1. The first step is to satisfy all of the hard constraints. We can meet most of them by assigning each class to a room and time. Per the first objective criteria, our assignment should aim to minimize the number of wasted seats. I would suggest formulating this as an integer linear program. If we list out all of the relevant variables:

$$\begin{aligned}
\text{classes} &= \{c_1, \ldots, c_i\} \\
\text{rooms} &= \{r_1, \ldots, r_j\} \\
\text{time slots} &= \{t_1, \ldots, t_k\} \\
\text{class enrollments} &= \{e_1, \ldots, e_i\} \\
\text{room capacities} &= \{r_1, \ldots, r_j\} \\
x_{crt} &= \begin{cases} 1 \text{ if class } c \text{ is assigned to room } r \text{ at time } t \\ 0 \text{ else} \end{cases}
\end{aligned}$$

This gives the objective function:

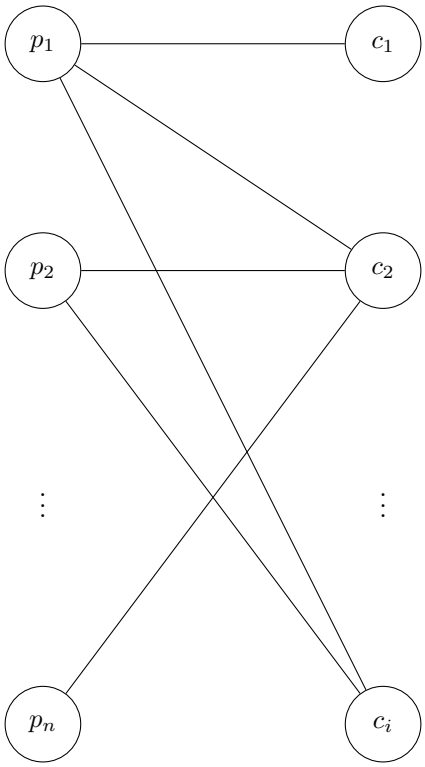
$$\begin{aligned}
\min \quad & \sum_{c,r,t} x_{crt}(f_r - e_c) \\
\text{s.t} \quad & x_{crt}(f_r - e_c) \geq 0 \quad \forall c, r, t && \text{(class enrollment} \leq \text{room capacity)} \\
& \sum_{c} x_{crt} \leq 1 \quad \forall r, t && \text{(two classes can't share the same room at the same time)} \\
& \sum_{r,t} x_{crt} = 1 \quad \forall c && \text{(every class has exactly one assigned room and time)}
\end{aligned}$$

whose solution is a valid schedule that maximizes room utility. There are likely Ruby gems that can solve these problems, so we should look around to see what our options are.

Concern: *This might cause trouble if the schedule puts a class with very few capable professors at an unfortunate time. If none of the available professors can work with the time, we're stuck. The ILP isn't aware of this, so this could be a potential pitfall.*

Idea: *A potential idea is to determine unfavorable times beforehand by looking at professor time preferences. We can assign each time slot a weight, which can be added to the objective function. This would ideally encourage the algorithm to avoid incurring high penalties associated with problematic times.*

2. Now we can match every class to a professor. We can view this as a bipartite matching:



However, we assume that $n \ll i$, since each professor is contracted to teach multiple classes. We usually require that both left and right sets have the same number of nodes in a bipartite match. To accomplish this, we duplicate each professor node, where the number of copies refers to the number of classes a professor will teach.

With the new graph, we draw a weighted edge $p \to c$ if professor p can teach class c. We determine the weights based on the professor's affinity for the course, where the higher the weight, the better the match. This scheduling problem turns into a max weight bipartite matching, which is a well-known problem we can find an algorithmic solution for.

Design Question: *How do we determine how many copies each professor has? Obviously, it must be between the minimum and maximum course load, but we'll probably need something more specific.*

Design Question: *How should we design our happiness function? The simplest would be a linear function with respect to class and time slot, though we could explore other options*