An Introduction to Numeric Planning

Representation and Search Algorithms

Alfonso Emilio Gerevini and Enrico Scala

Department of Information Engineering
University of Brescia, Brescia, Italy

UNIVERSITY
OF BRESCIA

ICAPS 2020 - Summer School

Goals and Assumptions of this Lecture

Introduction to the field of Numeric AI Planning in which we

- Define the problem, present the main representation language, main planning algorithms and systems
- Focus on the state-based model for (numeric) planning with
  - discrete time and "instantaneous" actions
  - full observability of states
  - deterministic action effects
  - no state changes other than those caused by the plan

Outline

Basic State-based Model (Classical Planning)

Plan generation as state-transition problem:

- a finite and discrete state space S (potentially huge!)
- a known initial state $s_0 \in S$
- a set $S_G \subseteq S$ of goal states
- a finite set of actions A
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic state transition function $s' = \gamma(a, s)$ for $a \in A(s)$
- action costs $c(a, s) \geq 0$

A **solution** is a sequence of applicable actions that maps s_0 into S_G

It is **optimal** if minimizes the sum of action costs

Related problems: plan existence, plan validation, plan revision, goal recognition, interleaved planning/execution

Propositional Planning Language

In a **propositional planning language**

- states represented by **propositions** (atoms, Boolean vars, facts)
- γ and A represented by **action preconditions/effects** over atoms

A problem in the STRIPS language is a tuple $\langle F, A, I, G \rangle$:

- F = set of all atoms
- A = set of all actions
- $I \subseteq F$ is an initial state
- $G \subseteq F$ is a set of goals

Note: all other states are *implicit*!

Each $a \in A$ represented by

- Positive effects $Add(a) \subseteq F$ and negative effects $Del(a) \subseteq F$
- Preconditions $Pre(a) \subseteq F$

Many extensions: conditional effects, non-atomic preconditions, domain axioms, state trajectory constraints, etc.

Why **Numeric** Planning?

Numbers are very useful for modelling problems involving, e.g.,

- *Available Resources* (fuel, energy, money, ...)
- *Physical quantities* (temperature, velocity, pressure, density, ...)
- *Spatial information* (cartesian coordinates, GPS position, size, ...)

$\Longrightarrow$ We add numbers in the planning model and in the language:

- Actions with **numeric preconditions** and **numeric effects**
  fuel-level $>$ *100 before moving and reduced by 88.5 after moving*
- **Numeric goals**
  target location has been visited and **fuel-level** $>$ **10**

Basic State-based Model for Numeric Planning

States are specified by variables whose values are (rational) **numbers**

- an **infinite** space of **numeric states** S
- a known initial **numeric** state $s_0 \in S$
- a **possibly infinite** set $S_G \subseteq S$ of goal states
- A finite set of actions A
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a deterministic state transition function $s' = \gamma(a, s)$ for $a \in A(s)$
- action costs $c(a, s) \geq 0$

Note: γ has a potentially **infinite number of state transitions**

$\Longrightarrow$ *How do we represent states, γ, and action preconditions/effects?*

Languages for Numeric Planning

PDDL (Planning Domain Definition Language) is the mostly used standard language for planning

Incrementally developed

- PDDL1 [McDermott and others, 1998]
- PDDL1.2 [Bacchus, 2000]
- PDDL2.1 [Fox and Long, 2003]
- PDDL2.2 [Hoffmann and Edelkamp, 2004]
- PDDL+ [Fox and Long, 2006]
- PDDL3.0 [Gerevini and Long, 2006]
- PDDL3.1 [Helmert et al., 2008]

PDDL2.1 introduced numeric planning and temporal planning in PDDL

PDDL+ and PDDL3: numeric planning problems with *events*, *processes*, *soft goals*, *metric state-trajectory constraints*

Numeric Planning Supported in PDDL2.1

A planning problem is a tuple $\langle F, X, A, s_0, G \rangle$ where

- F is a finite set of **Boolean variables** (propositions/atoms/facts)
- X is a set **numeric variables** over $\mathbb{Q}$ (**numeric fluents**)
- A is a set of **actions** (with preconditions and effects)
- s_0 is the **initial problem state** (an assignment to all vars in $F \cup X$)
- G are the **problem goals** (subset of atoms in F plus numeric conditions over the vars in X)

$\Longrightarrow$ *Which class of action models (preconditions and effects) and goals can be represented in PDDL2.1?*

Action Precondions/Effects & Goals (PDDL2.1 Models)

Preconditions ($Pre(a)$):

- Truth conditions on Boolean vars
- Conditions on numeric vars of the form "ξ *Rel* k"
  $k \in \mathbb{Q}$, ξ is *arithmetic expression*, *Rel* $\in \{\leq, <, =, >, \geq\}$

Effects ($Eff(a)$):

- Truth changes to Boolean vars
- Changes to numeric vars of the form *increase*, *decrease*, *assign*, *scale-up/down* by an *arithmetic expression*

$\Rightarrow$ **Should not conflict**! (e.g. *decrease* and *scale up* the same var)

Problem goals:

- Propositional goals as in propositional planning
- Numeric goals (same as the numeric action conditions)

State Dependency of Numeric Action Effects

Note that

In classical planning (STRIPS) action **effects are state-independent**:

every action has predefined effects that are the same for every state where the action can be applied

But **this does not hold for numeric effects**

E.g.: *(decrease fuel 10)* assigns 40 to *fuel* if it was 50, 0 if it was 10

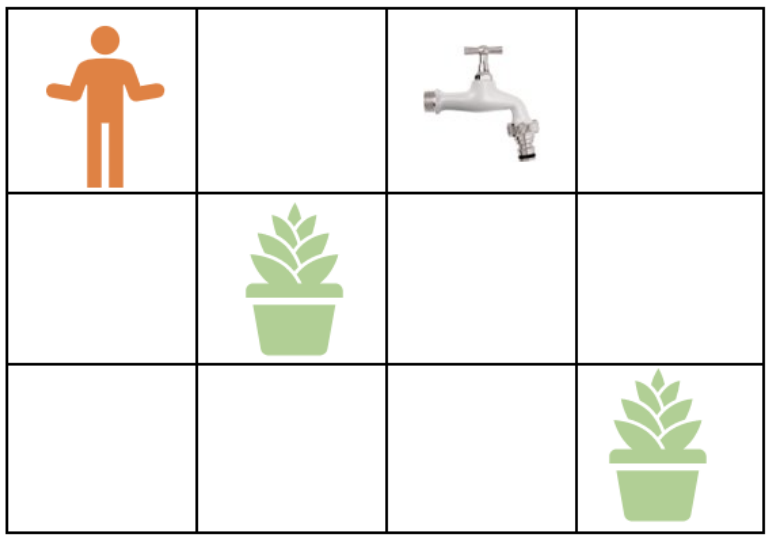
$\Longrightarrow$ The "state dependency" of the action effects can make the design of effective algorithms for numeric planning more difficult

(see heuristic search techniques in the 2nd part of this lecture)

Example in PDDL Syntax (plant watering)

Task: *one or more agents on a grid-like map have to water some plans at different locations using water loaded and carried from sources (taps) available at different locations*

- water tank of an agent has limited capacity
- water loaded in the tank by smaller container (multiple load actions)
- each plant needs its amount of water
- each move (up, down, left, right) and water load/unload has a **cost**



Example in PDDL (objects, predicates and functions)

PDDL supports the definition of

- *Problem objects with types*

  ```
  (:objects agent1 agent2 ...  - agent)
  ```

- *Predicates*

  ```
  (:predicates (open ?t - wathersource))
  ```

  Grounded predicate `(open tap1)` is a Boolean var/proposition

- *Functions* (numerical fluents/numeric vars)

  ```
  (:functions (x-pos ?a - agent)
              (y-pos ?a - agent))
  ```

  `(x-pos agent1)` and `(y-pos agent2)` are **numeric fluents (vars)**

- *Action models with parameters* (next slide)

Example of Action Schemata in PDDL2.1 Syntax

Problem action = PDDL action with instantiated parameters

```
(:action load-water
      :parameters (?a - agent ?t - watersource)
      :precondition (and (open ?t)
                         (= (x-pos ?a) (x ?t))
                         (= (y-pos ?a) (y ?t))
                         (<= (+ (carrying) 1) (max_load)) )
      :effect (increase (carrying) 1)
              (increase (total_loaded) 1) ) )
```

If `?a = agent1` and `?t = watercourse`, the involved numeric vars are:

`(x-pos agent1)` `(y-pos agent1)` `(carrying)` `(max_load)` `(total_loaded)`

Numeric Plans (PDDL2.1 Models)

Applicable action

- An action a is applicable in a state s when $s \models Pre(a)$
- An action a applied in s generates the state s' where the variable values of s are changed according to $Eff(a)$

Applicable action sequence

A sequence of actions $\langle a_0, ..., a_{n-1} \rangle$ is a **applicable in state** s_0 when

- a_0 is applicable in s_0
- each action a_i is applicable in the state s_i generated by the application of a_{i-1} in s_{i-1}

Valid plan (solution) is *a sequence of actions* $\langle a_0, ..., a_{n-1} \rangle$ *that, when applied in* s_0*, generates a sequence of states* $\langle s_1, ..., s_n \rangle$ *such that* $s_n \models G$*.*

Non-Linear Plans in PDDL2.1 (informally)

A valid plan may also be a **sequence of *sets* of actions** $\langle A_0, ..., A_{m-1} \rangle$ where **each pair of actions in A_i does not interfere** $(i = 0...m-1)$

Interference of a with b (informally)

- *An effect in $Eff(a)$ modifies the value of a var involved in $Pre(b)$*
  - `(decrease fuel 10)` $\in Eff(a)$ and `(fuel > 100)` $\in Pre(b)$
- *The effects in $Eff(a)$ do not commute with the effects in $Eff(b)$*
  - `(assign x 0)` $\in Eff(a)$ and `(assign y x)` $\in Eff(b)$

Plan Metrics for Numeric Planning in PDDL2.1

PDDL plans have a **plan quality** defined through numeric variables

- Action costs can be defined in terms of a numeric effect increasing var `total-cost`
- Quality of plans can be defined in terms of
  - minimizing `total-cost` or
  - minimizing a numeric (arithmetic) expression
- Example using simplified PDDL syntax:

  ```
  (:metric minimize (+ moving_cost (* 2 total_loaded)))
  ```

  Actions can only increase `moving_cost` and `total_loaded`

Complexity of Numeric Planning

Numeric planning with PDDL2.1 is **semi-decidable** [Helmert 2002] (harder than classical planning)

- In numeric planning plans can have no bound in their length
- In principle we can find a plan by enumerating all sequences of actions with increasing length
- But enumeration may not terminate if a plan does not exists (semi-decidability)

Decidable for some other fragments of PDDL2.1 ☺ [see Helmert 2002]

Fragments of PDDL2.1 for Numeric Planning

Researchers have studied numeric planning with one or more of the following restrictions to PDDL2.1:

- The numeric expression ξ in a precondition $\xi \; \mathit{Rel} \; k$ is **linear**
- *assign* effects **excluded**
- *scale-up*/*down* effects **excluded**
- *increase*/*decrease* effects modify the vars by **constants** instead of generic arithmetic expressions- E.g. `(increase x 5)`

A **Simple Numeric Problem** has all these but still *semi-decidable* ☹

(see how to solve them in the 2nd part of the lecture)

Some Limits and Extensions of PDDL2.1

☹ A numeric var may be **undefined** in a state (no initial value)

With x is undefined and $y = 5$, are these true/false/undefined?

`(x > 0)` `(not (x > 0))` `(or (x > 0) (y < 10))`

☹ Only arithmetic expressions: no *min*, *max*, *log*, *exp*, *sin*, *cos*, etc.

☹ Conflicts of numeric effects of type scale-up/down could be removed

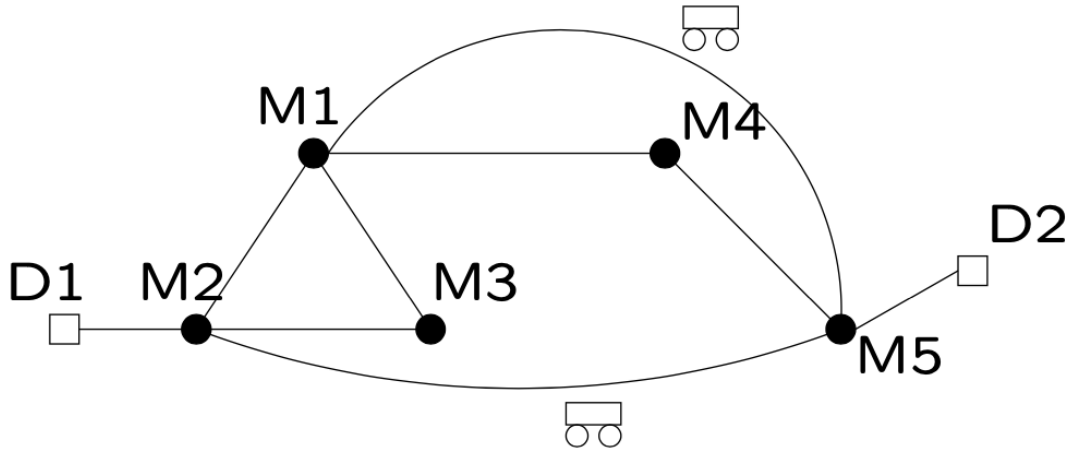
☹ Numeric parameters for an action are forbidden (finite action set)

☺ Nevertheless PDDL2.1 is a powerful language and researchers have addressed some of these limits [e.g., Savas-et-al 2016, Scala-et-al, 2016]

☺ Numeric planning can be combined with *temporal planning* (PDDL2.1/2.2) and enriched with: *state trajectory constraints* (PDDL3), *exogenous events* and *continuous processes* (PDDL+)

$\Longrightarrow$ *The full PDDL language is very powerful!*

PDDL Example: Travelling Purchaser Problem (TPP)



Given (1) a set of types of goods (2) a set of markets (**M**) selling different types and amounts of goods at different prices, (3) a demand of each type of goods to be purchased and transported to some depot (**D**),

$\Longrightarrow$ *satisfy the demand minimizing the routing cost of the trucks and the purchasing cost*

6 different PDDL versions (see IPC5) with simplifications and extensions

PDDL Example: Travelling Purchaser Problem (TPP)

```
(:predicates (at ?t - truck ?p - place))

(:functions (on-sale ?g - goods ?m - market)
            (drive-cost ?p1 ?p2 - place)
            (price ?g - goods ?m - market)
            (bought ?g - goods)
            (request ?g - goods)
            (total-cost))

(:objects
            market1 market2 ... - market
            depot1 depot2 ... - depot
            truck1 track2 ... - truck
            goods1 goods2 ... - goods)
```

PDDL Example: Travelling Purchaser Problem (TPP)

```
(:init (= (price goods1 market1) 15)
       (= (price goods1 market2) 8)
       ....
       (= (on-sale goods1 market1) 4)
       (= (on-sale goods1 market2) 9)
       ....
       (at truck1 depot0)
       (at truck2 depot1)
       ....
       (= (drive-cost depot0 market1) 381.20)
       (= (drive-cost market1 market2) 1033.70)
       ....
       (= (bought goods1) 0)
       (= (bought goods1) 0)
       ....
       (= (request goods1) 20)
       (= (request goods2) 23)
       ....
       (= (total-cost) 0))

(:goal (and (>= (bought goods1) (request goods1) ....)))

(:metric minimize (total-cost))
```

PDDL Example: Travelling Purchaser Problem (TPP)

```
(:action drive
          :parameters (?t - truck ?from ?to - place)
          :precondition (and (at ?t ?from))
          :effect (and (not (at ?t ?from)) (at ?t ?to)
                   (increase (total-cost) (drive-cost ?from ?to))))

(:action buy-all
   :parameters (?t - truck ?g - goods ?m - market)
   :precondition (and (at ?t ?m) (> (on-sale ?g ?m) 0)
                      (<= (on-sale ?g ?m) (- (request ?g) (bought ?g))))
    :effect (and ((assign (on-sale ?g ?m) 0)
              (increase (total-cost) (* (on-sale ?g ?m) (price ?g ?m)))
              (increase (bought ?g) (on-sale ?g ?m))))

(:action buy-allneeded
   :parameters (?t - truck ?g - goods ?m - market)
   :precondition (and (at ?t ?m) (> (on-sale ?g ?m) 0)
                      (> (on-sale ?g ?m) (- (request ?g) (bought ?g))))
   :effect (and (decrease (on-sale ?g ?m) (- (request ?g) (bought ?g)))
                (increase (total-cost) (* (- (request ?g) (bought ?g))
                                          (price ?g ?m)))
                (assign (bought ?g) (request ?g))))
```

More on Numeric Planning in PDDL?

- Many benchmarks from the International Planning Competitions
  https://www.icaps-conference.org/competitions/
- PDDL Wikipedia page:
  https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language
- Paper on PDDL 2.1 [Fox and Long, JAIR 2003]:
  https://www.jair.org/index.php/jair/article/view/10352
- Paper on PDDL3.0 [Gerevini et al., AIJ 2009]:
  https://www.sciencedirect.com/science/article/pii/S0004370208001847
- Complexity of PDDL Numeric Planning [Helmert, AIPS 2002]
- Book on PDDL: "An Introduction to the Planning Domain Definition Language", Haslum et al., 2019, Morgan & Claypool Publishers

Solving Numeric Planning Problems: Approaches

As for propositional planning, several different approaches and algorithms

- **Planning by compilation**: Translates the original problem into an equivalent one solved by existing solvers

  E.g., SAT for propositional planning, SMT for numeric planning

- **Planning as heuristic search**: We use general search algorithms with effective *domain-independent heuristics*

  E.g., Best-first search, A*, local search

  - directional (forward/backward) search in the space of states
    E.g., the Metric-FF planner
  - non-directional search in the space of partial plans
    E.g., the LPG planner

In this lecture we focus on forward state-based search

State Space Search

Informal Definition

State space search is about iterating over paths of a transition system until either

- there is no new path to explore
- a path proved to be a valid solution is found

Forward State Space Search is the variant where we always start from some initial condition

Forward State-Space Search Planning

Forward State Space Search for Planning revolves around a few, simple ideas:

- *Map problem into underlying transition system (TS) and seek trajectory belonging to it*. Two states s_0 and s_1 are connected whenever there is an action applicable in s_0 yielding s_1
- *Focus on reachable sequences only!* Simulate possible futures starting from initial condition
- *Discard Loops!* Focus only on simple plans; each state does not need to be visited more than once
- *Do not construct the whole TS upfront, but incrementally*. Realistic planning instances induce way too large TSs

Best-First-Search in the State Space

Best-First-Search(problem)

```
Q = { Node(problem.init, 0, h(problem.init))}
while Q is not empty do
    n = pop_best_node(Q)
    if n.s is a goal state then
        return extract_plan(n.s)
    for all s' ∈ succ(n.s) do
        insert(Q, Node(s', n.g+1, h(s')))
return problem is not solvable
```

- Q contains all plan-prefixes, each implicitly represented by a node with a pointer to its parent
- BFS operates Q in two ways:
  - pop next node according to some ordering function f
  - push new plan prefixes by appending new nodes obtained through successor function

For simplicity, duplicate checking and node re-opening is not detailed

Best-First-Search in the State Space

Best-First-Search(problem)

```
Q = { Node(problem.init, 0, h(problem.init))}
while Q is not empty do
    n = pop_best_node(Q)
    if n.s is a goal state then
        return extract_plan(n.s)
    for all s' ∈ succ(n.s) do
        insert(Q, Node(s', n.g+1, h(s')))
return problem is not solvable
```

- f determines the particular algorithm
  - UCS –> $f(n) = g(n)$
  - A* –> $f(n) = g(n) + h(n)$
  - WA* –> $f(n) = g(n) + w \times h(n)$
  - GBFS –> $f(n) = h(n)$

For simplicity, duplicate checking and node re-opening is not detailed

Forward State-Space Search for Numeric Planning

Solving numeric planning through forward state space search is straightforward:

1. State representation. Variables that keep track of values of numeric fluents
2. Extend successor function using numeric planning semantics
   - An action is applicable in state s iff its precondition evaluates to true in s. This needs to include reasoning over numeric conditions (e.g., $yx + z > 5$ is satisfied in s)
   - Successor of a state computed using actions' effects semantics
3. Goal test. Ability to evaluate formulas involving numeric terms

Best-First-Search in the State Space

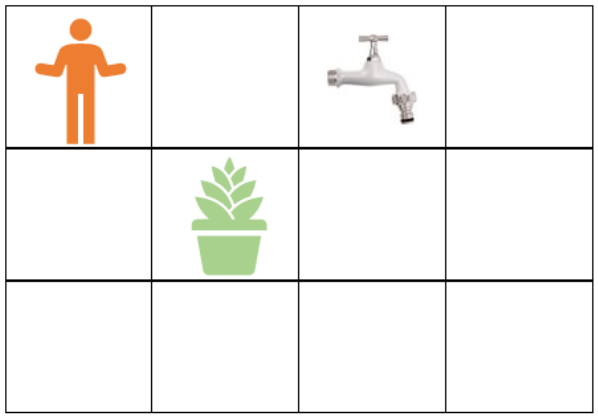
Best-First-Search(problem)

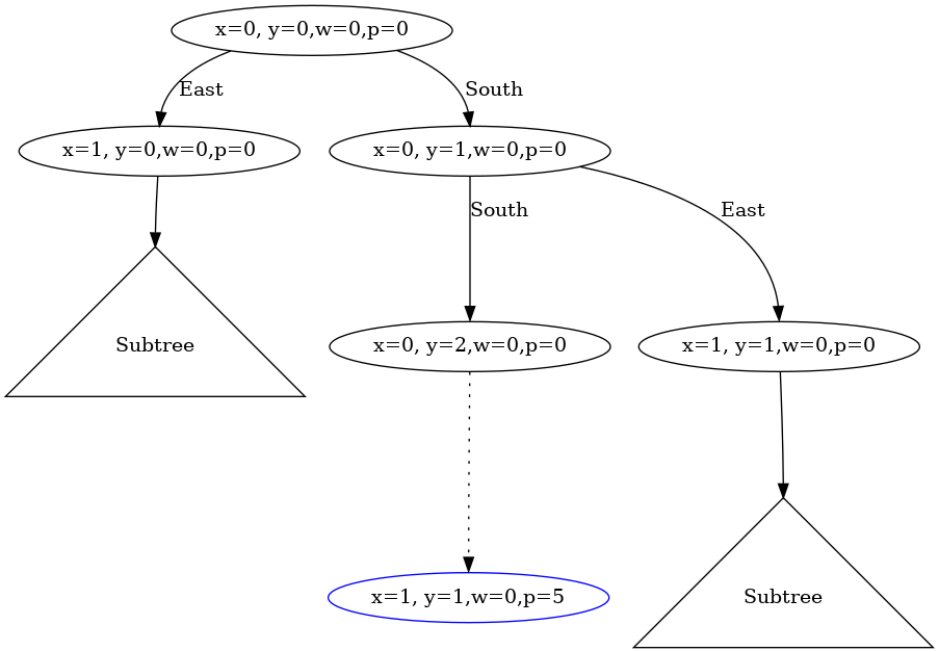
```
Q = { Node(problem.init, 0, h(problem.init))}
while Q is not empty do
    n = pop_best_node(Q)
    if n.s is a goal state then
        return extract_plan(n.s)
    for all s' ∈ succ(n.s) do
        insert(Q, Node(s', n.g+1, h(s')))
return problem is not solvable
```

goal state check, **succ** are the key points that need to be revised in order for getting the schema work for numeric planning problems

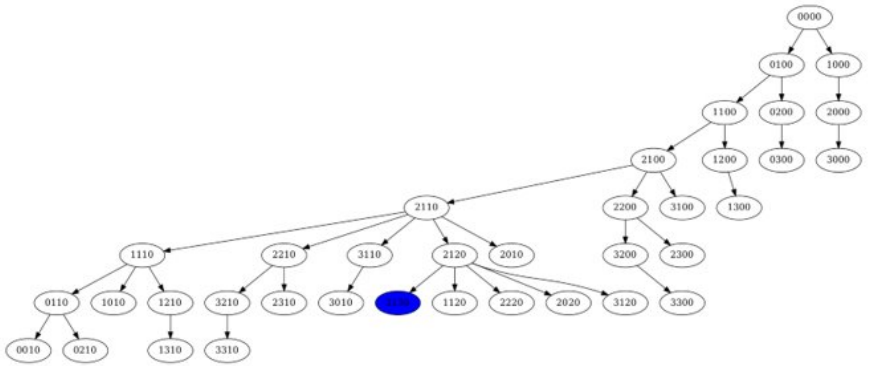
For simplicity, duplicate checking and node re-opening is not detailed

Plant-Watering Example Search Tree





- Actions: East, West, South, North, Load-Water, Pour-Plant
- Goal: Get plant poured



Best-First Search Performance

- Number of iterations (AKA size of the search tree) can be prohibitively large. Note that numeric planning transition system is infinite.
- The function f plays critical role. It is what guides the exploration
- Even though we cannot change the action costs, we can change the way we predict how distant the goal is and use an informed BFS (e.g., A*, GBFS, WA*)
- That is: **We can change the heuristic**

Heuristics

Known facts:

- If heuristic is perfect (true distance to the goal for each state), number of expansions is **linear with the length of the solution**
- The closer the heuristic is to the perfect estimate (called h^*), the less is the number of expanded nodes (Dechter and Pearl 1983)

- Most research look into making heuristic functions as much precise as possible

---

Almost perfect heuristics can still cause exponential blow ups, Helmert and Roger 2008.

Heuristics

Definition (Heuristic)

Heuristic is a function $h : S \to Q \cup \{\infty\}$

Definition (Admissibility)

Heuristic h is said to be admissible if $\forall s \in S.h(s) \leq h^*(s)$; A* with admissible heuristic finds optimal solutions.

Definition (Safe-Pruning)

Heuristic h is said to be safe-pruning if $\forall s \in S.h(s) = \infty \Rightarrow$ there is no solution from s

Definition (Tractability)

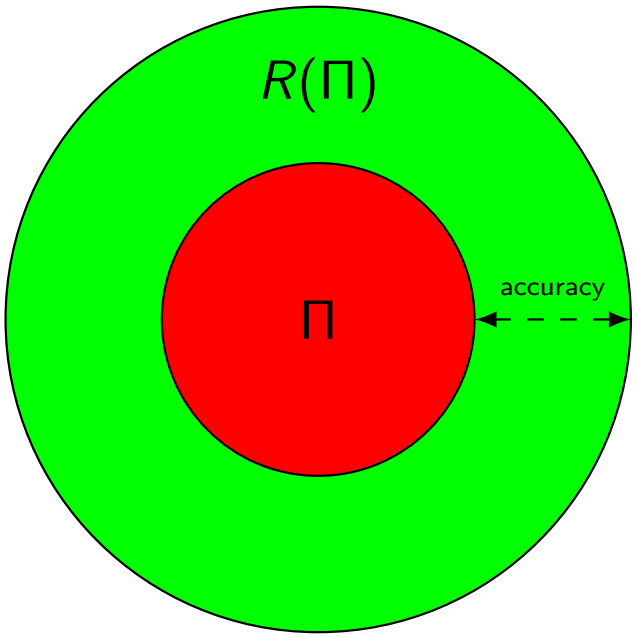
Heuristic is tractable if computable in poly-time (w.r.t. problem size). Heuristic should be easy than the problem they are approximating

How to Define Heuristics for Numeric Planning Problems

- The user defines it. Can work well, but how general can it ever be?
- Construct an algorithm that can look at the problem and tell us what the value of the heuristic is. Automatically!
- Heuristics of this kind are often called Domain Independent Heuristics

We will show domain independent heuristics for numeric planning, by investigating their properties. Admissibility, Safe-Pruning, Tractability

Relaxation-Based Heuristics. The Intuition



- **Devise a new problem that relaxes the problem** you are interested to solve
- Make sure the arising relaxed problem is easier, yet **contains enough information**
- Solve the relaxed problem, and use the solution to guide the solution of the new problem. **This is what gives you your heuristic**

Bonet and Geffner 2001, Hoffmann and Nebel 2001

Relaxation-based Approach in Numeric Planning

- Interval-based Relaxation (Hoffmann 2003, Aldinger et al. 2015)
  - **Decompose/ignore interactions among the numeric variables** of your problem
  - Characterized by starting from initial state up to a point in which the goal is relaxed satisfied
- Subgoaling-based Relaxation (Bonet and Geffner 2001, Scala et al. 2016)
  - **Decompose/ignore interactions among the numeric subgoals** of your problem
  - Characterized by starting regressively from the goal condition up to the point in which all action supporters have been found

Both frameworks can be used to devise a variety of heuristics

Interval-based Relaxation (Intuitively)

- Map numeric state variables into **intervals** (many states at once)
  - Ex. $s : \{x = 0, y = 0\} \rightarrow s^+ : \{x = [0, 0], y = [0, 0]\}$
- **Change semantics of effects** through **convex union** (intervals only grow)
  - Ex. $s^+[x+ = 1] = \{x = [0, 1], y = [0, 0]\}$ `Just update the max`
  - Ex. $s^+[x = 5] = \{x = [0, 5], y = [0, 0]\}$ `Intervals don't have holes!`
- **Change semantics of formulas** so as to over-approximate satisfiability
  - Ex. $x = 10$ is satisfied in $[0, 10]$
  - Ex. $x + y = 10$?

Interval Analysis

Each operation is computed so as to enclose all values that can ever be attained

Arithmetical Operations

- $x + y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$;
- $x - y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$;
- $x \times y = [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})]$;
- $x \div y = [min(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y}), max(\underline{x} \div \underline{y}, \underline{x} \div \overline{y}, \overline{x} \div \underline{y}, \overline{x} \div \overline{y})]$ (if $0 \notin y$ otherwise one of the bounds diverges).

Note

Interval analysis is a very powerful framework, Many other mathematical operations can be defined. More dtails in Moore et al. 2009, and their application into planning in Aldinger et al. 2015, Scala et al. 2016.

Examples

Example 1

Condition $\psi : x + y = 0$; Interval State $s : \{x = [0, 10], y = [-15, -15]\}$ Is condition satisfied? No, indeed $[0, 10] + [-15, -15] = [-15, -5]$

Example 2

Condition $\psi : x + y = 0$; Interval State $s : \{x = [0, 10], y = [-15, -5]\}$ Is condition satisfied? Yes, indeed $[0, 10] + [-15, -5] = [-15, 5]$ and it is possible to pick 0 within $[-15, 5]$ so as to satisfy the condition

Interval-based Relaxation. The problem

Interval-based Relaxation

Given a planning problem $\Pi = \langle s_0, A, G, X \rangle$, $\Pi^+ = \langle s_0^+, A^+, G^+, X^+ \rangle$ is its Interval-based relaxation where

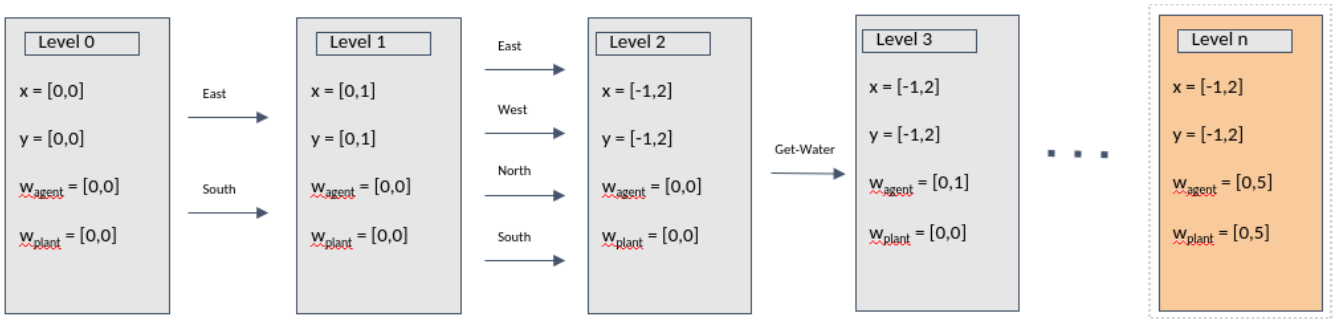
- s_0^+ is represented as a set of intervals.
- A^+ syntactical equivalent to A, but different semantics:
  1. Successor state via convex union
  2. Preconditions only relaxed satisfied ($\exists v \in [\underline{x}, \overline{x}] : v \models pre(a)$)
- G^+ is syntactical equivalent to G. Semantics as for action precondition
- $X^+ \equiv X$. Variables don't change, but expressions follow Interval Algebra

How to solve it?

Two step process:

1. Reachability analysis of the goal (forward analysis)
2. Extraction of a relaxed plan solution (backward analysis)

Reachability through Example



The process

- Determine action applicability using intervals
- Apply actions with convex union up to the point where goal is satisfied
- Store actions for each encountered level

Observations

1. Conditions satisfied at level i will also be satisfied in any level $j > i$
2. Applicable actions never decrease

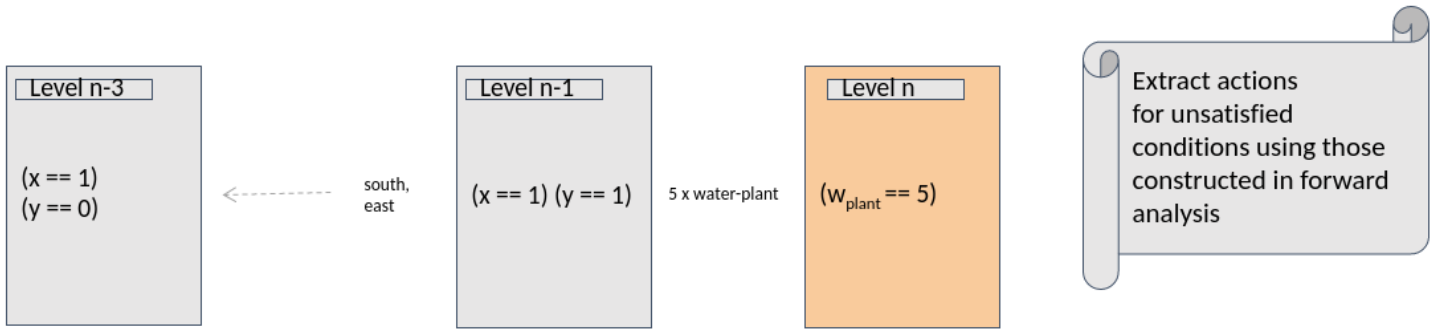
All good when the problem is solvable **What about when the problem isn't solvable?**

Termination Criteria

- Metric-ff **mneed criteria** (Hoffman 2003) (linear expressions, acyclic assignments)
  - Compute max sufficient value for each variable. Stop when fix point is reached, or all intervals contain mneed

- **Asymptotic reachability** (Aldinger et al. 2015, Scala et al. 2016) (linear, non-linear, state dependent effects that can be acyclic, too)
  - make hidden state dependency explicit in preconditions of supporters (constructed from actions). Substitute increase and decrease with infinity assignments

The Metric-FF heuristic



- Relaxed Plan is $\pi^+ = \{5 \times \text{water-plant}, \text{load-water}, 2 \times \text{east}, \text{south}\}$
- This plan solves Π^+...does it solve Π, too?
- No, but its cost can be used to guide the search:

$$h^{MFF} = \sum_{a \in \pi^+} a$$

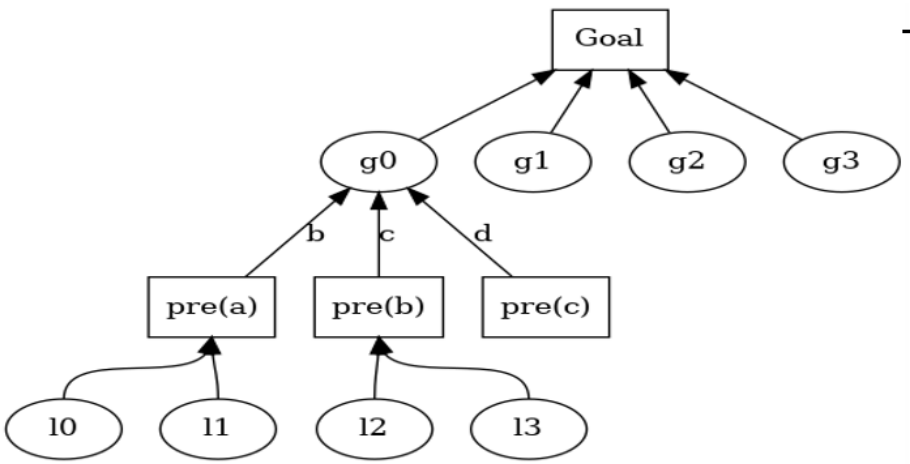
Interval-based Relaxation Properties

- Computable?
  - Yes! it eventually terminates because of the **mneed criteria**
- Tractable?
  - Yes! Π^+ can be decided in polytime
- Safe-pruning?
  - Yes! The existence of a plan for Π always implies that there is a relaxed plan for Π^+
- **Admissible?**
  - No!

Subgoaling-based Relaxation

Intuition

Backward from the goal and recursively over subgoals that need to be achieved. Focus on subgoals independently



Two interpretations

- Pessimistic: sum cost of subgoals
- Optimistic: maximize cost of subgoals

Regression in Planning

Intuitively

An operation that combines some formula ψ with an action a to obtain a new formula ψ' which establishes sufficient and necessary conditions for reaching ψ through a

Formally

Regression is a function $regr : \Gamma \times A \rightarrow \Gamma$ for which:

$$\forall \psi \in \Gamma . \forall a \in . \forall s \in S . s \models regr(\psi, a) \iff s[a] \models \psi$$

- Regression is at the basis of subgoaling-based relaxation.
- In classical planning, if ψ is an atom and action a adds ψ, $regr(\psi, a) = \text{pre}(a)$ - **This is fundamental for making subgoaling so effective..**

Subgoaling in Classical Planning

$$h^{max}(s,\psi) \doteq \begin{cases} 0 & \text{if } s \models \psi \\ \min\limits_{a \in ach(s,\psi)} \left(h^{max}(s, \text{pre}(a)) + c(a)\right) & \text{if } \psi \text{ is a PC} \\ \max\limits_{\psi' \in \psi} \left(h^{max}(s,\psi')\right) & \text{if } \psi \text{ is } \wedge \end{cases}$$

- $ach(s,\psi)$ denotes the set of achievers of a proposition
- Decomposition comes from ignoring interactions among conjucts
- Pillar in optimal planning (Helmert and Domshlak 2006)

Properties

h^{max} is:

- Admissible
- Safe-pruning
- Tractable

Question

How to extend it to consider numeric variables, but in particular numeric conditions?

Simple Numeric Planning

Definition (Simple Numeric Condition)

A numeric condition is said to be simple if it is linear and all actions interacting with it are either increase or decrease by a constant

Definition (Simple Numeric Planning)

A numeric planning problem is said to be simple if it only contains simple numeric conditions.

Regression in Simple Numeric Planning

Intuition

For each numeric formula, produce a new formula where each variable is substituted with the way such variable is modified by the action

Example

- $\psi : x + y \geq 10$, an action a where $\text{eff}(a) : x+ = 1$, and an action b where $\text{eff}(b) : x- = 1$
- $regr(\psi, a) = (x + 1) + y \geq 10$
- $regr(\psi, b) = (x - 1) + y \geq 10$

What can we say?

- Action a is a *possible achiever* for ψ. Also *it seems you need at least 10 executions of a* if you are in a state where both x and y are set to 0.
- Action b is not a *possible achiever* for ψ.
- **These observations are independent on the state in which the actions are applied**

How do we make this general?

Possible achievers through multi-time regression

Problem. Numeric effects are not idempotent, and we need a way to count the needed amount of actions

Observation: $regr(...regr(\psi, a), a)$ m times is equivalent to operator $\psi^{r(a,m)}$:

$$\psi^{r(a,m)} \equiv m \overbrace{\left(\sum_{x=\mathsf{lhs}(e), e \in \mathsf{eff}_{\mathsf{num}}(a)} w_{\psi,x} k_{a,x} \right)}^{N_{\psi,a}} + \overbrace{\left(\sum_{x \in X} w_{\psi,x} x \right) + k_\psi}^{\xi_\psi} \geq 0$$

This operator gives us:

- way to establish whether an action a is a possible achiever by inspecting the sign of $N_{\psi,a}$.
- easy counting mechanism:

$$\widehat{\mathsf{rep}}(a, \psi, s) = \begin{cases} 0 & \text{if } s \models \psi \\ \frac{-[\xi_\psi]^s}{N_{\psi,a}} & \text{else if } a \in ach(\psi) \\ \infty & \text{otherwise} \end{cases}$$

h^{max}_{hbd} for numeric planning

$$h^{max}_{hbd}(s,\psi) \doteq \begin{cases} 0 & \text{if } s \models \psi \\ \min\limits_{a \in ach(s,\psi)} (h^{max}_{hbd}(s, \text{pre}(a)) + c(a)) & \text{if } \psi \in \text{PC} \\ \min\limits_{a \in ach(s,\psi)} (\widehat{\text{rep}}(a,\psi,s) \cdot c(a) + h^{add}_{hbd}(\text{pre}(a))) & \text{if } \psi \in \text{SC} \\ \max\limits_{\psi' \in \psi} h^{add}_{hbd}(s,\psi') & \text{if } \psi \text{ is } \wedge \end{cases}$$

Note that $\widehat{\text{rep}}(\cdot,\cdot,\cdot)$ induces a continuous relaxation. The integral version of the minimisation problem is NP-Hard (Scala et al. 2020).

Properties

- Safe Pruning?
  - It can be shown by observing that one can always find all necessary achievers if unrelaxed problem is solvable
  - If heuristic yields infinity, the problem from that state is indeed unsolvable
- Tractable?
  - Can pose the problem as a blind search over atoms
  - Complexity polynomial on the number of actions and conditions of the problem
- Admissible?
  - NO!

Counterexample

Example Problem

- One goal $x + y > 9$, initial state is $\{x = 0, y = 0\}$.
- Two actions a and b such that
  - $\text{pre}(a) = (y \geq 10)$ $\text{eff}(a) = \{x+ = 1\}$
  - $\text{pre}(b) = \top$ $\text{eff}(b) = \{(x+ = 0.5), (y+ = 5)\}$

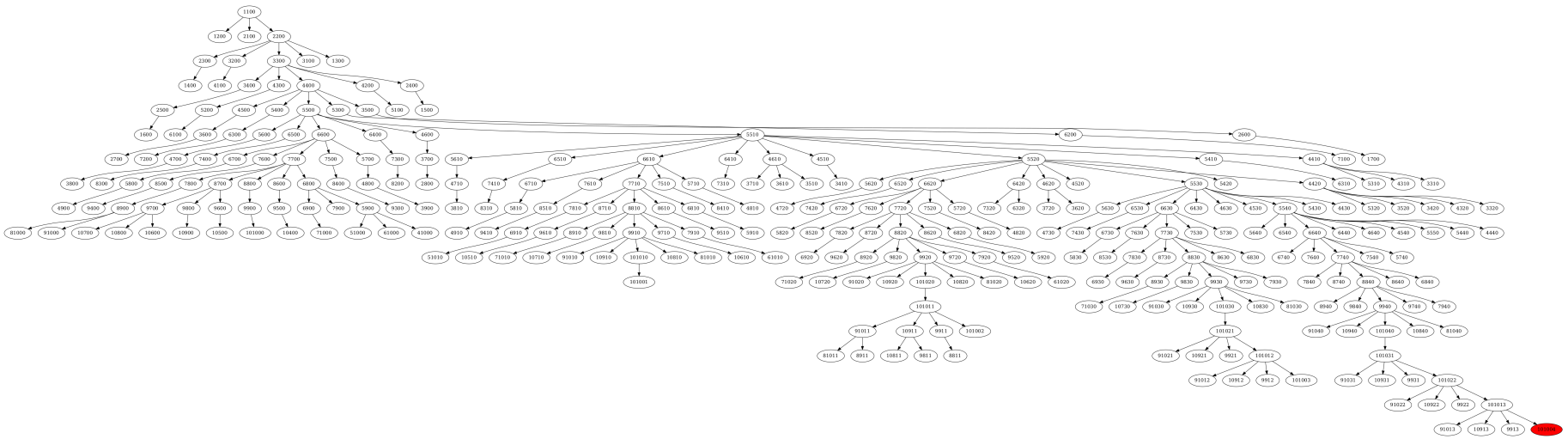
- Cost optimal plan for this problem? 11! I.e., $\langle 2 \times b, 9 \times a \rangle$
- Cost optimal through h^{max}_{hbd} is 12!
  - Obtained by choosing a as a best achiever, and recursively b for its precondition. Decomposition does not look into the positive interactions among the actions.
  - This is not a pathological case. Happens in real instances.

Make it admissible

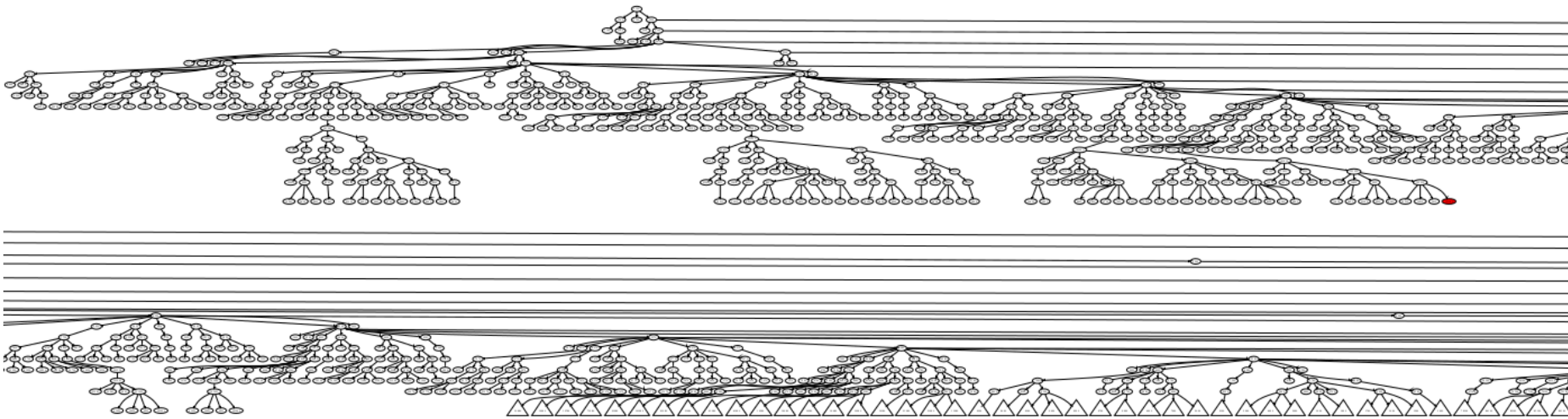
$$h^{max}_{hbd}(s,\psi) \doteq \begin{cases} 0 & \text{if } s \models \psi \\ \min\limits_{a \in ach(s,\psi)} \left(h^{max}_{hbd}(s, \text{pre}(a)) + c(a)\right) & \text{if } \psi \in \text{PC} \\ \cancel{\min\limits_{a \in ach(s,\psi)} \left(\widehat{\text{rep}}(a,\psi,s) \cdot c(a) + h^{add}_{hbd}(\text{pre}(a))\right)} & \text{if } \psi \in \text{SC} \\ \max\limits_{\psi' \in \psi} h^{add}_{hbd}(s,\psi') & \text{if } \psi \text{ is } \wedge \end{cases}$$

$$\left(\min_{a \in A^+ \cap ach(s,\psi)} \widehat{\text{rep}}(a,\psi,s) \cdot c(a)\right) + \left(\min_{a \in A^+ \cap ach(s,\psi)} \hat{h}^{max}_{hbd}(s, \text{pre}(a))\right)$$

Search Tree with h^{max}_{hbd} over a 10×10 Grid



Search Tree with no heuristic over a 10×10 Grid



Inadmissible Variant

$$h^{add}_{hbd}(s,\psi) \doteq \begin{cases} 0 & \text{if } s \models \psi \\ \min\limits_{a \in ach(s,\psi)} \left(h^{add}_{hbd}(s, \text{pre}(a)) + c(a)\right) & \text{if } \psi \in \text{PC} \\ \mathbf{\min\limits_{a \in ach(s,\psi)} (\widehat{rep}(a,\psi,s) \cdot \gamma(a) + h^{add}_{hbd}(pre(a)))} & \text{if } \psi \in \text{SC} \\ \sum\limits_{\psi' \in \psi} h^{add}_{hbd}(s,\psi') & \text{if } \psi \text{ is } \wedge \end{cases}$$

- Pretend all subproblems don't interact. That is, sum the cost of each subproblem
- Can work much (MUCH) better than admissible variant in practice.
- Lots of research tries to understand when things can be summed up or not. Underlying problem is NP-Hard.

Subgoaling or Intervals

- It depends on the problem!
- Subgoaling enables optimal planning. There are at the moment no heuristics based on intervals that do so (Piacentini et al. 2018 is a slightly different exception). Interval-based relaxation with cost limited, but something exists (Aldinger et al. 2015)
- Subgoaling is difficult to be extended with more general expressions, action effects.

Recap - Take home message

- Want to build a numeric planner?
  - Can use state-space search onto which you can use best-first search schema pretty much off the shelf.
- Heuristic reasoning is crucial. For this, two different schemata are known: based on Subgoals decomposition or Variables decomposition.
- Both have strengths and weaknesses. Be careful to understand your problem better first. Simplify if complexity is not necessary!
- Plenty of problems in combining the two relaxations (Piacentini et al 2018), and extend to more powerful constructs (e.g., non-linear problems, processes...)

Work not covered here

- LPRPG heuristic. Builds on intervals, but reason over resource-flows patterns much more effectively than what we have seen here (Coles et al. 2008)
- LPG planner uses different variants of relaxed plan heuristics that are aimed at capturing some action interference (Gerevini et al. 2008). The SAPA planner does something similar (Do and Kambhabati 2003)
- Using Linear programs to devise optimal estimates (Piacentini et al. 2018)
- Better integration of the heuristic in the search through helpful actions (Hoffmann 2003)
- Linear programs also in variants of h^{max} with the aim of capturing conflicts among subgoals (Scala et al. 2020)

Non-exhaustive List of Available Numeric Planners

- Optic, Colin (Kings College). PDDL 2.1, Fragments of PDDL+. No discretisation, Heuristic search via IBR. C++
- Metric-FF (Joerg Hoffmann). PDDL 2.1. Heuristic from IBR. Restricted to linear and acyclic tasks, C
- LPG (University of Brescia). PDDL 2.1, monotone in each variable separately. Local Search with IBR heuristic. C
- ENHSP (Enrico Scala, ANU, University of Brescia). PDDL2.1, PDDL+, optimality guarantees, highly customizable. Heuristic search via IBR, Subgoaling. JAVA
- SMTPLAN (King's College). Fragment of PDDL+. Depends on SMT engine. Frotend in Python
- TFD (G. Roger, P. Eyerich, C. Dornhege, R. Matmuller) PDDL2.1, heuristic search through variant of h^{hadd} heuristic, C++

All can be found easily by googling them

Main References for Numeric Planning

Modeling Language and Complexity Analysis

Fox, Maria, and Derek Long. "PDDL2.1: An extension to PDDL for expressing temporal planning domains." *Journal of artificial intelligence research* 20 (2003): 61-124.

Helmert, Malte. "Decidability and Undecidability Results for Planning with Numerical State Variables." In *AIPS*, pp. 44-53. 2002.

Haslum, Patrik, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. "An introduction to the planning domain definition language." *Synthesis Lectures on Artificial Intelligence and Machine Learning* 13, no. 2 (2019): 1-187.

Geffner, Hector, and Blai Bonet. "A concise introduction to models and methods for automated planning." Synthesis Lectures on Artificial Intelligence and Machine Learning 8, no. 1 (2013): 1-141.

Search and Heuristics

Hoffmann, Jörg. "The Metric-FF Planning System: Translating``Ignoring Delete Lists"to Numeric State Variables." *Journal of artificial intelligence research* 20 (2003): 291-341.

Aldinger, Johannes, and Bernhard Nebel. "Interval based relaxation heuristics for numeric planning with action costs." In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pp. 15-28. Springer, Cham, 2017.

Li, Dongxu, Enrico Scala, Patrik Haslum, and Sergiy Bogomolov. "Effect-Abstraction Based Relaxation for Linear Numeric Planning." In *IJCAI*, pp. 4787-4793. 2018.

Scala, Enrico, Patrik Haslum, Daniele Magazzeni, and Sylvie Thiébaux. "Landmarks for Numeric Planning Problems." In *IJCAI*, pp. 4384-4390. 2017.

Scala, Enrico, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. "Subgoaling Techniques for Satisficing and Optimal Numeric Planning." *Journal of Artificial Intelligence Research* 68 (2020): 691-752.

Scala, Enrico, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. "Interval-based relaxation for general numeric planning." In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, pp. 655-663. 2016.

Scala, Enrico, Alessandro Saetti, Ivan Serina, and Alfonso E. Gerevini. "Search-Guidance Mechanisms for Numeric Planning Through Subgoaling Relaxation." In Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, pp. 226-234. 2020.

Piacentini, Chiara, Margarita P. Castro, André Augusto Ciré, and J. Christopher Beck. "Linear and Integer Programming-Based Heuristics for Cost-Optimal Numeric Planning." In *AAAI*, pp. 6254-6261. 2018.

Piacentini, Chiara, Maria Fox, and Derek Long. "Planning with numeric timed initial fluents." In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 4196-4197. 2015.

Coles, Amanda, M. Fox, and D. Long. "A hybrid LP-RPG heuristic for modelling numeric resource flows in planning." *Journal of Artificial Intelligence Research* 46 (2013): 343-412.

Coles, Amanda, Andrew Coles, Maria Fox, and Derek Long. "Forward-chaining partial-order planning." In *Proceedings of the Twentieth International Conference on International Conference on Automated Planning and Scheduling*, pp. 42-49. 2010.

Gerevini, Alfonso E., Alessandro Saetti, and Ivan Serina. "An approach to efficient planning with numerical fluents and multi-criteria plan quality." *Artificial Intelligence* 172, no. 8-9 (2008): 899-944.

Do, Minh, and Subbarao Kambhampati. "Sapa: A multi-objective metric temporal planner." Journal of Artificial Intelligence Research 20 (2003): 155-194.

Mainly Used Available Planners (alphabetical order)

ENHSP - https://sites.google.com/view/enhsp/ E. Scala (see website for other contributors to the platform)
LPG - https://lpg.unibs.it/lpg/ A.E. Gerevini, A. Saetti, I. Serina
LPRPG - https://nms.kcl.ac.uk/planning/software/lprpg.html Amanda Coles, Andrew Coles, M. Fox, D. Long
Metric-FF - https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html J. Hoffmann
Optic - https://nms.kcl.ac.uk/planning/software/optic.html Amanda Coles, Andrew Coles, M. Fox, D. Long
SMTPLAN - https://github.com/KCL-Planning/SMTPlan M. Cashmore, D. Magazzeni, M. Fox, D. Long
TFD - http://gki.informatik.uni-freiburg.de/tools/tfd/ G. Röger, P. Eyerich, C. Dornhege, and R. Mattmüller.
UPMURPHI - https://github.com/gdellapenna/UPMurphi G. Della Penna, D. Magazzeni, F. Mercorio