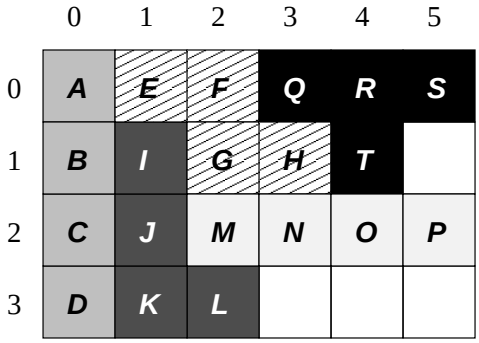
Discussion Problems: "Constraint Programming Encodings"
Declarative Methods (JHU 600.432/632)
Prof. Jason Eisner

1. You are trying to pack a two-dimensional suitcase with certain Tetris pieces. To make the problem simpler to encode, no rotation of the pieces is allowed. One solution is shown below, for packing 5 pieces (ABCD, EFGH, IJKL, MNOP, and QRST) into a 6×4 suitcase.



Here is a partial encoding of the problem instance whose solution is shown above, using ECL^{i}PS^e-like constraints on the (x, y) coordinates:

```
% The suitcase is 6 x 4
[Ax,Bx,Cx,Dx,Ex,Fx,Gx,Hx,Ix,Jx,Kx,Lx,Mx,Nx,Ox,Px,Qx,Rx,Sx,Tx] :: 0..5,
[Ay,By,Cy,Dy,Ey,Fy,Gy,Hy,Iy,Jy,Ky,Ly,My,Ny,Oy,Py,Qy,Ry,Sy,Ty] :: 0..3,

% the ABCD piece is vertical
Ax = Bx = Cx = Dx,
Ay+1 = By,   By+1 = Cy,   Cy+1 = Dy,

% the IJKL piece has an "L" shape
Ix = Jx = Kx,   Kx+1 = Lx,
Iy+1 = Jy,   Jy+1 = Ky,  Ky = Ly,

% etc.
```

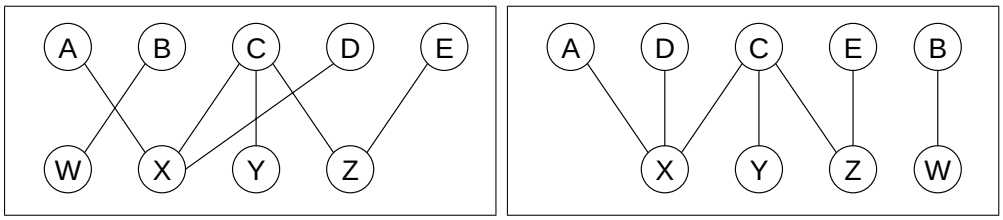
(a) [6 points] Give additional constraints to encode the fact that pieces may not overlap. (*Hint:* One nice solution uses `alldifferent`, though other approaches are fine too.)

(b) [5 points] The above approach forces every piece to have a position (and only one position) in the suitcase. But suppose you have too many pieces to fit. How might you revise your encoding so that it is possible to leave some pieces out? (Explain precisely what needs to be changed, but it is not necessary to give the actual constraints.)

(*Hint:* You could completely replace the original approach, but it is also possible to do this with a simple change. Think outside the box ...)

(c) [3 points] Your constraint program in question 1b may have many solutions. Explain how you would define a variable `Cost` so that solutions that get more pieces into the suitcase have a lower `Cost`.

2. You would like to draw a bipartite graph so that its edges don't cross. Here are two drawings of the same input graph. You want your solver to find the one on the right.



You are required to use this kind of two-layer drawing. So in the case above, your job is to find an ordering (permutation) of $\{A, B, C, D, E\}$ and an ordering of $\{W, X, Y, Z\}$ so that no *pair* of edges crosses.

Notice that AX and CX can't ever cross, because they share an endpoint X.

Whether AX crosses BW depends on the relative order of A and B, and the relative order of X and W. A no-crossing constraint between AX and BW could be written in ECLiPSe as `(A < B) #= (X < W)`.

(a) [10 points] Encode the no-crossing-layout problem for the above input graph as a list of ECLiPSe constraints. (It should be obvious how to extend your approach to larger graphs, but you only have to handle this particular graph.) **You must list *all* of the necessary no-crossing constraints—*as well as any other kinds of constraints you need!*—to prove to us that you have found a *systematic*, mechanical way of translating a graph into an Eclipse program.**

(b) [6 points] By joining (fusing) the two constraints `(X < Z) #= (C < E)` and `(X < Z) #= (A < C)`, and then projecting back to the triple (A, C, E), what new constraint could a solver infer on just the variables (A, C, E)? Express this constraint **both** in ECLiPSe and in English.

(c) [15 points] Now do part (a) again, but this time encode the same problem as a SAT problem. (*Hint:* Don't worry about the numerical position of the vertices, only their order. Try using variables with names like `A<B`.) **Make sure you don't overlook any necessary kinds of clauses. If there are too many individual clauses of some type to write out by hand, you can write just the first few and then explain the pattern.**

(d) [10 points] In fact, it is not necessary to use SAT or constraint solvers to find a crossing-*free* drawing of a bipartite graph. There's an easy polynomial-time algorithm that will find such a drawing if it exists.

However, "most" bipartite graphs don't have a crossing-free drawing. And it *is* NP-hard to find a drawing with the minimum number of crossings. So solving this much harder problem does require our fancy methods. Explain how to revise your solution to part (a) and/or (c) in order to do so.

3. In the game of Mastermind, your job is to guess a secret code P. The code is a sequence of n colors, such as $P = \text{GYGRR}$ (green, yellow, green, red, red) when $n = 5$.

   On each of your moves, you guess your own sequence Q of n colors, such as $Q = \text{GGRGB}$.

   This "challenge" Q is answered with a "response" (a, b), where a is the number of your colors that were correct and in the correct position, and b is the number of your colors that are correct but in the wrong position.

   It is helpful to transform this response to (a, c), where $c = a + b$. Then c is the number of

your colors that are correct (but not *necessarily* in the correct position).

What happens in our example?

- In this case, $a = 1$ since $Q_1 = P_1$ (but $Q_2 \neq P_2$, $Q_3, \neq P_3$, etc.).
- And $c = 3$ because your guesses Q_1, Q_2, and Q_3 can be matched respectively with P_1, P_3, and P_4. We do not consider Q_4 to be correct because all the greens in P have already been used up by matching Q_1 and Q_2—there are no additional greens in P for Q_4 to match.

  Another way of thinking about it: $c = 3$ means that by reordering Q (e.g., to GBGGR), we could drive a as high as 3.

  Another way of thinking about it: Since the two sequences include 3 and 2 greens, the maximum number of greens that can be matched is $2 = \min(3, 2)$. (It doesn't really matter *which* 2.) So greens can only contribute 2 to c. Reds contribute another 1 to c, for a total of c.

(a) [10 points] Let's take $n = 3$ to keep your answer short. (But your answer should be easily generalizable to longer n.)

Using ECL^{i}PS^e notation, write down the constraints on the unknown P = `[P1,P2,P3]` that follow from knowing that a particular challenge Q = `[Q1,Q2,Q3]` gave rise to a particular response (a, c). Your constraints should mention a, c, `P1`, `P2`, `P3`, `Q1`, `Q2`, `Q3`, and perhaps other variables.

(b) [5 points] Suppose you receive responses to k different challenges. How would you then determine whether there is a *unique* solution for P (so that you can guess P without any further challenges)? Mention the specific algorithms you would use.

(c) [5 points] In 2005, the Mastermind Satisfiability Problem (MSP) was proved to be NP-complete. From this fact, and from the name of the problem, you can probably guess exactly what the MSP is. What are the inputs to the MSP? What is the output?