A Universal Weak Method:

Summary of results

John E. Laird and Allen Newell
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA

ABSTRACT

The weak methods occur pervasively in AI systems and may form the basic methods for all intelligent systems. The purpose of this paper is to characterize the weak methods and to explain how and why they arise in intelligent systems. We propose an organization, called a *universal weak method,* that provides functionality of all the weak methods. A universal weak method is an organizational scheme for knowledge that produces the appropriate search behavior given the available task-domain knowledge. We present a problem solving architecture in which we realize a universal weak method. We also demonstrate the universal weak method with a variety of weak methods on a set of tasks.[1]

1. Introduction

A basic paradigm in artificial intelligence (AI) is to structure systems in terms of *goals* and *methods,* where a goal represents the intention to attain some object or state of affairs, and a method specifies the behavior to attain the goal. Some methods, for example, hill climbing and means-ends analysis, occur pervasively in existing AI systems. Such methods have been called *weak methods.* It has been hypothesized that they form the basic methods for all intelligent systems [5], The purpose of this paper is to characterize the weak methods and to explain how and why they arise in intelligent systems.[2] We propose an organization, called a *universal weak method,* that provides the functionality of all the weak methods We provide an implementation of this method in a production system architecture based on search in a problem space.

2. The Problem Space Hypothesis

A method (weak or otherwise) must be interpreted by an architecture. A key idea on which to base the architecture for an intelligent agent is *search.* Human problem solving and AI programs that work on problems of appreciable intellectual difficulty seem to always exhibit search. The case has been argued that a framework of search is involved in *all* human goal-directed behavior: a hypothesis that is called the *Problem Space Hypothesis* [6].

We will adopt this hypothesis of the centrality of search and will build an architecture for the weak methods around it. Problem search occurs in the attempt to attain a goal. The current situation exists in the agent in some representation, which will be called a *state.* The agent can transform this representation to yield a new representation with *operators.* The set of possible states plus the operators will be called the *problem space.* Together, the problem space and goal define the current *task* that the agent is working on. To attain a goal, the agent starts at an initial state, and applies a sequence of operators to reach some desired state. Some goals also have *path constraints* that limit the paths that are acceptable. *Search control* is the knowledge the agent has about the task that is used to make the decisions encountered while problem solving. The agent must select states, select and apply operators, decide if a subgoal should be used, and decide if a goal has succeeded, failed or should be suspended. A *method* corresponds to a specific pattern of search control knowledge that guides the agent in its decisions.

3. A Problem Solving Architecture

In this section we give a particular search based problem solving architecture: SOAR. SOAR has representations for the *objects* involved: goals, problem spaces, states, and operators. Each representation of an object can be *augmented* with additional information about the object or about the history of the object in the problem solving.

The *current context* of the architecture consists of a single object of each type:

goal, problem space, state, operator

Objects in the current context, together with additional available objects of each type, are called the *stock.* Although the stock is limited by the physical resources of the agent, we make the simplifying assumptions (for the present experiment) that there is unlimited access to the stock and that the stock has unlimited capacity and reliability.

A single generic action is available in SOAR to control the search in the problem space: *rcplacement* of an object in the current context by another object of the same type. After a replacement, the current objects to the right (in the above ordering) of the replacement become undefined. This *initialization* is necessary because each object depends on the partial context provided by other current objects. Different methods of search are realized by selecting and replacing objects in the context.

Search-control knowledge is brought to bear on this process by the *Elaboration Decision Application (EDA) cycle,* which involves three distinct phases of processing. The elaboration phase takes the objects from the stock as input, and augments the current-context objects. Existing data cannot be modified, only augmented. The elaboration phase terminates when no more augmentations can be made. The decision phase follows the elaboration phase and replaces an existing object in the current context based on votes. Using objects in the stock as input, and it collects *votes:* for, against and vetoes. All votes are totaled and SOAR replaces the left-most context-object type that has a new winning object. If the net votes for all objects of a type are negative, fail wins and

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

2 An extended account with full details is available In [2].

replaces the current object. Following the decision phase, if the current state and operator are defined, the operator is applied to the state to produce a new state. The new state replaces the current state, while the current operator (through initialization) becomes undefined The elaboration phase is then repeated.

The elaboration and decision phases constitute the search-control memory of the agent. We use a specialized production system [8] for them with productions of the form:

If C, and C„ and ... and C then A
1 2 n

The C are conditions that examine the current working context and the stock. The form of the conditions is limited to a class of patterns common to production systems [1]. A is an action that either adds knowledge to a current object via an augmentation (for an elaboration production), casts a vote for an object (for a decision production), or applies an operator to a state. A production is satisfied if the conjunction of its conditions is satisfied. All satisfied elaboration productions fire concurrently during the elaboration phase. All satisfied decision productions vote together during the decision phase.

To achieve goals in this architecture, productions must be created that define the task: (1) problem space operators, (2) detection of the desired states of the goal, (3) initialization of the current context and stock with the appropriate goal, problem space, and initial state. A method to control the search is then defined by elaboration and decision productions. Figure 3-1 shows the productions for simple hill climbing For these productions to be used on a specific task, they would have to be instantiated with domain knowledge to compare states and determine the applicability of operators. For Figure 3 1, the elaboration productions compute the evaluation of the current state in relation to the desirea state. The productions for goal decisions detect if the desired state has been reached, or if all the operators have been tried for a state and rejected (fail wins), meaning a local minimum has been reached. The first state decision production will take a step up a hill by voting for the current state if it is better than its ancestor state. The second production rejects a step down the hill by voting for the ancestor state if it is better than the current state The operator productions veto operators that have already been applied to the state, and vote for operators that will apply to the state.

Elaboration
State: If the current states has not been evaluated, compute the evaluation and add it to the state

Decision
Coal. If the current goal is solved, vote for supergoal.
Goal If the current operator is fail, vote for supergoal
State If the current state has an evaluation greater than its ancestor state, vote for the current state
State: If the current state has an evaluation less than its ancestor state, vote for its ancestor state
Operator. If an operator has been applied to a state before, veto it.
Operator If an operator will apply, vote for it

Figure 3-1: Search control for Simple Hill Climbing.

4. A Universal Weak Method

The architecture of the previous section is suitable for encoding many weak methods,[3] although we illustrated this only for hill climbing. For an architecture to be appropriate for realizing the weak methods generally, it must allow simple and direct encodings of all weak methods. One possibility is to have a separate description for each method and select one based on the current situation. Another possibility is to analyze the situation and synthesize the appropriate weak method.

We suggest a third alternative: there should exist something, call it a universal weak method (UWM), that responds directly to a situation by behaving according to the weak method appropriate to the knowledge the agent has of the task. Each weak method can then be characterized by the small amount of knowledge it has about the task. The UWM is what is left after this characteristic knowledge is removed. This is the default behavior: what is done, given nothing that is known about the task. When the knowledge (productions) for a method is added to the UWM, the system behaves according to that method.

Figure 4 1 gives the elaboration/decision productions that constitute the UWM. The first elaboration production detects if all problem spaces have been vetoed (meaning none of the problem spaces were adequate for achieving the goal) and marks the goal unacceptable (an augmentation). The goal must be elaborated with this information so that when it is no longer the current goal, it will not be selected as a subgoal. The next two elaboration productions serve the same purpose for problem spaces, and states. All the decision productions for the goal, problem space, and state vote for an acceptable object and veto unacceptable objects. The operator productions perform the same task for operators. Thus, even though the UWM does specify a nontrvial behavior, it is a simple specification that provides just enough control to search a problem space.

Elaboration:
Goal If the current problem spoce is fail, the goal as unacceptable
Problem Space Il the current state is fail, the problem space is unacceptable
State If the current operator is fail, the state is unacceptable

Decision:
Goal If there is an acceptable available goal, vote for it
Goal If there is an unacceptable available goal, veto it.
Problem Space Il an acceptable problem space is associated to the current goal, vote for it
Problem Space Il an unacceptable problem space is associated to the current goal, veto it.
State. If an acceptable state is in the current problem space, vote for it.
State If an unacceptable state is in the current problem space, veto it.
Operator If an acceptable operator is associated to the current problem space, vote for it.
Operator If an operator has already been applied to the current state, veto it

Figu re 4-1: The Universal Weak Method.

To achieve a goal with the UWM, we must still define the task (operators and attainment-test) within SOAR with productions. With just the UWM and the task productions, SOAR will search the problem space, but unguided by any special knowledge of the task. Search control knowledge must be added as elaboration/decision productions. These productions (along with the UWM) determine the behavior and thereby define a method. The elaboration productions define concepts such as depth, evaluation, difference or duplicate state based on the structure of the current task. The decision productions convert these concepts into action by voting for objects. Figure 4-2 shows the decision productions that define simple hill climbing for any task. Task dependent elaboration productions must be added to Figure 4-2 to compute the evaluation of states. With these productions, the agent has the knowledge to compute an evaluation function for a state of the task and to use

[3] It is also suitable for encoding extended domain-dependent methods, but this is not (he focus of the paper

the evaluation function to select a state. When these productions are added to the UWM, SOAR will search using simple hill climbing.

Simple Hill Climbing
State If the current state is not acceptable or has an evaluation worse than the ancestor state, vote for the ancestor state.
State If the current state is acceptable and has an evaluation better than the ancestor state, vote for the current state.

Figure 4-2: Simple Hill Climbing Search Control

5. Experimental Demonstration

In this section we demonstrate empirically that the UWM just defined is capable of producing many weak methods. We restrict the scope of the demonstration by not considering methods that involve subgoals. Subgoals, of course, are critical to problem solving generally and also to weak methods. However, the role of subgoals in SOAR rests on the (predicted) existence of another functional capacity of a general intelligent agent, *universal subgoaling* [3], to set up subgoals to cope with difficulties that arise in accomplishing a task. We expect that the UWM and universal subgoaling jointly will produce all that might reasonably be called weak methods We attend here only to the UWM in isolation.

SOAR is implemented in a parallel production system architecture, XAPS2 [7].[4] Twelve tasks were implemented in SOAR, with some of them sharing a common problem statement. They are mostly simple tasks familiar from the AI literature, plus a few even simpler decision and logical tasks. Such tasks are suitable for an initial test of a universal weak method, being familiar, easy to implement, and knowledge lean (the situation in which weak methods are used). Elaboration and decision productions containing knowledge about the task (similar to those if Figure 4-2) were added to control the search and produce the behavior of a weak method. Productions were added that achieved the following weak methods: avoid duplicates (AD), heuristic search with operator selection heuristics (OSHS), means ends analysis (MEA), breadth-first search (BRFS), depth first search (DFS), simple hill climbing (SHC). steepest ascent hill climbing (SAHC), best-first search (BFS), modified best-first search (MBFS, that is with one step lookahead), and A star (A*).

Task	UWM	AD	OSHS	MEA	BRFS	DFS	SHC	SAHC	BFS	MBFS	A*
Eight Puzzle	+	+	+	+	+	+	+	+	+	+	+
Tower of Hanoi	+	+	+		+	+					
Missionaries and Cannibals	+	+	+	+	+	+	+	+	+	+	+
Water Jug	+	+	+		+	+					
Picnic Problem 1	+										
Picnic Problem 2	+	+	+	+	+	+	+	+	+	+	+
Picnic Problem 3	+	+	+	+	+	+	+	+	+	+	+
Syllogisms	+			+							
Wason Logic Task	+		+								
Three Wizards	+		+								
Root Finding 1	+	+	+		+	+	+		+		
Root Finding 2	+										

Figure 5-1: All methods versus all tasks

Figure 5-1 shows the results of the tasks and the weak methods. In the cases labeled + , the behavior was that of the stipulated weak method. In the cases left blank there did not seem to be any structure in the task that allowed inclusion of search-control knowledge leading to the weak method. Search-control knowledge was added only if it had heuristic value. Although in principle determining whether a method is being followed could be an issue, there is no doubt at all for the runs in question. The structure of the combined set of productions shows that the method will occur and the trace of the actual run simply serves to verify this. Note that the success of a method on a task is not an issue (some succeeded, some did not), rather the question is whether the UWM plus the search-control production behaved appropriately.

6. Conclusion

We have attempted in this paper to take a step towards a theory of weak methods and an appropriate architecture for a general intelligent agent This has involved introducing a specific problem-solving architecture, SOAR, based on the problem-space hypothesis, and then a universal weak method that provides the ability to perform as any weak method given appropriate search-control increments that respond only to the special knowledge of the task used by the weak methods We demonstrated the generality of the UWM by using many different weak methods for twelve tasks that the weak method exploits.5 This suggests that the weak methods can be defined as the methods that can be obtained by adding simple search control knowledge to the UWM (with universal subgoaling).

References

1. Forgy, C. L. *OPS5 Manual* Computer Science Department, Carnegie Mellon University, 1981.
2. Laird, J. E. and Newell, A. A Universal Weak Method. Carnegie Mellon University, 1983. (Forthcoming)
3. Laird, J. E. and Newell, A. Universal Subgoaling: An Initial Investigation. Proceedings of the AAAI83, American Association for Artificial Intelligence, 1983. (Submitted)
4. Nau. D. S., Kumar, V. & Kanal, L. A general paradigm for A.I. search procedures. Proceedings of the AAAI82, American Association for Artificial Intelligence. 1982, pp. 120-123.
5. Newell, A. Heuristic programming: III structured problems. In Aronofsky, J., Ed., *Progress in Operations Research, III,* Wiley, New York, 1969, pp. 360-414.
6. Newell, A. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson, Ed. *Attention and Performance VIII,* Erlbaum, Hillsdale, NJ, 1980.
7. Rosenbloom, P. S. & Newell, A. Learning by chunking: A production system model of practice. Computer Science Department, Carnegie-Mellon University, Oct, 1982.
8. Waterman, D. A. & Hayes-Roth, F. (Eds.). *Pattern Directed Inference Systems.* Academic Press, New York, 1978.

[4] We have since implemented the system in OPS5 [1] with a modified conflict resolution to provide the required concurrency.

[5] Our enterprise bears some resemblance to Nau, Kumar and Kanal (1982), but differs in striving for generality in an operational agent, rather than generality in an encompassing analytical scheme.