Probabilistic extensions to Graphplan

John Langford

Problem: Given knowledge about the possible states of the world, the ability to transform from one state to another, an initial state, and a goal state, what is the optimal path from the initial state to the goal state? In full generality, the planning problem is inherently difficult involving an exponential size search. For deterministic settings decent algorithms exist, but in probabilistic settings, considerable research remains to be done.

Impact: Finding elegant solutions to the probabilistic planning problem would be useful for all sorts of planning agents. Currently, planning algorithms are severely limited by the requirement of full knowledge and deterministic domains. If these two restrictions can be removed without seriously slowing the planning algorithm, then planning can useful in the real world, including applications in robots, resource allocation, and efficient delivery.

State of the Art: Graphplan is a new planner built by Avrim Blum and Merrick Furst based upon the idea of first precompiling the planning problem onto a polynomial sized graph then using the structure of the graph to prune a search. Significant speed increases over previous planners have been gained. Planners which work on the same basic representation as graphplan include Prodigy and UCPOP. Another interesting approach which uses a different representation of the problem is reinforcement learning.

Approach: I came up with a way to embed probabilistic operators into the graph, allowing graphplan to conduct searches in spaces where the outcome of a particular action is unknown. Avrim further extended this approach to deal with probabilistic state and contingent outcomes. Currently we are thinking about ways to gain speed ups, ways to further extend graphplan into the probabilistic domain, some other new approaches.

Future Work: Currently, I'm thinking about two ideas for extending graphplan. These are generalized types and allowing overloading of the (currently built in) assumption that state does not change if you don't affect it. In all planners that I know of, predicates only have a 'boolean' type. Unfortunately, while it is possible to map all problems down to booleans it is not always easy to do so. Allowing predicates with an integer type could be useful in several situations, such as the 'water jugs' problem or the class of problems where there is some finite resource which gets used up.

The basic assumption that elements of the state are unchanged unless the agent acts to change them is right most of the time, but not right often enough that something needs to be done about this. Examples include objects on a slope or effects of other agents.