Planning and Optimization

B1. Overview of Classical Planning Algorithms

Malte Helmert and Thomas Keller

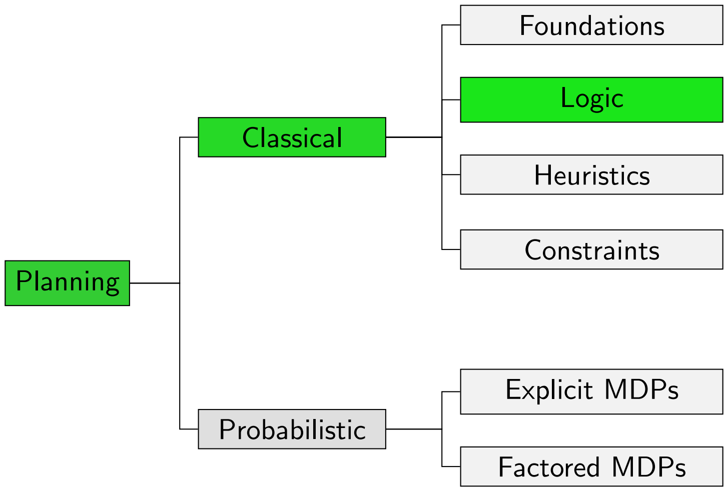
Universität Basel

October 2, 2019

Planning and Optimization

October 2, 2019 — B1. Overview of Classical Planning Algorithms

Content of this Course



B1.1 The Big Three

Classical Planning Algorithms

Let's start solving planning tasks!

This Chapter
very high-level overview of classical planning algorithms

- bird's eye view: no details, just some very brief ideas

The Big Three

Of the many planning approaches, three techniques stand out:

- explicit search ⇝ Chapters B2–B4, Parts C–F
- SAT planning ⇝ Chapters B5–B6
- symbolic search ⇝ Chapters B7–B8

also: many algorithm portfolios

Satisficing or Optimal Planning?

must carefully distinguish:

- satisficing planning: any plan is OK (cheaper ones preferred)
- optimal planning: plans must have minimum cost

solved by similar techniques, but:

- details very different
- almost no overlap between best techniques for satisficing planning and best techniques for optimal planning
- many tasks that are trivial for satisficing planners are impossibly hard for optimal planners

B1.2 Explicit Search

Explicit Search

You know this one already! (Hopefully.)

Reminder: State-Space Search

Need to Catch Up?

- We assume prior knowledge of basic search algorithms:
  - uninformed vs. informed (heuristic)
  - satisficing vs. optimal
  - heuristics and their properties
  - specific algorithms: e.g., breadth-first search, greedy best-first search, A^*
- If you are not familiar with them, we recommend Ch. 5–19 of the Foundations of Artificial Intelligence course: `https://dmi.unibas.ch/en/academics/computer-science/courses-spring-semester-2019/lecture-foundations-of-artificial-intelligence/`

Reminder: Interface for Heuristic Search Algorithms

Abstract Interface Needed for Heuristic Search Algorithms

- init() ⇝ returns initial state
- is_goal(s) ⇝ tests if s is a goal state
- succ(s) ⇝ returns all pairs $\langle a, s' \rangle$ with $s \xrightarrow{a} s'$
- cost(a) ⇝ returns cost of action a
- h(s) ⇝ returns heuristic value for state s

⇝ Foundations of Artificial Intelligence course, Chapters 6 and 13

State Space vs. Search Space

- Planning tasks induce transition systems (a.k.a. state spaces) with an initial state, labeled transitions and goal states.
- State-space search searches state spaces with an initial state, a successor function and goal states.

⇝ looks like an obvious correspondence

- However, in planning as search, the state space being searched can be different from the state space of the planning task.
- When we need to make a distinction, we speak of
  - the state space of the planning task whose states are called world states vs.
  - the search space of the search algorithm whose states are called search states.

Design Choice: Search Direction

How to apply explicit search to planning? ⇝ many design choices!

Design Choice: Search Direction

- progression: forward from initial state to goal
- regression: backward from goal states to initial state
- bidirectional search

⇝ Chapters B2–B4

Design Choice: Search Algorithm

How to apply explicit search to planning? ⇝ many design choices!

Design Choice: Search Algorithm

- uninformed search:
  depth-first, breadth-first, iterative depth-first, ...
- heuristic search (systematic):
  greedy best-first, A^*, weighted A^*, IDA^*, ...
- heuristic search (local):
  hill-climbing, simulated annealing, beam search, ...

Design Choice: Search Control

How to apply explicit search to planning? ⇝ many design choices!

Design Choice: Search Control

- heuristics for informed search algorithms
- pruning techniques: invariants, symmetry elimination, partial-order reduction, helpful actions pruning, ...

How do we find good heuristics in a domain-independent way?

⇝ one of the main focus areas of classical planning research

⇝ Parts C–F

B1.3 SAT Planning

SAT Planning: Basic Idea

- formalize problem of finding plan with a given horizon (length bound) as a propositional satisfiability problem and feed it to a generic SAT solver
- to obtain a (semi-) complete algorithm, try with increasing horizons until a plan is found (= the formula is satisfiable)
- important optimization: allow applying several non-conflicting actions "at the same time" so that a shorter horizon suffices

SAT Encodings: Variables

- given propositional planning task $\langle V, I, O, \gamma \rangle$
- given horizon $T \in \mathbb{N}_0$

Variables of SAT Encoding

- propositional variables v^i for all $v \in V$, $0 \leq i \leq T$ encode state after i steps of the plan
- propositional variables o^i for all $o \in O$, $1 \leq i \leq T$ encode operator(s) applied in i-th step of the plan

Design Choice: SAT Encoding

Again, there are several important design choices.

Design Choice: SAT Encoding

- sequential or parallel
- many ways of modeling planning semantics in logic

$\rightsquigarrow$ main focus of research on SAT planning

Design Choice: SAT Solver

Again, there are several important design choices.

Design Choice: SAT Solver

- out-of-the-box like MiniSAT, Glucose, Lingeling
- planning-specific modifications

Design Choice: Evaluation Strategy

Again, there are several important design choices.

Design Choice: Evaluation Strategy

- always advance horizon by $+1$ or more aggressively
- possibly probe multiple horizons concurrently

B1.4 Symbolic Search

Symbolic Search Planning: Basic Ideas

- search processes sets of states at a time
- operators, goal states, state sets reachable with a given cost etc. represented by binary decision diagrams (BDDs) (or similar data structures)
- hope: exponentially large state sets can be represented as polynomially sized BDDs, which can be efficiently processed
- perform symbolic breadth-first search (or something more sophisticated) on these set representations

Symbolic Breadth-First Progression Search

prototypical algorithm:

Symbolic Breadth-First Progression Search

def bfs-progression(V, I, O, γ):
    $goal\_states := models(\gamma)$
    $reached_0 := \{I\}$
    $i := 0$
    loop:
        if $reached_i \cap goal\_states \neq \emptyset$:
            return solution found
        $reached_{i+1} := reached_i \cup apply(reached_i, O)$
        if $reached_{i+1} = reached_i$:
            return no solution exists
        $i := i + 1$

$\rightsquigarrow$ If we can implement operations *models*, $\{I\}$, $\cap$, $\neq \emptyset$, $\cup$, *apply* and $=$ efficiently, this is a reasonable algorithm.

Design Choice: Symbolic Data Structure

Again, there are several important design choices.

Design Choice: Symbolic Data Structure

- BDDs
- ADDs
- EVMDDs
- SDDs

Other Design Choices

- additionally, same design choices as for explicit search:
  - search direction
  - search algorithm
  - search control (incl. heuristics)
- in practice, hard to make heuristics and other advanced search control efficient for symbolic search
  ⇝ rarely used

B1.5 Planning System Examples

Planning Systems: FF

FF (Hoffmann & Nebel, 2001)

- problem class: satisficing
- algorithm class: explicit search
- search direction: forward search
- search algorithm: enforced hill-climbing
- heuristic: FF heuristic (inadmissible)
- other aspects: helpful action pruning; goal agenda manager

⇝ breakthrough for heuristic search planning;
winner of IPC 2000

Planning Systems: LAMA

LAMA (Richter & Westphal, 2008)

- problem class: satisficing
- algorithm class: explicit search
- search direction: forward search
- search algorithm: restarting Weighted A* (anytime)
- heuristic: FF heuristic and landmark heuristic (inadmissible)
- other aspects: preferred operators; deferred heuristic evaluation; multi-queue search

⇝ still one of the leading satisficing planners;
winner of IPC 2008 and IPC 2011 (satisficing tracks)

Planning Systems: Fast Downward Stone Soup

Fast Downward Stone Soup (Helmert et al., 2011)

- problem class: optimal
- algorithm class: (portfolio of) explicit search
- search direction: forward search
- search algorithm: A^*
- heuristic: LM-cut; merge-and-shrink; landmarks; blind (admissible)

⇝ winner of IPC 2011 (optimal track)

Planning Systems: SymBA*

SymBA* (Torralba, 2015)

- problem class: optimal
- algorithm class: symbolic search
- symbolic data structure: BDDs
- search direction: birectional
- search algorithm: mixture of (symbolic) Dijkstra and A^*
- heuristic: perimeter abstractions/blind

⇝ winner of IPC 2014 (optimal track)

B1.6 Summary

Summary

big three classes of algorithms for classical planning:

- explicit search
  - design choices: search direction, search algorithm, search control (incl. heuristics)
- SAT planning
  - design choices: SAT encoding, SAT solver, evaluation strategy
- symbolic search
  - design choices: symbolic data structure + same ones as for explicit search