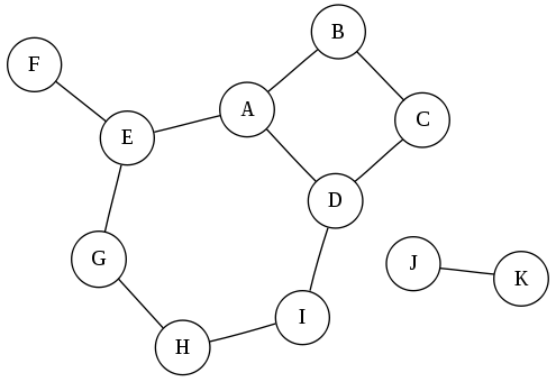
# Discussion Problems: "More Miscellany"
# Declarative Methods (JHU 600.432/632)
# Prof. Jason Eisner

1. Recall that a graph $k$-coloring problem consists of an undirected graph and $k$ colors. You have to assign a color to each of the $n$ vertices of the graph such that no two neighboring vertices (i.e., vertices connected by a single edge) have the same color.




This can be set up as an ECL$^i$PS$^e$ problem with $n$ variables, each one of which has domain `1..`$k$.

In this question, we will consider the case $k = 2$. This case can be solved *in polynomial time*: in fact, $O(n + m)$ time, where $n$ is the number of vertices and $m$ is the number of edges. However, some algorithms may be faster than others. We will consider a few algorithms.

(a) [3 points] One way to color the graph in polynomial time is to reduce it to a 2-SAT problem (since 2-SAT problems can be solved in polynomial time). This is possible for $k = 2$. For the graph above, write down a couple of the 2-SAT constraints that would be needed to encode a 2-coloring problem on this graph.

(b) [6 points] Alternatively, you could try using backtracking search with arc consistency. What would happen on the example? (How many decisions would be made? What would arc consistency do for you? Does variable ordering matter?)

(c) [6 points] Alternatively, you could use variable elimination. What would the result be of eliminating variable $D$ in the example above? What would the result be of then eliminating variable $A$?

*Hint:* Remember that if a variable $V$ is connected to other variables, it makes those other variables interact with each other indirectly. If $V$ is eliminated, then the interaction must be preserved by expressing the interaction directly.

2. [10 points] Here is a hard practical problem, which you should be able to attack using methods from this course.

You have $n$ processes distributed among $m$ processors on a network. For example, process 2 is currently running on processor 8, and so is process 5. We can encode this as $L_2 = 8$, $L_5 = 8$.

For efficiency reasons, you would now like to "migrate" some of the processes to other processors. How can you decide which processes would be good to move?

The problem is hard if you wish to take multiple considerations into account, e.g.:

- Large processes are expensive to migrate. For example, if process 5 is very large, then you would strongly prefer to leave it on processor 8.
- To the extent that two processes communicate a lot with each other, you would prefer to get them onto the same processor.
- Every processor should be running about the same number of processes, so that all processes make progress at the same rate.[1]

So what is a good general practical technique that you could use here? **Name the technique.** Explain in some detail how you would apply it in this case to obtain a new assignment, and discuss how to make it efficient.

You may assume that you will run your technique to completion on another computer (not one of the $k$ processors) to decide which processes to move, before you *actually* move any processes.

For full credit, your answer should indicate how to take the above three considerations into account. The following notation might be helpful:

- Let $S_i$ be the size of process $i$ (for $1 \leq i \leq n$).
- Let $C_{ij}$ be the amount of communication between $i$ and $j$ (for $1 \leq i < j \leq n$).
- Let $N_k$ be the number of processes running on processor $k$ (for $1 \leq k \leq m$).

[1] This is an oversimplification. One might want higher-priority processes to make faster progress, for example.

3. [5 points] Write a Prolog query of the form `member(...)` that will not terminate. Say what solutions it will give, if any.

4. (a) [3 points] Here is a simple ECL$^i$PS$^e$ program that illustrates finite-domain constraint programming:

```
solve(A,B) :-
  [A,B]::0..5,
  A+B #= 7,
  A*B #= 10,
  labeling([A,B]).
```

What set of satisfying assignments will be found by the query `solve(A,B)`?

(b) [5 points] Implement `solve`/2 in Prolog. Your Prolog program should find exactly the same set of satisfying assignments to the above query, using a simple backtracking generate-and-test strategy.

You may not use any ECL$^i$PS$^e$ features, so none of the 4 lines above is allowed. Your only arithmetic should be the Prolog constraints `7 is A+B` and `10 is A*B`. (Remember that these constraints require `A` and `B` to be grounded, or they will cause a runtime error.)

(c) [3 points] Let's consider the efficiency of your Prolog program. Suppose you modified your Prolog program in the obvious way to handle the domain `0..`$n$ rather than `0..5`. What is the Prolog program's runtime as $n$ gets large?

i. approximately proportional to $n$
ii. approximately proportional to $n^2$
iii. approximately proportional to $n^3$
iv. approximately proportional to $b^n$ for some $b$
v. none of the above

(d) [3 points] Now back to ECL$^i$PS$^e$, which does bounds propagation whenever possible. In the query of problem 4a, what are the reduced domains of A and B just before the

`labeling([A,B])` step is reached? In other words, what does ECL$^i$PS$^e$ already know about `A` and `B` before backtracking search begins? Justify your answer.

(**Note:** You could use arc consistency instead of bounds propagation; they have the same effect for this problem.)

(e) [6 points] How could you modify the ECL$^i$PS$^e$ program to avoid symmetric (mirror-image) solutions? How, if at all, would this affect the results of bounds propagation (question 4d)? Justify your answer.

(**Note:** You could use arc consistency instead of bounds propagation; they have the same effect for this problem.)

(f) [5 points] Prolog does not have bounds propagation. What *other* mechanism does it have that can rule out a huge portion of a search space in a single step (taking only $O(1)$ time)? Illustrate with a simple example.

(g) [12 points] A domain constraint `[A,B]::0..7` is equivalent to saying that `A` and `B` are both 3-bit binary integers (somewhere from 000 to 111). Assuming that is true, express problem 4a's constraint `A+B #= 7` as a conjunction of SAT clauses over boolean variables. You do **not** have to use CNF, e.g., it is all right to use operators like $\rightarrow$ and $\leftrightarrow$.

Your answer should generalize efficiently to the same problem for $n$-bit integers: `[A,B]::0..`$(2^n-1)$`, A+B #=` $2^n-1$. That is, you only have to write out the encoding for the case $n = 3$, but it should be clear that you could encode the problem in the same way for any $n$, using only $O(n)$ or $O(n^2)$ variables and constraints.

This means that your solution should *not* do something brute-force, like giving a big disjunction over all $2^n$ satisfying assignments to (`A`, `B`). Instead, think about how you would do binary addition by hand, carrying a 1 if necessary.

(h) [10 points] Express the constraint `A+B #= 7` in pure Prolog. That is, don't use ECL$^i$PS$^e$, and don't use Prolog's built-in arithmetic. You will have to define addition constraints in some other way.