ABSTRACT: This paper describes an implemented system, PLAND, for discovering substructures in observed action sequences. The goal is to show how a system can learn useful macro-operators by observing a task being performed. An intelligent robot using this system could learn how to perform new tasks by watching tasks being performed by someone else, even if the robot does not possess a complete understanding of the actions being observed.

Macro-operators are discovered within a specific context that provides the types of generalizations allowed in the discovery process and uses the previously proposed macro-operators to build new ones. Background knowledge is used to determine which generalizations are appropriate and to control search. The system can discover syntactic structures (grammars) without background knowledge, but more meaningful and useful structures are discovered when background knowledge is incorporated into the process.

The foundations of PLAND are in similarity-difference-based (SDBL) learning systems that perform conceptual clustering; however, unlike most SDBL systems, a large amount of background knowledge can be incorporated to improve learning effectiveness.
Substructure Discovery of Macro-Operators

Bradley L. Whitehall
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

ABSTRACT

This paper describes an implemented system, PLAND, for discovering substructures in observed action sequences. The goal is to show how a system can learn useful macro-operators by observing a task being performed. An intelligent robot using this system could learn how to perform new tasks by watching tasks being performed by someone else, even if the robot does not possess a complete understanding of the actions being observed.

Macro-operators are discovered within a specific context that provides the types of generalizations allowed in the discovery process and uses the previously proposed macro-operators to build new ones. Background knowledge is used to determine which generalizations are appropriate and to control search. The system can discover syntactic structures (grammars) without background knowledge, but more meaningful and useful structures are discovered when background knowledge is incorporated into the process.

The foundations of PLAND are in similarity-difference-based (SDBL) learning systems that perform conceptual clustering; however, unlike most SDBL systems, a large amount of background knowledge can be incorporated to improve learning effectiveness.

1 INTRODUCTION

The goal of the research presented in this paper is to discover (plausible) structures in observed action sequences. Specifically, the PLAND (PLAN Discovery) system discovers macro-operators (macros) of action subsequences by searching for interesting substructures in observed action traces. By constructing a hierarchical representation from the flat (linear) structure of actions observed, the system gains insight into the interconnections of the actions being observed.

Macro-operators are discovered within a specific context that provides the types of generalizations allowed in the discovery process and uses the previously proposed macro-operators to build new ones. Background knowledge is used to determine which generalizations are appropriate and to control search. The system can discover syntactic structures (grammars) without background knowledge, but more meaningful and useful structures are discovered when background knowledge is incorporated into the process.

The foundations of PLAND are in similarity-difference-based (SDBL) learning systems that perform conceptual clustering; however, unlike most SDBL systems, a large amount of background knowledge can be incorporated to improve learning effectiveness.

2 SUBSTRUCTURE DISCOVERY

A brief description of substructures and substructure discovery is presented in this section; for more detail see [22] and [8]. A substructure is defined as a collection of relations and the nodes associated with those relations. The nodes and relations constitute a connected portion of the structure within a complete event. All the nodes are connected by relations in the graph theoretic sense where the nodes are vertices and the relations are edges.

Discovering structure within given input events requires the use of part-to-whole generalization [3]. In other words, from the observed pieces the system explores how the complete event is best described. There are two types of structure being found by the PLAND system. The linear structure of actions is composed into macros and a hierarchical structure is created when macros use previously defined macro-operators as though they were single actions. These found macros can be used to break the task into manageable chunks and to facilitate the recognition of the problem solver's goals. The system works at discovering the first kind of structure; the hierarchical structure of macros is a byproduct of macros internalizing other macros.

Structure in this system consists of actions and the relationships between them. Logical sequences of actions that perform a
Three types of macros are discovered by the PLAND system: referred to as similarity-based learning (SBL) or empirical. However, these sequences, loops, and conditionals.

Events also play an important role.

Wolff's SNPR system [24] for discovering to allow actions to have associated attributes, a feature that aids the discovery process.

The PLAND system is concerned with discovering possible macros that are known to accomplish some task in the given execution trace, but whose general applicability is as yet unknown. Macro-operators discovered by the system could be passed to an explanation-based learning (EBL) system [5, 12] to determine an explanation based evaluation of the macro's usefulness and to be further generalized. Having PLAND propose discovered macros to an EBL system means that the EBL system is deriving proofs only for macros that have some empirical support.

PLAND is based upon similarity-difference-based learning** (SDBL) systems [6, 7, 19, 21]. As with all similarity-difference-based systems it does not try to prove the validity of the discovered macros. There is a leap of faith in the generalization done in the discovery process. For example, it is possible for the system to classify an anomaly as a macro. But in order for this to happen under typical heuristic biases, the anomaly would need to occur many times. In such a case one must question how irregular the observed actions really are.

The PLAND system takes a single sequence of observed actions as input. From this stream of actions it must discover logical units that can reduce the complexity of the trace. This is similar to the NODDY system of Andreae [1], but that system is given examples of a single iteration through the body of a loop from which it can learn conditionals and the loop structure. PLAND must itself break the action sequence into examples and work with those chunks to discover macros. This complicates the problem because one is never sure that one is working with correct "examples." Wolff's SNPR system [24] for discovering grammars performs some of the functions of PLAND. The main differences being that PLAND discovers loops explicitly where SNPR does not, and that SNPR does not incorporate knowledge to allow actions to have associated attributes, a feature that aids the discovery process.

3.1 Types of Macros Discovered

Three types of macros are discovered by the PLAND system: sequences, loops, and conditionals.

**Similarity-difference-based learning is the same as what was previously referred to as similarity-based learning (SBL) or empirical. However, these systems do not learn by observing only similarities. The differences between events also play an important role.

- Sequences
  - The most basic macro is a simple sequence of steps. A sequence is a block of actions that have been used in many places in the input trace. Actions in a sequence have not occurred consecutively enough times to be considered a loop.
  - Loops
    - Loops are defined as sequences that appear juxtaposed for at least a minimum number (a parameter) of iterations. In the normal meaning of the word, loops have test conditions to stop their execution. PLAND does not determine what those stopping criteria are but learns only the sequence of actions that compose the body of the loop1. As an example, consider an input string of $ABCDCDCEFCDCC$, from which the loop macro, using formal grammar syntax, $(CD)^*$ is discovered. The string can now be described as being generated by $AB(CD)^*BE(CD)^*$.
  - Conditionals
    - The third type of macro found is conditionals. A conditional allows a choice of actions for some particular point in time. PLAND defines conditionals as macros that have more than one choice point within them. A particular choice point is not limited to just two alternatives. For example, if given $ABCDACEABCD$ the system discovers the macro $(A(B+D+E)C)^*$. In a manner analogous to loops, the situations that cause a specific branch of a choice point to be performed are not learned.

These are the three types of macro-operators that the system can discover. By nesting previously discovered macros within other macro-operators the system is able to discover complex relationships between different macros and build up a hierarchical structure of an observed sequence of actions. The algorithm to discover these constructs is a major portion of the knowledge built into the system. Additional heuristics and generalizations done by the system are supplied by background knowledge.

3.2 Top Level Organization

The system works on multiple levels of generalization called contexts. A context contains all the information needed to process a set of actions for a given level of abstraction. This includes the input sequence of actions, the previously discovered macros, agendas, and information about how macros overlap and subsume each other. The system can proceed to a more abstract level by creating a new context in which the actions are generalized. The actions can be generalized by replacing groups of actions with macros or by the use of a fuzzy matching algorithm.

The system can change the level it is working on by replacing the current context. This flexibility is useful if the system is unsure which level of generalization is appropriate for the problem. The system can work on one context for a specified amount of time, then swap contexts and work on a different one.

Another important data structure is an agenda. An agenda indicates where to look for new macros in the given example. There are many agendas competing for processor time, and a simple agenda control system manages their priorities. An agenda

---

1Learning the exit conditions for a looping construct requires a system like BAGGER [15, 11] because exit conditions are reflected in the state of the system but not in the actions performed.
The result of the last context was:

**CONTENTS**

<table>
<thead>
<tr>
<th>COGSAY</th>
<th>NAME</th>
<th>MACROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.0</td>
<td>M1</td>
<td>(&lt;\text{WAKE-UP} \text{ EAT } \text{GOTO-GYM} + (\cdot) \text{ GOTO-WORK} \text{ GOTO-HOME EAT GOTO-BED}&gt;)</td>
</tr>
</tbody>
</table>

Ready to start another cycle.

The result of the last context was:

**CONTENTS**

<table>
<thead>
<tr>
<th>COGSAY</th>
<th>NAME</th>
<th>MACROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.0</td>
<td>M1</td>
<td>(&lt;\text{WAKE-UP} \text{ GET-SNACK} \text{ GOTO BED} [\text{M1]} \text{ SLEEP WALK}&gt;)</td>
</tr>
</tbody>
</table>

All interesting macros were discovered.

This example is finished.

### Figure 1: Student Example

contains information on where in the action sequence the search for a macro is to begin. The previously found macros that can be used in building up the current macro are specified in the agenda. The type of macro wanted is indicated, either a loop or a conditional. Searching directly for sequence macros is not done, but information regarding a possible sequence macro is updated whenever a new macro is discovered. Agendas compete only with other agendas of the same context.

#### 3.3 Use of Background Knowledge

The AI community has recognized that in order to learn substantive concepts a system must possess knowledge about the domain [15, 23]. Some SDBL systems, e.g., [5, 7, 9, 19], have not used background knowledge in a flexible manner to guide the learning process, but rather have used knowledge to control the types of generalizations allowed.

Recently, researchers have incorporated more background knowledge into the systems to help with the discovery process [10, 13, 20]. PLAND continues this trend. This section describes the multiple ways in which domain specific knowledge is used by the system.

To help explain the use of background knowledge in the system, a simple example is presented. The system is capable of dealing with far more complex examples. Background knowledge is expressed by rules and the system does backward chaining through the rules to obtain an answer to a query about the current processing. For this simple example the input (as shown in figure 1) consists of a week's worth of actions performed by a hypothetical (simple minded) graduate student. The goal of the system is to discover a macro that will define a typical day in the life of this student. The background knowledge specifies that going to the gym is optional and that a day must start by waking up in the morning, end by going to bed, and a student must also get some work done during the day. A conceptual version of the rule for the constraints on the desired macro is given in figure 2.1

![Figure 2: Background Knowledge Rule](image_url)

where MATCH is a function that makes use of a fuzzy matching algorithm to determine whether two actions are equivalent. MATCH does not require that two actions be "EQ" equal. The fuzzy matcher will enable predefined patterns to determine how a match for particular items may occur. Simple things such as allowing numeric values to fall within a range, ignoring parts of an action, and forcing strict equivalence are built into the matching routines. More complex matching, such as traversing defined ISA links for actions, can be specified by defining LISP code. SOME-ACTION just checks that the first argument MATCHes at least one action in the second argument (a macro).

Although simple names or letters are used for actions here, it should be clear that exact syntactic matches are not required. Thus a loop indicated by \(X^*\) could actually represent a loop where an occurrence is in a list of actions from the set \(X_1, X_2, X_3, \ldots, X_n\), where each \(X_i\) is a known way to achieve \(X\). \(X_i\) need not be a primitive action, but could be a macro that describes very complex actions whose result is known. For example, in figure 1 there are many ways a student can eat: fix a bowl of cereal, grab some fast food, fix a meal at home, etc., thus the reason for different subscripts.

In the example, PLAND finds a macro for the student which is wake up, optionally go to the gym, work, optionally go to the gym, head home, eat, and go to bed. Although the output of the system in figure 1 makes the problem look simple, it was not solvable without the domain knowledge in figure 2.1.

PLAND uses background knowledge in three distinct ways. Figure 3 presents an outline of the PLAND algorithm showing the different levels of background knowledge (BK) used. At the highest level, the background knowledge acts as meta-knowledge. The system works on levels of generalization called contexts. The system queries the knowledge base to determine what the current working context should be. The current context is retained if there are good opportunities for discovering new macros within it. If a different context is suggested, the knowledge base returns the new context to be used. At this level, the knowledge is used to control the level of generalization of the action steps. Thus the system can process the input at a higher level of abstraction after some macros have been found. If the search at the higher level is fruitless then the system can return to the more detailed level. This is a powerful problem solving mechanism because it

---

1Due to the amount of regularity in the example, a large number of macros were created. The handling of these macros shows the discovery process. The simple condition of forcing days to start by waking up and end by going to bed is enough to prune the number of possible macros so that the system can function. This example shows that the addition of simple knowledge can greatly improve the prospects of discovering the correct solution.
allows the system to pursue many possible goals. If the system
discovers a macro that indicates a certain environment is present,
then it can create a context that generalizes some of these actions
to help confirm that notion. This type of hypothesis formation
is rare in SDBL systems.

When the top level decides to change contexts, background
knowledge is consulted as to what type of generalizations should
be performed on the actions (via the fuzzy matcher) of the cur-
rent context. Consultation with the background knowledge in
this fashion allows the system to discover macros that would be
impossible to discover otherwise. A system without knowledge
specific to the domain could not make logical guesses at which
of the many possible generalizations has meaning to the problem
at hand.

For the example given earlier, no such knowledge exists. But
for more complex problems the actions could be interpreted in nu-
merous ways. Knowledge of these different interpretations would
allow the creation of multiple contexts where the type of matches
allowed for the actions was different. The system could then let
each context run a few selected agendas and continue with the
context that had found the best macros (as measured by cognitive
savings or other heuristic criteria).

The background knowledge used at the intermediate level
helps direct the searching process for the macros through agenda
control. Before any agenda is given control, the background
knowledge is queried to approve the agenda's applicability. If
the knowledge indicates that macros are not to be searched be-
ond a certain point in the input sequence, then those agendas
can be pruned. Information contained in the agenda could sig-
ify that the agenda should not be performed. The knowledge
used in this method can save substantial amounts of processing
and significantly prune the search tree.

If the system had a large number of actions it observed while
the student was sleeping, such as snoring, rolling over, etc., then
knowledge could eliminate searching this area. For example, a
rule might reject an agenda entry for finding loops that start in
a position between an action of going to bed and waking up.

At the lowest level, background knowledge is used to control
the macros allowed by the system. After finding the sequence
of actions for a macro-operator, a query to background knowl-
edge determines if the new sequence meets any simple criteria
expressed for macros. With simple rules for checking macro se-
quences, like the rule presented in figure 2, the system is able
to eliminate the generation of useless macros. This saves stor-
age and time. Knowledge at the lowest level can rid the system
of macros having low utility. Although this seems trivial, this
level of control determines whether a system finds a solution or
runs out of space and/or time. Control like this is missing in
many SDBL systems. Those systems can only make guesses at
what is useful, much as this system does when no background
knowledge is present. The implication that nothing valid can be
done without background knowledge is not intended. In fact this
system can find useful results even when no knowledge is given.
In these cases the system acts like a finite state machine builder.
The input actions are treated as the letters of a string. Then
the discovered macros act as formal grammars defining portions
of the input string and together constitute a generalized regular
grammar that can generate the input and other "similar" strings.

3.4 Cognitive Savings

The utility of a discovered macro-operator is measured by cog-
nitive savings. PLAND uses the cognitive savings values to de-
termine which of two substrutures has the best potential for
extension and applicability. The intent of this value is to capture
the mental savings one gains by working with the substructure
instead of the primitive actions that compose it. A simple for-
uila for cognitive savings is

\[ \text{cognitive savings} = \left( \frac{\text{number of structure occurrences} - 1}{\text{size of structure}} \right) \]

where size of structure can be defined as number of nodes, num-
ber of relations, or some other formula using the components of
the structure. This formula incorporates components of Wolff's
compression principles [24]. A macro that can chunk a large
number of primitive actions and occurs many times is very use-
ful. There is a trade off when a macro expansion causes some
prior occurrences to cease to be covered. The exact threshold for
this cut off point is domain dependent.

4 MACRO DISCOVERY DETAILS

There is only room here to describe at the highest level how
loops and conditionals are discovered. Details of the processes
are explained in [22].

The most basic concept underlying loop macro discovery is if
a sequence occurs many times, with one occurrence following the
other, then reduce the sequence. This is a simple concept that
has been used before to reduce given sequences [14, 18]. But
even this simple concept is difficult to implement in practice.
Finding answers to simple questions can explode in exponential
time when the examples are not explicitly given. Such questions
as "where does the loop body begin?," "how long is the sequence
of the body of the loop?," and "does the action that begins the
loop also occur within the loop body (thus not always indicating
a new iteration)?" are difficult questions to answer. The loop
discovery module in PLAND expects parameters in the agenda to guide the search for answers to these questions.

This section describes in a schematic way the algorithm used for loop discovery. An association list (a-list) is created for each type of action in the input sequence. The car of the list is the type of action and the rest is an ordered list of positions (index number) of that action type in the input sequence. Consecutive occurrences of the action, represented by adjacent numbers in the a-list, indicate the (possible) start of a new loop iteration. The system starts with a particular action and tries to find loops that begin with that action. Two occurrences of an action are taken as the delimiters for the loop. The system then tries to “fill in” the body of the loop by determining if the actions that follow the first instance of the proposed loop head also follow the second proposed loop head. The actions between two possible iterations are tested incrementally. As long as the sequence generated thus far does not completely describe the actions between two starting positions of the proposed loop then the process continues. The sequence is grown by all possible macros in addition to the single actions. This does not explode, as there is a fixed number of macros that are usable (as defined by the agenda), and each iteration of the proposed loop body acts as a constraint on what is allowed. If there is a sequence that occurs between the loop starting actions then that sequence is the loop body.

The discovery of conditionals is more complex than the discovery of loops. The problem with discovering conditionals is that anything can be made optional. In the extreme case a sequence could be described by a loop of length one with the body consisting of a single conditional for all possible actions. The algorithm used by PLAND avoids this pitfall by requiring that all conditionals have a base or key point that cannot be part of a choice set. The basic principle in discovering conditionals is find actions that occur on fixed intervals, then make conditionals out of what lies between these key points.

The first step in the discovery of conditionals is to build a difference array. The difference computed is the number of actions between two sequential occurrences of an action type. Next an array is built that indicates the number of consecutive differences of equal spacing for each difference array. This is done so that in the next step the action with the largest number of iterations (key action) can be found. From the explanation thus far, it should be clear that the conditionals discovered must be part of a loop. It is the repetition of actions at a fixed distance from each other in the sequence that allows the conditionals to be discovered. The actions do not have to be a fixed distance from each other in the primitive version of the observed trace. But they must be a fixed “action” distance apart which means a variable length macro (such as a loop) could be used in the conditional. Filling in the actions around the key is the fourth step. This is where the actual choice sets get constructed. The last step of the algorithm is to convert the best descriptions of the conditionals into macro structures and find the other occurrences of the conditional. If there are ties for the best conditional then all of them are returned.

5 ANOTHER EXAMPLE

The example, whose run is shown in figure 4, demonstrates that conditionals may be found with embedded macros. The system can discover conditionals and loops nested to an arbitrary depth. For brevity of presentation no background knowledge was incorporated in this example and the actions are letters. When no background knowledge is used the system finds regular expressions that define chunks of the input string. The first macro found by the system, M1, is a conditional but it does not yet have the embedded loop macro. This is because the loop macro for X is not been discovered at this point; M2 is discovered next. Now the system finds the better conditional, M3, which expresses the complete string, \((X^* + B) A\)^n. The system continues to find other macros which are less interesting as indicated by the cognitive savings value. Notice that the macro numbers are not sequential. Nonsequential macro numbers indicate the system has discovered macros that are subsumed by previously defined macros. The subsumed macros are noted and not used further by the system. In this case, the best macro, M3, defines the whole input sequence.

6 CONCLUSION

The PLAND system demonstrates substructure discovery is useful for finding macro-operators. Substructure discovery allows a system to learn more complex relationships than are possible with attribute only systems. The hierarchical structure of an observed sequence of actions can be constructed without complete background knowledge or explicitly stated examples. Background knowledge can be incorporated to allow the system to discover more appropriate macro-operators with less effort.

PLAND is not a grammar induction system. It is able to induce grammars to define the observed sequences but it goes for
beyond capabilities of such systems. It is more powerful because it discovers loops explicitly and uses background knowledge in combination with the fuzzy matcher to allow the system to discover more than just syntactic structures.

The system is novel in the area of macro-operator processing because it does not perform problem solving and store the results. The system discovers the macro-operators by passively observing the actions of another agent performing a task and inducing the structure of the macros.

7 ACKNOWLEDGMENTS

This work benefited from discussions with Robert Stepp, Larry Holder, Bob Reinke, Bharat Rao, and Diane Cook.

REFERENCES


