# Constraint Programming

Basic working is that we have a constraint store and a search. The constraint store contains all the constraints.

* The Constraint Store is repeatedly trying to prune the search space
* When no more pruning can be done it returns control to the search
* The search then adds a new constraint (i.e. It assigns a value to one of the decision variables)
* the constraint store then checks with each constraint to see if it’s still feasible
* if any of the constraints aren’t feasible then it returns to the search telling it that the new constraint isn’t acceptable and to try a different one
* if all the constraints are feasible then it tries to prune the search space again

## Global Constraints

These are better than just single constraints as they involve all the variables and values and allow us to prune the search space quicker and more efficiently

### AllDifferent Global constraint

States that all variables must have a different value

To prove feasibility:

1. Lay the variables and values out as a bipartite graph so that there is a vertice from each variable to the values that are in it’s domain
2. To prove feasibility need to find a Maximum Match (M) for the graph (this means that there is a line from every variable to a value so the number of edges in M is equal to the number of variables)
   1. Do this by finding any match and modeling it as a directed graph
      1. edges that are in the match go up from value to variable
      2. edges that are not in match go down from variable to value
   2. Now iteratively improve that match until it can no longer be improved – if at that point it’s a Maximum match then the constraint is feasible, if it’s not then it isn’t feasible. We do this using a depth first or best first search approach
      1. Take a variable that doesn’t have an edge in the match
      2. Travel down one of it’s edges to the value
      3. Travel back up the matched edge to the variable the value had been matched to
      4. Keep doing this until we eventually reach a value that wasn’t previously in the match
         1. if we do this then that will mean that we’ve added an additional variable to the match and therefore improved it. If this is a Maximum Match then we’ve proved feasibility, if not then go back to i. for this match and keep going until we either find a Maximum Match or we’ve exhausted all possibilities and shown that this constraint isn’t feasible
         2. if we can’t do this then that will mean that we can’t improve the match by using that variable and edge

To Prune:

1. Lay the variable and values out as a bipartite graph again but this time reverse the direction of the direction (so edges in the match go down and edges not in the match go up)
2. Need to find any edges that are not part of any of the following – if they aren’t then we can prune them (so remove the value from the domain of the corresponding variable for the edge):
   1. Part of the match that we found when proving feasibility (M)
   2. Part of an ‘even alternating path starting with a free value vertex’ (P)
      1. For each free value travel up an edge that wasn’t in the match and then down an edge that was – keep doing this until we can’t travel any further. If we end on another value vertex then this is a P
   3. Part of a strongly connected Component
      1. Use depth first search
         1. Start with a variable and go down an edge that was in M
         2. Go back up an edge that wasn’t in M to another variable
         3. Keep doing this until we eventually either reach the variable we started off with or can’t go any further
            1. if get back to the variable we started with then this is a strongly connected component so all these edges are part of some maximum match
            2. if we can’t go any further then this isn’t a strongly connected component so try the next option

## First Fail principal

Start with the place you’re most likely to fail – that way you don’t do the easy stuff and then get to the hard stuff and find you’ve wasted your time as you can’t do it.

In search this means start with the places where the domain is the smallest but not 1 (a domain of 1 is easy as you only have one choice) but a domain of 2 is hardest and also doing that instead of a domain of 4 means you have a smaller search space.

If there is more than one variable that has the smallest (non 1) domain then choose the one that is the most constrained (i.e. is involved in the most constraints) – in the country graph colouring example this would be the country that touches the most other countries.