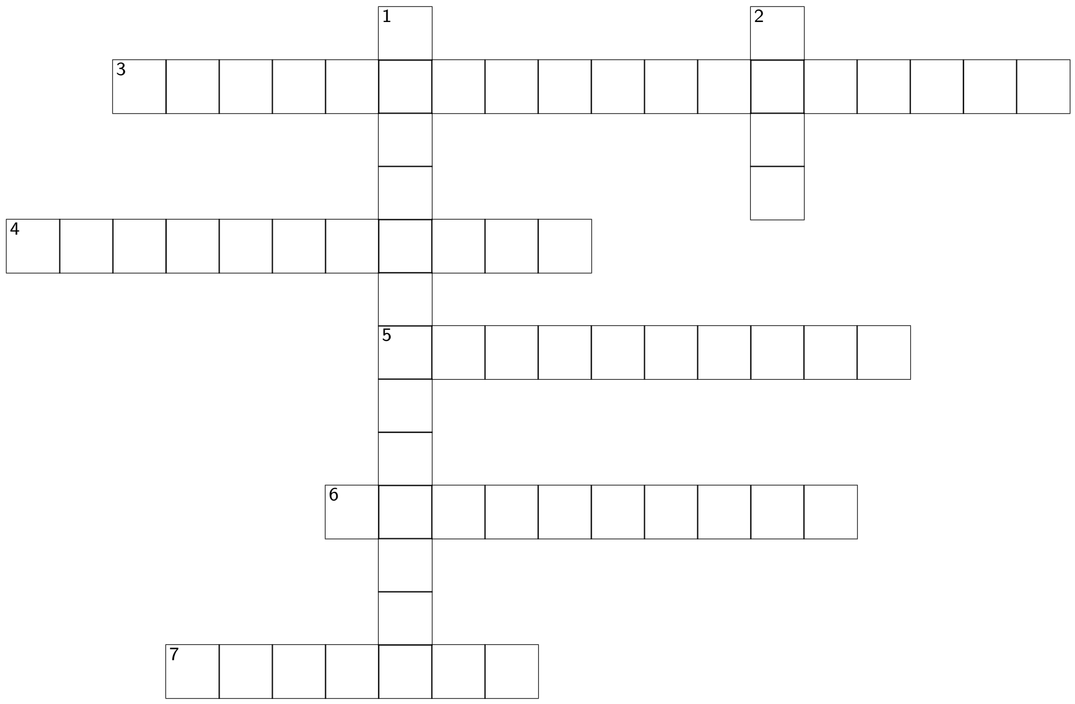
1 Missing in the Mountains

The 15-281 Course Staff decided to climb the Rocky Mountains together over Winter Break. On their way back down from the mountains, they realize they left Harlene at the top of the tallest mountain! They have no idea where on the mountain they are or which mountain they are on, and are worried about how they will find Harlene before lecture on Monday. Help the 281 Staff remember all of the local search algorithms they have learned so they can save Harlene!

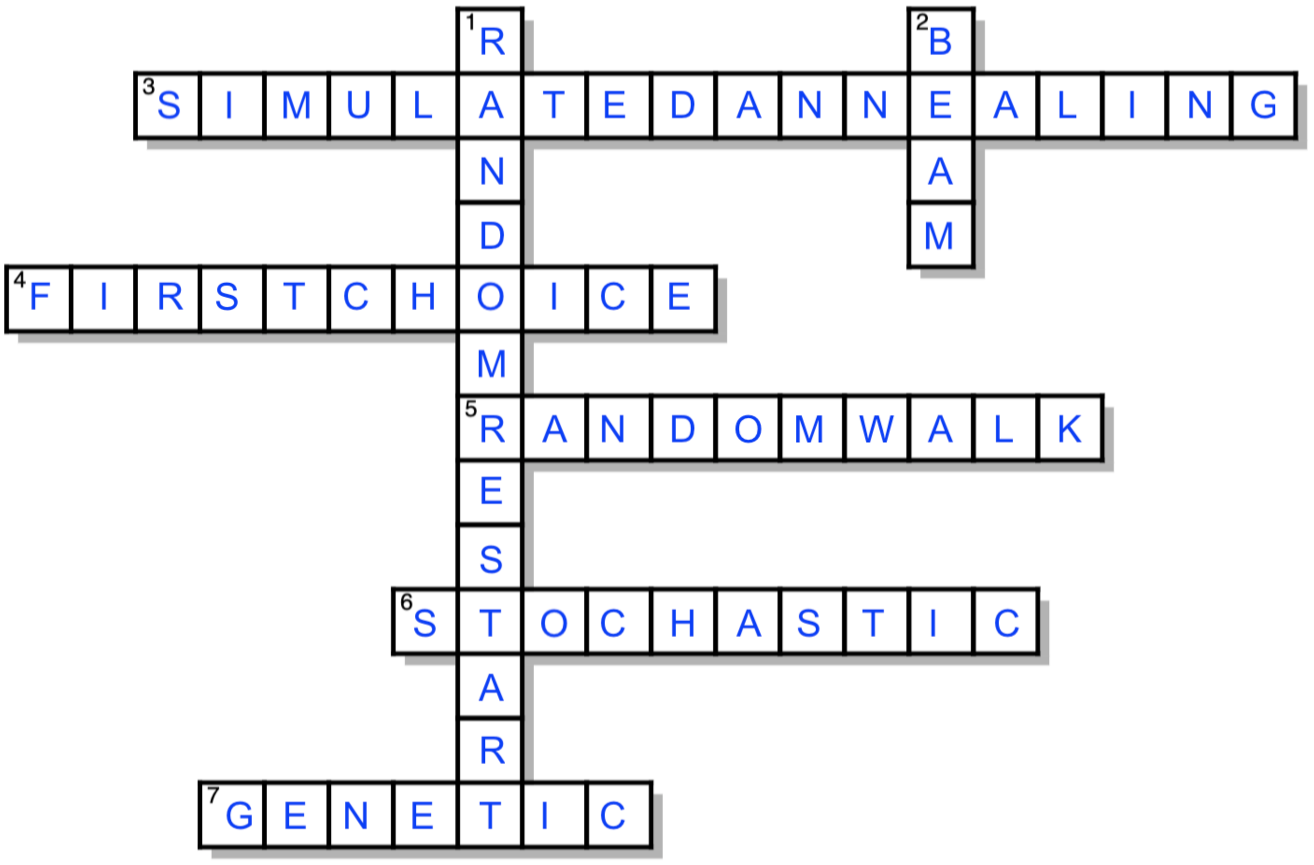


Down

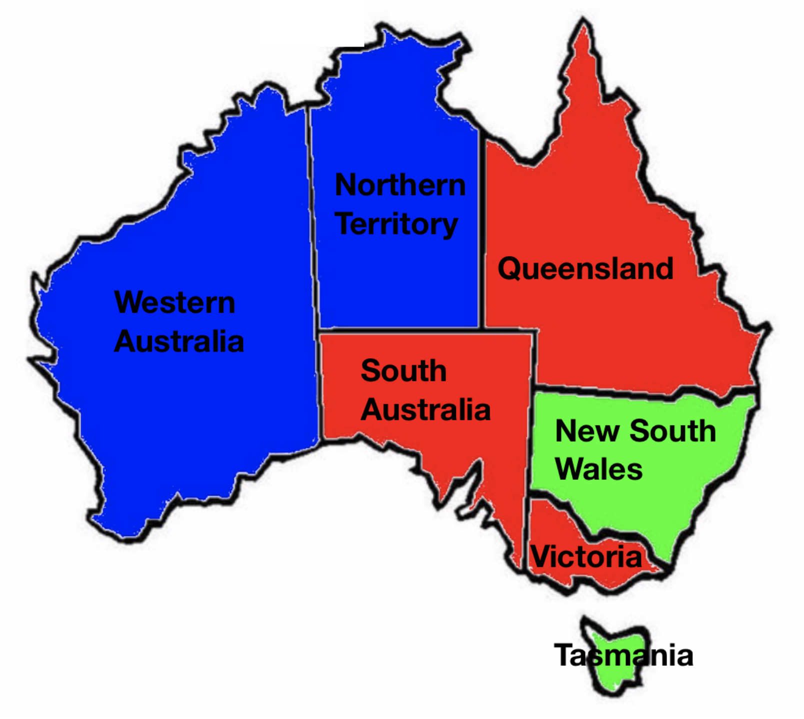
1. A variant of hill-climbing where you conduct a series of searches from randomly generated starting states until the goal is found.
2. While keeping track of k states, generate all successors and retain the best k successors of all of them.

Across

3. A local search technique where you allow for downhill moves but make them rarer as time goes on.
4. A variant of hill-climbing where you generate successors randomly (one by one) until a better one is found.
5. A local search technique where you uniformly randomly choose a neighbor to move to.
6. A variant of hill climbing in which you choose a move randomly from the uphill moves, with the probability of a move being chosen dependent on the "steepness" (amount of improvement from making that move).
7. A variant of stochastic beam search where successors are generated by combining two parent states instead of modifying a single state.


1 RANDOMRESTART
2 BEAM
3 SIMULATEDANNEALING
4 FIRSTCHOICE
5 RANDOMWALK
6 STOCHASTIC
7 GENETIC

2 Map Coloring with Local Search



Recall the various local search algorithms presented in lecture. Local search differs from previously discussed search methods in that it begins with a complete, potentially conflicting state and iteratively improves it by reassigning values. We will consider a simple map coloring problem, and will attempt to solve it with hill climbing.

(a) How is the map coloring problem defined (In other words, what are variables, domain and constraints of the problem)? How do you define states in this coloring problem?

- Variables: WA, NT, SA, Q, NSW, V, T (States in Australia)

- Domain: Green, Red, Blue

- Constraints: Adjacent countries can't have the same color assignment.
  e.g: Implicit: $WA \neq NT$
  Explicit: (WA, NT) $\in$ (red, blue), (red, green), (blue, red), (blue, green), (green, red), (green, blue)

- Problem state: a full coloring of the map (i.e., color assignments to all variables).

(b) Given a complete state (coloring), how could we define a neighboring state?

A neighboring state could be a full coloring of the graph with a different color assignment to only one variable.

(c) What could be a good heuristic be in this problem for local search? What is the initial value of this heuristic?

The heuristic could be the number of variable pairs that have conflicting colors. In the initial state, the following 3 pairs (WA-NT, Q-SA, SA-V) are conflicting, so the heuristic h = 3. (Note: there could be other possible heuristics for this problem.)

(d) Use hill climbing to find a solution based on the coloring provided in the graph.

Let h be our heuristic value.
In the original graph, we have 3 coloring conflicts as stated in (c). Depending on the search order, the assignment order might be different and the searched path lengths as well as coloring can also vary. We represent the coloring of states in a list with the following order: [WA, NT, Q, SA, NW, V, T]. Below are two examples of potential search paths.

- Step 1: h = 3. Conflicts are WA-NT, Q-SA, SA-V. We start with WA-NT. Coloring NT with Green would resolve the WA-NT conflict. Coloring: [B, G, R, R, G, R, G]
  Step 2: h = 2. Conflicts are Q-SA, SA-V. We can pick SA-Q pair and assign Blue to SA, which would resolve SA-Q and SA-V conflicts but will add coloring conflict for WA-SA pair. Still, it decreases the number of conflicts and is a better neighboring state. Coloring: [B, G, R, B, G, R, G]
  Step 3: h = 1. Conflict is just WA-SA pair. We can simply assign WA with Red to resolve this conflict, where we completed the search and found a solution to the problem. Coloring: [R, G, R, B, G, R, G]

We got pretty lucky in the search above and found a solution in 3 steps. However, local search may not always resolve conflicts optimally. Below is an example where it has to resolve the conflicts with more steps.

- Step 1: h = 3. Conflicts are WA-NT, Q-SA, SA-V. We start with WA-NT. Coloring WA with Green would resolve the WA-NT conflict. Coloring: [G, B, R, R, G, R, G]
  Step 2: h = 2. Conflicts are Q-SA, SA-V. We can resolve both of these conflicts by assigning Blue to SA, which will lead us to a better neighboring state with only one conflict: NT-SA. Coloring: [G, B, R, B, G, R, G]
  Step 3: h = 1. Conflicts are NT-SA. However, looking at all possible assignments to both of these two states, we see that no matter what color we assign to either one of them, we can't find a better neighboring state. Therefore the current iteration of hill climbing would end and we would restart from the initial state if we are applying random-restart hill climbing. An alternative is to use simulated annealing, which would allow us to sometimes move to states of higher heuristic value in order to escape local minima.

We see that in the second search, we need more steps to complete search. There are other possible search steps sequences depending on the choice on the order of conflicts to resolve, and the color assignment when resolving each conflict.

We can also use a genetic algorithm approach to find a solution to color the map. We could let the number of non-conflicting State pairs (in this case, WA-SA, NT-SA, Q-NT, SA-NW, Q-NW, NW-V) minus the number of conflicts in each state be our fitness function, and represent the variables in a list as above: [WA, NT, Q, SA, NW, V, T].

With a genetic algorithm, we first randomly select $2k$ full colorings of the graph with replacement (i.e. k pairs of parents), with probability proportional to their fitness function values. For each pair of parents, we would choose a cross over point $\in \{1...6\}$ to split the variable list in two halves and combine the color assignments with cross over to generate two offspring states.
(There are many other ways we could've implemented this crossover - this is just following the n-queens example from lecture.)

Then with a small probability, we could mutate the color assignments to the variables by one color, which means we would modify one of the values assigned to the variable list to a value in $\{Red, Green, Blue\}$. We would repeat the process until we find a good enough coloring that does not result in any conflicts within the map, or until it exceeds the max number of iterations.

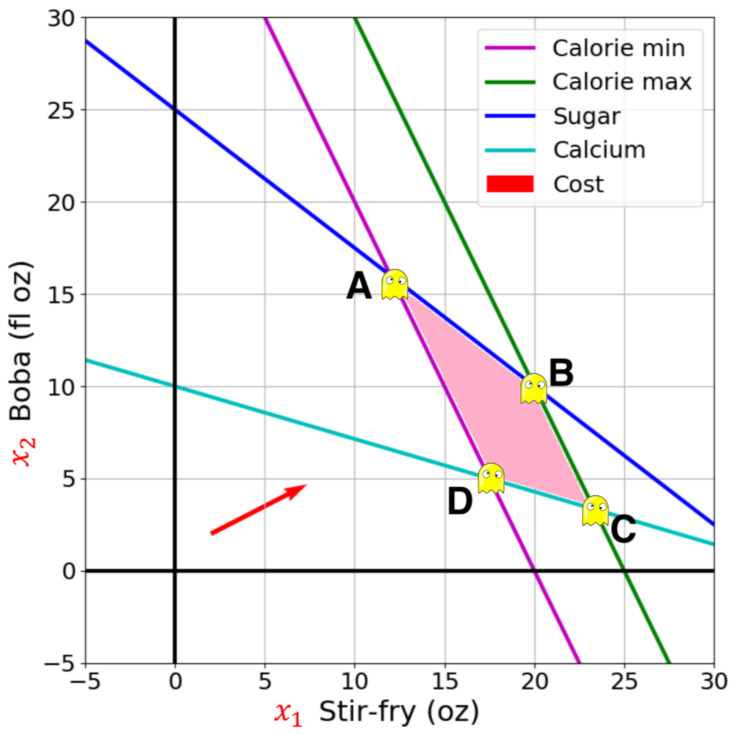
(e) How is local search different from tree search?

Tree search keeps all unexplored alternatives on the frontier, where local search does not have a frontier but only keeps improving a single option until there's no better neighboring states.

3 Algorithms for Solving Linear Programming

In Wednesday's lecture, we went through two algorithms for solving linear programming programs - vector enumeration and the simplex algorithm.

Recall the "Healthy Squad Goals" example from Wednesday's lecture. The goal is to minimize the cost, and the cost vector (red) is perpendicular to the purple and green lines.



1. Briefly describe both algorithms and explain how they differ. (hint: use terms such as vertices, intersections and neighbors).

   Vertex enumeration: Find all vertices of feasible region (feasible intersections), check objective value
   Simplex: Start with an arbitrary vertex. Iteratively move to a best neighboring vertex until no better neighbor found

   Intersection is found by solving a pair of constraints (although some constraint pairs might not intersect). A neighboring intersection differs by only one constraint.

2. Run simplex algorithm starting from point B. Now try running the algorithm starting from point C. How do their solutions differ?

   Starting from point B returns a solution of point A, while starting from point C returns a solution of point D. Notice that points A and D are equally optimal.

   Starting from point B, we have neighboring vertices point A and C. Point A is chosen as it is better. From point A, we have neighboring vertices point B and D. point B is worse than point A, and point D is equally optimal as point A. Thus, no better neighbor is found and the algorithm returns point A.

Starting from point C, we have neighboring vertices point B and D. Point D is chosen as it is better. From point D, the neighboring vertices A and C are either worse or equal to point D. Thus, no better neighbor is found and the algorithm returns point D.

4 Cargo Plane: Linear Programming Formulation

A cargo plane has three compartments for storing cargo: front, centre and rear. These compartments have the following limits on both weight and space:

Compartment	**Weight capacity (tonnes)**	**Space capacity (cubic metres)**
Front	10	6800
Centre	16	8700
Rear	8	5300

The following four cargoes are available for shipment on the next flight:

Cargo	**Weight (tonnes)**	**Volume (cubic metres/tonne)**	**Profit ($/tonne)**
C1	18	480	310
C2	15	650	380
C3	23	580	350
C4	12	390	285

Any proportion of these cargoes can be accepted. The objective is to determine how much of each cargo C1, C2, C3 and C4 should be accepted and how to distribute each among the compartments so that the total profit for the flight is maximised. **Formulate** the above problem as a linear program (what is the objective and the constraints?). Think about the assumptions you are making when formulating this problem as a linear program.

Variables:

We need to decide how much of each of the four cargoes to put in each of the three compartments. Hence let $x_{i,j}$ be the number of tonnes of cargo i (i=1,2,3,4 for C1, C2, C3 and C4 respectively) that is put into compartment j (j=1 for Front, j=2 for Centre and j=3 for Rear) where $x_{i,j} \geq 0; i = 1, 2, 3, 4; j = 1, 2, 3$.

(Note here that we are explicitly told we can split the cargoes into any proportions (fractions) that we like.)

Constraints:

1. We cannot pack more of each of the four cargoes than we have available.

$$x_{1,1} + x_{1,2} + x_{1,3} \leq 18$$

$$x_{2,1} + x_{2,2} + x_{2,3} \leq 15$$

$$x_{3,1} + x_{3,2} + x_{3,3} \leq 23$$

$$x_{4,1} + x_{4,2} + x_{4,3} \leq 12$$

2. The weight capacity of each compartment must be respected.

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} \leq 10$$

$$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} \leq 16$$

$$x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} \leq 8$$

3. The volume (space) capacity of each compartment must be respected.

$$480x_{1,1} + 650x_{2,1} + 580x_{3,1} + 390x_{4,1} \leq 6800$$

$$480x_{1,2} + 650x_{2,2} + 580x_{3,2} + 390x_{4,2} \leq 8700$$

$$480x_{1,3} + 650x_{2,3} + 580x_{3,3} + 390x_{4,3} \leq 5300$$

Objective: The objective is to maximise total profit, i.e.

maximise $310(x_{1,1} + x_{1,2} + x_{1,3}) + 380(x_{2,1} + x_{2,2} + x_{2,3}) + 350(x_{3,1} + x_{3,2} + x_{3,3}) + 285(x_{4,1} + x_{4,2} + x_{4,3})$

The basic assumptions are:

1. that each cargo can be split into whatever proportions/fractions we desire
2. that each cargo can be split between two or more compartments if we so desire
3. that the cargo can be packed into each compartment (for example if the cargo was spherical it would not be possible to pack a compartment to volume capacity, some free space is inevitable in sphere packing)
4. all the data/numbers given are accurate

Something also to note is that we can also solve this linear programming problem using one of the various tools available online. One in particular you may find interesting is Google's OR-tools (Click here!). If you want to see an example of this being used, we have written up a solution to this recitation problem, which can be found at this link: Linear Programming code. We do not require that you learn how to use these tools, but it is a cool resource if you want to check it out! On all homeworks and exams, you will be expected to solve it by hand if we ask you to.