Plan-Then-Compile Architectures

Tom M. Mitchell

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Tom.Mitchell@cs.cmu.edu

Abstract

Several architectures have recently been developed which integrate planning, reacting, and knowledge compilation learning, in order to improve system reactivity while retaining system flexibility. This paper assesses strengths and weaknesses of such Plan-Then-Compile architectures, by analyzing one such architecture, Theo-Agent [Mitchell 90]. We describe capabilities and limitations of the Theo-Agent, and use these to try to understand the generic capabilities and limits of Plan-Then-Compile architectures more generally.

1. Introduction

Several integrated architectures have recently been developed which may be characterized as Plan-Then-Compile architectures in that they share a number of common features. In particular, we define a Plan-Then-Compile architecture to be any architecture that:

- **Includes both a planner and a reactive decision maker.**
- **Implements the reactive decision-maker by a set of stimulus-response rules.** These rules map directly from the agent's observed world to the action it performs next.
- **Reacts whenever possible.** Whenever the agent must choose an action to perform, it first consults its set of stimulus-response (S-R) rules which constitute its reactive decision maker. If a rule applies to the current situation, it is invoked.
- **Plans when necessary.** If no rules apply, a search-based planner is invoked in order to infer an appropriate action.
- **Learns when forced to plan.** After planning, a new S-R rule is learned (compiled), to cover the just-planned decision and world state. The learned rule is constructed from the plan rationale so that it makes exactly the same decision that the planner would, over some appropriate generalization of the current world state.

RoboSoar [Laird and Rosenbloom 90] and Theo-Agent [Mitchell 90] are two Plan-Then-Compile architectures that have been applied to robot control problems. In addition, the majority of current work on explanation-based learning of control knowledge is based on similar architectural assumptions (e.g., Prodigy [Minton, et al. 89], and much of the work of DeJong and his group).

This paper assesses one such Plan-Then-Compile architecture, the Theo-Agent, according the several criteria of concern to this workshop. To a large degree, this assessment applies to the entire class of Plan-Then-Compile architectures.

2. Theo-Agent

The Theo-Agent architecture is described in detail in [Mitchell 90]. We summarize the main characteristics here. Theo-Agent is a Plan-Then-Compile architecture as defined above -- it reacts when possible, plans when necessary, and learns a new S-R rule when forced to plan. Several additional characteristics distinguish it from the generic Plan-Then-Compile architecture described above:

- **Internal generation of goals.** Goals are created internally by the Theo-Agent, based on prespecified *drives* and the currently observed world state. Each drive specifies a parameterized goal and conditions under which it is to be activated and satisfied. For example, a drive for the goal Recharge-Battery might specify that this goal is to be activated whenever the battery level is sensed to be less than 75% of its maximum, and satisfied whenever the battery level is at least 98% of its maximum. If several goals are simultaneously activated (e.g., Recharge-Battery and Avoid-Obstacle), one is selected for attention based on additionally prespecified rules for prioritization.
- **Caching policy.** All the agent's inferred beliefs about its world, goals, and what it should do at any given point, depend directly or indirectly on sensor input regarding its world. All such inferred beliefs are cached, along with dependencies that describe why each belief is believed. As sense data changes, dependent beliefs are deleted, and lazily rederived only if and when they are needed.
- **Control policy.** The Theo-Agent is controlled by executing the following loop:

  Do Forever:

  1. **Sense** and update readings for all *eagerly sensed* sensor features, and delete any

cached values for *lazily sensed* features.

2. **Decide** which Action to perform
3. **Execute** the chosen Action

The **Decide** portion of the above cycle attempts the following steps in sequence until one succeeds:

1. Retrieve the cached belief regarding which action to perform (if available)
2. Infer the chosen action from available S-R rules
3. Select the first step of the agent's Plan (inferring a plan if necessary)
4. Select the default action (e.g., WAIT)

- **Sensing policy.** The agent distinguishes between *lazy* and *eager* sensing of world features. As noted in the above description of the control policy, eagerly sensed features are updated automatically at the beginning of each sense-decide-execute loop. Lazily sensed features are instead deleted, and are re-sensed if and when they are subsequently needed. This sensing policy provides a simple focus of attention mechanism that helps minimize average reaction time of the agent, while still allowing it to access a large variety of sense features.

2.1. Experimental Results

The Theo-Agent architecture has been used to implement a family of simple mobile robot agents with varying goals and actions. In these programs, the sensor inputs to the robot include a variety of sonar inputs and simple visual features. Available actions include moving forward and backward, turning, and applying sense routines to measure objects near the robot. Typical goals include getting nearer to unknown distant objects, getting away from objects that are too close, and identifying unknown objects. Over a variety of such agent instantiations, the Theo-Agent architecture succeeds in acquiring a collection of S-R rules that encode successful reactive strategies. A more complete description of the Theo-Agent and its performance is available elsewhere [Mitchell 90]. The main results of our experiments may be summarized as follows:

- The architecture reduces response time from minutes to subsecond for a variety of simple agents, as a result of acquiring S-R rules that allow its reactive decision-maker to succeed with increasing frequency as its experience grows.
- In some cases, the preconditions of the learned S-R rules refer to world features that are expensive to sense. In such cases, the dominant cost in matching the rule is the cost of obtaining the necessary sense data.
- While knowledge-compilation style learning is the cause of speedup for the agent, the caching policy also contributes significantly.

3. Characterization

Below we characterize the Theo-Agent according to the list of criteria suggested for this workshop. Although we discuss the Theo-Agent in particular, most of this characterization applies as well to the entire class of Plan-Then-Compile architectures.

Generality: What are the tasks and environments for which this architecture is appropriate? In order for Plan-Then-Compile architectures to lead to increasingly reactive behavior, it appears necessary that:

- **A correct initial model of actions be available.** This is perhaps the biggest constraint on applicability. While knowledge compilation learning can speed up the reactive time of the agent, it cannot (at least in its purest form) alter the correctness of the agent's decisions. In this sense, it shares some of the limitations of non-learning architectures. Much current work on explanation-based learning is aimed toward removing this limitation.
- **An effective reactive strategy can be represented as a collection of if-then rules.** No knowledge-compilation method (or manual coding method, for that matter) can converge to an effective, reactive strategy unless the task, sensors, and environment allow such a strategy to be encoded in the given representation. In the case of the Theo-Agent, this requirement is most easily violated when the number of S-R rules required is very large (see discussion of Scalability below).
- **The agent's environment be forgiving of initial slow reaction time.** Plan-Then-Compile architectures that converge to fast response times may still exhibit very slow initial response times before sufficient compilation has taken place. In cases where slow response time results in the agent's demise, it will never survive to achieve its asymptotic behavior.

Versatility: What types of goals, methods, and behaviors are supported? The Theo-Agent supports only a discrete, categorical notion of goals, as opposed to a continuous reward metric (as used, for example, in many reinforcement learning architectures). While one could imagine extending it to allow such continuous rewards (e.g., the Recharge-Battery is activated/satisfied to the degree that the battery is fully charged), it is not clear how to extend the knowledge compilation techniques to go along with such goals. There are many other specific limitations of the Theo-Agent. For instance, it has only a simple forward-chaining planner. This limitation, however, is more due to a lack of programming effort than any fundamental limitation of Plan-Then-Compile architectures. RoboSoar, for example, supports a variety of planning strategies, and explanation-based learning can be applied to any plan which has a logically-described rationale, regardless of the planning process used to discover it.

Rationality: Are the actions the agent performs always consistent with its knowledge and goals. Theo-Agent is rational. This is one of its primary characteristics. The plans it creates are guaranteed, relative to its action models, to succeed. Furthermore, the learned rules are constructed by the EBG algorithm so that they preserve the correctness (or incorrectness) of the planner's decisions. Learned rules will always make the same decisions as the planner. This rationality leads to success in cases where the agent has a correct model of its actions, and it is tractable to construct complete plans.

Ability to manually add new knowledge: New action descriptions (e.g., pre and postcondition models of the agent's actions) can be added, as well as new drives (i.e., goals and their corresponding activation and satisfaction conditions). A large part of the motivation behind knowledge compilation in Plan-Then-Compile architectures lies in the ability to accept and operationalize new knowledge in the form of new actions or drives. However, if actions or drives are added or changed, then previously compiled S-R rules should be deleted, as they will have "compiled in" assumptions about the agent's previous set of drives and actions.

Ability to learn: New S-R rules are learned from plan rationales, via explanation-based generalization. This type of learning is able to improve the efficiency of decision making in some cases, but note that it is not capable of changing the output decisions of the agent (i.e., the correctness of the decisions is unchanged by this form of learning).

Taskability: Can it perform different tasks based on external commands? At present, the Theo-Agent does not listen for commanded goals from the external world. Instead, it chooses its own goals from its predefined "drives". However, it appears to be straightforward to add a new drive which is activated whenever a new requested goal is input, and which activates this goal. Hence, this limitation seems to be a detail of the current implementation, and not a fundamental limitation of Plan-Then-Compile architectures.

Scalability: Does it scale to larger and larger problems requiring more knowledge? There are two scaling problems that can lead to difficulties for the Theo-Agent. First, as the complexity of actions and world states increases the size of the planning search can grow combinatorially. This is the typical search-space scaling problem faced by search-based planners. The Theo-Agent has no better means for minimizing this combinatorial explosion than non-learning architectures. However, explanation-based learning has been used in many other systems to learn control knowledge to constrain this search. Such techniques are compatible with the Theo-Agent and with Plan-Then-Compile architectures in general. The second scaling problem is that the reactive component of the Theo-Agent slows down as the number of compiled rules grows (due to the cost of matching the growing number of rules). This is a problem common to all Plan-Then-Compile architectures. Approaches to minimizing this problem include (1) empirically determining the utility of learned rules and deleting those whose costs outweigh their benefits [Minton, et al. 89], (2) specializing rules which are especially costly to match by instantiating rule variables, and (3) indexing learned rules so that one need not match every rule explicitly in every world state. Both of these scaling issues are serious problems for Plan-Then-Compile architectures, as they are for non-learning architectures using similar representations for planning or reactive decision makers.

Reactivity: Can it respond to changes in an unpredictable environment? Increasingly so. As it acquires new S-R rules, a greater proportion of its decisions will be made by the reactive component, and fewer by the planner. Unless the reactive component becomes very slow (due to a huge number of rules that must be matched), the overall system will become increasingly reactive.

Efficiency: Does it guarantee real-time response? In practice, the Theo-Agent becomes quite efficient after learning, for the simple but real sets of robot goals and actions we have experimented with. In best cases, learning results in the robot's reaction time going from minutes to subsecond [Mitchell 90]. However, as noted in the above discussion about scaling, the final efficiency depends strongly on how many reactive rules are required to characterize the desired behavior of the agent.

Psychological validity: Is it a reasonable model of human behavior? The Theo-Agent architecture was not designed with this criterion in mind. However, there is evidence (e.g., [Chi and VanLehn 91] that humans accomplish similar kinds of speedup learning via some variant of explanation-based learning. Furthermore, the Soar architecture, on which RoboSoar is based, is explicitly designed to be a model of human cognitive processes.

4. Summary

Plan-Then-Compile architectures offer a means of compiling out efficient reactive rules from knowledge of the agent's goals and actions. Several such architectures exist, and much of the current research on explanation-based learning of control knowledge is compatible with the assumptions underlying these architectures (and much of it is directed toward extending their capabilities and scope). The discussion here informally identifies some of the properties of environments, tasks, goals and actions that are requisites for success for such architectures. This analysis is still incomplete, and additions and corrections are solicited.

Given our weak understanding of the space of integrated architectures, it is reasonable to proceed by analyzing case studies of specific architectures. This paper attempts to go one step further, by identifying a class of related architectures (the Plan-Then-Compile architectures), and to try to characterize properties of the class. Hopefully one result of the workshop will be to identify other classes as well, so that we begin to discuss regions rather than points in the space.

Acknowledgments. Many of the ideas expressed here have come out of discussions with other members of the Theo group, and with participants in the Spring 1991 seminar on Intelligent Agents at CMU. This work is supported by the Defense Advanced Research Projects Agency under research contract N00014-85-K-0116.

References

[Chi and VanLehn 91]
Chi, M., and VanLehn, K.
The Content of Self-Explanations.
Journal of the Learning Sciences , 1991.
in press.

[Laird and Rosenbloom 90]
Laird, J.E. and Rosenbloom, P.S.
Integrating Planning, Execution, and Learning in Soar for External Environments.
In *Proceedings of AAAI '90*. AAAI, 1990.

[Minton, et al. 89] Minton, S., Carbonell, J.G., Knoblock, C.A., Kuokka, D.R., Etzioni, O. and Y. Gil,
Explanation Based Learning: A Problem Solving Perspective.
Artificial Intelligence , 1989.
Also available as technical report CMU-CS-89-103.

[Mitchell 90] Mitchell, T.M.
Becoming Increasingly Reactive.
In *Proceedings of AAAI '90*. AAAI, 1990.

Table of Contents