Course on Automated Planning: Introduction

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, Spain
Plan for the Course (6 days)

1. Intro to AI and Automated Problem Solving

2. Classical Planning as Heuristic Search and SAT

3. Beyond Classical Planning: Transformations
   - Soft goals, Conformant Planning, Finite State Controllers, Plan Recognition

4. Planning with Uncertainty: Markov Decision Processes (MDPs)

5. Planning with Incomplete Information: Partially Observable MDPs (POMDPs)

6. Open Challenges in the field; Wrap up
First Lecture

- Some AI history
- The Problem of Generality in AI
- Models and Solvers
- Planning
The proposal (for the meeting) is to proceed on the basis of the conjecture that every aspect of . . . intelligence can in principle be so precisely described that a machine can be made to simulate it.
An early collection of AI papers and programs for playing chess and checkers, proving theorems in logic and geometry, planning, etc.
Importance of Programs in Early AI Work

In preface of 1963 edition of *Computers and Thought*

We have tried to focus on papers that report results. In this collection, the papers . . . describe actual working computer programs . . . Because of the limited space, we chose to avoid the more speculative . . . pieces.

In preface of 1995 AAAI edition

A critical selection criterion was that the paper had to describe . . . a running computer program . . . All else was talk, philosophy not science . . . (L)ittle has come out of the “talk”.

Many of the key AI contributions in 60’s, 70’s, and early 80’s had to do with programming and the representation of knowledge in programs:

- Lisp (Functional Programming)
- Prolog (Logic Programming)
- Rule-based Programming
- Interactive Programming Environments and Lisp Machines
- Frame, Scripts, Semantic Networks
- ‘Expert Systems’ Shells and Architectures
AI methodology: Theories as Programs

• For writing an AI dissertation in the 60’s, 70’s and 80’s, it was common to:
  ▶ pick up a task and domain \( X \)
  ▶ analyze/introspect/find out how task is solved
  ▶ capture this reasoning in a program

• The dissertation was then
  ▶ a \textbf{theory} about \( X \) (scientific discovery, circuit analysis, computational humor, story understanding, etc), and
  ▶ a \textbf{program} implementing the theory, \textbf{tested} over a few examples.

Many great ideas came out of this work . . . but there was a problem . . .
Methodological Problem:

Theories expressed as programs cannot be proved wrong: when a program fails, it can always be blamed on ’missing knowledge’

Three approaches to this problem

- narrow the domain (expert systems)
  - problem: lack of generality
- accept the program is just an illustration, a demo
  - problem: limited scientific value
- fill up the missing knowledge (intuition, commonsense)
  - problem: not successful so far
AI in the 80’s

The knowledge-based approach reached an **impasse** in the 80’s, a time also of debates and controversies:

- **Good Old Fashioned AI** is "rule application” but intelligence is not (Haugeland)
- **Situated AI**: representation not needed and gets in the way (Brooks)
- **Neural Networks**: inference needed is not logical but probabilistic (PDP Group)

Many of these criticisms of mainstream AI partially valid then; less valid now.
AI Research in 2010

Recent issues of AIJ, JAIR, AAAI or IJCAI shows papers on:

1. SAT and Constraints
2. Search and Planning
3. Probabilistic Reasoning
4. Probabilistic Planning
5. Inference in First-Order Logic
6. Machine Learning
7. Natural Language
8. Vision and Robotics
9. Multi-Agent Systems

I’ll focus on 1–4: these areas often deemed about techniques, but more accurate to regard them as models and solvers.
Example: Solver for Linear Equations

\[ \text{Problem} \implies \text{Solver} \implies \text{Solution} \]

- **Problem**: The age of John is 3 times the age of Peter. In 10 years, it will be only 2 times. How old are John and Peter?

- **Expressed as**: \( J = 3P \); \( J + 10 = 2(P + 10) \)

- **Solver**: Gauss-Jordan (Variable Elimination)

- **Solution**: \( P = 10 \); \( J = 30 \)

Solver is **general** as deals with any problem expressed as an instance of **model**

Linear Equations Model, however, is **tractable**, AI models are not . . .
AI Solvers

\[ \text{Problem} \implies \text{Solver} \implies \text{Solution} \]

- The basic models and tasks include
  - **Constraint Satisfaction/SAT**: find state that satisfies constraints
  - **Bayesian Networks**: find probability over variable given observations
  - **Planning Problems**: find action sequence that produces desired state
  - **Planning with Feedback**: find strategy for producing desired state

- All of these models are **intractable**, and some extremely powerful (POMDPs)
- The challenge is computational: **how to scale up**
- For this, solvers must **recognize and exploit structure** of the problems
- Methodology is **empirical**: benchmarks and competitions
- Significant **progress** in recent years
**SAT and CSPs**

- **SAT:** determine if there is a *truth assignment* that satisfies a set of clauses

\[ x \lor \neg y \lor z \lor \neg w \lor \cdots \]

- Problem is NP-Complete, which in practice means worst-case behavior of SAT algorithms is *exponential* in number of variables.

- Yet current SAT solvers manage to solve problems with **thousands of variables and clauses**, and used widely (circuit design, verification, planning, etc).

- Key is **efficient (poly-time) inference** in every node of search tree: *unit resolution, conflict-based learning, . . .*

- Many other ideas *logically possible*, but **do not work** (don’t scale up).

- Same for **Constraint Satisfaction Problems (CSPs)**
Related Tasks: From SAT to Bayesian Networks

- **Weighted MAX-SAT**: find assignment $\sigma$ that minimizes total cost $w(C)$ of violated clauses

$$\sum_{C : \sigma \not\models C} w(C)$$

- **Weighted Model Counting**: Adds up 'weights' of satisfying assignments:

$$\sum_{\sigma : \sigma \models \mathbf{T}} \prod_{L \in \sigma} w(L)$$

SAT methods extended to these other tasks, closely connected to **probabilistic** reasoning tasks over **Bayesian Networks** (and Neural Networks):

- **Most Probable Explanation (MPE)** easily cast as Weighted MAX-SAT
- **Probability Assessment** $P(X|\text{Obs})$ easily cast as Weighted Model Counting

Current **best BN solvers** built over this formulation (ACE, Weighted Cachet)
Further ties between Deductive and Probabilistic Inference

- Underlying **structure** of SAT, CSPs, and BNets can be expressed by graph $G$

  ![Graph G](image)

- A parameter called the **(induced) treewidth** $w(G)$ measures then how 'close' is $G$ to a Tree, $w(G) = 2$ for $G$ above, and $w(G) = 1$ if $G$ is a tree.

- All SAT, CSP, and BN tasks are **exponential** in $w(G)$, and thus solvable in **polynomial time** for **bounded** $w(G)$ (e.g., trees)

- These models often referred to as **graphical models** (Dechter 03)

- Resolution methods for all these models closely related
Planning Models: Producing States by Applying Actions

- **(Classical) Planning** concerned with finding a sequence of actions that transforms an initial state into a goal state. This is called a **plan**.

- **States** are truth assignments as before, represented by the atoms that are true.

- Actions add certain atoms and delete others, provided their **preconditions** hold.

- A **planner** is a solver that takes a planning problem (initial and goal states, and actions) and outputs a **plan**.

- The **cost** of a plan is given by the number of actions.

\[
\text{Init, Actions, Goals} \implies \text{Planner} \implies \text{Plan}
\]
Given the actions that move a 'clear' block to the table or onto another 'clear' block, find a plan to achieve the goal.

Problem becomes finding a path in a directed graph.
How planning problems are solved?

- How do we find a route in a map from Barcelona to Madrid?

- Need **sense of direction**: whether an action takes us towards the goal or not

- In AI, this is captured by **heuristic functions**: functions $h(s)$ that provide an **estimate of the cost** (number of actions) from any state $s$ to the goal

- **Key new idea in planning** is that useful heuristics $h(s)$ can be obtained **automatically** from the problem encoding

- **How?** Solving a **relaxed problem** where **deletes** are dropped

  - Heuristic $h(s)$ is cost of solution found for **relaxed problem** in **poly-time**
How is our problem solved?

- Provided with the **heuristic** $h$, plan found without search by **hill-climbing**
- Actually, only states reached by actions in **blue** need to be evaluated
Summary: AI and Automated Problem Solving

- A research agenda that has emerged in last 20 years: solvers for a range of intractable models

- Solvers unlike other programs are general as they do not target individual problems but families of problems (models)

- The challenge is computational: how to scale up

- Sheer size of problem shouldn’t be impediment to meaningful solution

- Structure of given problem must recognized and exploited

- Lots of room for ideas but methodology empirical

- Consistent progress
  - efficient but effective inference methods (derivation of $h$, conflict-learning)
  - islands of tractability (treewidth methods and relaxations)
  - transformations (compiling away incomplete info, extended goals, . . . )