[AIND 2018 Term 1] Carl Wennstam

Research review

- Introduction
    - Definitions of AI
    - Practical use
    - Choice of research papers
- Planning in Games: An Overview and Lessons Learned
    - Classical planning (STRIPS)
    - Hierarchial Networks (HTN)
    - Related techniques
    - Lessons learned
- Multi-Agent Planning with Planning Graph
    - Agent relationships
- PDDL: A Language with a Purpose?
    - Picat’s planner
    - Tabling
    - Comparison to PDDL
- PDDL: A Language with a Purpose?

Introduction

Definitions of AI

Several definitions of AI exist, along two dimensions Thinking (Humanly vs

Rationally) and Acting (Humanly vs Rationally). **Planning** seems to be a field of AI which fits well the two definitions given by our lecturers:

> *AI is programming a computer to do the right thing when you don't know what the right things is. - **Peter Norvig***

This is the case in planning problems, as we are generating a behavior (which is generally on the Rationalist side but might be towards the Humanist side depending on the application) without knowing what that the optimal behavior will be.

> *AI is clever solutions to exponential problems. - **Thad Starner***

This is also featured here, as we are dealing with employing **search** and **planning graphs** to **NP-hard** problems.

Practical use

So, in theory, planning seems at the heart of AI. But how useful is it? With the recent hype around **neural networks** and **deep learning**, is planning (and planning graphs) still relevant? Some of my classmates did not seem convinced. For this reason, I decided to focus on some practical applications of planning and PDDL.

Choice of research papers

- Planning in Games: Planning in Games: An Overview and Lessons

Learned

    - practical implications in industry applications
    - Discusses hiearical planning

- PDDL: A Language with a Purpose?

    - critique of PDDL usefullness in theoretical and practical terms

- Planning as Tabled Logic Programming

    - Argues that the features of tabled logic (Picat) model make it a more appropriate language than PDDL for modeling and solving planning problems.

Planning in Games: An Overview and Lessons Learned

Classical planning

Classing planning, where we go from an initial state to a goal using a STRIPS style planner, requires a good definition of the problem so that A* can be applied. The first major game to apply this was F.E.A.R. in 2005. A planner was implemented to generate NPC behavior. This was feasible even 13 years ago due to the problems having short sequences. The game spawned a sequel, two expansions and inspired at least 5 other title. One of the later iterations, Stalker (2008), added a nested structure so that more complex sequences could be generated out of the core STRIPS search.

Hierarchial networks

HTN (mentioned in AIND 11.2) searches through actions that break down **recursively** using **task decomposition** and was first featured in Killzone 2, as the designers they wanted more control over the behaviors (approaching the *Acting Humanly* side). This aspect of HTN, letting the agent draw on information from the designers, is also discussed in AIND. A **SHOP** (Simple Hierarchial Ordered Planner) turned out a big success, and allegedly the players were often suprised that the in-game bots were not in fact online players, thereby passing a type of **Turing test**. The later game Transformers : War for Cybertron used a similar **SHOP** algorithm which also tracked the (side)effects of all actions.

Generally, SHOP focuses on two core issues in planning:
For many types of planners, including HTN, their limitation is in the flexibility of the domain. Custom procedural functions can be hard to integrate when you don’t know how the planner may combine actions while planning.
To deal with this, SHOP limits the expressiveness of the planner (by imposing an order on the plans) to make it easy to insert custom logic.

Related techniques

A **behavior tree** is similar to a HTN but stops at current action if “this is likely to work out”, rather than searching the full graph. These have been in use in the industry since 2004, inspired by robotics and virtual agents from decades before.

A utility system is the term used to describe a voting/scoring system, and they are often applied to sub-systems of games like selecting objects/positions based on the results of a spread-sheet like calculation. Featured in **The Sims**.

Lessons learned

- According to the author, the open questions for using planners in game AI are about **design**, how to tweak the behaviors resulting from planners, and thinking more in terms of systems and emergent AIs. This is where the efforts have been focused and this has led to the incremental transition towards hierarchical approaches.

- In well understood domains, simpler techniques work well. In the cases of action/combat games, we can easily build robust AI that looks deliberate using simple **reactive** techniques like behavior trees.

- Planning has most benefits in **unknown domains**. One of the benefits of planning is in **prototyping**, creating new behaviors quickly by letting the planner generate behavior given new actions or goals to work with. Planners also have shown to be more beneficial in **open worlds**, where the sandbox simulation has significantly more complexity.

> *Regardless of whether developers use planning techniques or not, an*

> *architecture that **separates the AI's goals** (or WHAT to do) and the **AI's decision making** (or HOW to do it), has proven to be very effective.*

Multi-Agent Planning with Planning Graph

While AIND 11.4 contains an overview of Multi-Agent Planning, paper describes implemntation details for planning for multi-agents situations in STRIPS-like domains with planning graph.

Agent relationships

Three possible relationships between agents' goals are considered in order to evaluate plans: the agents may be **collaborative**, **adversarial** or **indifferent**. Algorithms to deal with each situation are proposed.
The collaborative situations can be easily dealt with the original GRAPHPLAN algorithm by redefining the domain in a proper way. Forward-chaining and backward chaining algorithms are discussed to find infallible plans in adversarial situations. In case such plans cannot be found, the agent can still attempt to find a plan for achieving some part of the goals. A forward-chaining algorithm is also proposed to find plans for agents with independent goals.

Planning as Tabled Logic Programming

Picat’s planner

This paper describes **Picat’s planner**, its implementation, and planning models as an **alternative to PDDL style planners**.
I could not find a mention of this recent planner in AIND. Broadly, this paper demonstrates the e?ectiveness of tabled logic programming for planning, and argues the **importance of modeling** despite recent signi? cant progress in domain-independent PDDL planners. This ranges from designing state representations to facilitate data sharing and symmetry breaking, encoding actions with operations for e?cient precondition checking and state updating, to incorporating domain knowledge and heuristics.

Tabling

Tabling is a technique used in logic and functional programming systems, which caches the results of certain calculations in memory and reuses them in subsequent calculations through a quick table lookup.
Like state marking used in search algorithms, tabling can prevent the same state from being expanded more than once during search. Tabling has been found useful in many search problems, including theorem proving, program analysis and model checking. Recently, tabled logic programming has been successfully employed to solve speci?c planning problems.

During search, every state encountered is tabled, and tabled states are used to e?ectively perform resource-bounded search. In Picat, structured data can be used to **avoid enumerating all possible permutations** of

objects, and term sharing is used to avoid duplication of common state data.

Comparison to PDDL

As a modeling language for planning, Picat di?ers from PDDL in that it:

- allows use of structures to represent states.
- supports explicit commitment and nondeterministic actions, which enables users to have better control over action applications
- provides facilities for describing domain knowledge and heuristics for pruning search space.

As a solving system, Picat’s planner implements several techniques for better performance.

- Tabling of every state encountered during search and avoids repeating the exploration of the same state.
- Adopts a hash-consing technique to share common state data and to speed up the equality testing of states.
- Using the tabled states to e?ectively perform resource-bounded search.

PDDL: A Language with a Purpose?

Argues that, going forward, more efforts are needed to define the purpose of PDDL (theoretical modelling or practical planning applications) and formalize and develop the language in that direction.

> *Good planning algorithms are hard to devise, but fairly easy to evaluate; on the other hand, modelling languages are fairly easy to devise, but hard to evaluate.*

References

- *Planning in Games: An Overview and Lessons Learned*, **Alex J. CHAMPANDARD**, AiGameDev.com (March, 2013) http://aigamedev.com/open/review/planning-in-games/

- *Planning as Tabled Logic Programming*, **N-F. ZHOU**, CUNY Brooklyn College and Graduation Center (July, 2015) https://arxiv.org/pdf/1507.03979.pdf

- *Multi-Agent Planning with Planning Graph* **T.D. BUIL**,Parlevink Group, University of Twente, Netherlands (2003) http://krak.ipipan.waw.pl/~wjamroga/papers/plangraph03eunite.pdf

- *PDDL: A Language with a Purpose?*, **T. L. MCCLUSKEY**, Department of Computing and Mathematical Science, School of Computing and Engineering,University of Huddersfield, UK (June, 2003)