December 1998

The Dual Algorithm

Jerome Heath

Follow this and additional works at: http://aisel.aisnet.org/amcis1998

Recommended Citation
http://aisel.aisnet.org/amcis1998/57

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 1998 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
The Dual Algorithm

Jerome Heath

Introduction

This project was developed out of philosophical concerns rather than systems analysis. Wittgenstein said that the foundations of the language game cannot be part of the language. I reinterpret this comment as dividing mental activity into two algorithms. The foreground includes the language game. The illusive background is more abstract than language. I posit that the foreground includes the language game of Wittgenstein. I also posit that the foreground includes picture or graphic references that are more abstract than language, but not as abstract as the true background. With Wittgenstein the background (foundations) is more action or “doing” oriented. The foreground of pictures and language is more structure oriented.

The foreground algorithm is structured in limited ways so that its operation can be "visible." The data in the foreground follows predictable rules as to relationships so that there is a visibility to the processes. The data of the foreground is particularly significant in this characteristic, that is, strong stable relations that make it easy to interpret (be visible), and the processes that occur are relatively obvious.

The background is more abstract and elusive. The data, if it can be called that, is a basis for relationships that are the important part of the algorithm. The relationships are changeable as distinct from the clearly understood and stable relationships of the foreground. If a person could “view” this mental background, it would give very little information about its workings. The abstractness of the background must be turned into some foreground “data” in order to be understood at all.

The problem I chose to work on was the block problem. This problem requires artificial intelligence, but is workable with a small program. I believe the algorithm could be expanded to other problems with suitable changes in operating details, but most of these problems would require a much larger program. I hope in the future to work on such an expanded program.

I chose to work on the problem that gives GPS so much difficulty to show that this algorithm, with a very limited search, of short duration, can solve the problem:

```
A
C   B
B   A   C
```

Another problem was tried, which is actually a little harder for our algorithm:

```
C   A
B   B
A   C
```

Goals of the Project

The main goal of the project was to develop a program in which there was a foreground algorithm, of obvious structure and processing, that communicated with a background algorithm that was more abstract. The background algorithm should be the harder algorithm to develop. The background algorithm of the brain is believed by some to resemble a neural network, although that is not important to the theory. The important part of the background is that it have a number of different nodes which could be referenced separately and independently. The background also needs to show an abstract reference method that could solve the problem in abstraction. I posit that it should also be action oriented.

I expected to see the possibility of planning in the algorithm. I also expected to see learning. In addition I was looking for areas in the program, particularly in the communication between the two algorithms, where other artificial intelligence processes could be included in the future.

Of course, one goal of the project is to solve the block problem. The program needs to have a methodology that will find a solution to the problem at hand. But this solution cannot be hard-wired. I was looking for an algorithm that will find a solution in a more abstract way.

The Details of How the Project Works

The project foreground algorithm first establishes a start and finish state. It then sets up the pretraining array. After this, the foreground makes two pointers that are to carry the problem into the background. Then the background is called.

The background algorithm, when called, determines the next activity that is "in context" and tries the process that it refers to. Winding out of the algorithm occurs when done is returned or when all the activities that are “in context” in the upper (first) node have been tried and failed.

See Figure 1 for the block diagram. Note, that the foreground and background are the main modules of the program. The heading module, in the figure, is to provide a group of headings to show the user what the activity list refers to. The planning module sets the activity pretraining list to the planning level based on the start state and the end state.
The write_list (write the whole list) module sends each of the pointers tried, in a linked list, to the write_out (write one line) module which writes out one pointer tried to the screen. The pointer tried is actually a state of the system that is then saved in a linked list. The write_out module is also called, independently, in order to write out one line from a given pointer that is tried. The set_error and error_node modules are used to adjust the pretraining levels of the activity list depending on circumstances.

The duplicate module makes a copy of each pointer tried in order to have a separate pointer area to be processed. Changing this storage area does not affect the original copy. The pretrain module is used to set the activity list when a new node and activity list are started.

The test_x_off_y and test_x_on_y modules test to see if the preconditions, of the called for procedure, are viable. If the preconditions are viable, each pointer to be tried is changed to the test conditions.

The do_check module is used to check to see if the problem is solved yet, and if not, if the test creates a state that already existed. If the solution has been reached, do_check returns done. If the test state has already exited, do_check returns an error. Otherwise do_check returns OK.

### Problems Solved in the Development Process

Starting with philosophical considerations is not a standard way of developing a computer program. The first trials did not look like the system could accomplish something like the desired process. In spite of these concerns the changes that were made to correct early bugs were done while retaining the philosophical base. In the end the process was close enough to the philosophically desired concepts that I consider that this as a reasonable test of the theory. Perhaps no computer program can completely accomplish the processing our mind does internally, because there seems to be so much parallel processing.

There are two characteristics about the background algorithm that are important. The algorithm should be able to try different paths to solutions, but with some control over how it tries various paths. The algorithm should also result in action occurring, on finding a particular activity.

A particular difficulty with using the dual algorithm is the foreground structure or the "visualization" process. For each problem there are unique characteristics that can be used to show the state of the problem. Keeping those characteristics simple but effective is an important part of the dual algorithm. If the state characteristics, that are used, do not define the problem, then the solution will not be real. If the state is overdetermined there may be incongruous results or weird intermediate stages. The use of a_on, b_on, c_on did give all the information needed for the block problem, and thus, was used to convey the place in the solution that the system has reached.

Leaving the background when done, or when there are no path choices left, was difficult to program correctly. There is a kind maze to the loops used to provide variability to the background. There was also a big problem with the "tail" of a solution that is being discarded when the system is backing up. The tail had to be killed at just the right moment because if this tail remained the next "try" would end up pointing at itself. This caused an endless loop when presenting the forward algorithm lists at the end of the program.

I, of course, had to be careful that other pointers were doing what I wanted with them. One interesting process was the concept of duplicating a pointer structure, which allowed for two structures that were the same but didn’t actually point at the same configuration. This was done so that the system could alter one of these pointers and use the other to compare with, as the start state.

See Figure 2 for a picture of the run process.

### AI Techniques

The first AI technique I used was a kind of minimum planning. The way the algorithm worked, the minimum planning got us close enough so the algorithm could rapidly determine the actual moves needed. Only certain moves are allowed, in any given problem, so the algorithm had to determine what the order of those moves was to be. The algorithm was actually quite fast at determining that order. This is comparable to the GPS-search, of Norvig (1992), which seems to need to search every possibility in order to work.

There is an incremental level of learning. When the system puts 99 in an activity array location it has learned that this move was not proper. This is not used effectively in the program, but could help to speed the solution process if this learning was interpreted and used in other ways as the system goes.
through different nodes. Once a process received a 99 it should only be changed back to a 1 after backing up past a previous process. This would be possible but tricky to program. This would speed up the process of solving the problem.

The forward algorithm is meant to do something related to intelligence, which I call "visualization." The algorithm can "see" the process of a series of states. This can act as large scale learning that far exceeds the power of the network learning represented by the 99's. The present algorithm can picture the solutions by using this process to save successful run values. In the future a new start/finish state could be compared with these charts and the information used to solve a block puzzle.

I also see an advantage of using this foreground information during the solving of a puzzle. The process would involve an interim means-ends test like the planning procedure but which occurs during the running of the background. The pretraining settings would be changed with successful moves. The difficult part would be the necessity of "unpretraining" during a backing up process. The information is there, but the processing is tricky.

Just using the information that a given process was accomplished would also speed things up. Reversing this information might not be necessary if the system backed up here because backing up means that process did not work. Of course the process might work at a much later time, if something else is done first, or if the system backed up past a previous process. All this would have to be considered in our pretraining rules.

**Things To Do If There Were More Time**

Presently, the algorithm can only do blocks, even though it does those consistently well. Processing of other problems would require some differences, particularly in the foreground. The actual action done by the background would, of course, be different in another problem area. I am hopeful that in most problems the actual activity list (the way of choosing which process is done next) would be much the same.

I was very satisfied with the algorithm as it appears in this program. If I had more time I would try a context problem. It is hoped that I can find a problem that actually had more than one context so I could study the context area. If the way context works is consistent from one problem to another, solving one "context" problem would give us a good idea how context worked. Of course, each problem would require different actions and a different "visualization."

One suggestion for trying this would be a simple sentence parser, using very simple rules to test the idea. I consider something like Figure 3. Another suggestion would be to do something in category recognition. I am interested in a project like this because this could be part of my dissertation. If I can prove some more significant parts of the dual algorithm and its power, I have a start in the right direction.

![Figure 3. Parse.](image)

The addition of context to the problem might make the algorithm more powerful, particularly for problems that have hierarchy or levels of processing. I think the context would be like a policeman to the activity list, which would limit choice to different levels of the activity list with different context.

Another possible problem area for this algorithm is statistical simulation. If the activities are distributed in value by probability rules and the context is chosen as a random number, simulation of probability processes could be handled by this algorithm.

If each activity has the probability given in the table, the activity number would be according to the last column of the table.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity 1</td>
<td>2%</td>
</tr>
<tr>
<td>activity 2</td>
<td>5%</td>
</tr>
<tr>
<td>activity 3</td>
<td>5%</td>
</tr>
<tr>
<td>activity 4</td>
<td>8%</td>
</tr>
<tr>
<td>activity 5</td>
<td>10%</td>
</tr>
<tr>
<td>activity 6</td>
<td>10%</td>
</tr>
<tr>
<td>activity 7</td>
<td>15%</td>
</tr>
<tr>
<td>activity 8</td>
<td>15%</td>
</tr>
<tr>
<td>activity 9</td>
<td>15%</td>
</tr>
<tr>
<td>activity 10</td>
<td>15%</td>
</tr>
</tbody>
</table>

The condition of choosing an activity is that context (chosen randomly) is greater than the activity. Learning is more difficult to program here. On finding that an activity is not feasible, the activity numbers will need to distribute the 100% of probability among the remaining activities.

**Analysis**

The program does some interesting things in solving these simple blocks problems. I can see the possibilities of doing other things with the algorithm because there is information available from one or the other algorithm that could be used to improve the solution to a problem.

The fact that there is some special information there, is born out by the fact that the program solves each problem relatively fast and looks like it could solve any problem in about the same time and effort. This compares with the GPS that cannot solve the first problem of this program.

This algorithm is not as fast as running the solution after a detailed planning program (like NOAH) that would specify the order of processes quite completely. But the time spent on planning before running, may be more than the time spent by this algorithm in the minimal searching of the problem space. An advantage is gained because the algorithm's search is limited by the initial planning.

There is a possibility of a faster learning process with the dual algorithm. The incremental learning of the background is slow and the meaning is difficult (but not impossible) to convey to other parts of the problem. The foreground algorithm can support
a more over-the-whole-process learning. That is, the saved runs can show how the system can get from some start state to another state. If the problem was solved before, the results of the search of the hold array, could be used directly without going through the background.

There is also a kind of flexibility in this combination. Although some of this flexibility causes problems with ironing out the details of the program, the flexibility indicates the availability of methods and information to handle a variety of problems.

In an all ways search, the search takes a considerable amount of time and processing. The background algorithm of this program does a shortened version of a solution search. This shortened version is the result of planning. The shortness means it is not cumbersome or greatly time consuming. It also appears to handle any problem in "about" the same way, so the start and end state seem irrelevant as far as difficulty is concerned.

References