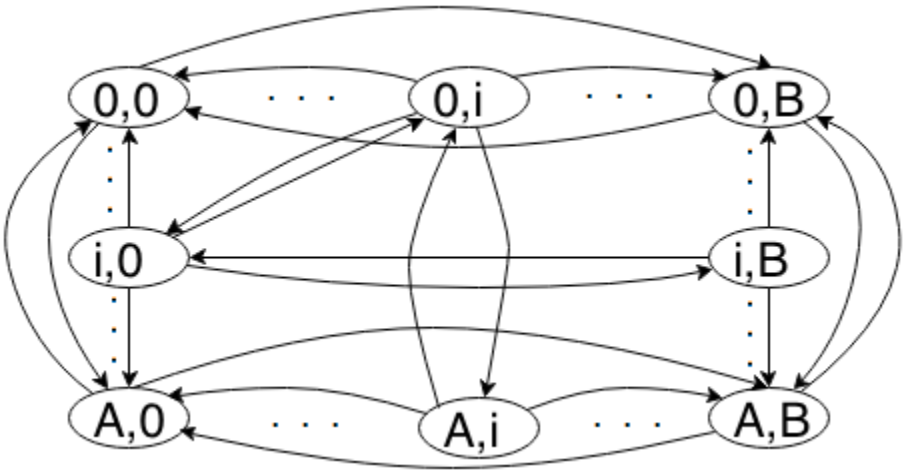
Two Jugs Problem
Zach Sisco

Problem:
Two jugs have capacities of **A** and **B** quarts. Use the jugs to measure out exactly **C** quarts of water, while obeying the following restrictions:

i. Either jug can be filled to capacity from a water tap;
ii. The contents of either jug can be emptied into a drain;
iii. Either jug may be poured into the other.

Model:
The Two Jugs Problem can be modeled as a network problem. Each node in the network contains two values, representing the possible capacities of the two jugs. A directed arc from one node to another represents a "pour" that adheres to the given constraints in the problem statement. Thus, the problem can be solved by finding a path from the beginning node to a node which has the value that needs to be measured. The model presented here will find the shortest path (i.e. the minimal number of "pouring" operations needed to measure the desired capacity).

An example network is shown below. **A** and **B** are the jug capacities.



Explanation of AMPL Code:

```
param a:=5;
param b:=8;
set A:= 0 .. a;
set B:= 0 .. b;

param goal_cap := 4;  # Goal quantity to measure out

set N := {i in A, j in B : i=0 or j=0 or i=a or j=b};
```

The above code declares three parameters. Parameters *a* and *b* denote the two jugs' respective capacities. Parameter *goal_cap* is the capacity that needs to be measured. The sets *A* and *B* are sets that contain integers from 0 – *a* or 0 – *b*. These are all the possible values that jugs A and B can hold. The set *N* declares the set of all possible nodes in the graph. Each node is an ordered pair. The first number taking values in *A*, and the second taking values in *B*.

```
set E := {(i,j) in N, (k,l) in N:
        (i!=k or j!=l) and
        ( (k=0 and l=j) or
```

```
          (k=a and l=j) or
          (l=0 and i=k) or
          (l=b and i=k) or
          i+j=k+l
        )};

set N_exit :={(i,j) in N: i=goal_cap or j=goal_cap};
set E_exit :={(i,j,k,l) in E: k=goal_cap or l=goal_cap};
set E_exit2 :={(i,j,k,l) in E: i=goal_cap or j=goal_cap};
```

This section of code declares sets of edges. *E* is the set of all possible edges in the graph. In the AMPL model, an edge is represented as a 4-tuple. Its first 2 values represent the jugs' capacities before an operation, and its second 2 values represents the jugs' capacities after an operation. The logical constraints in the definition for *E* restricts the set of edges to only valid "pours." The first logical statement prevents edges that do not do anything. For example, going from node (2,2) to (2,2). The next statement of disjunctions say that either 3 operations can happen:

1. jug A is filled to capacity or emptied, while jug B stays the same;
2. jug B is filled to capacity or emptied, while jug A stays the same;
3. the sum of A and B before the pour equals the sum of A and B after the pour.

The set *E_exit* is the set of all edges that connect to nodes containing the goal capacity. The set *E_exit2* is the set of all edges that connect from nodes containing the goal capacity. *N_exit* is the set of all nodes that contain the goal capacity in either the first or second value of the node.

```
var Use {(i,j,k,l) in E} >= 0;  # 1 iff (i,j,k,l) in shortest path

minimize Total_Time: sum {(i,j,k,l) in E} Use[i,j,k,l];
```

The *Use* variable is defined over all edges and is equal to 1 if an edge (*i,j,k,l*) is contained in the shortest path. The objective function *Total_Time* is defined to minimize the total number of edges in the solution (shortest path).

```
subject to Start: sum {(0,0,i,j) in E} Use[0,0,i,j] = 1;

subject to Exit: sum{(i,j,k,l) in E_exit} Use[i,j,k,l] = 1;

subject to Exit2: sum{(i,j,k,l) in E_exit2} Use[i,j,k,l] = 0;

subject to Balance {(k,l) in N diff ({(0,0)} union N_exit)}:
        sum {(i1,j1,k,l) in E} Use[i1,j1,k,l] = sum {(k,l,i2,j2) in E} Use[k,l,i2,j2];
```

This section of code defines the constraints. The first constraint, *Start*, says that only one edge leaving from (0,0) can exist in the solution. The *Exit* constraint is defined so that only one edge in the set *E_exit* can be use in the solution. The *Exit2* constraint prevents any edges leaving from the "goal capacity" nodes to be used in the solution. The *Balance* constraint is defined over all nodes except the starting node (0,0) and any of the exit nodes. It ensures that for any node in the solution there is an edge leaving it and an edge entering it.

Here is the complete AMPL code:

```
param a:=5;
param b:=8;
set A:= 0 .. a;
set B:= 0 .. b;
```

```
param goal_cap := 4;  # Goal quantity to measure out

set N := {i in A, j in B : i=0 or j=0 or i=a or j=b};

set E := {(i,j) in N, (k,l) in N:
        (i!=k or j!=l) and
        ( (k=0 and l=j) or
          (k=a and l=j) or
          (l=0 and i=k) or
          (l=b and i=k) or
          i+j=k+l
        )};

set N_exit :={(i,j) in N: i=goal_cap or j=goal_cap};
set E_exit :={(i,j,k,l) in E: k=goal_cap or l=goal_cap};
set E_exit2 :={(i,j,k,l) in E: i=goal_cap or j=goal_cap};

var Use {(i,j,k,l) in E} >= 0;  # 1 iff (i,j,k,l) in shortest path

minimize Total_Time: sum {(i,j,k,l) in E} Use[i,j,k,l];

subject to Start: sum {(0,0,i,j) in E} Use[0,0,i,j] = 1;

subject to Exit: sum{(i,j,k,l) in E_exit} Use[i,j,k,l] = 1;

subject to Exit2: sum{(i,j,k,l) in E_exit2} Use[i,j,k,l] = 0;

subject to Balance {(k,l) in N diff ({(0,0)} union N_exit)}:
        sum {(i1,j1,k,l) in E} Use[i1,j1,k,l] = sum {(k,l,i2,j2) in E} Use[k,l,i2,j2];
```

Experimental Analysis:

C denotes the capacity that needs to be measured.
A, B are the two jugs and their respective capacities.
Each entry is the minimal amount of operations needed to measure out **C** with jugs **A** and **B**.
'x' denotes an infeasible solution.

	C=1	C=2	C=3	C=4	C=5	C=6	C=7	C=8
A=2, B=3	2	1	1					
A=2, B=4	x	1	x	1				
A=2, B=5	4	1	2	4	1			
A=2, B=6	x	1	x	2	x	1		
A=2, B=7	6	1	4	4	2	6	1	
A=2, B=8	x	1	x	4	x	2	x	1
A=2, B=9	8	1	6	4	4	6	2	8
A=3, B=4	2	4	1	1				
A=3, B=5	4	2	1	6	1			
A=3, B=6	x	x	1	x	x	1		
A=3, B=7	4	6	1	2	8	4	1	
A=3, B=8	6	4	1	10	2	4	8	1
A=3, B=9	x	x	1	x	x	2	x	x
A=4, B=5	2	6	4	1	1			
A=4, B=6	x	2	x	1	x	1		
A=4, B=7	4	8	2	1	8	6	1	
A=4, B=8	x	x	x	1	x	x	x	1
A=4, B=9	4	10	6	1	2	8	10	4
A=5, B=6	2	6	8	4	1	1		
A=5, B=7	8	2	4	6	1	10	1	
A=5, B=8	8	4	2	10	1	6	8	1
A=5, B=9	4	10	8	2	1	8	12	6
A=7, B=8	2	6	10	12	8	4	1	1
A=7, B=9	12	2	8	6	4	10	1	14

Most of the infeasible solutions were scenarios where both jugs had even capacities and an odd capacity needed to be measured. However, there were also situations where jugs with even capacities could not measure an even number (e.g. in the case of **A**=4, **B**=8 and **C**=2,6). And there are also cases where jugs with one even and one odd capacity produced infeasible solutions (e.g. **A**=3 and **B**=6). What causes infeasible solutions?

It turns out that Boldi et al. [1] proved that **C** is measurable if and only if it is a multiple of the greatest common divisor of **A** and **B**. From this is follows that if GCD(**A**,**B**) = 1, than any capacity **C**, where $1 \leq \mathbf{C} \leq max\{\mathbf{A},\mathbf{B}\}$ can be measured.

Conjecture: Non-trivial cases (where the minimal number of operations > 1) always take an even number of operations.

Questions: What makes certain cases hard? Given two jugs, is there a way to know which capacities will be the most difficult to measure?

Literature Review: The following briefly reviews two other publications that study the jugs measuring problem. The first publication by M. Shieh and S. Tsai [2] studies the generalized jugs problem. They

proved lower and upper bounds on the minimum number of measuring operations for *n* jugs. Also, they reduced the general optimal jugs measuring problem to an NP-hard problem (the shortest GCD multiplier problem). Thus, showing that the problem is NP-hard.

A second publication that was reviewed was by Y. K. Man [3]. This paper gives an arithmetic solution to solving the two jugs problem. The problem is modeled by the Diophantine equation, $mx+ny=d$. It says that since the Diophantine equation is solvable if and only if GCD(m,n) divides d, then the two jugs problem is solvable if and only if the greatest common divisor of the capacities of the two jugs divides the capacity needed to be measured. The author gives two algorithms for solving the two jugs problem arithmetically, and illustrates how solutions can be achieved with several examples. The author argues that this approach is suitable for hand calculations and can be easily implemented in a typical programming language. Further areas of research for this approach include generalizing the algorithm for *n* number of jugs. It may be worthwhile to see if this arithmetic approach (implemented in a programming language) is computationally faster than the network model provided above.

References

[1] P. Boldi, M. Santini and S.Vigna. Measuring with jugs, Theoretical Computer Science, 282 (2002) 259–270.

[2] M. Shieh and S. Tsai. Jug measuring: algorithms and complexity, Theoretical Computer Science, 396 (2008) 50-62.

[3] Y. K. Man. An Arithmetic Approach to the General Two Water Jugs Problem. Proceedings of the World Congress on Engineering 2013 Vol I, WCE 2013.