AIND–Planning Module

Research Review Reading

Aaron Caroltin

June 5, 2017

Planning by SAT and constraint satisfaction

Many transition systems have too many states to consider them explicitly one by one, and then factored representations that allow representing large numbers of states and state sequences may be a more efficient alternative. Such representations and search methods are known as *symbolic* or *factored*. The earliest symbolic search methods were based on Binary Decision Diagrams (BDDs), pursued in state-space research since late 1980s. The currently most scalable method (since the late 1990s) is based on reduction to the propositional satisfiability problem SAT.

SAT methods are less prone to excessive memory consumption than BDDs. Unlike explicit state-space search, their memory consumption can be (far) less than linear in the number of visited (stored) states, and they can be implemented with a memory consumption that is linear in the length of a plan or transition sequence (as opposed to the in general exponential memory consumption of explicit state-space search.) However, the currently leading algorithms for the SAT problem gain much of their efficiency by consuming more memory than theoretically optimal algorithms, and in practice consume as much memory as explicit state-space search.

Meta-level search strategies

When solving a planning problem, there are some meta-level strategies that can be applied to any type of search algorithm (or their combinations).

Algorithm portfolios

The idea of using a combination of several techniques has already been mentioned in the context of heuristics, where aggregates of two or more unrelated heuristics can be formed. Algorithm portfolios do the same at the highest level of a program that solves a planning problem. The general idea is to utilize the complementarities between different search methods. If the strengths of algorithm 1 are sufficiently complementary to the strengths of algorithm 2, the combination of these two algorithms may be much stronger than either of the components.

A portfolio can be used in different ways.

- **Selection:** Choose one algorithm from the portfolio, based on the properties of the problem instance, and run it.
- **Sequential composition:** Run several algorithms according to a given schedule, for example the first algorithm for *n* seconds or until some termination criterion is satisfied, and if no solution has been found, terminate the run and continue with the next algorithm.
- **Parallel composition:** Run several algorithms in parallel (in one or several CPUs/cores), with the same or different rates, until one of them finds a solution.

The advantage of parallel composition is that it finds a solution quickly if one of the component algorithms finds a solution quickly. Its disadvantage is that all the algorithms are run simultaneously, requiring more memory.

Explicit state-space search

Explicit state-space search, meaning the generation of states reachable from the initial state one by one, is the earliest and most straightforward method for solving some of the most important problems about transition systems, including model-checking (verification), planning, and others, widely used since at least the 1980s. All the current main techniques for it were fully developed by 1990s, including symmetry and partial order methods, informed search algorithms, and optimal search algorithms (A* already in the 1960s). It is relatively easy to implement efficiently, but, when the number of states is high, its applicability is limited by the necessity to do the search only one state at a time. However, when the number of states is less than some tens of millions, this approach is efficient and can give guarantees of finding solutions in a limited amount of time.

Search algorithms

There are several types of search algorithms that are routinely applied in planning. These include the following.

1. well-known uninformed search algorithms like depth-first search, breadth-first search
2. systematic heuristic search algorithms with optimality guarantees, for example A* and its variants like IDA*, WA*
3. systematic heuristic search algorithms without optimality guarantees, for example the standard "best-first" search algorithm which is like A* but ignores the cost-so-far component of the valuation function
4. incomplete unsystematic search algorithms, most notably stochastic search algorithms

Incomplete search algorithms can be useful for specific problems when good heuristics are available, or as a part of a portfolio of search algorithms.

References

1. D. Mitchell, A SAT Solver Primer, EATCS Bulletin, 85, pp. 112-133, February 2005.
2. P. Beame, H. Kautz, and A. Sabharwal, Towards Understanding and Harnessing the Potential of Clause Learning, Journal of AI Research, 22, pp. 319-351, 2004.
3. Gomes & Selman, Algorithm portfolio design: theory vs. practice, Uncertainty in AI, 1997.
4. P. E. Hart, N. J. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum-cost paths, IEEE Transactions on System Sciences and Cybernetics, SSC-4(2), pp. 100-107, 1968.
5. J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, 1984.
6. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach.