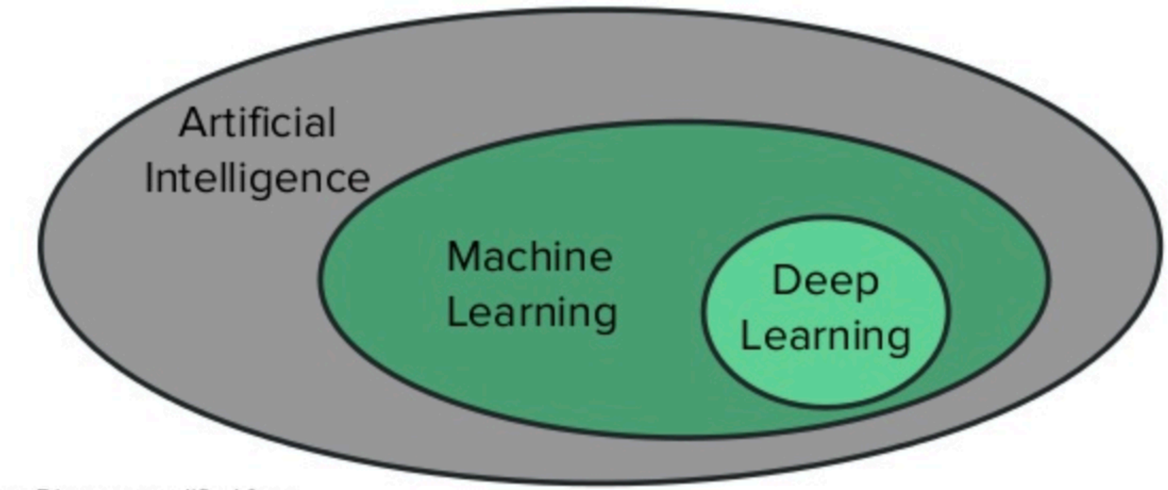
Machine Learning for Graph Coloring

Assoc. Prof. Yusuf Sahillioğlu

Computer Eng. Dept, MIDDLE EAST TECHNICAL UNIVERSITY, Turkey

Machine Learning

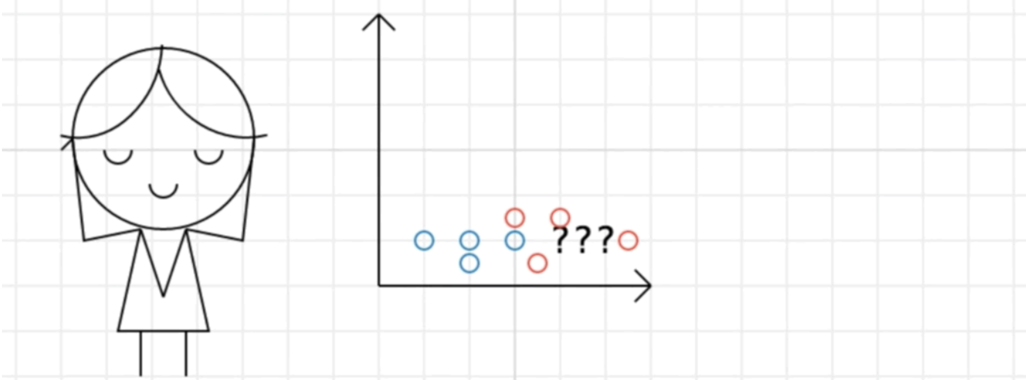
- ✓ Artificial Intelligence (AI) & Machine Learning (ML) & Deep Learning (DL)
- ✓ AI ~ anything related to making computers do stuff that are traditionally done by humans; sorting, gaming, etc.
- ✓ ML ~ algorithms that learn models from data; neural nets, SVMs, etc.
- ✓ DL ~ application of multi-layer neural nets to learning tasks.



Machine Learning

- ✓ How do we decide the best class?
- ✓ Experience the past (training) and decide accordingly (query).

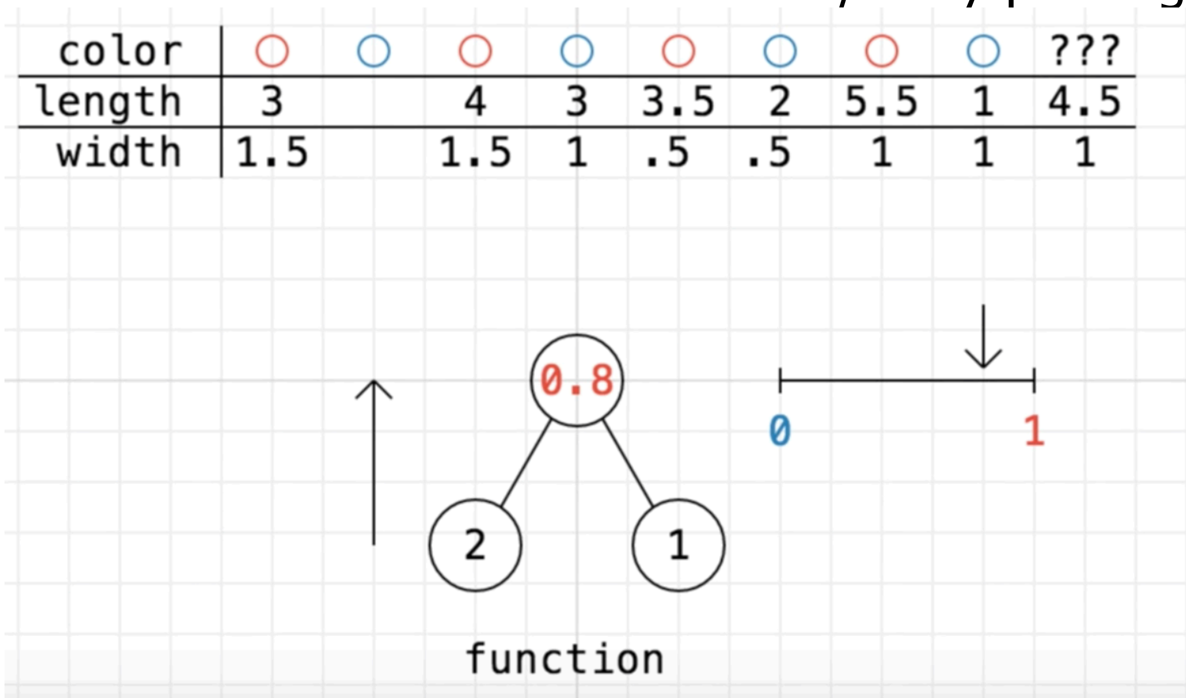
color	○	○	○	○	○	○	○	○	???
length	3	2	4	3	3.5	2	5.5	1	4.5
width	1.5	1	1.5	1	.5	.5	1	1	1



- ✓ Plot your experience.
- ✓ Mystery flower above (???) lands closer to reds, so decision: red.

Machine Learning using Neural Networks (NNs)

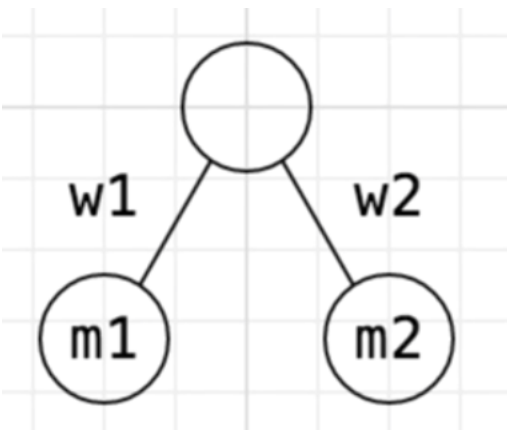
- ✓ Neural nets can do this classification for us w/o any plotting or such.

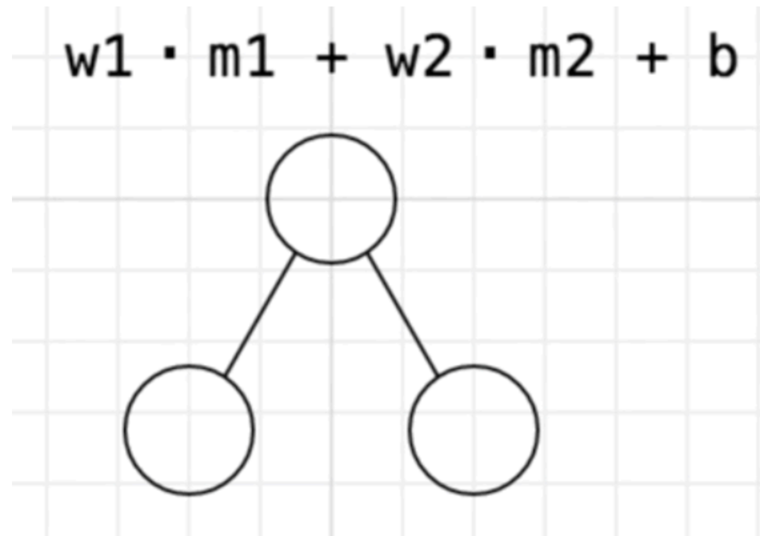


- ✓ Feed the input (width, length) to our net (bottom) and get an output as their weighted combination (top). If closer to 1, net tells us it is red.
- ✓ Currently net is wrong (‘cos 2 & 1 from a blue flower). Adjust weights.

Weights of the NN

- ✓ Output is a weighted (w1 and w2) combination of the input.



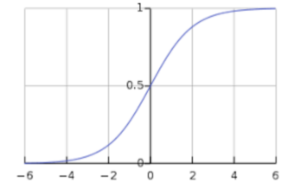


- ✓ Adjust these weights and the bias term (b) to make your net behave the way you want.
- ✓ We want: respect the input-output pairs we provide (train with):
  - ✓ 2 & 1 → Blue (so output ≤ 0.5), 5.5 & 1 → Red (so output > 0.5), etc.

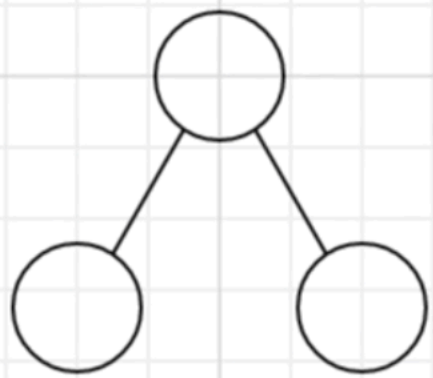
color	○ (red)	○ (blue)	○ (red)	○ (blue)	○ (red)	○ (blue)	○ (red)	○ (blue)
length	3		4	3	3.5	2	5.5	1
width	1.5		1.5	1	.5	.5	1	1

Weights of the NN

✓ Squash the values to be in [0,1]: $\text{sigmoid}(x) = \dfrac{1}{1 + e^{-x}}$



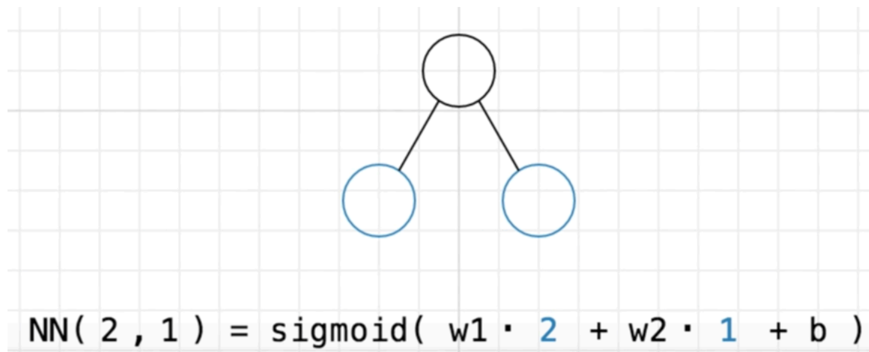
$$\text{sigmoid}(w1 \cdot m1 + w2 \cdot m2 + b)$$



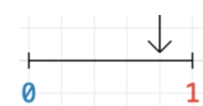
Weights of the NN

- ✓ Weights and biases start randomly (to be adjusted later).

color	○	○	○	○	○	○	○	○	???
length	3	2	4	3	3.5	2	5.5	1	4.5
width	1.5	1	1.5	1	.5	.5	1	1	1



- ✓ w1 = .5, w2 = .2, b = .3 ➔ NN(2,1) = sigmoid(1.5) = .8

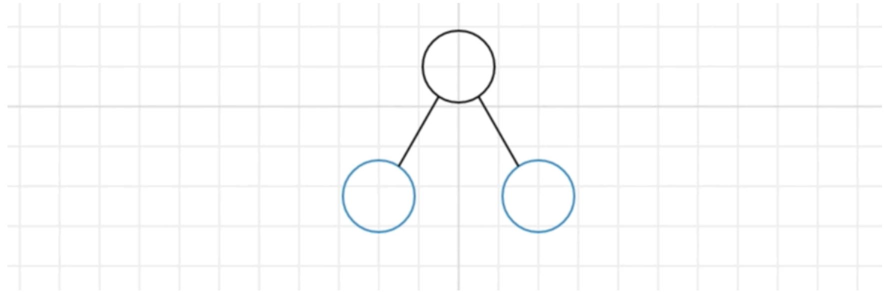


- ✓ NN thinks it is red; we'd have preferred output to be closer to 0 ☹.
- ✓ Solution: adjust weights & biases (via Backpropagation method).
  - ✓ Cost function: $(\text{prediction - target})^2 = (.8 - 0)^2$
  - ✓ Since prediction depends on weights & biases variables, take partial derivative w.r.t. those (gradient descent) and get the adjustment that minimizes the cost.

Weights of the NN

✓ Weights and biases start randomly (to be adjusted later).

color	○	○	○	○	○	○	○	○	???
length	3	2	4	3	3.5	2	5.5	1	4.5
width	1.5	1	1.5	1	.5	.5	1	1	1

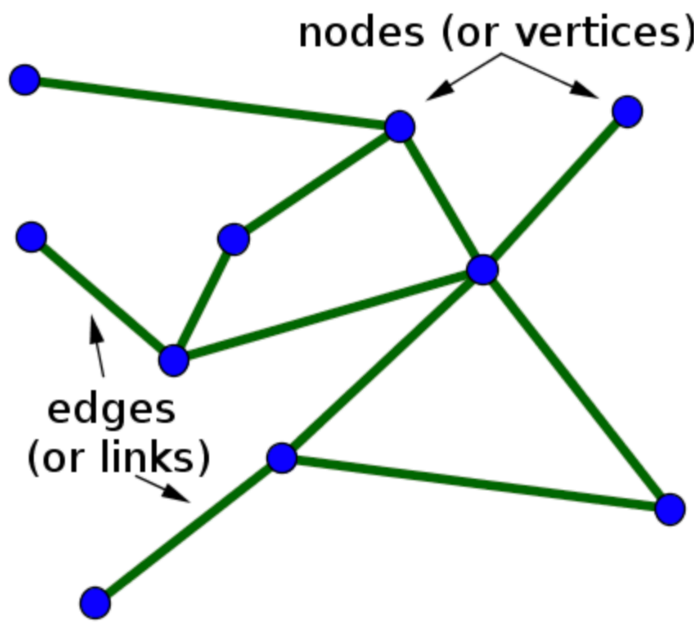


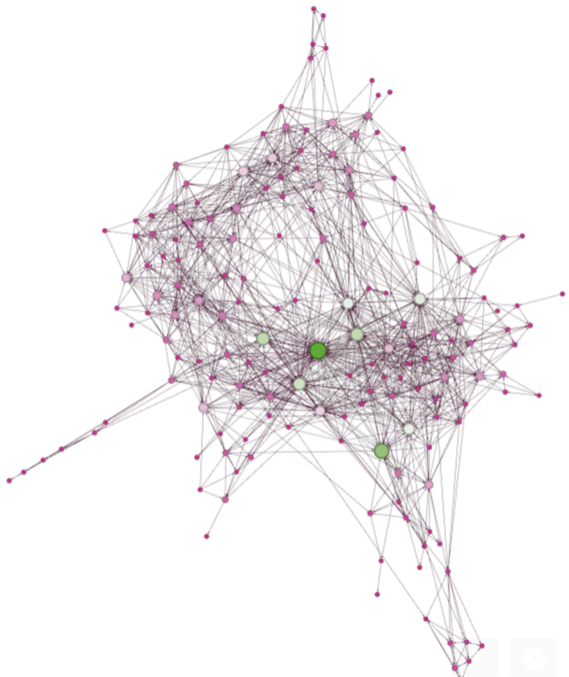
✓ Once all weights & biases are adjusted based on the observed data, we essentially constructed our model (NN).

✓ Feed the parameters of the new flower to this constructed model in order to classify it instantly (and hopefully accurately).

Graph

✓ Graph: set of vertices and edges that model many problems in CS.



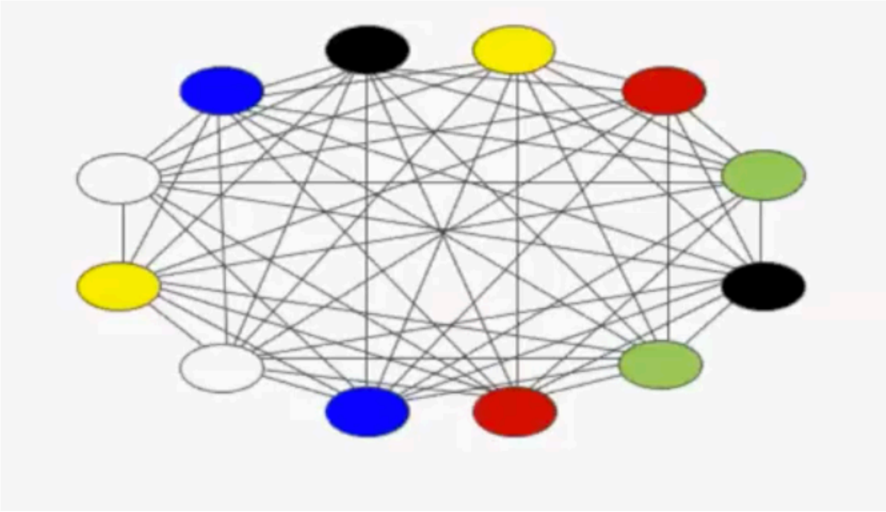


✓ Footballers (vertices) are connected (edges) if they played at the same team anytime in their careers.

✓ People are connected if they are friends, e.g., Facebook network.

Graph Coloring

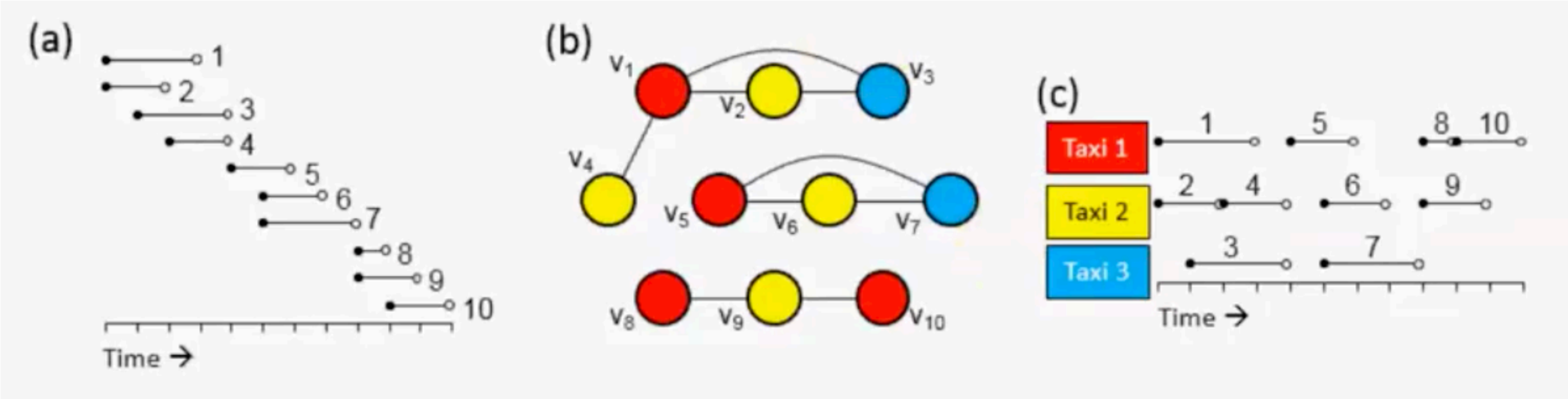
- ✓ Assignment of colors to vertices s.t. neighbor verts've different colors.



- ✓ Use as few colors as possible (chromatic number).
- ✓ Why do we care?

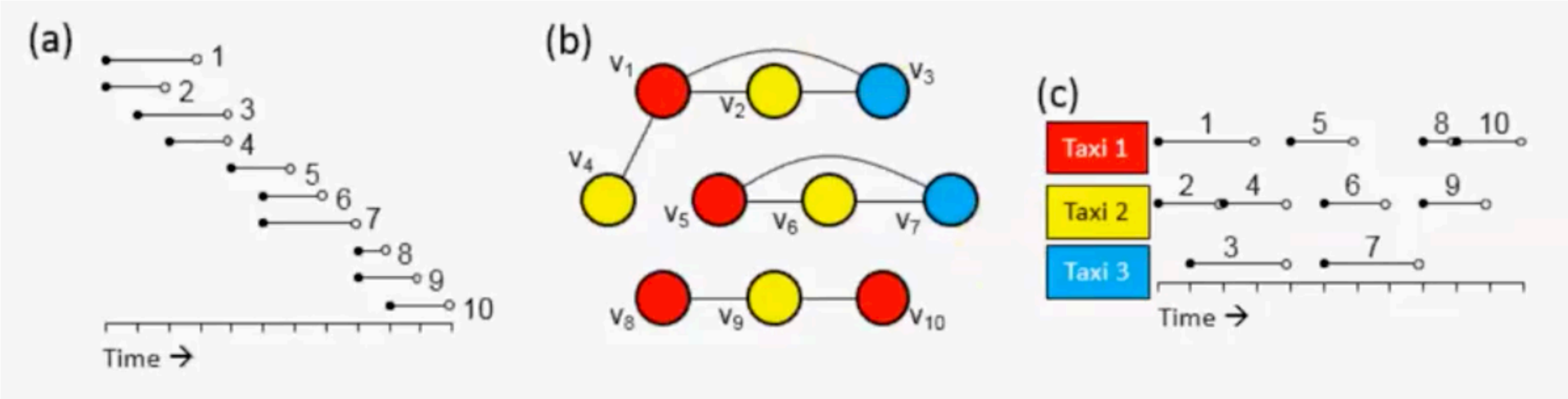
Graph Coloring for Scheduling

- ✓ Series of taxi journeys with a start time (filled) and an end time (empty).
- ✓ A taxi cannot be used on another journey until it returns.
- ✓ 10 taxis obviously suffice to serve these requests but expensive ☹.
- ✓ Can do with just 3 ☺.
- ✓ Graph: journey (vertex), overlapping in time (edge).



Graph Coloring for Scheduling

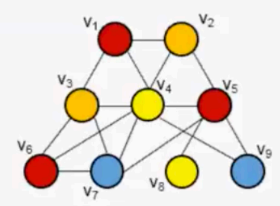
- ✓ Series of flights with a start time (filled) and an end time (empty).
- ✓ A gate cannot be used while occupied by a plane.
- ✓ 10 gates obviously suffice to serve these requests but expensive ☹.
- ✓ Can do with just 3 ☺. Min # of gates for these flights is 3.
- ✓ Graph: flight (vertex), overlapping in time (edge).



Graph Coloring for Scheduling

- ✓ Schedule exams for courses.
- ✓ Two courses clash if some student taking them both.
- ✓ 9 timeslots obviously suffice to serve these requests but expensive ☹.
- ✓ Can do with just 4 ☺.
- ✓ Graph: course (vertex), clashing (edge).

Lecture Name	Clashes with...
1) Algebra	2, 3, 4
2) Probability Theory	1, 4, 5
3) Graph Theory	1, 4, 6, 7
4) French Literature	1, 2, 3, 5, 6, 7, 9
5) English Literature	2, 4, 7, 8, 9
6) Welsh Literature	3, 4, 7
7) European Economics	3, 4, 5, 6
8) Algorithms	5
9) C++ Programming	4, 5



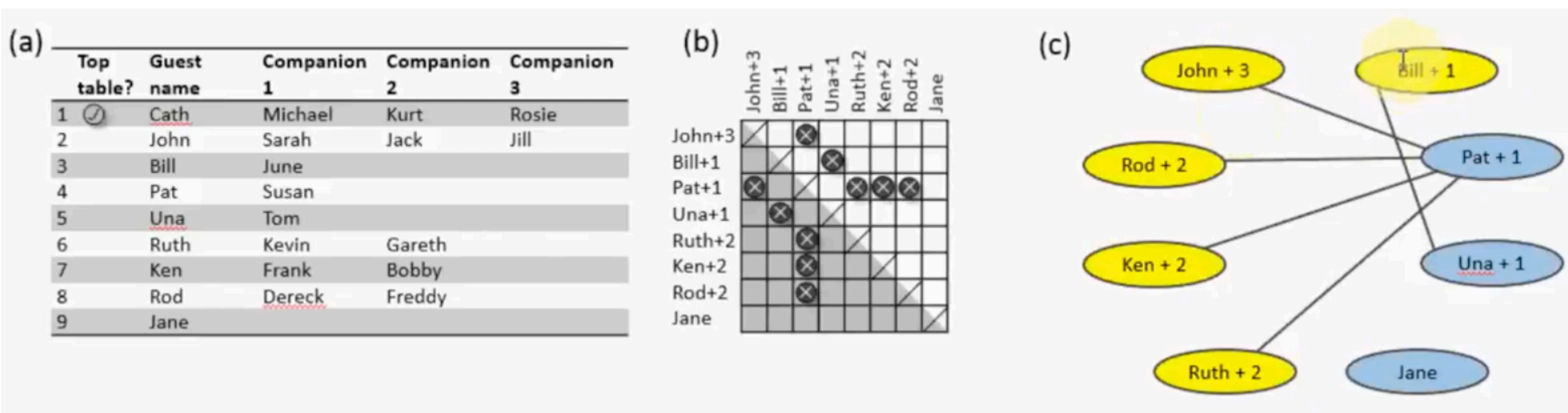
Timeslot 1	2	3	4
Lecture 1 Algebra	Lecture 2 Probability Theory	Lecture 7 European Economics	Lecture 4 French Literature
Lecture 5 English Literature	Lecture 3 Graph Theory	Lecture 9 C++ Programming	Lecture 8 Algorithms
Lecture 6 Welsh Literature			

Graph Coloring for Scheduling

- ✓ Separate cages in a zoo.
- ✓ Two species may not get along together.
- ✓ Graph: animals (vertex), hating (edge). Min # of cages.

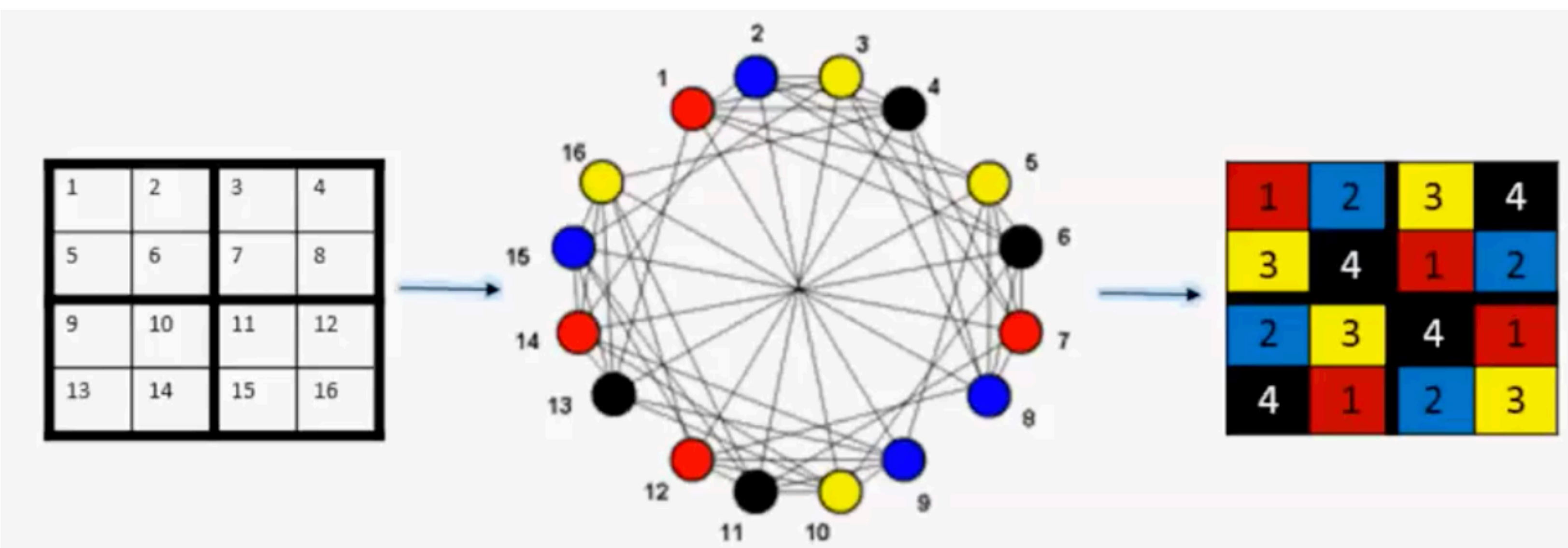
Graph Coloring for Scheduling

- ✓ Design seating plans for weddings.
- ✓ Some people do not want to seat together (drama).
- ✓ 9 tables obviously suffice to serve these requests but expensive ☹.
- ✓ Can do with just 2 ☺.
- ✓ Graph: party (vertex), hating each other (edge).



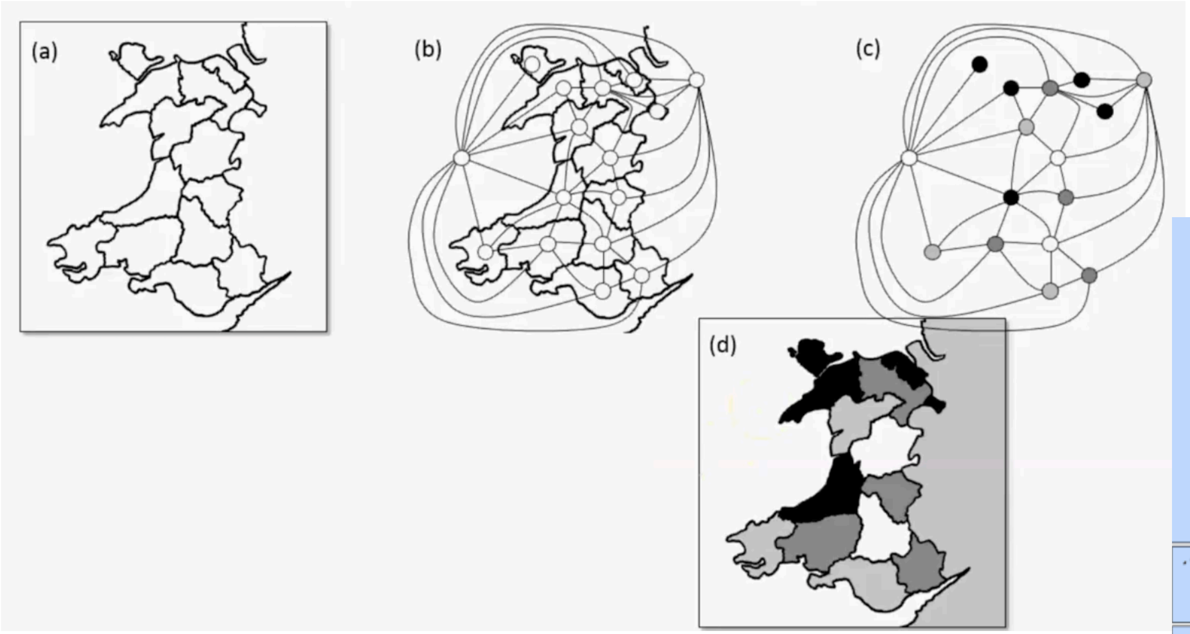
Graph Coloring for Sudoku

- ✓ Solve Sudoku puzzles.
- ✓ Fill in the blank cells s.t. each row, col, and 2x2 box has 1-4 just once.
- ✓ Graph: cell (vertex), same row, col, or box (edge).
- ✓ 4-coloring of this graph corresponds to a Sudoku solution.
- ✓ Some cells filled already (clues) = some vertices already colored for u.

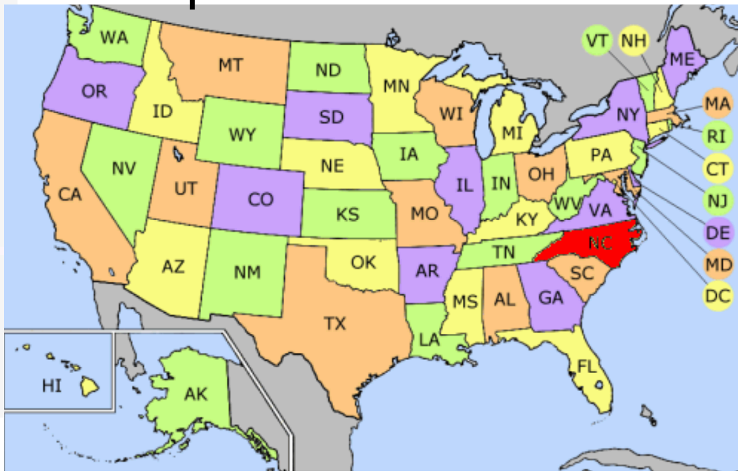


Graph Coloring for Maps

- ✓ Color maps to separate neighboring regions robustly.
- ✓ Theorem: 4 colors suffice for all possible maps.
- ✓ Graph: region (vertex), neighboring (edge).



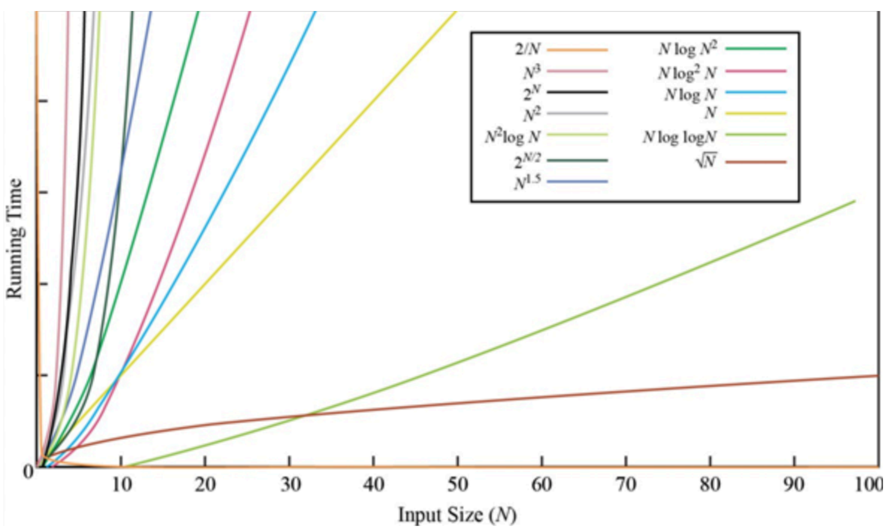
Suboptimal: 5 colors used.



Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Exact solution: Check each of the k^n assignments of k colors to n vertices for legality. Repeat for $k = 1,2,..,n-1$.
- ✓ Too slow ‘cos this is a brute-force exponential solution.
  - ✓ Growth-rates of functions.

Function	n: 10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$



$O(n^2)$ algorithm	$2n^2$ instructions	10^9 inst/second	2000 secs.
$O(n \lg n)$ algorithm	$100\, n \lg n$ instructions	10^7 inst/second	60 secs.

Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Approximate solution: Based on heuristics. No optimality guarantees.
- ✓ This is where machine learning comes in.
  - ✓ Some heuristics goods for some graphs.
  - ✓ Train a neural network: input graph G1, output Heuristic2.
    input graph G7, output Heuristic1.
    .
    .
    input graph G166, output Heuristic 2.
    .
    .
  - ✓ Given a query graph, decide the best heuristic for it and apply it.

Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Previously, we used flower features (width, length) to flower color (red, blue) mappings to train our NN. Then, decided the color of a new query point using this NN.
- ✓ Now, use graph features* to preferred heuristic (H1, H2) maps to train the NN and decide the better heuristic for a new query graph.

* 13 features measured on graphs:

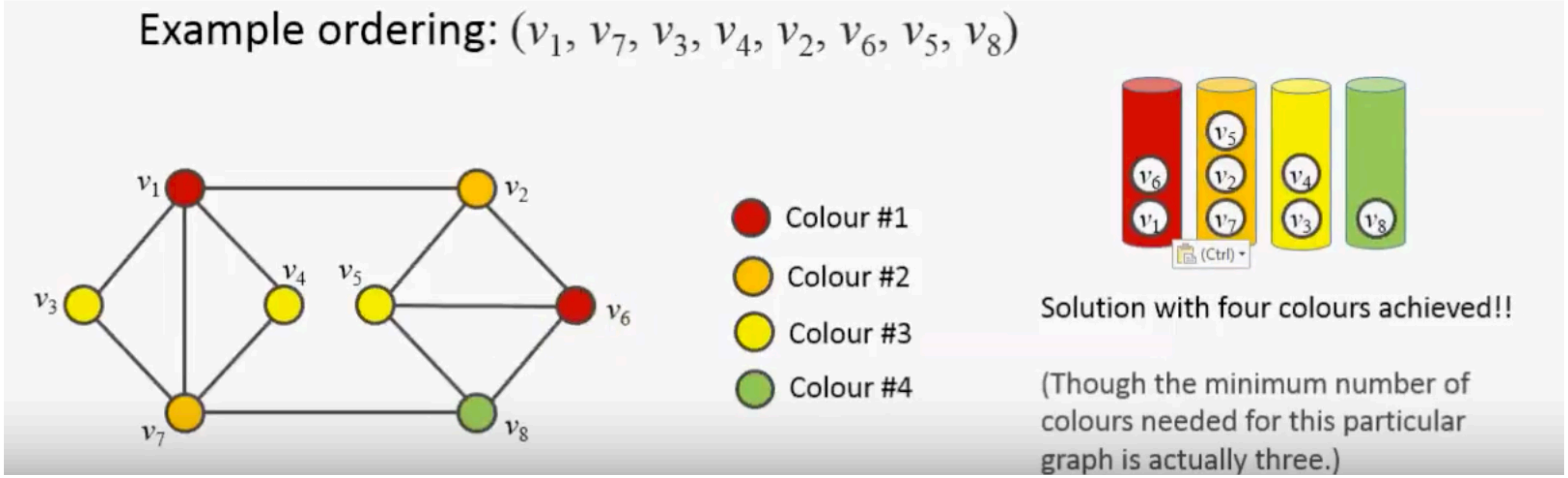
1: **no. of nodes:** n
2: **no. of edges:** m
3,4: **ratio:** $\frac{n}{m}, \frac{m}{n}$
5: **density:** $\frac{2 \cdot m}{n \cdot (n-1)}$

6-13: **nodes degree statistics:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy

** Heuristic preference: run both heuristics on each training graph and pick the one using fewer colors (or taking less time).

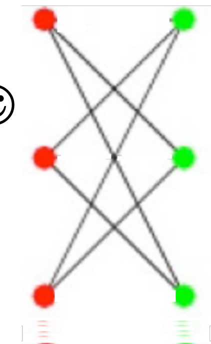
Graph Coloring Algorithms

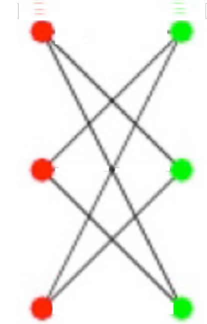
- ✓ So how to solve this problem?
- ✓ Heuristic 1: order vertices arbitrarily v_1, v_2, .., v_n. You have available colors c_1, c_2, .., c_n.
- ✓ For i=1 to n: Color v_i with the lowest legal color c_j //make it optimal by calling this loop n! times for each possible ordering. (O(n)->O(n!)).



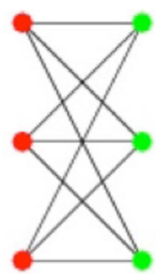
Graph Coloring Algorithms

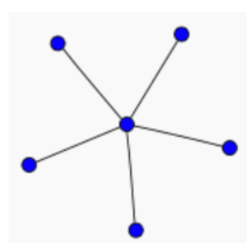
- ✓ So how to solve this problem?
- ✓ Heuristic 1: order vertices arbitrarily v_1, v_2, .., v_n. You have available colors c_1, c_2, .., c_n.
- ✓ For i=1 to n: Color v_i with the lowest legal color c_j //make it optimal by calling this loop n! times for each possible ordering. (O(n)->O(n!)).
- ✓ Bad ordering: left-right-down-left-right-down-.. → n/2 colors ☹
- ✓ Good ordering: left-down-left-down-..right-down-.. → 2 colors ☺





- ✓ Always optimal regardless of ordering (2 colors):



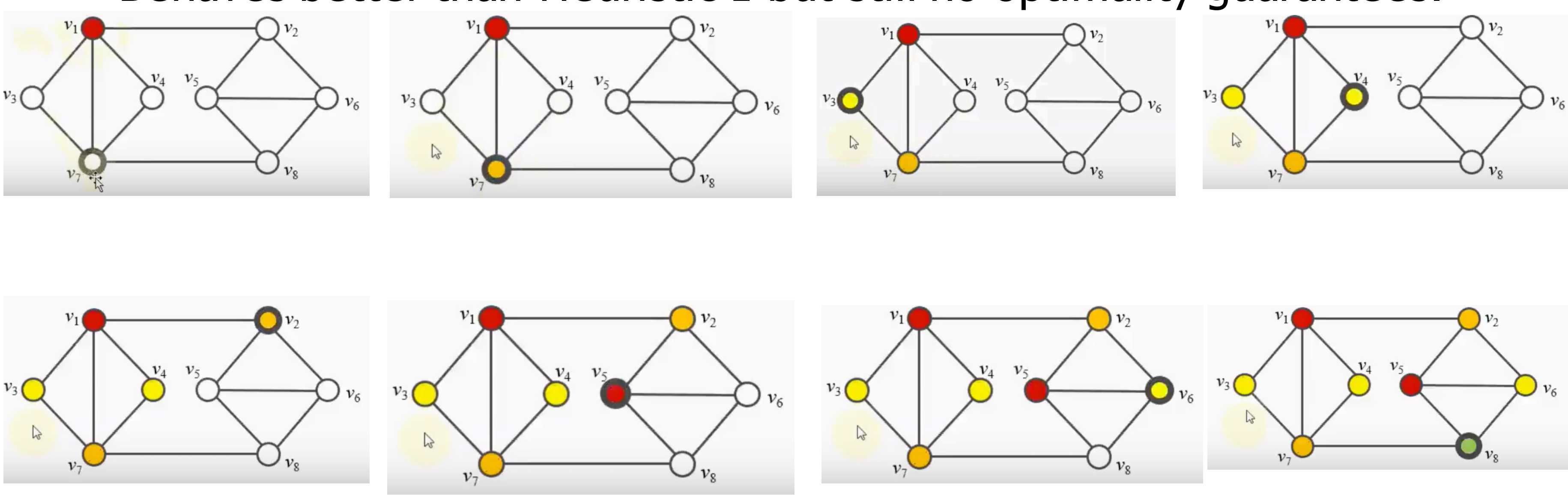


Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Heuristic 1: order vertices arbitrarily v_1, v_2, .., v_n. You have available colors c_1, c_2, .., c_n.
- ✓ For i=1 to n: Color v_i with the lowest legal color c_j //make it optimal by calling this loop n! times for each possible ordering. (O(n)->O(n!)).
- ✓ Upper bound on # of colors to be used: d+1, if max degree is d.
- ✓ Proof:
  - ✓ Basis: 1-vertex graph (max degree is d=0) requires 0+1=1 color. Done.
  - ✓ Induction: Assume statement is True for all n-vertex graphs. Show also True for n+1-vertex graphs. Here I show: $v_1,v_2,v_3,......,v_n,v_{n+1}$.
  - ✓ Red subgraph has n vertices and max degree ≤ d, so by induction it uses at most d+1 colors.
  - ✓ For v_{n+1}, even if all its neighbors (at most d neighbors) have different colors (worst-case), pick the (d+1)th color for v_{n+1}. Done.

Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Heuristic 2: Choose the uncolored vertex w/ the highest # of different neighbor colors and color it legally. Break ties by choosing the vertex w/ the highest degree.
- ✓ Behaves better than Heuristic 1 but still no optimality guarantees.



Graph Coloring Algorithms

- ✓ So how to solve this problem?
- ✓ Heuristic 2: Choose the uncolored vertex w/ the highest # of different neighbor colors and color it legally. Break ties by choosing the vertex w/ the highest degree.
- ✓ Behaves better than Heuristic 1 but still no optimality guarantees.
- ✓ Always optimal regardless of ordering (3 colors, 2 colors):