Becoming Reactive By Concretization

Armand Prieditis and Bhaskar Janakiraman
Department of Computer Science
University of California
Davis, CA 95616

Abstract

One way to build a reactive system is to construct an action table indexed by the current situation or stimulus. The action table describes what course of action to pursue for each situation or stimulus. This paper describes an incremental approach to constructing the action table through achieving goals with a hierarchical search system. These hierarchies are generated with transformations called *concretizations*, which add constraints to a problem and which can reduce the search space. The basic idea is that an action for a state is looked up in the action table and executed whenever the action table has an entry for that state; otherwise, a path is found to the nearest (cost-wise in a graph with cost-weighted arcs) state that has a mappring from a state in the next highest hierarchy. For each state along the solution path, the successor state in the path is cached in the action table entry for that state. Without caching, the hierarchical search system can logarithmically reduce search. When the table is complete the system no longer searches: it simply reacts by proceeding to the state listed in the table for each state. Since the cached information is specific only to the nearest state in the next highest hierarchy and not the goal, inter-goal transfer of reactivity is possible. To illustrate our approach, we show how an implemented hierarchical search system can completely reactive.

1 Introduction and Motivation

Intelligent interaction with the world can be viewed as a combination of planning to achieve some goal and of reaction to external stimuli in the course of executing a plan. A pure planning system produces a complete plan of actions before executing it [4, 8, 3, 5]. In contrast, a pure reactive system quickly selects and executes a single action based on an external stimulus [2, 1, 6]. Planning systems appear to work well when the predictability of the world is precisely captured in the planner's actions, whereas reactive systems appear to work well in worlds that are fraught with uncertainty or unpredictability—where plans have little chance of succeeding in their entirety, where the ability to plan to completion is not a virtue. This paper describes how a planning system can incrementally become more reactive through interaction with its world, By becoming more reactive, the system reduces its decision-making time.

Previous approaches to building reactive systems from non-reactive ones include compilation and learning from examples. Firby [5] and Rosenschein [13] show how to compile high-level input descriptions of actions and goals into reactive systems. Similarly, Rosenschein and Kaelbling describe a technique to compile constraint expressions into directly executable circuits for a robotic control system [14]. Mitchell uses Explanation-Based Learning to incrementally learn the general conditions under which a particular action, which helps achieve a particular planning goal, should be applied [10]. If the conditions are matched, the same action is applied—irrespective of the system's current goal. The advantage of learning over compiling is that examples focus on those parts of the environment with which an intelligent agent actually interacts; only those actions that are relevant to that interaction are compiled for reactivity.

The problem with the Explanation-Based Learning approach is that multiple goals can lead to multiple action suggestions for the same state, which results in deliberation as to which action to apply and therefore less reactivity. This anomaly is commonly called the *wandering bottleneck* problem in the machine learning literature: as a result of eliminating one time bottleneck (e.g. time taken to react) another one unexpectedly arises (e.g. time taken to decide how to react). More precisely, in a problem with n problem-solving states, each state can have as many as n possible action suggestions since there can be as many as n goals from which the action suggestions are learned. Moreover, to store such a network of states and actions can require as much as $O(n^2 \log n)$ space over n goals and n states, since $O(\log n)$ space is required to store each action suggestion. If n is an exponential of problem size, then this approach is generally not feasible.

This paper describes a technique to avoid the wandering bottleneck problem by hierarchically organizing the state space such that at most one action is learned for each state. As a side-benefit, this hierarchical organization reduces the worst-case space requirements by a factor of n.

The rest of this paper is organized as follows. Section 2 defines the notion of a concretization and derives several important properties of concretizations in

search. Section 3 describes our approach to becoming reactive by concretization. Section 4 presents experimental results of applying our approach. Finally, Section 5 summarizes the conclusions of this work and discusses a few promising avenues of future research.

2 Concretizations

Intuitively, a concretization of a problem is one that has added constraints. The importance of these added constraints is that they reduce the branching factor during search. To formalize this intuitive notion requires a definition of search. The definition that we will assume is standard in the AI literature [11]. A search problem can be thought of as consisting of a graph of nodes, which represent states, and directed arcs that represent the application of an operator. These arcs are typically weighted to represent the cost of applying the corresponding operator. Search can be thought of as finding a finite path in the graph from a node representing a given initial state to a node representing a given goal state. The graph can be specified explicitly or implicitly. In an explicit specification, the nodes and arcs with associated costs might be supplied in a table that includes every node in the graph and a list of its successors and the costs of associated arcs. This information might also be specified by a matrix that stores the costs of associated arcs for every pair of nodes (an infinite cost arc represents the absence of an arc). In an implicit specification, only that portion of the graph that is sufficient to include a goal node is made explicit by applying operators using a search algorithm such as A^* [11]. For example, in the Eight Puzzle problem, the set of states consists of all tile permutations and operators only allow swapping the blank with an adjacent tile (i.e. the cost function on a pair of states returns 1 if one state is reachable from the other by swapping the blank with an adjacent tile, and ∞ otherwise). The goal state might specify that the tiles are in a particular order.

More formally, let a search problem be a 3-tuple $\langle S, c \rangle$, where S is a set of states describing situations of the world; $c : S \times S \rightarrow \Re$ is a positive cost function that represents the cost of applying the corresponding action from one state to another; and $G \subseteq S$ is a set of goal states. An *instance* of a search problem includes a 2–tuple $\langle i, g \rangle$ where $i \in S$ is the initial state and $g \in S$ is the goal state (for simplicity, we assume that there is only one goal state). The objective is to find a finite length finite cost path from i to g.

A problem $\langle S', c' \rangle$ is a *concretization* of another problem $\langle S, c \rangle$ with respect to $\phi : S' \rightarrow S$ iff ϕ *reduces cost:* $(\forall s', t' \in S') c(\phi(s'), \phi(t')) \leq c(s', t')$.

For example, Figure 1 shows a concretization of the Towers of Hanoi problem. The original problem is composed of operators that stack smaller disks on top of larger disks from pin to pin; states are simply disks stacked in increasing size on various pins. The initial and goal states for a typical three disk instance of the Towers of Hanoi problem are also shown in the figure. If the disks are numbered from top to bottom and then the operators are constrained such that they never place an odd-numbered disk on an even-numbered disk and vice versa, then this new problem is concretization of the original problem with respect to a mapping function that ignores disk parity. The reason is because the cost is reduced: operators apply more often in the original problem. Notice that any solution in the concrete space is guaranteed to be a solution in the original space because the concretized problem is more restricted. Since the branching factor will be lower for the concretized problem, solution generation will be more efficient (though slightly longer solutions will generally result). This property, which we call *solution-soundness*, is perhaps most powerful when a problem can be concretized into one for which an efficient solution generator exists. Any solution to the concretized problem can then be directly mapped onto a solution to the original problem. For example, a Blocks World problem with three table locations can be concretized into a Towers of Hanoi problem, which has an associated divide-and-conquer algorithm, by assigning a "size" to each block (say, small to large for each block on every stack, consistent in the initial and goal states). Any solution to the corresponding Towers of Hanoi problem can be mapped onto a solution to the original problem simply by ignoring size.

Tenenberg describes a similar property, which he calls the *downward solution property*, in the context of planning with a certain type of operator representation [16]. In his terminology, a transformed problem has the downward solution property if every solution in the transformed space can be mapped onto one for the original problem. Solution soundness is a generalization of the downward solution property since it does not depend on specific operator representations.

Despite the solution-soundness property of concretizations, a solvable problem in the original space

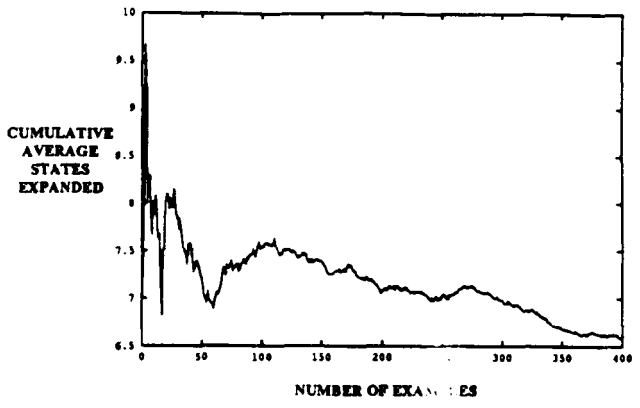

Figure 4 Becoming Reactive Through Interaction with the World

only those states that are most frequently encountered or apply learning techniques to reduce table size. In particular, we are currently investigating applying our ideas to a less artificial problem (a robot routing task), which includes explicitly specified operators with inputs from external sensors such as in a robot routing task. It might be possible to apply Explanation-Based or inductive learning to learn the *class* of states that lead to the nearest state in the next highest hierarchy.

Another problem is that constructing concretization hierarchies is generally a difficult problem. However, a catalog of problem transformations such as those of Absolver II [12] might prove helpful. Another method might be to use clustering algorithms to group similar states into equivalence classes. Problem-solving performance with more meaningful groupings—those that exploit the structure of the search graph and similarity of states—should be improved over the results we obtained with random hierarchical groupings.

Ultimately, we would like to test our ideas in a dynamic world where an intelligent agent's plans to achieve goals are continually thwarted by unforeseen events to which the agent has to react immediately, recover, and then proceed towards achieving the goal. We believe that a hierarchical learning system of the sort described here may be especially suited for such worlds. We are currently modeling a dynamic world and testing this hypothesis.

References

[1] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87*, pages 268–272. Seattle, WA, 1987. American Association for Artificial Intelligence.

[2] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986.

[3] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–378, 1987.

[4] R. Fikes, P. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251–288, 1972. Also in *Readings in Artificial Intelligence*, Webber, B. L. and Nilsson, N. J., (Eds.).

[5] J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, 1989.

[6] L Kaelbling. An architecture for intelligent reactive systems. In M. Georgeff and A. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, 1986.

[7] L. Kleinrock and F Kamoun. Hierarchical routing for large networks. *Computer Networks*, 1:155–174, 1977.

[8] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(2):97–109, 1985.

[9] R. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[10] T. Mitchell. Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, July 1990. American Association for Artificial Intelligence.

[11] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Palo Alto, CA, 1980.

[12] A. Prieditis. Machine discovery of effective admissible heuristics. *Machine Learning*, October 1992. To Appear.

[13] S. Rosenschein. Formal theories of knowledge in ai and robotics. *New Generation Computing*, (3):345–357, 1985.

[14] S. Rosenschein and L. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Theoretical Aspects of Reasoning about Knowledge*, pages 83–98, San Mateo, CA, 1988. Morgan Kaufmann.

[15] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[16] J. Tenenberg. *Abstraction in Planning*. PhD thesis, University of Rochester, 1988.