

COMP90054 – Week 5 tutorial

Prepared by Zijie Xu. Last updated: 23 August 2023

Generating heuristic function via relaxation

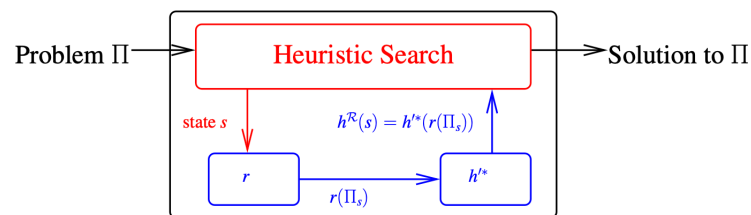
Objective

- Find an estimate of the perfect heuristic h^* for problem \mathcal{P}

Relaxation

- Relaxation is a method to compute heuristic functions
- Steps
 - Simplifying the original problem \mathcal{P} to a relaxed problem \mathcal{P}'
 - \mathcal{P}' has perfect heuristic h'^* which can be used to admissibly estimate h^*
 - Define a transformation r that simplify \mathcal{P} to \mathcal{P}'
 - Given a specific planning task $\Pi \in \mathcal{P}$, estimate $h^*(\Pi)$ by $h'^*(r(\Pi))$
- Relaxation can be native, efficiently constructible ($r(\Pi)$), and/or efficiently computable ($h'^*(\Pi')$)
 - if r or h'^* cannot be efficiently computed, we may choose to approximate them, design them so that they are feasible in most cases, or just use brute force 😊

Using a relaxation $\mathcal{R} = (\mathcal{P}', r, h'^*)$ during search:



→ Π_s : Π with initial state replaced by s , i.e., $\Pi = (F, A, c, I, G)$ changed to (F, A, c, s, G) .

Example

- Problem:** Grid world (move up/down/left/right, can't move through walls)
- Simplified problem \mathcal{P}' :** Grid world but can move through walls
- Transformation r :** Remove preconditions of checking wall at target cell
- Optimal Heuristic for \mathcal{P}' h'^* :** Manhattan distance from current position to goal

PDDL

Propositional Domain Definition Language

- Components of a PDDL planning task
 - Objects: Things in the world we defined
 - Predicates: Properties of the objects, can be true or false
 - Initial state: The state of the world that we start in
 - Goal specification: Things that we want to be true
 - Actions/Operators: Ways of changing the state of the world
- An implementation of PDDL planning task consists of two files
 - Domain file: defines the predicates and operators
 - Problem file: defines the objects, the initial state and the goal state
- Tips
 - $predicate(x, y)$ does not imply $predicate(y, x)$
 - `:typing` can be useful if have multiple types of objects, so PDDL only search the relevant type of variables during search

TSP Example

- Let C be the set of cities, E be the set of directed edges showing connected cities
- $P = \langle F, O, I, G \rangle$
 - $F = \{at(x), visited(x), connected(x, y) \mid x \in C, (x, y) \in E\}$
 - $I = \{at(sydney), visited(sydney), connected(x, y) \mid (x, y) \in E\}$
 - $G = \{visited(x) \mid x \in C\}$
 - $O = \left\{ move(x, y) \left\{ \begin{array}{l} pre: at(x), connected(x, y) \\ add: at(y), visited(y) \mid (x, y) \in E \\ del: at(x) \end{array} \right. \right\}$

- Domain

```
;; TSP PDDL ***Domain File***

(define (domain tsp)
  (:requirements :typing)
  (:types node)

  ;; Define the facts in the problem
  ;; "?" denotes a variable, "-" a type
  (:predicates
    (at ?pos - node)
    (connected ?start ?end - node)
    (visited ?end - node)
  )

  ;; Define the action(s)
  (:action move
    :parameters (?start ?end - node)
    :precondition (and
      (at ?start)
      (connected ?start ?end)
    )
    :effect (and
      (at ?end)
      (visited ?end)
      (not (at ?start))
    )
  )
)
```

- Problem

```
;; TSP PDDL ***Problem File***

(define (problem tsp-01)
  (:domain tsp)
  (:objects Sydney Adelaide Brisbane Perth Darwin - node)

  ;; Define the initial situation
  (:init (connected Sydney Brisbane)
         (connected Brisbane Sydney)
         (connected Adelaide Sydney)
         (connected Sydney Adelaide)
         (connected Adelaide Perth)
         (connected Perth Adelaide)
         (connected Adelaide Darwin)
         (connected Darwin Adelaide)
         (at Sydney)
  )
  (:goal
    (and
      (at Sydney)
      (visited Sydney)
      (visited Adelaide)
      (visited Brisbane)
      (visited Perth)
      (visited Darwin)
    )
  )
)
```