1 Reductions

This section is intended to familiarize students with thinking about CSPs in general terms, and relating them to one another.

A *reduction* from problem X to problem Y is possible if an algorithm that solves Y can be used to solve X. In order to do so, a problem statement in the form of problem X, must be converted into a problem statement in the form of problem Y.

1.1 Graph Coloring

Graph coloring is defined as follows. Given a graph $G = (V, E)$, and a set of colors C, is it possible to assign a color from C to every vertex such that no two contiguous vertices have the same color?

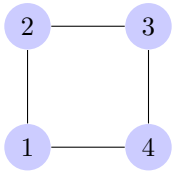


Figure 1: 2 Colorable Graph

Figure 1 is two colorable. Vertices $\{1, 3\}$ can be colored C_1 and vertices $\{2, 4\}$ can be colored C_2. This implies that it is also, 3 colorable, 4 colorable, and so on. But it is not 1 colorable.

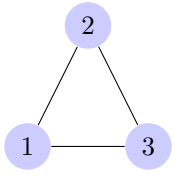


Figure 2: 3 Colorable Graph

In figure 2, every node has to have a different color, or else it would collide with one of the other two nodes. Therefore this graph is 3 colorable.

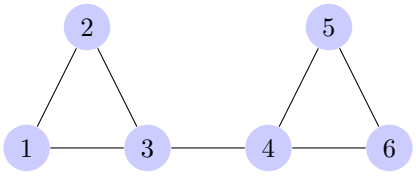


Figure 3: 3 Colorable Graph

Figure 3, is also 3 colorable. So long as nodes $\{3, 4\}$ are different colors, $\{1, 2, 3\}$ and $\{4, 5, 6\}$ can both be filled in using 3 colors.

1.2 Cooking Reduction

A head chef is preparing her dinner menu for tonight and has just finished her first draft. Given the current menu, there are S possible stations that can be ran tonight in the kitchen. She has to assign her chefs, C, to each station, but some chefs don't work very well together. All sations have infite capacity. She has comprised a set of chef tuples that don't work well together, $\{(C_1, C_4), (C_6, C_3), (C_4, C_2), etc\}$. She wants to finalize the menu, but needs to know whether there it is possible to assign chefs to stations such that all chefs who don't work well together will be at different stations. **Note, not all stations have to be manned**. She knows how to solve graph coloring problems. How can this problem be turned into a graph coloring problem?

Answer: Every station can be thought of as a color. Every chef can be thought of as a node. And every pair of chefs that don't work well together can an edge between their nodes in the graph. If there exists a graph coloring for this, then such an assignment of chefs is possible.

1.3 Meetings Reduction

This problem was discussed in class, but I think it worth revisiting, as it may make more sense after the previous example.

Joe is in charge of scheduling all the meetings for his company this year. He is given a set of all the days that are available to him to choose from, D. Additionally he is given a set of all meetings that need to be held at some point during this year, M. Some meetings can be scheduled on the same day, and others cannot. For meetings that can be scheduled on the same day, there is an infinite capacity of rooms available. He has a set of all meetings that can't be scheduled on the same day, $\{(M_4, M_1), (M_3, M6), etc\}$. He wants to know whether it is possible to schedule all meetings given these constraints. Again, how can this be formalized as a graph coloring problem?

Answer: Every day can be thought of as a color. Every meeting can be thought of as a node. And every pair of meetings that can't occur on the same day can

an edge between their nodes in the graph. If there exists a graph coloring for this, then such an assignment of meetings is possible.

1.4 Discussion

Despite sounding quite different at face value, these can all be thought of as similar problems. Let look at them from a high level to see why.

All problems require the assignment of variables to a domain of values. And all are subject to the constraints that pairwise variables must have different values assigned to them. If we think about each in these terms, rather than on their stories, it becomes more obvious.

2 Defining a CSP

Assume again that we are trying to schedule a list of meetings, M, to days, D. In this case, the constraints are pairwise $\{(M_1, M_2), (M_6, M_2), (M_3, M_4), etc\}$, where the pair (M_i, M_j) signifies that Meeting M_i must come before the meeting M_j. Assume that no two meeting can occur on the same day.

2.1 Defining CSP Components

Ask the students to define what the variables, domains, and constraints of this problem are.

Example:

Variables: The meetings $M_1, M_2, ...M_n$

Domains: The days that are available $D_1, D_2, ...D_m$

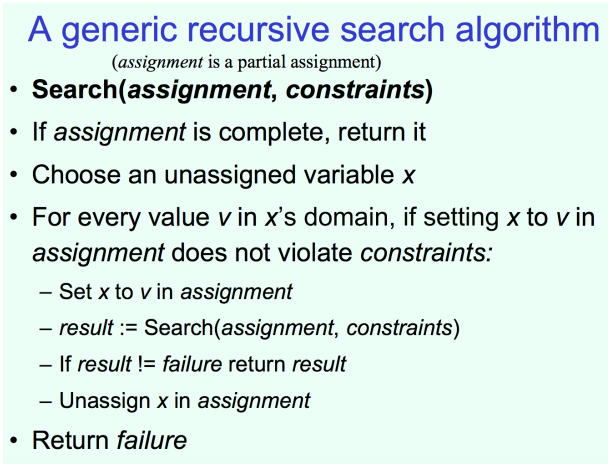
Constraints: For every constraint in the set of constraint C, (M_i, M_j) the day assigned to M_i must come before the day assigned to M_j

$$\forall (M_i, M_j) \in C \quad date(M_i) < date(M_j)$$

2.2 Solving With Search

For students interested, this problem can be solved quite efficiently using topological sorting, though this is not within the scope of this class.

Let's assume we wanted to solve this problem using a generic recursive search algorithm as was discussed in class. What might this code look like?



Above is a copy of the search algorithm. It exhaustively tries every possible value in the domain of every possible variable. Therefore we know if a solution exists, it will find it.

You may notice the algorithm is recursing on itself, meaning it is using a stack (DFS) rather than a queue (BFS). Why is DFS the better choice here than BFS?

Answer: Well, we know we have to assign all variables, and the number of variables is a constant. Therefore, the depth at which we find a solution is always the same (depth is equal to number of variables). If the depth of the solution is always the same, then DFS is better as it doesn't require the memory of BFS, and also doesn't affect the optimality of our answer.

2.3 On Board Walk Through

Consider the following example from the problem above.

You have 3 days, and 3 meetings. Meeting 1 must come before meeting 2, and meeting 3 must also come before meeting 2. The only valid orderings of meetings are 1,3,2 and 3,1,2. What other states may be tried by the search algorithm? Feel free to walk through the search trees on the board.

Answer: The search algorithm could try any permutation of {1,2,3} so any of (1,2,3) (1,3,2) (2,3,1) (2,1,3) (3,1,2) (3,2,1) are possible search states.

3 Constraint Optimization

Constraint optimization problems seek to find the best solution given an objective. For example consider the problem where queens can move horizontal,

vertical, diagonal, and in an L shape like a knight. It is not possible to place such queens on an 8 by 8 board and have no two queens attack each other, so instead we try to minimize the number of attacking queens.

3.1 Meetings Cost Optimization

Again assume that we are scheduling meetings. Now we are using an external vendor as a meeting facility, and we have to pay at a day by day granularity. They give us a list of all possible dates. Some dates are more popular than others, and thus cost more to rent. We still have a list of hard constraints, certain meetings cannot fall on the same day (otherwise multiple meetings can occur on the same day), and some meetings must be booked for dates that come before others. We also have a soft constraint we are trying to optimize, spend as little money booking the dates as possible.

The booking agency is able to give a cost for each set of days we may book. Additionally, we know that the cost of adding another room is always zero or greater. In other words, booking an additional day never decreases the price.

3.1.1 State Representation

What would be the representation for the states, successor function, and goal check?

For example: A state can be a valid assignment of some subset of the meetings to days. A successor is achieved by adding any additional meeting to the valid assignment, creating another valid assignment. A goal is reached when a state contains all meetings.

3.1.2 Search Techniques

All search techniques discussed last week can be applied here. Just like before, there is a successor function, a goal state, and initial state, and a cost associated with every state. We are trying to find the minimum cost goal state. For example, DFS, BFS, Dijkstra, and so on could all be used. With an admissible heuristic, so could A*.

Another new algorithm brought up in class, depth first branch and bound can also be used (dfbb). Dfbb keeps track of the best solution it has seen so far. Whenever it comes upon a path with a higher cost than the best solution found so far, it can skip that path knowing the cost will only increase as they traverse along it.