2)

- To elaborate the simple heuristic we can used:
  1. the number of actions (S)
  2. the number of open goals (OC)
  3. the number of causal links (CL)
  4. the number of threats (UC)
- For instance UCPOP uses : S + OC + UC
- Experiments show that S + OC works relatively well compared with other heuristic combinaisons

Note

Due to causal links addition refinement mechanism, $f(\pi)$ is not admissible

Regression AND/OR Graph heuristic

Regression AND/OR Graph heuristic

For each $OC(\pi)$, the heuristic compute an AND/OR graph along regression steps defined by γ^{-1} down to some fixed level k. Let $\eta_k(OC(\pi))$ be the weighted sum of:

1. the number of actions in this graph that are not in π and
2. the number of subgoals remaining in its leaves that are not in the initial state s_0

Note

- η_k incurs a significant overhead

Heuristic based on planning graph

Planning Graph Heuristic

Instead of computing for each $OC(\pi)$ a regression AND/OR graph, this heuristic builds a planning graph once for the planning domain and uses it as follow in order to estimate $\eta_k(OC(\pi))$:

$$\eta_k(OC(\pi)) = \left\{ \begin{array}{ll} 0 & \text{if } OC(\pi) \subseteq s_0 \\ \infty & \text{if } \forall a \in A, a \text{ is not revelant for } OC(\pi) \\ \multicolumn{2}{l}{\max_p\{\delta_\pi(a) + \eta(\gamma^{-1}, a)) \mid p \in OC(\pi) \cap \text{effects}^+(a)} \\ & \text{and } a \text{ is relevant for } OC(\pi)\} \text{ otherwise} \end{array} \right\}$$

with $\delta_\pi(a) = 0$ when a is in π and $\delta_\pi(a) = 1$ otherwise

To go further

Exercice

Exercice 1

How many serialization trees are there for the AND/OR tree in slide 306 ?

Further readings

- X. Nguyen, S. Kambhampati, and R. Nigenda.
  Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search.
  Artificial Intelligence, 135(1-2):73 124, 2002.
- A. Gerevini and L. Schubert.
  Accelerating partial-order planners: Some techniques for effective search control and pruning.
  Journal of Artificial Intelligence Research, 5(1):95-137, 1996.
- B. Bonet and H. Geffner.
  Planning as heuristic search: New results.
  In Proceedings of European Conference on Artificial Intelligence, pages 360 372, 1999.