In this week's lab we are going to look again at *Lab 2*'s crypt-arithmetic problem. Our goal is to develop algorithms that solve the problem and emulate to some degree the process that a person with some problem-solving expertise would use to solve the problem. We'll be referring explicitly to Polya's problem-solving principles (start by reading that handout).

The Problem

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and when the letters have been replaced by their assigned numbers, the following equation is satisfied: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

and 5 is assigned to the letter D.

Polya's First Principle: Understand the Problem

The stated problem has 10 variables, namely the letters A, B, D, E, G, L, N, O, R and T.

A solution to the problem is an assignment of integers { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } to the variables such that the following **constraints** are satisfied:

- No two variables are assigned the same integer. That is,

$$\begin{array}{cccccccccc} A\neq B, & A\neq D, & A\neq E, & A\neq G, & A\neq L, & A\neq N, & A\neq O, & A\neq R, & A\neq T, \\ & B\neq D, & B\neq E, & & \cdots & & & & B\neq T, \\ & & D\neq E, & D\neq G & & \cdots & & & D\neq T, \\ & & & \ddots & & \cdots & & & \vdots \\ & & & & & & & O\neq R, & O\neq T, \\ & & & & & & & & R\neq T; \end{array}$$

- the equation DONALD + GERALD = ROBERT is satisfied;
- and D = 5.

How many undirected inequality constraints are there in this problem?[1]

[1] There are $(9 \times 10) / 2 = 45$ undirected inequality constraints – do you see why?

What does it mean to say that the equation DONALD + GERALD = ROBERT is satisfied?

We need to be precise about the meaning of addition for numbers represented as multi-digit numerals. Thinking about the equation in the following form:

$$\begin{array}{r} \text{D O N A L D} \\ +\ \text{G E R A L D} \\ \hline \text{R O B E R T} \end{array}$$

leads us to the following six explicit constraints involving the original ten variables and the five possible carry digits *v*, *w*, *x*, *y* and *z*:

$D + D = v * 10 + T$, where $v \in \{ 0, 1 \}$;[2]

$v + 2L = w * 10 + R$, where $w \in \{ 0, 1 \}$;[3]

$w + 2A = x * 10 + E$, where $x \in \{ 0, 1 \}$;

$x + N + R = y * 10 + B$, where $y \in \{ 0, 1 \}$;

$y + O + E = z * 10 + O$, where $z \in \{ 0, 1 \}$; and

$z + D + G = R$.

Note that the final addition does *not* result in a two-digit integer.

The five new variables for possible carry digits are examples of what are sometimes called *nuisance parameters*. They get this name because although they are part of the problem they are not the variables we are really interested in. As we will see, however, including them in the problem formulation actually makes the problem easier to solve!

Polya's Second Principle: Devise a Plan

We haven't quite finished understanding the problem because we haven't yet made use of the fact that *D* = 5. Using this fact will allow us to

- eliminate the first of the six equation constraints,
- reduce the number of unknown variable values in the problem, and
- reduce the number of undirected inequality constraints.

---

[2] The sum of two distinct integers from the set { 0, 1, 2, …, 9 } lies in the set {1, 2, …, 17}, which includes only integers with a 0 or a 1 in the ten's place.

The notation " $x \in \{ \ldots \}$ " is read " *x* is an element of the set { … }."

[3] Since $v \in \{ 0, 1 \}$ and $L \in \{ 0, 1, 2, \ldots, 9 \}$, we know that $v + 2L \in \{0, 1, \ldots, 19\}$, which includes only integers with a 0 or a 1 in the ten's place.

Polya's Third Principle: Carry Out the Plan

Use the fact that $D = 5$ to determine the values of v and T.

Knowing both that $D = 5$ and $T = 0$ enables us to eliminate these two digits from the candidates for assignment. Knowing as well that $v = 1$ enables us to simplify one of the equation constraints.

We can now reformulate the problem as follows:

Reformulated Problem

The stated problem has 12 variables, namely the letters A, B, E, G, L, N, O, and R, along with the nuisance parameters *w*, *x*, *y*, and *z*.

A solution to the problem is an assignment of integers { 1, 2, 3, 4, 6, 7, 8, 9 } to the variables A, B, E, G, L, N O, and R, and an assignment of integers { 0, 1, } to the nuisance parameters *w*, *x*, *y*, and *z*, satisfying these *undirected inequality constraints*:

$$\begin{array}{rrrrrrrrrrrr} A \neq B, & A \neq E, & A \neq G, & A \neq L, & A \neq N, & A \neq O, & A \neq R, \\ & B \neq E, & & & \cdots & & B \neq R, \\ & & E \neq G & & \cdots & & E \neq R, \\ & & & \ddots & \cdots & & \vdots \\ & & & & & N \neq O, & N \neq R, \\ & & & & & & O \neq R; \end{array}$$

and the following *equation constraints*:

$$1 + 2L = w * 10 + R,$$

$$w + 2A = x * 10 + E,$$

$$x + N + R = y * 10 + B,$$

$$y + O + E = z * 10 + O, \text{ and}$$

$$z + 5 + G = R.$$

Polya's Fourth Principle: Look Back

How many constraints are there in the reformulated problem?[4]

We've reduced the total number of constraints from 45 + 6 = 51 to 28 + 5 = 33.

We've increased the total number of variables from 10 to 12.

We've also increased the number of possible assignments of values to variables from

$$9! = 9 \times 8! = 362{,}880 \text{ to } 2^4 \times 8! = 16 \times 8! = 645{,}120.$$[5]

Of course, if we know the values of the 8 remaining letter variables we can determine the values of all the nuisance variables (because we could just carry out the addition DONALD + GERALD and make note of any carried digits). This observation reduces the number of assignments we need to check to

$$8! = 40{,}320.$$

We can now apply Polya's principles to the reformulated problem.

Polya's First Principle: Understand the Problem

We'll go for now with the *reformulated* problem stated above.

Polya's Second Principle: Devise a Plan

The strategy used last week in the function **Lab2_CryptArithmetic** is called **generate and test**. It simply involves looking at all the possible assignments of values to variables and testing each assignment to determine whether or not it is the solution.

As discussed above, we've made some progress beyond last week's understanding of the problem, dramatically reducing the number of assignments we need to test.

---

[4] There are $(8 \times 7)/2 = 28$ inequality constraints and 5 equation constraints.

[5] There are two choices for *each* of the four remaining nuisance parameter values, so $2^4 = 16$ possible value assignments to the remaining nuisance parameters as a whole.

Polya's Third Principle: Carry Out the Plan

Start up MATLAB using the *Matlab Cog Sci 242* icon in the *Cognitive Science* folder under "Applications".

Copy the file **Lab2_CryptArithmetic.m**, currently in the

S:\Cognitive Science\MATLAB\Cog Sci 242

directory, into your G:\MATLAB folder.

WORKING WITHIN THE MATLAB EDITOR, open this file and save it as

Lab3_CryptArithmetic

in your working directory.

Recall that the function *Lab2_CryptArithmetic* generated an array *States* whose rows were permutations of the vector

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

In our reformulation of the problem we've eliminated the need to assign the value 0 to a letter (since we know T = 0).

Edit the function *Lab3_CryptArithmetic*, making two changes to the program lines that reflect this change in assignment possibilities.

WORKING WITHIN THE MATLAB EDITOR, now open your function *Lab2_CheckSolution* and save it as *Lab3_CheckSolution*. You'll need to make some modifications to this function. What changes do you need to make, and why?[6]

CHECK POINT: Call me over to show me the changes you've made in editing these two functions.

You have one last change to make to *Lab3_CryptArithmetic*. Change each call to *Lab2_CheckSolution* to a call to *Lab3_CheckSolution*.

Now run both programs:

>> Lab2_CryptArithmetic

>>Lab3_CryptArithmetic

Compare the results. You should, of course, get the same answer (taking into account that you already know that T=0 when you run *Lab3_CryptArithmetic.)*

Yet *Lab3_CryptArithmetic* is the better program. Why?

[6] Remember that the letters which still need values assigned are A, B, E, G, L, N, O, and R, in that order. You already know that D = 5 and T = 0.

Timing Program Runs in MATLAB

As a practical matter, programmers are concerned with minimizing both the amount of memory a program uses and the amount of time it takes to run.

One reason that *Lab3_CryptArithmetic* is a better program than *Lab2_CryptArithmetic* is that it uses less memory.

In *Lab2_CryptArithmetic,* array *States* contains $9! \times 9 = 362{,}880 \times 9$ integers.

In *Lab3_CryptArithmetic,* array *States* contains $8! \times 8 = 40{,}320 \times 8$ integers.

Thus, for the *States* array alone *Lab3_CryptArithmetic* uses <u>less than a tenth</u> of the memory required by *Lab2_CryptArithmetic*.[7]

However, the two programs also differ in their speed of execution. If you look closely, you'll notice lines in *Lab2_CryptArithmetic* (and hence in your *Lab3_CryptArithmetic* function as well) using the MATLAB functions **tic** and **toc**.

The MATLAB function **tic** precedes the first line of the program (after the function definition), while the function **toc** appears just before **return** causes the program to terminate.

The MATLAB functions **tic** and **toc** work together like a stopwatch. The stopwatch is started by **tic** and stopped by **toc.** The output of **toc** is the elapsed time, in seconds, since **tic** started the stopwatch.

Note the use of **disp** to report the elapsed time.

Run both *Lab2_CryptArithmetic* and *Lab3_CryptArithmetic* again.
Record the number of seconds each takes to run. Which one is faster?
Do several runs of each in making your comparison.

[7] $(8! \times 8)/(9! \times 9) = (8! \times 8)/(9 \times 8! \times 9) = 8/(9 \times 9) = 8/81 < 0.10$

Polya's Fourth Principle: Look Back

Between last week's lab and this one, we've developed *two* programming solutions to the crypt-arithmetic problem. Both solutions used the **generate-and-test** strategy.

The first solution was very straightforward but not efficient in either memory or time. The second solution made use of some but not all of the available constraints to reduce the size of the problem before applying the generate-and-test strategy. This accounts for its greater efficiency. We can do even better if we take all the available constraints into account!

Polya's First Principle: Understand the Problem

We'll accept the reformulation of the problem that we use for *Lab3_CryptArithmetic*, but seek a solution method that makes better use of the available constraints.

Polya's Second Principle: Formulate a Plan

The basic strategy is to carefully examine the constraints to see if we can restrict the possible values for the variables to such a degree that we ultimately find a solution.

Polya's Third Principle: Carry Out the Plan

Here are the equation constraints, labeled for subsequent reference

$$1 + 2L = w * 10 + R, \qquad \text{(C1)}$$

$$w + 2A = x * 10 + E, \qquad \text{(C2)}$$

$$x + N + R = y * 10 + B, \qquad \text{(C3)}$$

$$y + O + E = z * 10 + O, \qquad \text{(C4)}$$

$$z + 5 + G = R. \qquad \text{(C5)}$$

Variables A, B, E, G, L, N, O and R take unique values in the set { 1, 2, 3, 4, 6, 7, 8, 9 } while the nuisance variables *w*, *x*, *y* and *z* take values in the set { 0, 1 }.

Constraint C4 can obviously be simplified, so we'll start there:

$y + O + E = z * 10 + O$ implies $y + E = z * 10$, which implies $E = z * 10 - y$.

Since $0 < E < 10$, we must have $z = 1$ and $y = 1$, implying that $E = 9$ and

$$6 + G = R. \qquad \text{(C5)}$$

The remaining values for the main variables comprise the set { 1, 2, 3, 4, 6, 7, 8 }, but from constraint C5 we know that

$$G \in \{ 1, 2 \} \text{ and } R \in \{ 7, 8 \}.$$

The variable *R* also appears in constraint C1:

$$1 + 2L = w * 10 + R. \qquad \text{(C1)}$$

This tells us that *R* must be an odd number. Therefore $R = 7$.

Returning to constraint C5 we then conclude that $G = 1$.

The remaining values for the main variables comprise the set { 2, 3, 4, 6, 8 }, and the remaining equation constraints are

$$1 + 2L = w * 10 + 7, \qquad \text{(C1)}$$

$$w + 2A = x * 10 + 9, \qquad \text{(C2)}$$

$$x + N + 7 = y * 10 + B. \qquad \text{(C3)}$$

Solving constraint C2 for the variable *A* we get

$$A = (x * 10 + 9 - w)/2.$$

This means that $x * 10 + 9 - w$ is even, and less than or equal to 16 since the largest value that *A* could have is 8. But this can happen only if $x = 0$ and $w = 1$. Therefore,

$$A = 4.$$

Since we now know the value of *w*, we can solve equation C1 for *L*:

$$L = (10 + 7 - 1)/2 = 8.$$

The possible values for the remaining three main variables comprise the set { 2, 3, 6 }, and the remaining equation constraint is

$$N + 7 = y * 10 + B. \qquad \text{(C3)}$$

Testing the different possible values we find that $N = 6$, $y = 1$ and $B = 3$.

Therefore, O must have the one value remaining: $O = 2$.

Solution: (A, B, E, G, L, N, O, R) = (4, 3, 9, 1, 8, 6, 2, 7).

Polya's Fourth Principle: Look Back

We were ultimately able to find a solution to the reformulated problem by applying each constraint in turn to dramatically reduce the number of variable assignments that we needed to check. This *analytic* solution didn't involve any digital computing.

However, we can take *inspiration* from this analytic solution, a product of the human brain, to develop a *different* algorithm for solving this problem using a computer.

Polya's First Principle: Understand the Problem

We are going to view the reformulated problem as a **constraint satisfaction problem** (CSP)[8], consisting of

- a set of variables,
- a domain for each variable (its set of possible values), and
- a set of constraints.

A solution for the problem is an assignment of values to variables that satisfies the constraints.

In our case, the advantage of thinking about the problem this way – rather than as a search through *all* possible assignments – is that there are many *fewer* constraints than there are possible assignments.

Constraint satisfaction problems have been studied by computer scientists for some time, and there are good algorithms for solving them efficiently. Representing a CSP as a constraint network will help us understand these algorithms.

The **constraint network** for a CSP consists of the following:

- There is a **node** *for each variable*, a *circle* enclosing the variable name.

  Within a variable's node is the set of possible values for the variable, i.e. the variable's **domain**.

- There is a *node for each constraint*, a *rectangle* enclosing the constraint.
- For every constraint *c,* and every variable *X* involved in the constraint *c*, (i.e. every variable in the **scope** of constraint c) there is an **arc** connecting the node for *X* and the node for *c.*

[8] A good source for learning about CSPs is Poole & Mackworth, Artificial Intelligence: Foundations of Computational Agents (Cambridge University Press, 2010), freely available online at http://artint.info.

We'll treat the nuisance parameters *w*, *x*, *y*, and *z* a bit differently from the other variables. Remember there are only $2^4 = 16$ possible assignments of the values 0 or 1 to the different nuisance parameters. Since this is a small number of possibilities, we'll just *choose* a *particular* assignment of nuisance parameter values and then try to solve the rest of the problem for the main variable values. If that assignment of nuisance parameter values doesn't lead to a solution we'll try another one.

In the reformulated problem there are 28 undirected inequality constraints (such as $A \neq B$) and five equation constraints (labeled C1 through C5).

There are also 8 main variables.

Carefully read the definition of a constraint network again. **Draw** the portion of the network that involves the 8 main variables and constraints C1 through C5.

CHECK POINT: Call me over to look at your network.

Remember that we're trying to develop a computer program to solve this problem that focuses on the constraints. An important implementation choice we have to make is how to represent the constraints. For the algorithm we're going to be using and for the programming language we're using, it will be a good idea to represent a constraint as an *array*, as follows:

Recall that the *scope* of a constraint is the set of variables involved in specifying the constraint condition.

Suppose the scope of the constraint consists of the variables $X_1, X_2, .., X_\ell$.

The constraint will be represented as an array -- we can call it *Constraint* – having ℓ columns, one for each variable in the scope.

A *row* of *Constraint* will be an assignment of values to the variables in the scope that does satisfy the particular constraint condition. These values will be drawn from the current domains of each of the variables in the scope.

Thus, if a particular row of *Constraint* is [1, 6, 5, 2], we know that this corresponds to the assignment $X_1 = 1$, $X_2 = 6$, $X_3 = 5$ and $X_4 = 2$. We also know that this assignment satisfies the condition of that particular constraint.

Example

Suppose we have variable *X*, with domain { 1, 2, 3 }, and the variable *Y*, with domain { 2, 4, 5 }.

Write the constraint $X \neq Y$ as an array below.

You can check your work as follows using a MATLAB function I wrote called **NotEqualConstraint**:

```
>> X = [ 1, 2, 3 ];
>> Y = [ 2, 4, 5 ];
>> [ failure, Constraint ] = NotEqualConstraint( X, Y )        ( no semi-colon here )
```

You should have the same rows as the ones my function produces, although you may have them in a different order.

The "failure" flag is 0 if values finding the constraint are found
(meaning, "we did not fail to find values satisfying the constraint),
and is 1 if no values are found
(meaning, "we did fail to find values satisfying the constraint").

To see the details of how this function works,

>> **type NotEqualConstraint**

CHECK-POINT: Call me over to show me your constraint array and to ask me any questions you have about how **NotEqualConstraint** works.

Now create a function **Lab3_Constraint_C1** that implements constraint C1 for the reformulated problem:

$$1 + 2L = w * 10 + R.$$

The first line of your file should read:

function [failure, Constraint] = **Lab3_Constraint_C1**(L, R, w)

where L and R are row vectors, and the nuisance parameter w is a scalar.

You should model this function closely on **NotEqualConstraint** and save it to your G:\MATLAB folder.

CHECK-POINT: When you have your function working, call me over to see it.
Be sure to comment it appropriately, including identifying all members of your team. Then upload it to the Moodle folder for this function. Follow the file naming conventions in the instructions.

We'll continue with a CSP approach to solving DONALD + GERALD = ROBERT next week.

By that time you will need to have written constraint functions for each of the five main equation constraints in the reformulated problem.

That is, we will need the following constraint functions (with the syntax shown):

[failure, Constraint] = **NotEqualConstraint**(X, Y)

[failure, Constraint] = **Lab3_Constraint_C1**(L, R, w)

[failure, Constraint] = **Lab3_Constraint_C2**(A, E, w, x)

[failure, Constraint] = **Lab3_Constraint_C3**(B, N, R, x , y)

[failure, Constraint] = **Lab3_Constraint_C4**(E, O, y, z)

[failure, Constraint] = **Lab3_Constraint_C5**(G, R, z)

implementing the following constraints:

$A \neq B,\ A \neq E$, ..., $O \neq R$, and

$1 + 2L = w * 10 + R$ (C1)

$w + 2A = x * 10 + E$ (C2)

$x + N + R = y * 10 + B$ (C3)

$y + O + E = z * 10 + O$ (C4)

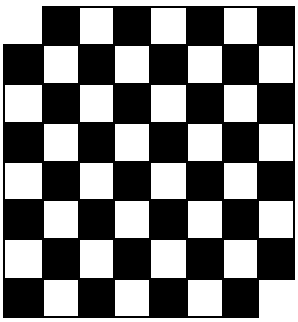
$z + 5 + G = R.$ (C5)

The remaining constraint functions will closely resemble the function you just wrote. Note that the scope for constraint C3 consists of *three* variables, not two. This will require a third **for** loop.

I recommend finishing as many of these constraint functions in lab this week as you can. You do not need to upload them to Moodle, but they do need to be ready to use by the start of lab next week.

In any time remaining, solve the following puzzles:

The Mutilated Checkerboard Problem



The checkerboard above has been "mutilated" by removing the squares at the top left and bottom right corners. There are 62 squares remaining.

Can this checkerboard be covered exactly with 31 tiles, each a 1×2 rectangle covering exactly two squares?

Cats and Dogs

There are 3 cats & 3 dogs on the left bank of a river.
There is one raft, starting on the left bank.
The raft can carry at most two animals, and must carry at least one (going both directions).
Dogs can't outnumber cats at any location (either on the bank of the river or on the raft).

Goal: End up with all 3 cats & 3 dogs on the right bank of river.