Cog Sci 242 **Lab 2: Problem-Solving & Search** *Fall 2018*

Problem-Solving

I have set up four puzzle stations around the room:

Crypt-arithmetic, *Entanglement Puzzles*, *Reassembly*, and *Walking Through Paper*.

Each station has instructions.

As a pair, you should attempt at least <u>two</u> of these. For one of the puzzles you choose, one of you should be the solver and the other the observer. For the other puzzle you should switch roles.

Be sure to include the Crypt-arithmetic puzzle among the puzzles you try.

(If you are familiar with one of these puzzles, you should take the observer role for that one and be careful not to drop hints about its solution!)

It is not necessary to find the solution to a puzzle – but the solver should try their best to do so! Use your time wisely in order to be able to fit at least two puzzles in. Do NOT look on the internet for solutions!

You may ask me for clarification about the goal and rules for each puzzle, but I will not be giving your solution hints.

<u>Protocol Analysis</u>

The process of observing someone solving a problem and analyzing their problem-solving behavior is called protocol analysis.

Your goal in this protocol analysis is to gain some insight into how people solve problems of the sort reprsented by these puzzles. (Of course, you can also draw on your own personal experience.)

Roles:

- **Solver**: try to articulate verbally what you are thinking as you are working on solving the problem or puzzle.

- **Observer**: record what the solver says and does. Only ask questions if the solver isn't giving your verbal information about what they are thinking.

Cog Sci 242 **Protocol Analysis – Data Form** *Fall 2018*

Puzzle: _____ Date: _____

Solver: _____ Observer: _____

Solver Does	**Solver Says ....**

Solver Does	**Solver Says ....**

Cog Sci 242 **Entanglement Puzzles** *Fall 2018*

This puzzle station has three puzzles. You cannot cut or force any parts in solving them.

1) *Ring on a "Post"*

- Remove the metal ring from the "post," i.e. so that you have the ring in one hand and the rest of the puzzle in the other hand, completely separated. Then put restore the puzzle to its original configuration.

There are two different versions of this puzzle.

2) *Two on a Loop*

- This puzzle has two cord loops and two metal rings. Your goal is to get both rings on the same loop, and then to restore the puzzle to its original configuration.

3) *Tied Together*

Call me if you are ready for this puzzle.

This station has two objects, one of which is a solid sphere and the other of which is a many-pointed solid.

Both objects fall apart into a number of pieces.

Your goals is to take one of the objects apart and put it back together again.

This crypt-arithmetic problem was used by Newell and Simon in their original work using protocol analysis to study problem-solving[1].

Find an assignment of the integers 0 – 9 to the letters in the words

DONALD, GERALD and ROBERT such that:

each *integer* is assigned to a unique *letter*;

each *letter* is assigned to a unique *integer*;

and when the letters have been replaced by their assigned numbers, the following equation is satisfied:

DONALD + GERALD = ROBERT;

Note: 5 is assigned to the letter D.

[1] Newell, Allen, "Studies in problem solving; subject 3 on the crypt-arithmetic task Donald + Gerald = Robert" (1967). *Computer Science Department, Carnegie-Mellon University*. Paper 1731. http://repository.cmu.edu/compsci/1731

Cut a hole in this sheet of paper big enough for you to walk through. You may make *two* cuts, and the resulting hole must consist of an unbroken circle of paper. No taping allowed!

Analyzing Your Data

Having collected your saying/doing data about problem-solving, study it together with your partner.

This analysis is qualitative.

You are seeking to answer two questions:

- What types of **strategies** (ways of attacking the problem) were used?

- What types of **errors** were made, and in what contexts?

You should give specific evidence from your problem-solving observations to support your answers.

Write your answers for the two problems you and your partner solved in a Word document, save it as a PDF file, and **upload it to the Moodle site** in the *Lab 2 - Protocol Analysis* upload folder. In each case, clearly identify the problem solved, the person solving it, and the answers to the two questions. **Do this now.** Be sure that everyone in your group is in agreement with what is written, or that any disagreements are documented in what you write. Be sure the initials of you and your partner are at the end of the file name, eg. **Lab 2 Protocol Analysis_AK_KC**.

In their pioneering work, Newell and Simon thought of human problem-solving as goal-oriented behavior, moving from an initial state to a solution state.

The term **state** refers to the information needed to characterize a given step in the problem-solving process.

To illustrate this characterization of problem-solving as a sequence of **state transitions**, we will consider the crypt-arithmetic problem you worked on earlier in the lab.

Find an assignment of the integers 0 – 9 to the letters in the words

DONALD, GERALD and ROBERT such that:

each *integer* is assigned to a unique *letter*;

each *letter* is assigned to a unique *integer*;

and when the letters have been replaced by their assigned numbers, the following equation is satisfied:

DONALD + GERALD = ROBERT;

Note: 5 is assigned to the letter D.

From this problem description, the current state of problem-solving can be specified by stating a proposed assignment of integers to letters satisfying the following constraints:

- Each of the nine integers 0, 1, 2, 3, 4, 6, 7, 8, 9 is assigned to a unique letter among the nine letters A, B, E, G, L, N, O, R, T;
- and each of these nine letters is uniquely assigned to one of these nine integers.

(*Note*: We already know that the integer 5 has been assigned to the letter D.)

If we use the letters as headings, then the state can be economically represented as an ordered sequence of integers.

Look at my example, then write an integer sequence corresponding to another state.

My example:	A	B	E	G	L	N	O	R	T
	0	9	8	1	2	7	6	3	4

Your example:	A	B	E	G	L	N	O	R	T

We don't yet know which state corresponds to a solution (if we did, then the problem would be solved). But we do have a **certificate** for a problem solution – a method of determining whether or not a given state is a solution:

> "When the letters have been replaced by their assigned numbers,
> the following equation is satisfied:
>
> DONALD + GERALD = ROBERT."

To illustrate, I'll use this certificate to test whether or not my example state is a solution:

My example:

A	B	E	G	L	N	O	R	T	, together with the constraint	D
0	9	8	1	2	7	6	3	4		5

implies

567025 + 183025 = 369834.

Is this true? That is, does my example state satisfy the solution certificate?[2]

Now check whether or not *your* example state satisfies the solution certificate:

The problem statement *doesn't* specify an **initial state**, so it doesn't matter which state we begin with. (We'll decide on the initial state later.)

Finally, note that provided we keep the letter heading order the same (for example, in alphabetical order), the state can simply be specified as a *permutation* (possible re-ordering) of this sequence of integers:

0 1 2 3 4 6 7 8 9

WE WILL THEREFORE ASSUME THAT THE LETTER HEADINGS FOR THE STATE ARE IN ALPHABETICAL ORDER.

---

[2] **No**, since 567025 + 183025 = 750050, not 369834.

Programming a Problem Solver

Computational Theory

If you consider "yourself solving this cryptarithmetic problem" as an information processing system, then, following Marr's theory of analyzing information systems, the problem statement recast as a **state transition problem:**

- definition of a state
- an initial state
- rules stating how to transition from state to state
- solution certificate

constitutes a *computational theory* for the system.

We have discussed everything we need at this level of explanation except the transition rules. No such rules were given explicitly in the original problem statement, so we'll simply add the following pragmatic rule:

"Don't check a state that you have already checked."

Representation and Algorithm

We actually have already started work on this level of explanation in our decision to **represent** a state as a *permutation* (possible reordering) of the integers

0 1 2 3 4 6 7 8 9.

Given this representation, *one possible* **algorithm** to solve this problem would be:

1: Generate a list of all possible permutations of

0 1 2 3 4 6 7 8 9.

2: Set the current state to be the first permutation in this list.

3: Check whether the current state satisfies the solution certificate.

If so, return the current state as the solution.

Otherwise, set the next state in the list to be the current state,
and return to step 2.

CHECK POINT:

Discuss with your partner(s) whether or not you are convinced that the algorithm described is guaranteed to find a solution to the problem, if it exists.

When you have settled this matter between you (one way or another), call me over to check your reasoning.

IMPORTANT

This is *an* algorithm for solving the problem, but we are not claiming that this is necessarily *the* algorithm a human would use to solve the problem.

That is, we have not taken into account the data you collected from your protocol analysis about the ways humans would solve the problem.

At this point, we are in a gray area between human cognitive modeling (trying to account for human cognition) and **artificial intelligence** (whose goal is to engineer systems that can solve problems human intelligence can solve – but not necessarily in the same way).

Implementation

According to Marr, the last level of explanation of an information processing system is implementation. Here we will not be concerned with neural implementation, but rather with *software implementation* using MATLAB.

At this level, we are concerned with answering "how" questions whose answers depend on specific features of the MATLAB language. The "how" questions correspond to features of our representation and algorithm level of description.

How can we represent a permutation of 0 1 2 3 4 6 7 8 9 in MATLAB?

- This is most easily done with a **row vector**

  v = [0, 1, 2, 3, 4, 6, 7, 8, 9];

  (or whatever the particular permutation happens to be).

How can MATLAB generate a list of *all possible* permutations of the row vector

[0, 1, 2, 3, 4, 6, 7, 8, 9] ?

One of the great advantages of choosing MATLAB as a programming language is that it has functions to do all sorts of mathematical things. In fact, there is a MATLAB function **perms** that solves just this problem:

```
v = [ 0, 1, 2, 3, 4, 6, 7, 8, 9 ];

States = perms( v );
```

States is actually a matrix (array). Each row of the matrix has 9 entries – so the matrix has 9 columns. Rows & columns for matrices are indexed beginning with "1."

States has 9! = 9×8×7×6×5×4×3×2×1 (read "9 factorial") rows,
one for each possible permutation of this vector.[3]

How can a given row of the *States* matrix be assigned to the current state?

Assume we have the row index of the vector we want stored in the variable *k*. Then

```
current_state = States( k, : );
```

makes the assignment. "States(k, :)" means "select row *k*, all (:) columns" of *States*.

How can MATLAB check whether a given state vector satisfies the solution certificate?

We will return to this sub-problem in a moment, when you will write a MATLAB function *Lab2_CheckSolution* to accomplish this using the following syntax:

```
solution_found = Lab2_CheckSolution( current_state );
```

It will output a value of **true** to the variable *solution_found* if the current state is a solution, and a value **false** otherwise.

The input to this function is the permutation vector assigned to *current_state*.

---

[3] See MATLAB documentation for **perms** at http://www.mathworks.com/help/matlab/ref/perms.html. The rows in *States* are ordered according to something called *reverse lexicographic order*, a detail which need not concern us in this algorithm.

How can we return the program to Step 2 if a solution hasn't been found?

This is done with a "f**or** loop" and an "**if ... else ...** block."

The syntax for the **for** loop specifies the range of row index values that may still need to be examined. If we are unlucky, then *all* the rows of *States* will have to be examined until we find a solution, so we need to know how many rows there are.

There are as many rows in *States* as there are permutations of a vector with 9 distinct elements. To count this number of permutations, we reason:

We have 9 choices for the element to use in the first position.
We have 8 remaining choices of element to use in the second position.
We have 7 remaining choices of element to use in the third position.
⋮ ⋮
We have 2 remaining choices of element to use in the eighth position.
We have 1 remaining choices of element to use in the ninth position.

There are as many permutations as there are distinct sets of these choices[4], namely

$$9 \times 8 \times 7 \times \cdots \times 2 \times 1 = 9! \qquad \text{(read "9 factorial").}$$

MATLAB has a built-in function for computing factorials; we calculate

last_row_index = **factorial**(9);

The role of the **if ... else ...** block is to distinguish between the actions that will be taken when a solution as been found and the actions that will be taken when a solution hasn't been found.

- If a solution has been found, we want to output it and exit the function.
- If a solution hasn't been found, we want to check the next row of *States*.

So we write:

```
if solution_found
        solution = current_state;
        return;
else
        current_state = States(k, :);
        solution_found = Lab2_CheckSolution( current_state );
end
```

[4] 9! = 362,880

The *indenting* and the *keywords* **if**, **else**, and **end** are all important here.

The function **return** terminates the program, returning control to any function which may have called it. (However, we'll just be running the function implementing this program on its own.)

Putting it altogether, we get the following MATLAB function. (Note that I have added comments that are similar to the original English description of the algorithm.)

```
function solution = Lab2_CryptArithmetic()       % No input for this function.

% This function solves the crypt-arithmetic problem from Cog Sci 241, Lab 2.
% It calls the user-defined function Lab2_CheckSolution.
% The solution is a permutation of the vector [0,1,2,3,4,6,7,8,9].
% See the Lab 2 handout for information on how to interpret the solution.
%
% Author: Alan P. Knoerr          Last Update: September 8, 2015

v = [0,1,2,3,4,6,7,8,9];
States = perms(v);                % Find all permutations of vector v.
k = 1;                            % Initialize the row index for States.
last_row_index = factorial(9);    % Calculate the number of rows in States.

current_state = States(k,:);
solution_found = Lab2_CheckSolution( current_state );   % Check initial state.

for k = 2:last_row_index       % Examine all remaining permutations, if needed.
  if solution_found
        solution = current_state;   % Return solution when it is found,
        return;                     % then exit the function.
  else
     current_state = States(k,:);
     solution_found = Lab2_CheckSolution( current_state );
  end                              % End of the if block.
end                                % End of the for loop.

end                                % End of the function.
```

CHECK POINT:

Call me over if there is anything you don't understand about the function *Lab2_CryptArithmetic.*

I have placed this function in the *S:\Cognitive Science\MATLAB\Cog Sci 242* directory. You can *read* it in MATLAB using the **type** command, but you can't run it yet.

Explain why *Lab2_CryptArithmetic* won't work yet! Call me over if you have a question about this.

The issue, of course, is that the function *Lab2_CheckSolution* (which *Lab2_Cryptarithmetic* needs) doesn't exist yet! You need to write that function now. To do so, you need to know more about MATLAB.

More MATLAB

Functions and Scripts

Last week you worked with scripts, which are simply lists of commands you would otherwise execute in the workspace.

A MATLAB *function* is similar to a script, but differs in several ways.

- All variables are *local* in a function, unless declared otherwise.
- Functions generally have *explicit inputs and outputs*.
- Functions often have *control statements* that allow other statements to be chosen or reused.

MATLAB programming typically involves writing functions rather than scripts.

Programming the Function *Lab2_CheckSolution*

Computational Theory

From our previous work, we know that this function must have the syntax:

solution_found = **Lab2_CheckSolution**(current_state);

It must assign a value of **true** to the *output* variable *solution_found* if the current state is a solution, and a value **false** otherwise.

The current state is a solution if, when we replace the letters

A B E G L N O R T

with the integer in the corresponding position of the vector *current_state*, and also replace the letter D with the integer 5, the following equation is satisfied:

DONALD + GERALD = ROBERT

The *input* to this function is the permutation vector assigned as the current state.

This specification of the function constitutes a computational theory for it.

Representation and Algorithm

We already know that *current_state* is a vector of length 9 whose entries are integers.

Working with your partner, figure out an algorithm that uses the information in current state to check the solution certificate for this problem (see the computational theory)[5].

Specify the steps of your algorithm in English, or a combination of English and symbols (which may include MATLAB if you are comfortable with it).

[5] *Hint*: Remember that the numeral 342 stands for $3 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$.

CHECK POINT:

Call me over to discuss your algorithm with you before you proceed.

Implementation

Figure out how to implement each step of your algorithm using MATLAB.

If you have any questions about MATLAB, call me over for assistance.

When this work is done, you should be able to write your function, with comments, using my listing for *Lab2_CryptArithmetic* as a guide.

Now you need to type your function program into a file that MATLAB can read.

Start MATLAB if you haven't already done so.

Edit a new M-file with

function solution_found = **Lab2_CheckSolution**(current_state)

as the first line.

Type in the rest of your function, **with appropriate comments**.

Test it in the command window by assigning an appropriate permulation vector to *current_state* and verifying that it returns the appropriate truth value (as "0" or "1").

When you are satisfied it is working, run *Lab2_CryptArithmetic* as

>> solution = **Lab2_CryptArithmetic**( )

Verity that the solution vector returned does in fact represent a correct solution to the original problem.

Upload your *Lab2_CheckSolution* file to the Moodle site in the *Lab 2 - CheckSolution* folder when you are done. Be sure to include the initials of all partners in the file name of the uploaded file, such as **Lab2_CheckSolution_AK_JL.m**. Note that you should also keep a version of the file without initials at the end in your directory, eg. **Lab2_CheckSolution.m**.

If you don't finish this during the lab period this week, try to arrange time with your partner to finish and upload it before your lab period next week.