Artificial Intelligence Planning

Gholamreza Ghassem-Sani

Sharif University of Technology

Lecture slides are mainly based on

Dana Nau’s AI Planning

University of Maryland
Related Reading

- For technical details:
  - Malik Ghallab, Dan Nau, and Paolo Traverso
    *Automated Planning: Theory and Practice*
    Morgan Kaufmann, May 2004
    ISBN 1-55860-856-7

- Web site:
  - http://www.laas.fr/planning
Outline

- Planning Successful Applications
- Conceptual model for planning
- Example planning algorithms
Planning Successful Applications

- Space Exploration
- Manufacturing
- Games
Space Exploration

- Autonomous planning, scheduling, control
  - NASA
- Remote Agent Experiment (RAX)
  - Deep Space 1
- Mars Exploration Rover (MER)
Manufacturing

- Sheet-metal bending machines - Amada Corporation
  - Software to plan the sequence of bends
    [Gupta and Bourne, *J. Manufacturing Sci. and Engr.*, 1999]
Games

- *Bridge Baron* - Great Game Products
  - 1997 world champion of computer bridge
    [Smith, Nau, and Throop, *AI Magazine*, 1998]
  - 2004: 2nd place
Conceptual Model

1. Environment

\[ \Sigma = (S, A, E, \gamma) \]
State Transition System

\[ \Sigma = (S, A, E, \gamma) \]

- **S** = \{states\}
- **A** = \{actions\}
- **E** = \{exogenous events\}
- State-transition function \( \gamma: S \times (A \cup E) \rightarrow 2^S \)
  - **S** = \{s_0, \ldots, s_5\}
  - **A** = \{move1, move2, put, take, load, unload\}
  - **E** = \{\}
  - \( \gamma \): see the arrows

The Dock Worker Robots (DWR) domain
Conceptual Model
2. Controller

State transition system
\[ \Sigma = (S, A, E, \gamma) \]

Controller

Description of \( \Sigma \)

Planner

Observations

Execution status

Initial state

Observation function
\( h: S \rightarrow O \)

Given observation \( o \) in \( O \), produces action \( a \) in \( A \)

System \( \Sigma \)

Actions

Events
Complete observability:  
\( h(s) = s \)

Observation function  
\( h: S \rightarrow O \)
Conceptual Model
3. Planner’s Input

State transition system
\[ \Sigma = (S, A, E, \gamma) \]

Depends on whether planning is online or offline

Observation function \( h: S \rightarrow O \)

Given observation \( o \) in \( O \), produces action \( a \) in \( A \)

Planning problem

Observation function

Initial state

Objectives

Execution status

Planner

Controller

System \( \Sigma \)

Plans

Events

Actions
Planning Problem

Description of $\Sigma$
Initial state or set of states
   Initial state = $s_0$
Objective
   Goal state, set of goal states, set of tasks, “trajectory” of states, objective function, …
   Goal state = $s_5$

The Dock Worker Robots (DWR) domain
State transition system
\[ \Sigma = (S,A,E,\gamma) \]

Given observation \( o \) in \( O \), produces action \( a \) in \( A \)

Planning problem

Observation function
\[ h(s) = s \]

Depends on whether planning is online or offline

Instructions to the controller

Conceptual Model
4. Planner’s Output

Planning problem

Initial state

Objectives

Execution status

Planner

Plans

Controller

Actions

System \( \Sigma \)

Events

State transition system
\[ \Sigma = (S,A,E,\gamma) \]
Plans

Classical plan: a sequence of actions

\langle \text{take}, \text{move}1, \text{load}, \text{move}2 \rangle

Policy: partial function from $S$ into $A$

\{(s_0, \text{take}), (s_1, \text{move}1), (s_3, \text{load}), (s_4, \text{move}2)\}

The Dock Worker Robots (DWR) domain
Planning Versus Scheduling

- **Scheduling**
  - Decide when and how to perform a given set of actions
    - Time constraints
    - Resource constraints
    - Objective functions
  - Typically NP-complete

- **Planning**
  - Decide what actions to use to achieve some set of objectives
  - Can be much worse than NP-complete; worst case is undecidable
Three Main Types of Planners

1. Domain-specific
2. Domain-independent
3. Configurable
Types of Planners: 1. Domain-Specific (Chapter 19-23)

- Made or tuned for a specific domain
- Won’t work well (if at all) in any other domain
- Most successful real-world planning systems work this way
Types of Planners

2. Domain-Independent

- In principle, a domain-independent planner works in any planning domain
- Uses no domain-specific knowledge except the definitions of the basic actions
Types of Planners
2. Domain-Independent

- In practice,
  - Not feasible to develop domain-independent planners that work in every possible domain

- Make simplifying assumptions to restrict the set of domains
  - *Classical planning*
  - Historical focus of most automated-planning research
Restrictive Assumptions

- **A0: Finite system:**
  - finitely many states, actions, events

- **A1: Fully observable:**
  - the controller always $\Sigma$’s current state

- **A2: Deterministic:**
  - each action has only one outcome

- **A3: Static** (no exogenous events):
  - no changes but the controller’s actions

- **A4: Attainment goals:**
  - a set of goal states $S_g$

- **A5: Sequential plans:**
  - a plan is a linearly ordered sequence of actions $(a_1, a_2, \ldots a_n)$

- **A6: Implicit time:**
  - no time durations; linear sequence of instantaneous states

- **A7: Off-line planning:**
  - planner doesn’t know the execution status
Classical Planning (Chapters 2-9)

- Classical planning requires all eight restrictive assumptions
  - Offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time
- Reduces to the following problem:
  - Given \((\Sigma, s_0, S_g)\)
  - Find a sequence of actions \((a_1, a_2, \ldots, a_n)\) that produces a sequence of state transitions \((s_1, s_2, \ldots, s_n)\) such that \(s_n\) is in \(S_g\).
- This is just path-searching in a graph
  - Nodes = states
  - Edges = actions
- *Is this trivial?*
Classical Planning

- Generalize the earlier example:
  - Five locations, three robot carts, 100 containers, three piles
    - Then there are $10^{277}$ states
- Number of particles in the universe is only about $10^{87}$
  - The example is more than $10^{190}$ times as large!
- Automated-planning research has been heavily dominated by classical planning
  - Dozens (hundreds?) of different algorithms
  - A brief description of a few of the best-known ones
Plan-Space Planning (Chapter 5)

- Decompose sets of goals into the individual goals
- Plan for them separately
  - Bookkeeping info to detect and resolve interactions

For classical planning, not used very much any more
RAX and MER use temporal-planning extensions of it
Planning Graphs
(Chapter 6)

- Relaxed problem
  [Blum & Furst, 1995]
- Apply all applicable actions at once
- Next “level” contains all the effects of all of those actions
Graphplan

- For $n = 1, 2, \ldots$
  - Make planning graph of $n$ levels (*polynomial time*)
  - State-space search *within the planning graph*
- Graphplan’s many children
  - IPP, CGP, DGP, LGP, PGP, SGP, TGP, …
Heuristic Search (Chapter 9)

- Can we do an A*-style heuristic search?
- For many years, nobody could come up with a good $h$ function
  - But planning graphs make it feasible
    » Can extract $h$ from the planning graph

- Problem: A* quickly runs out of memory
  - So do a greedy search

- Greedy search can get trapped in local minima
  - Greedy search plus local search at local minima

- HSP [Bonet & Geffner]
- FastForward [Hoffmann]
Translation to Other Domains (Chapters 7, 8)

- Translate the planning problem or the planning graph into another kind of problem for which there are efficient solvers
  - Find a solution to that problem
  - Translate the solution back into a plan

- Satisfiability solvers, especially those that use local search
  - Satplan and Blackbox [Kautz & Selman]
Types of Planners: 3. Configurable

- Domain-independent planners are quite slow compared with domain-specific planners
  - Blocks world in linear time [Slaney and Thiébaux, A.I., 2001]
  - Can get analogous results in many other domains
- But we don’t want to write a whole new planner for every domain!
- Configurable planners
  - Domain-independent planning engine
  - Input includes info about how to solve problems in the domain
    » Hierarchical Task Network (HTN) planning
HTN Planning (Chapter 11)

- Problem reduction
  - *Tasks* (activities) rather than goals
  - *Methods* to decompose tasks into subtasks
  - Enforce constraints, backtrack if necessary
- Real-world applications
  - Noah, Nonlin, O-Plan, SIPE, SIPE-2, SHOP, SHOP2

**Task:** travel($x$, $y$)

**Method:** taxi-travel($x$, $y$)

- get-taxi
- ride($x$, $y$)
- pay-driver

**Method:** air-travel($x$, $y$)

- get-ticket($a(x)$, $a(y)$)
- fly($a(x)$, $a(y)$)
- travel($x$, $a(x)$)

**BACKTRACK**

- get-ticket($a(y)$, $a(x)$)
- travel($x$, $a(x)$)
- fly($a(x)$, $a(y)$)

- go-to-Orbitz
- find-flights($BWI$, $TLS$)
- buy-ticket($IAD$, $TLS$)

- go-to-Orbitz
- find-flights($BWI$, $TLS$)
- travel($UMD$, $Toulouse$)

- get-ticket($BWI$, $TLS$)
- get-ticket($IAD$, $TLS$)
- travel($IAD$, $TLS$)

- get-taxi
- ride($UMD$, $IAD$)
- pay-driver

- fly($BWI$, Toulouse)
- travel($TLS$, $LAAS$)

- get-taxi
- ride($TLS$, Toulouse)
- pay-driver
Comparisons

- Domain-specific planner
  - Write an entire computer program - lots of work
  - Lots of domain-specific performance improvements
- Domain-independent planner
  - Just give it the basic actions - not much effort
  - Not very efficient
Comparisons

- A domain-specific planner only works in one domain.

- **In principle**, configurable and domain-independent planners should both be able to work in any domain.

- **In practice**, configurable planners work in a larger variety of domains:
  - Partly due to efficiency
  - Partly due to expressive power

Coverage:

- Configurable
- Domain-independent
- Domain-specific
Typical characteristics of application domains

- Dynamic world
- Multiple agents
- Imperfect/uncertain info
- External info sources
  - users, sensors, databases
- Durations, time constraints, asynchronous actions
- Numeric computations
  - geometry, probability, etc.
- Classical planning excludes all of these
Relax the Assumptions

- Relax A0 (finite $\Sigma$):
  - Discrete, *e.g.* 1st-order logic:
  - Continuous, *e.g.* numeric variables
- Relax A1 (fully observable $\Sigma$):
  - If we don’t relax any other restrictions, then the only uncertainty is about $s_0$

\[ \Sigma = (S,A,E,\gamma) \]
- $S = \{\text{states}\}$
- $A = \{\text{actions}\}$
- $E = \{\text{events}\}$
- $\gamma: S \times (A \cup E) \rightarrow 2^S$
Relax the Assumptions

- Relax A2 (deterministic $\Sigma$):
  - Actions have more than one possible outcome
  - Contingency plan
  - With probabilities:
    - Discrete Markov Decision Processes (MDPs)

- Without probabilities:
  - Nondeterministic transition systems

\[ \Sigma = (S,A,E,\gamma) \]
\[ S = \{ \text{states} \} \]
\[ A = \{ \text{actions} \} \]
\[ E = \{ \text{events} \} \]
\[ \gamma: S \times (A \cup E) \rightarrow 2^S \]
Relax the Assumptions

- Relax A3 (static $\Sigma$):
  - Other agents or dynamic environment
    - Finite perfect-info zero-sum games
- Relax A1 and A3
  - Imperfect-information games (bridge)

\[
\Sigma = (S, A, E, \gamma)
\]
\[
S = \{ \text{states} \}
\]
\[
A = \{ \text{actions} \}
\]
\[
E = \{ \text{events} \}
\]
\[
\gamma: S \times (A \cup E) \rightarrow 2^S
\]
Relax the Assumptions

- Relax A5 (sequential plans) and A6 (implicit time):
  - Temporal planning
- Relax A0, A5, A6
  - Planning and resource scheduling
- And many other combinations

\[ \Sigma = (S,A,E,\gamma) \]
\[ S = \{ \text{states} \} \]
\[ A = \{ \text{actions} \} \]
\[ E = \{ \text{events} \} \]
\[ \gamma: S \times (A \cup E) \rightarrow 2^S \]
A running example: Dock Worker Robots

- Generalization of the earlier example
  - A harbor with several locations
    - e.g., docks, docked ships, storage areas, parking areas
  - Containers
    - going to/from ships
  - Robot carts
    - can move containers
  - Cranes
    - can load and unload containers
A running example: Dock Worker Robots

- **Locations**: $l_1$, $l_2$, …
- **Containers**: $c_1$, $c_2$, …
  - can be stacked in piles, loaded onto robots, or held by cranes
- **Piles**: $p_1$, $p_2$, …
  - fixed areas where containers are stacked
  - pallet at the bottom of each pile
- **Robot carts**: $r_1$, $r_2$, …
  - can move to adjacent locations
  - carry at most one container
- **Cranes**: $k_1$, $k_2$, …
  - each belongs to a single location
  - move containers between piles and robots
  - if there is a pile at a location, there must also be a crane there
A running example: Dock Worker Robots

- Fixed relations: same in all states
  \[ \text{adjacent}(l,l') \quad \text{attached}(p,l) \quad \text{belong}(k,l) \]

- Dynamic relations: differ from one state to another
  \[ \text{occupied}(l) \quad \text{at}(r,l) \]
  \[ \text{loaded}(r,c) \quad \text{unloaded}(r) \]
  \[ \text{holding}(k,c) \quad \text{empty}(k) \]
  \[ \text{in}(c,p) \quad \text{on}(c,c') \]
  \[ \text{top}(c,p) \quad \text{top}(\text{pallet},p) \]

- Actions:
  \[ \text{take}(c,k,p) \quad \text{put}(c,k,p) \]
  \[ \text{load}(r,c,k) \quad \text{unload}(r) \quad \text{move}(r,l,l') \]