Beating Kasparov is great . . . but how to play Mau-Mau??

- You (and your brother/sister/little nephew) are better than Deep Blue at everything – except playing Chess.
- Is that (artificial) "Intelligence"?
- How to build machines that automatically solve new problems?

Agenda

- What?
- Why?
- How?
- Summary
General Problem Solving

Write one program that can solve all problems.

\[ X \in \{ \text{algorithms} \} : \text{for all } Y \in \{ \text{"problems"} \} : X \text{"solves"} Y \]

Write one program that can solve a large class of problems.

General Game Playing

Write one program that can play all (2-player, 0-sum, \ldots) games.

- Describe “game” in a declarative language (based on logics).
- Variables (board position/cards), operators (moves), win/loss.
- Chess, but also Räuberschach, and Backgammon, and \ldots

Planning

Write one program that can solve all classical search problems.

What is a “classical search problem”?
- Variables: \( x \) with finite number of possible values \( D(x) \); state gives value to each variable (e.g., \( x = 1 \)).
- Actions: precondition (some variable values); effect (changes to some variable values); e.g., action “increment \( x \)\): pre \( x = 1 \), eff \( x := 2 \)\).
- Initial state: What the world is like now (e.g., \( x = 1 \)).
- Goal: What we want the world to change into (some variable values, e.g., \( x = 2 \)).
- Plan: Sequence of actions achieving the goal (e.g., “increment \( x \)”).

Example: FreeCell

Variables: card positions (\( position_{spades} = Q_{hearts} \)).
- Actions: card moves (\( move_{spades, Q_{hearts, freeCell}} \)).
- Initial state: start configuration.
- Goal: all cards “home”.
- Plan: Card moves solving this game.
Natural Language Sentence Generation

- **Input**: Tree-adjoining grammar, intended meaning.
- **Output**: Sentence expressing that meaning.

Generating Business Process Templates at SAP

- **Input**: SAP-scale model of behavior of activities on Business Objects, process endpoint.
- **Output**: Process template leading to this point.

Automatic Network Security Testing

- **Input**: Network configuration, location of sensible data.
- **Output**: Sequence of exploits giving access to that data.

Planning!

- Planning Domain Definition Language (PDDL) $\mapsto$ Planning System
Is AlphaGo a General Problem Solver?
(A): Yes  (B): No

→ At a high level, the answer is a clear no: AlphaGo does Go and nothing else.

Digging a bit deeper:
- Sometimes argued: The learning mechanisms are general/generic.
- But their architecture in Go was investigated by 150 DeepMind researchers.
  → In this sense, the learning is not any more general/generic than, e.g., search.
- Most impressive imho: the human-like ability to evaluate Go positions, and the human-like learning suggesting that similar things can be done in many other domains too.
  → Search + learning seems to be a key formula for General Problem Solving.

Any problem that can be formulated in the modeling language.
Don’t write the C++ code, just describe the problem!
Don’t maintain the C++ code, maintain the description!

- Powerful: In some applications, generality is absolutely necessary. (E.g. SAP)
- Quick: Rapid prototyping: 10s lines of problem description vs. 1000s lines of C++ code. (E.g. language generation)
- Flexible: Adapt/maintain the description. (E.g. network security)
- Intelligent: Determines automatically how to solve a complex problem effectively! (The ultimate goal, no?!)
- Efficiency loss: Domain-specific knowledge/engineering usually is key to performance . . .
  → Trade-off between “automatic and general” vs. “manual work but effective”.

How to make fully automatic algorithms effective?

- We are solving hard problems (NP-hard is the “easy case”)
- Enumerate all possibilities . . .

Search guidance ⇒ detect/learn problem structure!
(My) Research in General Problem Solving

- **Modeling**
  - PDDL, Logic, ...  
  - Hierarchies of generality (add/drop features).  
  - Computational complexity: How hard are these problems?
  - Expressive power: Which problems can be represented compactly? Which language features can increase that set?
  - Model efficiency: How does the model affect solver performance? How can we support the design of “good” models?

- **Algorithms**

- **Implementation**

- **Experiments**

- **Applications**

---

Search guidance: Lower bound functions $h_i$ enable us to prune parts of the search space.

Tractable fragments: Automatically relax input problem $X$ into tractable class $Y$, solve $Y$ to obtain lower bound $h$ for $X$.

Performance analysis: How does problem structure affect the quality of the bound $h_i$? Can we detect this automatically?

Learning: Learn search guidance (e.g., a Neural Network state-evaluation) from experience in solving similar problems.

---

Data structures: Clever coding can easily account for a factor 10 in performance.

Planning systems are big: I did > 100000 lines C during my PhD.
(My) Research in General Problem Solving

- Modeling
- Algorithms
- Implementation
- Experiments
  - Across domains: Assess general value of technique (> 50 benchmark domains in Planning: FreeCell, Sokoban, airport ground-traffic, printer control, elevator control, ...)
  - Performance analysis: What are the relevant structural properties affecting performance? Systematic exploration via controlled experiments.
  - Planning competition: Win the IPC!
- Applications
  - Computational Linguistics: Lots of very general search problems in, e.g., sentence generation and language alignment.
  - Attack simulation: More accurate models, more effective algorithms, socio-technical attacks, ... 
  - Verification: Finding error states in software is a planning problem; program synthesis is, too.
  - Agent control in virtual reality, plan recognition in intelligent user interfaces, dialogue planning, ...

Summary

The vision of general problem solving:

(some new problem)

describe problem → use off-the-shelf solver

(solution competitive with a human-made specialized program)

Beating humans at coming up with clever solution methods!

A Note on Terminology

- The term General Problem Solving was introduced by Newell and Simon (1963), then forgotten.
- However, several areas today follow its approach:
  - Separate the problem description from the strategy how to solve it. Develop technology targeted at solving any problem.
  - General Game Playing, Planning, SAT, CP, Theorem Proving.
- Nobody except me uses “General Problem Solving” quite yet . . .
  - Michael Thielscher: “General Intelligence”
  - Hector Geffner: “Model-based Approach to Autonomous Behavior”
- . . . so don’t expect to find this term in the literature.
Thanks for listening.

Need a night-time read?
Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)
