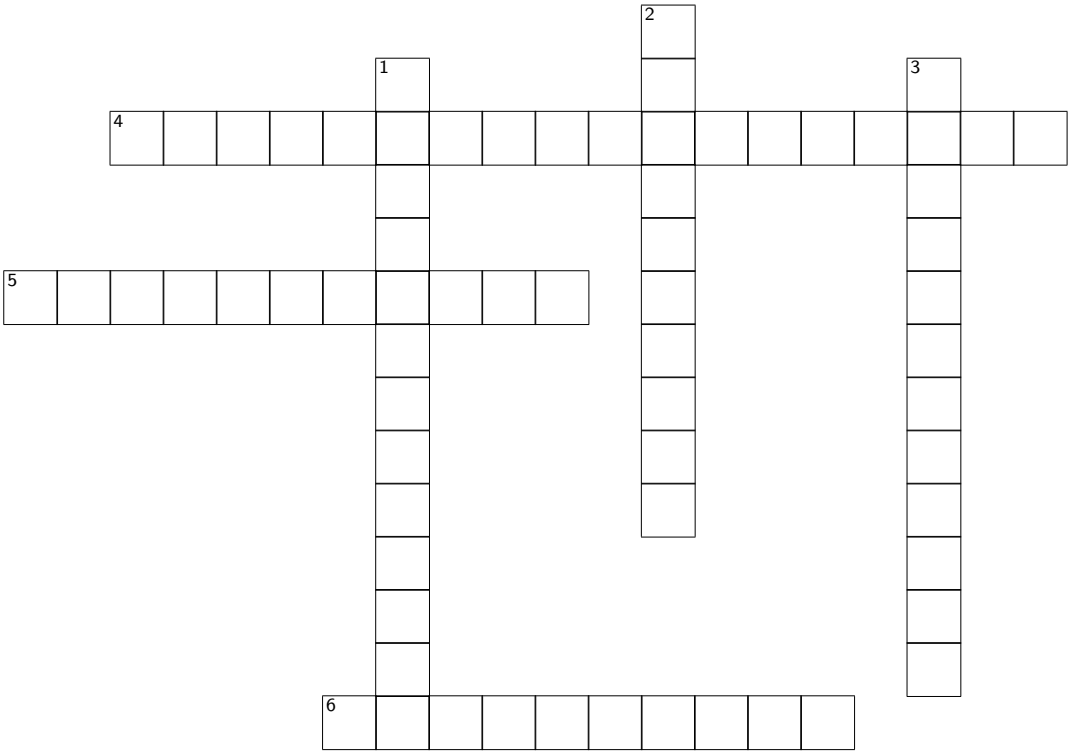
1 Missing in the Mountains

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 Simrit 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 Simrit before lecture. Help the 281 Staff remember all of the local search algorithms they have learned so they can save Simrit!

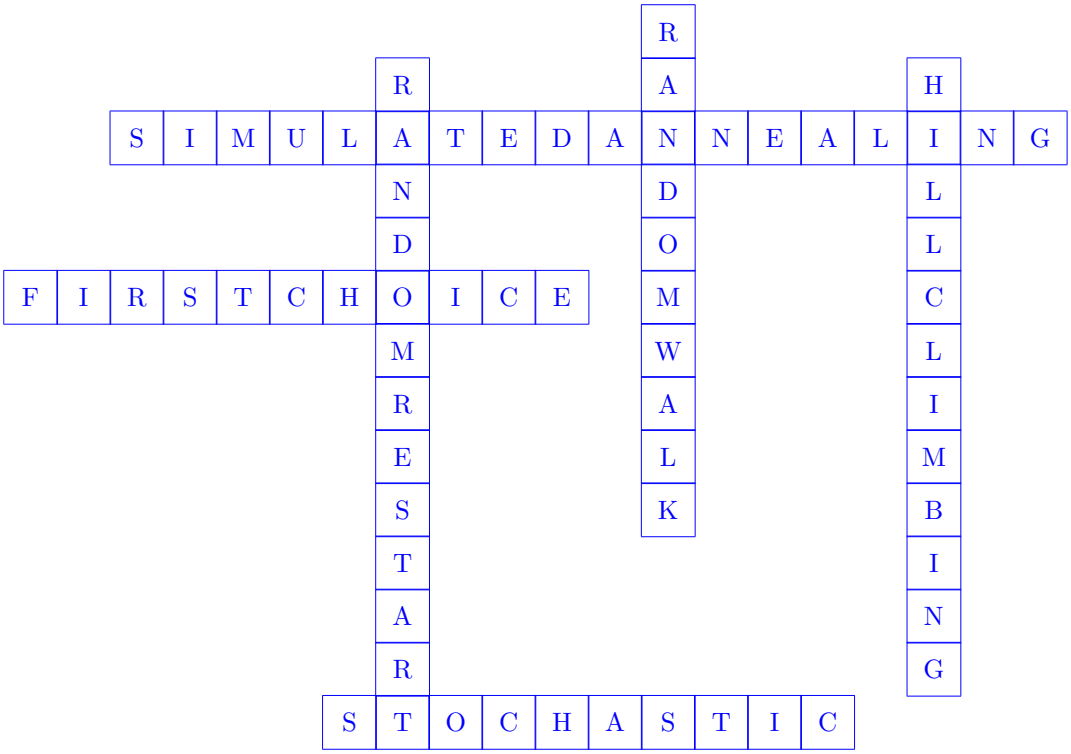


Down

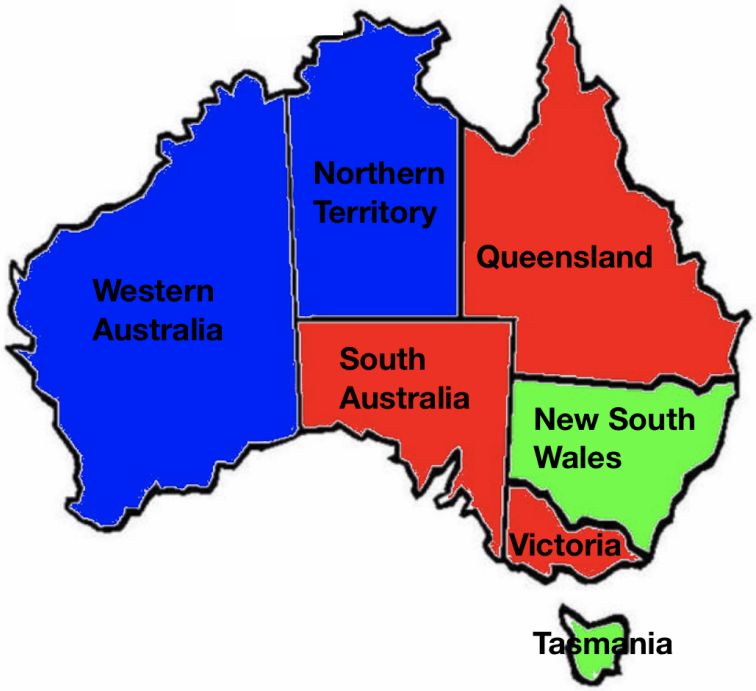
1. A variant of hill-climbing where you conduct a series of searches from randomly generated starting states until the goal is found.
2. A local search technique where you uniformly randomly choose a neighbor to move to.
3. A type of greedy local search where you move uphill to local maxima.

Across

4. A local search technique where you allow for downhill moves but make them rarer as time goes on.
5. A variant of hill-climbing where you generate successors randomly (one by one) until a better one is found.
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).


R
R A H
S I M U L A T E D A N N E A L I N G
N D L
D O L
F I R S T C H O I C E M C
M W L
R A I
E L M
S K B
T I
A N
R G
S T O C H A S T I C

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.

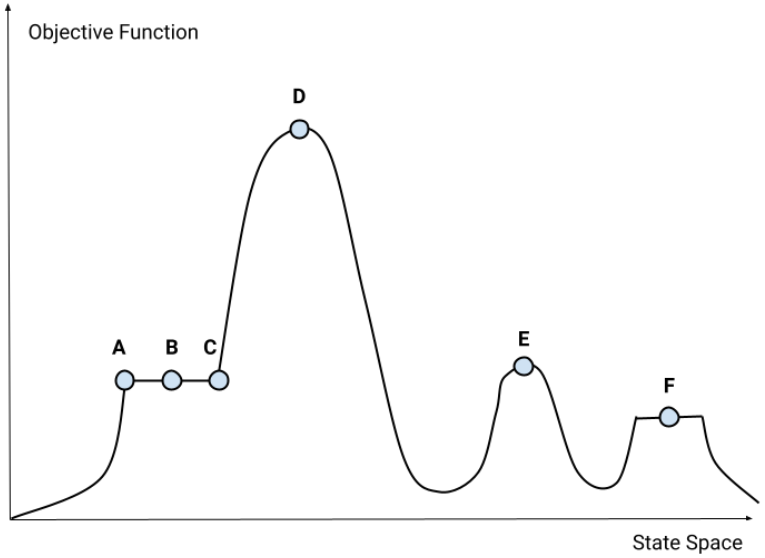
- Step 1: h = 3. Conflicts are WA-NT, Q-SA, SA-V. We start with WA-NT. Coloring WA with Green would resolve the WA-NT conflict. Coloring: [G, B, R, R, G, R, G]
- Step 2: h = 2. Conflicts are Q-SA, SA-V. We can resolve both of these conflicts by assigning Blue to SA, which will lead us to a better neighboring 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.

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

Tree search has a frontier while local search does not. Local search also never backtracks if it gets stuck.

3 Local Search Discussion Questions



Consider the state space above in the context of local search. Recall that our goal is to find the state that maximizes the objective.

1. Consider the points A, B, C, D, E, and F on the graph.

   (a) Which of the points on the graph are on a shoulder? Which of those points are local maximums?

   A, B, and C are on a shoulder. A and B are local maximums, but C is not.

   A is considered a local maximum since it does not have a neighbor with better objective value. We can use the same reasoning for B.

   C is not considered a local maximum because it has a neighbor to the right with a better objective value.

   (b) Which of the points on the graph are a "flat" local maximum?

   B and F are on a "flat" local maximum.

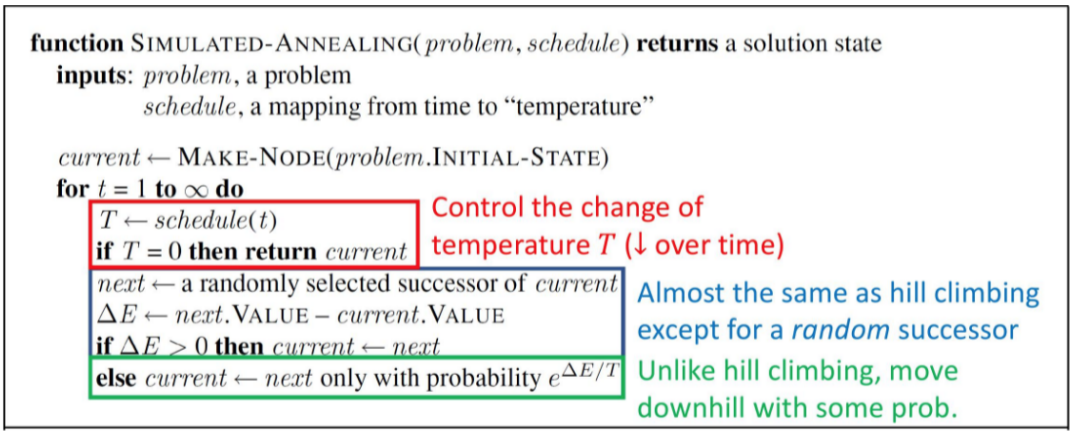
   (c) What is the difference between a shoulder and a "flat" local maximum?

   The key difference between a "flat" local maximum and a shoulder is that there is no uphill exit from a flat local maximum, whereas from a shoulder, uphill progress is technically possible from one of the endpoints of the shoulder.

2. Let's take a look at simulated annealing. Simulated Annealing is quite similar to hill climbing.

   - Instead of picking the best move, it picks a random move.
   - If the move improves the situation, the move is always accepted.

- Otherwise, it accepts the move with some probability less than 1



(a) How does the sign of ΔE reflect the "badness" of a move?

ΔE is negative for a "bad" move.

(b) In simulated annealing, we control the temperature T. How does the value of T impact the probability with which we choose a "bad" move?

0.5in Recall that the probability of choosing a "bad move" is $\frac{1}{e^{|\Delta E|/T}}$ since ΔE is negative for a bad move. For smaller values of T, our denominator is larger, so the probability of choosing a bad move is low. For larger values of T, our denominator is smaller, so the probability of choosing a bad move is high.

3. Mark True or False for each of the following statements.

(a) Regular hill climbing is optimal (i.e., will always find the global maximum)

False. Regular hill climbing, a.k.a. greedy local search, is not optimal since it may choose a local optima as the solution.

(b) Random restart hill climbing is optimal when given an infinite amount of time.

True. Random restart hill climbing is optimal since it will eventually restart at an initial state that allows it to find the global optimum.

(c) Simulated annealing allows for downward moves according to some fixed constant temperature T.

False. Simulated annealing allows for downhill moves according to a decreasing temperature T in order to make them rarer as time goes on.

(d) Simulated annealing is generally less time efficient than random walk.

False. Simulated annealing combines the efficiency of hill climbing with the completeness of random walk, making it generally much faster than random walk on its own.

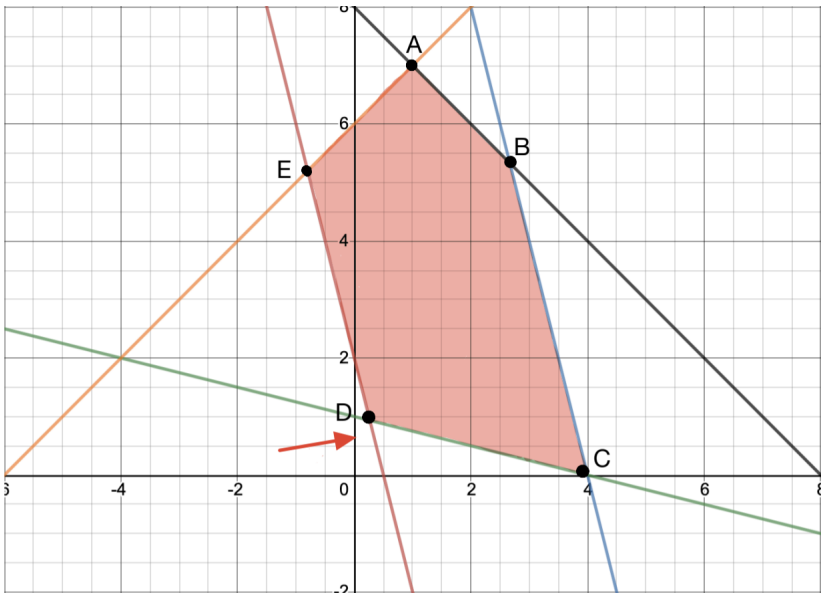
(e) A random walk algorithm is more likely to choose a better neighbor than a worse one.

False. A random walk search does not consider the optimality of the neighbors at all.

4 Algorithms for Solving Linear Programming

In lecture, we went through two algorithms for solving linear programming programs - vertex enumeration and the hill climbing algorithm.

Consider this linear programming problem. The goal is to minimize the cost, and the cost vector (red) is perpendicular to the blue and red 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.

   Hill climbing: Start with an arbitrary vertex. Iteratively move to a best neighboring vertex until no better neighbor is 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 the hill climbing 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 E, while starting from point C returns a solution of point D. Notice that points E 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 E and B. Point B is worse than point A, and point E is better point A. From point E, we have neighboring vertices point A and D. Point A is worse than point E, and point D is equally optimal. Thus, no better neighbor is found and the algorithm returns point E.

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.

5 Cargo Plane: Linear Programming Formulation

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

Compartment	**Weight capacity (tons)**	**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 (tons)**	**Volume (cubic metres/ton)**	**Profit ($/ton)**
C1	18	480	310
C2	15	650	380
C3	23	580	350
C4	12	390	285

Any proportion of these cargos 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 cargos 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 Center 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 cargos into any proportions (fractions) that we like.)

Constraints:

1. We cannot pack more of each of the four cargos 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.

If you were to put this linear program into standard form, what would be the dimensions of $A, \mathbf{b}, \mathbf{c}, \mathbf{x}$?

There are 12 variables, so $\dim \mathbf{x} = 12 \times 1$ and $\dim \mathbf{c} = 12 \times 1$. There are 10 constraints, so $\dim \mathbf{b} = 10 \times 1$ and since each constraint involves 12 variables, $\dim A = 10 \times 12$. Note that while A has 120 elements, most of them will be 0 as each constraint only involves a few of the variables, making it a sparse matrix.

Now consider a simpler problem. There is a cargo plane with a single compartment with limit on 20 tons weight and 2400 cubic meters limit on space. You want to use this cargo plane to transport boxes of oranges and pineapples to sell in a market overseas.

Your goal is to maximize the number of gold pieces under following constraints:

- The market only allows each person to sell 14 boxes.
- 1 box of oranges has weight 1 ton and volume of 100 cubic meters.
- 1 box of pineapples has weight 2 tons and volume of 300 cubic meters.
- You earn 5 gold pieces for 1 box of oranges.
- You earn 12 gold pieces for 1 box of pineapples.

We will now formulate and solve the LP.

1. Write the LP in inequality form.
2. Graph the constraints, cost vector, and at least 3 cost contours. Indicate the feasible region.
3. What is the **optimal number** of boxes of oranges and pineapples? How much gold does this earn?

Formulating problem as linear programming formulation, we have have following cost function and constraints where box of orange is x_1 and box of pineapples is x_2.

$$\begin{gathered}\text{Minimize } -5x_1 - 12x_2 \text{ where} \\ -x_1 \leq 0 \\ -x_2 \leq 0 \\ x_1 + x_2 \leq 14 \\ x_1 + 2x_2 \leq 20 \\ 100x_1 + 300x_2 \leq 2400\end{gathered}$$

Putting this problem in inequality form we have:

$$\min_{\mathbf{x}} \mathbf{c}^\top \mathbf{x} \; s.t. \; A\mathbf{x} \leq \mathbf{b}$$

where

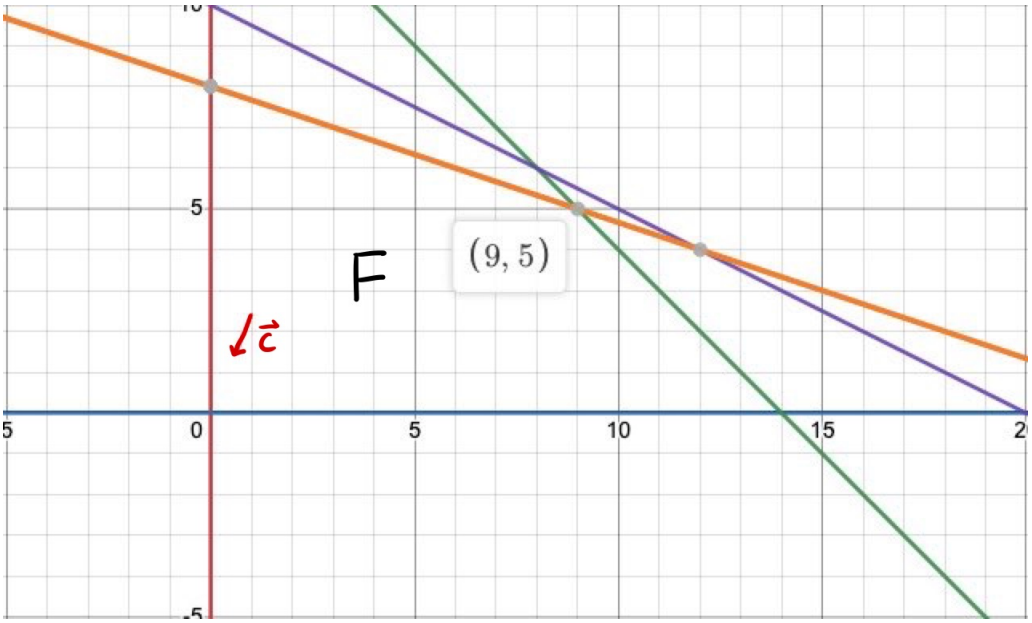
$$\mathbf{c} = [-5 \; -12]^\top$$

$$\mathbf{x} = [x_1 \; x_2]^\top$$

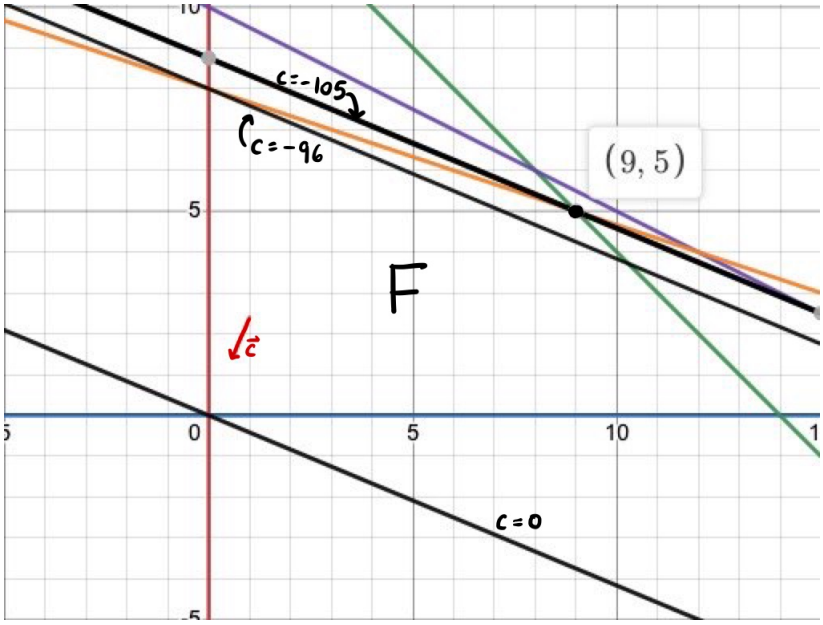
$$\mathbf{b} = [0 \; 0 \; 14 \; 20 \; 2400]$$

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \\ 1 & 2 \\ 100 & 300 \end{bmatrix}$$

We can solve these constraints by either drawing a graph and find intersection of constraints, or through systems of equations. Below is the graph of constraints with the cost vector labelled. Remember the cost vector points in the direction of increasing cost. (Solution: 9 boxes of oranges, 5 boxes of pineapples, 105 gold pieces)



Below is the graph with the cost contours (in black). Each of the contours represent a line of equal cost (labelled next to it) and are perpendicular to the cost vector. Subsequently, we can also use the contour lines to see that the one with cost = -105 is the last one to intersect the feasible region in the direction away from where the cost vector is pointing (since we want to minimize cost). The intersection point is (9,5) as we saw from the solution in the above image. Remember this cost is in terms of the standard form LP where we converted the maximization to a minimization. So, we must flip the resulting cost to be positive and get a result of 105 gold pieces.


c=-105
c=-96
(9, 5)
F
$\vec{c}$
5
0
5
10
c=0