Employee Schedule Optimization

Kenneth Cross
Math 177
San Jose State University

December 8, 2015

1 The Problem

The problem to be optimized is and employee schedule. More specifically, who might be the best people to schedule in a restaurant on any given day? While this problem uses a restaurant as its main example, the solutions can be applied to any shift-based industry. Since the problem stems from the availability of people, the solution is not obvious, other things must be performed before a *integer linear optimization* model can be formed.

2 The Solution

First, we start by representing peoples availability in a matrix, showing which jobs they can perform and which days they can work. Since there are variables that suggest it would be more beneficial for one person to work on a given day then another, they are attached as the last columns. This gives us a matrix that can be optimized. And the goal is to maximize the sum of the last few columns which will later be reorganized leaving $x_1 + x_2 + x_3$. x_1 is whether this employee is a preferred employee, or rather, they earn more sales than others do. x_2 is whether this employee is scheduled for a position they prefer. x_3 is the consideration of the difficulty of the day. If the day is difficult, then a skilled employee should be favored.

For each day of the week, there is a matrix like this,

	Server	Host	Runner	Busser	Bar-back	Bar	Pref. Emp.	Pos. Pref.	Diff.
Ryo	0	0	0	0	0	4	1	6	4
Joji	0	0	0	0	0	3	0	6	3
Erinn	0	1	0	0	2	0	0	5	2
Masa	2	5	1	0	0	1	1	2	5
Mari	3	2	1	0	0	0	0	1	4
Dan	3	0	0	0	0	0	1	1	2
Jay	2	0	0	0	0	0	0	1	2
Machiko	2	0	1	0	0	0	1	1	2
Masami	3	0	0	0	0	0	1	1	3
Nana	3	0	1	0	0	0	0	1	3
Shiho	4	0	4	4	0	0	0	4	4
Saya	0	0	2	0	0	0	1	3	2
Stephanie	0	0	2	0	0	0	1	3	2
Tamiko	2	0	2	0	0	0	0	1	2
Tomoyo	0	0	0	3	0	0	0	3	3
Naoko	1	1	1	1	1	1	1	1	3

Since multiple people may work the same shift, a separate matrix that shows shift scheduling demands is also used for further analysis and approximations.

	Server	Host	Runner	Busser	Bar-back	Bar-Tender	Difficulty
Monday	2	1	1	1	0	1	0
Tues.	2	1	1	1	0	1	0
Wed.	2	1	1	1	0	1	0
Thur.	2	1	1	1	0	1	0
Fri.	3	1	1	0	1	1	1
Sat.	3	1	1	0	1	1	1
Sunday	2	1	1	1	0	1	1

Instead of solving for combinatorial output of every person that can work together in a given day, an approximation is done, where there is something like a scan line performed. It works by looking at all available shifts and people that can fill them, it assumes that each job can be filled by one person, then it

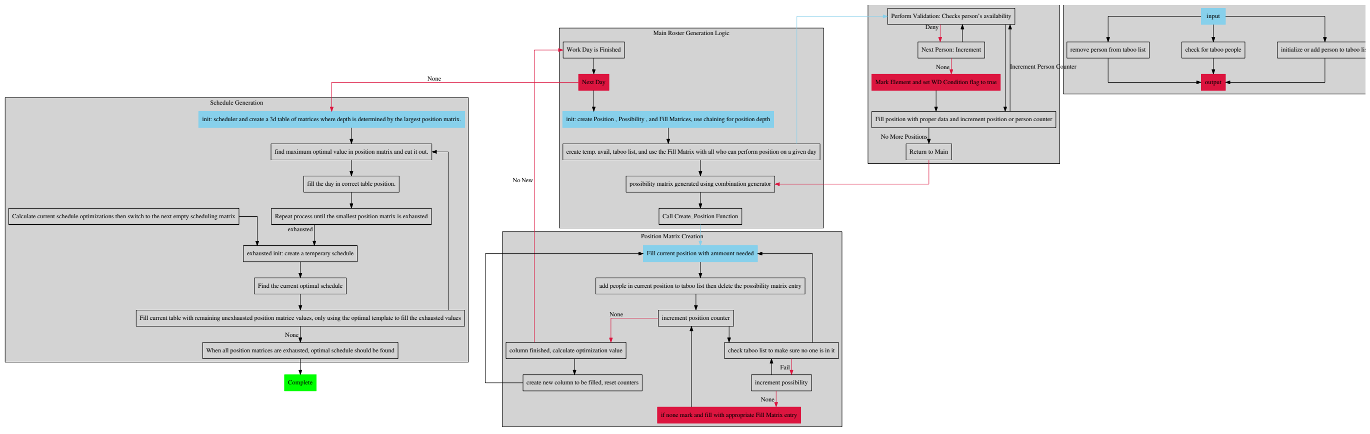
None
Schedule Generation
init: scheduler and create a 3d table of matrices where depth is determined by the largest position matrix.
find maximum optimal value in position matrix and cut it out.
fill the day in correct table position.
Calculate current schedule optimizations then switch to the next empty scheduling matrix
Repeat process until the smallest position matrix is exhausted
exhausted
exhausted init: create a temperary schedule
Find the current optimal schedule
Fill current table with remaining unexhausted position matrice values, only using the optimal template to fill the exhausted values
None
When all position matrices are exhausted, optimal schedule should be found
Complete
Main Roster Generation Logic
Work Day is Finished
Next Day
init: create Position , Possibility , and Fill Matrices, use chaining for position depth
create temp. avail, taboo list, and use the Fill Matrix with all who can perform position on a given day
No New
possibility matrix generated using combination generator
Call Create_Position Function
Position Matrix Creation
Fill current position with ammount needed
add people in current position to taboo list then delete the possibility matrix entry
None
increment position counter
column finished, calculate optimization value
check taboo list to make sure no one is in it
Fail
create new column to be filled, reset counters
increment possibility
None
if none mark and fill with appropriate Fill Matrix entry
Perform Validation: Checks person's availability
Deny
Next Person: Increment
None
Increment Person Counter
Mark Element and set WD Condition flag to true
Fill position with proper data and increment position or person counter
No More Positions
Return to Main
input
remove person from taboo list
check for taboo people
initialize or add person to taboo li
output

Figure 1: Algorithm Digraph

will calculate the optimal solution for the people who can fill it and append the result to the corresponding columns until all the jobs are filled.

3 Methodology

1) Staff Generation: Stored in a possibility matrix
2) Staff Optimization
3) Schedule Based on Staff Optimization Sum
4) Schedule Satisfaction Optimization

4 Modules and Functions

The entire depth of the inner workings of the program/algorithm are outlined here. The subsections try to follow a logical working order of pragmatic flow.

4.1 Rostering Optimizations

There are currently four values considered for the final roster optimizer value. The day difficulty, outstanding employee, position preference, and the employee day preference values and their relationship are what determine the optimizer value. Any undesirable relationships and traits are given zero points or subtract points, as optimizer values can be negative. Day preferences are given on the availability grid with a value of zero through five, where zero indicates the impossibility of work and five is the most highly desired day to work. The position preference value currently only determines the preference of one position, and it is a number, the position index to which they prefer. The other two values are boolean, which is easy to sum.

All positive relationships and values are given one point and summed, except all the points on day preferences are added instead of a single point.

The relationships needed to accrue points are the following: 1) Outstanding employee and difficult day 2) Difficult day with no outstanding employee in preferred position, then a point is subtracted.

The entire roster is sorted from highest to lowest by optimizer value then given to the scheduler function.

4.2 Multi-Shift Support

This should be taken care of in the scheduler algorithm rather than the rostering algorithm. This makes sense because rostering will calculate all possibilities of a person's availability rather than overlook other possibilities for other shifts. The scheduler should have a section to combine and compare full shift schedules where logic should be performed for optimizations and shifting conflicts or other legal parameters.

5 Data Representations

Incoming input is currently contained in two separate CSV files. One file contains employee availability where the position section is a preference as to how many times per week someone would like to perform that position. The other file contains what types of positions are required and how many of each for every day of the week. The CSV files are then read into python where they are stored as integer matrices. All of the following operations are performed in python, but the actual program has not been written to completion.