# A Utilitarian Approach to the Employee Staffing Problem

**by John C  Lawrence**

**j.c.lawrence@cox.net**

**June 20, 2025**

## Abstract

Enterprises such as hospitals and large businesses such as Walmart and Amazon must make sure they have the required number of employees on duty 24 hours a day although that number may vary for different time slots throughout the day. This also holds for worker-owned cooperative enterprises for which the profit motive takes second place to employee satisfaction. For our purposes this scheduling problem requires filling all the time slots with employees in a fair way taking into account the wishes and desires of the employees themselves. A utilitarian approach assumes that the employees will provide their inputs in terms of their preferred schedules as well as their preferences regarding compensation. The solution would maximize employees' collective utility while insuring that all shifts are covered, and that this is done within the total budget of the enterprise. Computational complexity has been hitherto so demanding that solutions have been difficult to attain. Presently, powerful computer chips which have been used for AI have also provided the possibility of brute force algorithmic solutions to computationally complex problems such as this one.

## Introduction

The problem is filling all the time slots in an enterprise with employees in a fair way taking into account the preferences of the employees themselves. The USDN website states: "The Canadian Worker Coop Federation describes the goals of the coop governance structure as "...service to its employees and its community rather than in service to the owners of capital. The goal is to provide the best possible employment conditions for the members and to provide the customers and community with a service or product at a fair price that meets their needs and leads to a sustainable community." Nurse self scheduling (Falcone, 2023) has been responsible for better work life balance and various other benefits for the employees. According to Noelle Forseth (2024): "researchers found out that 87% of nurses consider self scheduling the main solution for creating a flexible work environment at healthcare facilities." Self scheduling for other types of employees should have the same salutary benefits. For instance, some employees might prefer to work only days for less pay per hour and some might prefer to work nights at a higher pay per hour. In our model we consider that an employee can submit a number of programs, each program covering one week and indicating their preferences for time slots and compensation levels for that week. We consider a basic time slot of 4 hours so that a program consists of a specification of which 4 hour time slots over a week's period that an employee

would like to work. They can combine these four hour time slots in any way they wish including consecutively. We eliminate the consideration of "official" designations such as part time, full time and overtime We assume that each program can have a time slot utility (Hillinger, 2005; Smith, 2023) between 0 and 1 with "1" representing the highest utility and "0" indicating no utility, which means that the employee would absolutely not want this time slot. Each program can also have a range of compensation level specifications with utilities between 0 and 1. An employee would not be required to work for a compensation with a utility equal to zero nor in a program not on their list. Therefore, an employee would not be required to work in a program or for a compensation of utility 0. Management would set a maximum and a minimum compensation level. Employees can specify a range of compensation levels with associated utilities within those limits. Time slot utilities are considered to be independent from compensation utilities. If a specific program were very desirable, an employee might assign it a time slot utility of one, the highest utility. For each time slot program there may be associated a range of compensation levels with associated utilities.

Compensation utilities may be different for different programs. Employees' highest and lowest submitted compensations might not be the same as the maximum or minimum compensations possible within the system. In particular the lower the compensation asked for at a utility level of 1, the more likely it would be that a particular employee would be assigned that program, and it is assumed that they would not be assigned a program with a compensation less than the one associated with a utility of zero. The specification of a higher time slot utility and lower compensation utilities might give an employee a better chance to gain a more popular program preference. Conversely, specification of a higher time slot utility along with higher compensation utilities might give an employee a good chance to gain a less popular time slot preference at higher pay. Each employee's total utility would be the sum of the time slot and compensation level utilities for their assigned program. The social utility would be the summation of both utilities over all employees. There are more sophisticated ways to process the utility information other than simple summation (Lawrence, 2023), and also to guarantee that all employees have at least a minimum amount of utility (Lawrence, 2024), but they will not be considered here. The idea is to maximize social utility - in this case the combined utilities of all the employees - within the total budget and providing all time slots are filled exactly as necessary. These two constraints - total compensation over all employees and the specification of the exact number of employees to fill every time slot - are determined by enterprise management. The

problem for the social choice is to take in all this information and come up with an assignment of employees to time slots and compensation levels in such a way as to meet the requirements of total budget and staffing requirements of the enterprise. After the computations each employee would know their compensation level and time slot assignments for a particular weekly period.

The first step is to catalog all sets of combinations of employee's programs that meet the enterprise's requirements in terms of covering all time slots with the exact number of employees. There may be several members of this set each with its unique social utility. We assume that there is at least one. Next for each member of this set we find all combinations of compensation levels with associated utilities such that the maximum budget is not exceeded. The sum of these two utilities which maximizes the total or social utility is the one which determines the assignment of time slot programs and compensation levels for each employee for the week. An algorithm for accomplishing the special case of time slot utility maximization is presented in Appendix A. An algorithm for maximizing utility for compensation preferences would be similar, but is not considered here. Due to the extremely high number of calculations per second of the most advanced AI chips, brute force algorithmic methods for providing solutions are entirely feasible whereas previously the solutions of computational problems as complex as this one were not deemed practical.

Enterprises must insure that employee staffing is sufficient at all times although staffing needs may vary by time slot within a 24 hour day and a 7 day week. At the same time this must be accomplished in such a way that the total budget is not exceeded. While for profit enterprises will attempt to minimize the total staffing budget, a utilitarian approach accepts the total budget as a constraint and then attempts to maximize the employees' total utility both for time slot and compensation assignments while not exceeding the enterprise's total budget. Each employee submits a number of possible programs consisting of preferred time slot specifications along with associated utilities as well as preferred compensation levels with associated utilities for each time slot program. The sum of time slot and compensation level utilities over all employees equals the total utility for each week. A utilitarian solution attempts to maximize this value.

## The Data Structures

The employees are each assigned numbers in an array such as employee[i]. Employees'

names can be held in this array in alphabetical order i.e employee[1] = abbot, employee[2] = baker etc. Each employee would submit the parameters which would represent their preferences for time slots and compensation levels. We call such a submission a program. In our model an employee can submit as many programs as they want each with different utilities for time slots and compensations.

The enterprise administration would set the template which consists of the required number of employees for each time slot. They would also set the total budget. Within those two constraints, there are many possible time slot and compensation level assignments for each employee. A further assumption is that each employee has to be assigned at least one of their submitted programs every week, and that there is at least one set of employees' combined time slot programs which satisfies these constraints.

We assume that the time slots are split into 4 hour segments. For example, from 8 AM to 12 PM, 12 PM to 4 PM, 4 PM to 8 PM, 8 PM to 12 AM, 12 AM to 4 AM, and 4 AM to 8 AM. There are 6 four hour time slots per day and 42 per week. We number these from 1 to 42 starting with the 8 AM to 12 PM time slot on Monday. Time slots are numbered sequentially up to time slot 42 which would be midnight to 8 AM the following Sunday. The graphical user interface could simply show a series of squares representing the different time slots, and the employee would click on the squares they would like to work according to each program they submit. This data is stored in an array consisting of 42 computer bits. The template would specify how many employees are needed for each time slot. If this array were to be denoted as temp[q], where q varies from 1 to 42, then, for instance, temp[10] = 4 signifies that 4 employees are needed for the time slot from 8 PM to 12 AM on Tuesday. This defines the template in terms of staffing requirements.

A program consists of a series of time slots the employee would like to work. For instance, program 1 for employee 1 might consist of time slots 3,4,5, 8,9,10, 24,25,30. This comes to a total of 36 hours for the week. Program 2 for employee 1 might consist of the following time slots: 8,9,10, 30, 31, 40, 41, 42. This is a 32 hour week. A employee can submit as many programs as they want with associated utilities, uts[i][j], for each program where the index i designates the employee and the index j designates the program. Let's assume that the utilities are in the following set:
uts[i][j] = {0.0, 0.1, ... , 1.0}. Also a employee can't be assigned a program that has less utility than 0.1

Each employee's interaction with the graphical user interface would consist of their being queried by the GUI to submit their program preferences. First they would be asked to submit their time slot preferences, followed by the time slot utility for their first program. Then they would be asked for the same information for their next program and so on until every program that they wanted to be considered for had been entered. For each program they would be asked to submit their compensation preferences with associated utilities.

The employees' GUI might look like the following. They would just click on the appropriate squares:

| | | | | | |
|---|---|---|---|---|---|
| Monday<br>12 AM - 4 AM | Monday<br>4 AM - 8 AM | Monday<br>8 AM - 12 PM | Monday<br>12 PM - 4 PM | Monday<br>4 PM - 8 PM | Monday<br>8 PM - 12 AM |
| Tuesday<br>12 AM - 4 AM | Tuesday<br>4 AM - 8 AM | Tuesday<br>8 AM - 12 PM | Tuesday<br>12 PM - 4 PM | Tuesday<br>4 PM - 8 PM | Tuesday<br>8 PM - 12 AM |
| Wednesday<br>12 AM - 4 AM | Wednesday<br>4 AM - 8 AM | Wednesday<br>8 AM - 12 PM | Wednesday<br>12 PM - 4 PM | Wednesday<br>4 PM - 8 PM | Wednesday<br>8 PM - 12 AM |
| Thursday<br>12 AM - 4 AM | Thursday<br>4 AM - 8 AM | Thursday<br>8 AM - 12 PM | Thursday<br>12 PM - 4 PM | Thursday<br>4 PM - 8 PM | Thursday<br>8 PM - 12 AM |
| Friday<br>12 AM - 4 AM | Friday<br>4 AM - 8 AM | Friday<br>8 AM - 12 PM | Friday<br>12 PM - 4 PM | Friday<br>4 PM - 8 PM | Friday<br>8 PM - 12 AM |
| Saturday<br>12 AM - 4 AM | Saturday<br>4 AM - 8 AM | Saturday<br>8 AM - 12 PM | Saturday<br>12 PM - 4 PM | Saturday<br>4 PM - 8 PM | Saturday<br>8 PM - 12 AM |
| Sunday<br>12 AM - 4 AM | Sunday<br>4 AM - 8 AM | Sunday<br>8 AM - 12 PM | Sunday<br>12 PM - 4 PM | Sunday<br>4 PM - 8 PM | Sunday<br>8 PM - 12 AM |

The corresponding data structure in the computer would consist of 42 bits where "1" indicates that a particular time slot is part of the current program and "0" indicates that a particular time slot is not part of the current program for a particular employee. The programs are stored sequentially for each employee. For instance employee x could submit a program such as the following: Monday though Friday 8 AM to 4 PM which results in a 40 hour week. The data structure would look like the following:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Another example might be a employee who wanted to work 8 AM to 12 PM Monday through Friday and then 8 PM Saturday to 8 AM Sunday for a total of 28 hours for the week. Presumably, they might be able to make a lot more money on their overnight shift than on their morning shifts. The data structure for this program would be the following:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |

## The Variables

The problem is to assign employees to programs and to determine corresponding compensations levels for each employee for the week. Following are the variables:

| | |
|---|---|
| numemp | **total number of available employees |
| i | **employee identification index |
| j | **program identification index |
| prg[i] | **current program decision for employee i |
| numprg[i] | **number of submitted programs of employee i |

| | |
|---|---|
| uts[i][j] | **time slot utility for employee i and program j - 0 <uts[i][j] < 1 |
| cmp[i][j][k] | **$k^{th}$ compensation level of program j for employee i |
| ucm[i][j][k] | **compensation level utilities for program j of employee i - 0 < ucm[i][j][k] < 1 |
| numcmp[i][j] | **number of compensation levels for program j of employee i |
| tset[q][i] | **set of combinations of programs that meet time slot requirements, one for **each employee |
| cset[q][i][k] | **set of compensations for each member of set q that meet total budget **requirements |
| ts[i][j][k] | **time slot information for employee i, program j, time slot k. ts[i][j][k] = {0,1} |

## The Algorithm

First we want to identify all the possible combinations of work programs in which all time slots are filled with the exact number of employees needed as specified by the template. It is assumed that there is at least one combination of employees' programs such that all time slots are exactly covered and that every employee is included in the final work assignments. If this were not the case, some adjustments might have to be made in employees' assigned programs. This will not be considered here.

All of the information regarding time slot and compensation requests must be combined in such a way that each employee is given their preferences as much as possible regarding the times that they work and the compensation they receive. This must be done subject to the constraints of total budget and staffing requirements 24 hours a day. We assume that there is a minimum and maximum compensation set by the enterprise, but an employee is free to specify their minimum and maximum acceptable compensations within those limits. The goal is to maximize the employees' collective social utility both with respect to time slot preferences and with regard to compensation. For our model there are separate utilities for time slot and compensation, and each employee's total individual utility is the sum of the two. We first find all those combinations of work assignments that satisfy time slot requirements. Then for those we find the ones that maximize utility for individual compensation assignments.

The algorithm proceeds as follows. First we identify the combined employees' programs that exactly fill the enterprise's requirements for staffing for the week. We check time requirements first to make sure all time slots are covered with the correct number of employees. The set of combined employees' programs which fulfill this requirement are saved in tset[q][i] with q being the set number and i being the employee number. Employees are considered in order. The programs of each employee are also considered in order so that we can specify, for instance, a particular time slot of a particular program of a particular employee in a 3-dimensional array e.g. ts[i][j][k] which represents the $k^{th}$ time slot of the $j^{th}$ program of the $i^{th}$ employee. For instance, ts[i][j][k] = {1,0} and $1 \leq k \leq 42$.

List all the programs of all employees in the following order. First list all the programs of employee 1. Second list all the programs of employee 2 and so on. This resembles a tree like structure as illustrated in Figure 1 where the employees represent the trunk, the programs represent the branches and compensation levels are represented by the twigs. There are three possibilities: at each step, for the employee's program under consideration and considering all previous employees in the tree, there will be for each time slot, an exact fill, an underfill or an overfill with respect to staffing requirements as determined by the template. Start with the first program of employee 1. Consider the first program of the next employee in numerical order and check that their combined requested time slots don't exceed staffing requirements for any time slot. If there is an overfill, consider the next program in order of the employee under consideration. If all their programs have been considered without any one of them causing an exact fill or an underfill, go back to the previous employee and consider their next program in order. If this program does not result in an overfill, proceed to the next employee in order and consider their programs in order starting with their first program until an exact fill or an underfill is found. Then go to the next employee in order. Proceed in this way going forward when all previous employee's programs in order have resulted in an exact fill or an underfill and proceeding backward when an overfill is found and all programs of the employee under consideration have been considered.

A record is kept of progress through the chain of employees and programs in order. This record can be kept in the form prg[i] where i indicates the employee number. For instance, prg[10] = 2 would mean that, as we proceed through the tree, the current program decision for the $10^{th}$ employee is 2 i.e. their

second submitted program. This may change if no path through the tree is found in which this program for this employee is included. It is assumed that all employees prior to employee 10 and including employee 10 have been assigned programs which do not overfill any time slots. At any stage when a program is found that underfills or exactly fills shift requirements, record that in prg[i] and proceed to the next employee. When all employees have been considered and the process is completed with an exact fill or an underfill, record the exact fills in tset[q][i] and eliminate the underfills. Then proceed to find other possible exact fills and/or underfills. Go to the next program of the last employee and see if this results in an exact fill. If it does, record this in tset[q][i] and consider the next program of the last employee. For instance, tset[3][10] = prg[i] = 2 would indicate that for the third acceptable set, the $10^{th}$ employee would be assigned their second program. Once all the last employee's programs have been considered, proceed backward by one employee and go to that employee's next program. If that program does not overfill any time slot, proceed forward again. At any point if an overfill is detected and all of the programs for the employee under consideration have been considered, proceed backward by one employee and consider their next program in sequence and then proceed forward to the next employee if this program does not overfill.

Proceed until either more shifts are exactly filled, underfilled or a shift is overfilled. For the last employee sequentially there will be ether an exact fill, an underfill or an overfill. If an exact fill, mark this as a possible solution in tset[q][i]. Proceed to the next program of the last employee in order to find all of their programs that result in exact fills, and result, therefore, in acceptable sets. When all of the last employee's programs have been considered, proceed backward and repeat the same process. Eventually, by proceeding backward we will get all the way back to employee 1. After all of their programs have been considered, the process will terminate, and all of the sets of programs for which there are exact fills will have been found. We designate the total number of these sets by the variable m. The combined utilities for each member of tset[q][i] can then be computed. For instance, the utility of set q equals the sum of uts[i][prg[i]] for $1 \leq i \leq$ numemp. These utilities will then be added to the compensation utilities for each acceptable time slot set, and the sum of the utilities for time slot and compensation assignments over all employees which is a maximum will determine the employees' assignments for the week.

The next step is to compute the compensation levels for each employee in tset[q][j] which result in total budgets within the constraint of total compensation as defined by the template. We proceed in a similar fashion as we did with the computations for the time slot part of the program. First consider the highest requested compensation of employee 1 in tset[1][j]. Then add the highest requested compensation of employee 2 etc.. Continue in this way until the maximum compensation level is exceeded. Then go back one employee and add in their next highest compensation level and then proceed forward. Proceed backward and forward in this tree like structure until an acceptable set of compensation levels is found which does not exceed the total budget. There may be several of these sets of acceptable compensation levels for each acceptable time slot program. Then do the same for tset[2][j] etc. The algorithm proceeds in a tree like structure of employees' compensation levels as shown in Figure 1. Continue until all acceptable combinations have been found and stored in cset[q][j][k]. Then compute the social utility of each member of this set which is the sum of individual utilities. The member of the set with maximum utility is not necessarily the one which most closely approaches the total budget constraint.

Finally, for each member of tset[q][j] choose that member of cset[q][j][k] whose social utility when added to the social utility of tset[q][j] is a maximum. This determines the individual assignments of time slot and compensation level programs for the week.

# The Tree Structure

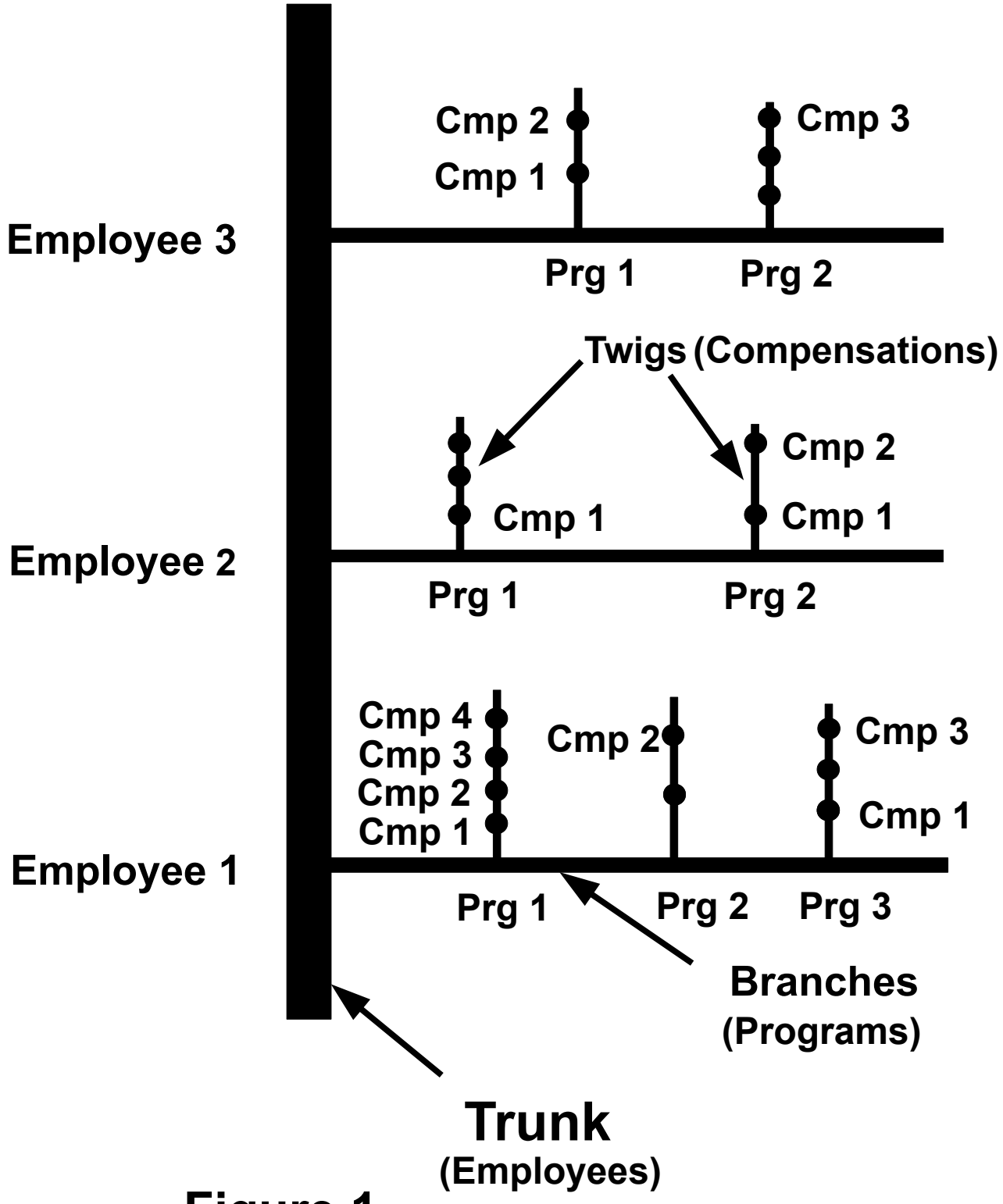


Figure 1

## Summary and Conclusions

We have considered a solution to the employee staffing problem which assigns employees to time slots and compensation levels according to the preferences of the employees themselves. Each employee submits their preferences for a number of programs they would like to be considered for where a program is a perfectly general selection of 4 hour time slots over a week's period, and also a specification of preferred compensation levels with associated utilities for each program. The utilities are chosen from the set {0, 0.1, ... , 1} where a utility of 1 means that that time slot or compensation level is most preferred. Time slot utilities are independent from compensation utilities. For all the acceptable combinations of employees' time slot programs which meet staffing requirements, we compute the total utility which is the sum of all the employees' individual utilities. We also compute for each acceptable set of time slot programs that set of individual compensation levels such that the budget set by management is not exceeded. The sum of total time slot utilities and total compensation utilities which results in a maximum then determines the assignment of programs for individual employees for the week.

Management sets the template which consists of the number of employees needed for each 4 hour time slot, and also the total budget for the week. A utilitarian solution comprises finding the set of employees' programs which fulfills the time slot and budget requirements in such a way as to maximize employees' total utility. The social utility is the sum of all the employees' individual utilities for the assigned programs. There is a separate social utility for time slot programs and for compensation levels. The assignment of programs maximizes the sum of time slot utilities plus compensation level utilities over all employees.

Cooperative and employee owned enterprises are interested in maximizing the satisfaction or utility of employees instead of making a profit for owners. These enterprises would be likely to profit from an analysis such as this one which purports to maximize employee satisfaction both in terms of the time slots worked and in terms of compensation.

Recently, the application of AI to nursing has been considered (Clancy, 2020) although primarily from the point of view of disease diagnosis and other clinical applications. For the purposes of this paper, we take advantage of the processing power of advanced AI chips, but use it for an algorithmic solution.

Brute force solutions for problems such as the one considered here have heretofore been considered intractable because of the large number of computations involved. For instance, consider n employees each having m programs. Their are mn possible paths through the tree of Figure 1 not even considering the twigs. With n = 100 and m = 4, this would compute to $3^{100} = 5.15\text{x}10^{47}$ calculations. According to The Verge (2024) "Nvidia says the new B200 GPU offers up to 20 *petaflops* of FP4 horsepower from its 208 billion transistors." Petaflop is a unit of computing speed equal to one thousand million million ($10^{15}$) floating point operations per second. FP4 means four bits of floating point precision per operation. However, for the problem considered here not every path through the tree needs to be followed. This makes it likely that chips such as the Nvidia B200 would be able to provide a solution in a reasonable amount of time. We do not ask the AI to come up with the algorithm for finding the solution to this problem. We provide an algorithmic programmed solution and then ask the AI chip to come up with the results in a reasonable amount of time.

## Appendix A

The following is a generic computer program which identifies the acceptable combination of employees' programs which maximizes time slot utility. We don't consider here the calculation of compensation levels. We assume that there is at least one acceptable set i.e. one set for which every employee is assigned a time slot program that exactly fills time slot requirements and meets budget requirements. The following is a bare bones computer program which uses generic language such as for statements, go to statements, if-then-else statements, continue statements and end statements and shows the logical flow without assigning data types etc. A basic knowledge of computer arrays is also necessary. Copious comments explain the logic.

numemp **total number of employees

i **number of current employee under consideration. $1 \leq i \leq$numemp

numprg[i] **total number of programs of employee i

j **number of current program under consideration. $1 \leq j \leq$ numprg[i]

prg[i] **$j^{th}$ program of employee i. Current program decision.

uts[i][j] **time slot utility of program j for employee i

k **time slot indicator $1 \leq k \leq 42$

ts[i][j][k] **time slot information for employee i, program j, time slot k. ts[i][j][k] = {1,0}

count[k] **running count of employees who want time slot k

max[k] **maximum number of employees needed for time slot k

tset[q][i] **q = number of an acceptable set of program decisions - one such that every
**employee is assigned a program and the combination of programs exactly fills
**all time slot requirements. Program decision for employee i is prg[i]

m **number of acceptable combinations of tset[q][i]. $q \leq m$

utstotal[q] **total time slot utility over all employees for $q^{th}$ acceptable set

```
for( k=1, 42)              **initializations
        count[k] = 0
```

```
end k
m=0                                   ** m is total number of acceptable sets
i = 1                                 **first employee
j = 1                                 ** first program of first employee

b:    continue

      for (k=1, 42)                   **Consider all time slots

        if (ts[i][j][k] = 1)          **a 1 means that that employee i wants time slot k
                                      **in program j. Otherwise, ts[i][j][k] = 0
          count[k] = count[k] + 1     **there are 3 cases: exact fill, underfill and overfill
                                      **if exact fill or underfill, keep going to employee
                                      **i+1, program j =1
                                      **if overfill, go to next program of current
                                      **employee i

          if (count[k] > max[k])      **an overfill. consider next program of current
                                      **employee i
              then
                 for(k1=1,k)
                    count[k] = count[k] - ts[i][j][k1]   **subtract off k count for current
                                                         **program which has been disqualified
                 end k1

                 go to c

              else

      end k                           **continue with for loop for current value of j

c:        j = j + 1                   **consider next program of current employee
          if ( j = numprg[i]+ 1)      **if all programs of current employee have been
                                      **considered
              go to d                 **go back to previous employee
```

```
                    else

                        go to b          **start for loop again for next program of current employee

                continue                 **continue with for loop if k ≤ 42
                                         **if k=42, for loop ended without overfilling
                                         **any time slot. time slots are underfilled or
                                         **exactly filled. Go to next employee

                end k                    **for loop ended without overfilling any time slot

                prg[i] = j               **current program decision for employee i

                if (i = numemp)          **all employees have been considered
                    go to e              **consider underfilled combination

                else
                    i = i + 1            **consider first program of next employee
                    j = 1

                go to b                          **go to first program of next employee:

d:              i = i - 1                        **go back to previous employee

                if (i  =  0)                     **all underfilled or exactly filled combinations have
                                                 **been found

                    go to e                      **consider underfilled combinations

                j = prg[i] + 1                   **go to next program of previous employee

                if (j = numprg[i] + 1)           **if yes, all programs of this employee have been
                                                 **considered

                        go to d                  **go back to previous employee in order
                else
                        go to b                  **start checking time slot availability with next
                                                 **program of this employee

e:              for (k = 1, 42)

                    if (count[k] = max[k)        **is time slot k exactly filled?
```

```
                continue
            else

                go to f          **there is an underfill. start with first employee again and
                                 **fill underfilled time slots

        end k                    **all time slots have been exactly filled

        go to g
```

For each value of k, the number of people who want a particular time slot, that is underfilled, find a program of an employee in order that adds to the sum of k values without overfilling any time slot. Check to see that the new program adds more to the k values than the current program decision. Then go to next employee and go through same sequence

```
f:      for (i=1, numemp)        see if next program of this employee adds to k count
                                 or not. check every value of k and count total
                                 number of k's over every time slot and compare with
                                 values of sum of k's from current program.

                                 if there is a net increase in k's choose this program
                                 temporarily. then check next program. Finally choose the
                                 program for this employee that results in the largest
                                 number of total k's without overfilling any time slot.

                                 prg[j] is current program decision for employee i

                               **if underfilled k, check difference in k values. if positive

                               **and total is positive without taking away any from those
                               **exactly filled or if there is a net gain in filling, choose next
                                **program. does this program overfill any time slots. if so
                               **go to next program. if not and if the time slot is not exactly
                                **filled and if (ts[i][j+1][k] - ts[i][j][k] > 0, add 1 to k

                               **count. If total k count over all time slots is
                               **positive, choose this program over original program


h:      for ( j = prg[i], numprg[i])   **do k count for every program after current program
        tot = 0                         **prg[i] is current program decision for employee i
        for (k=1, 42)
        if (count[k] < max[k])         **count[k] is total number of employees who want time slot k
          if (ts[i][j+1][k] - ts[i][j[k] > 0    ** for each employee, or each time slot, k can be 1 or 0
```

```
                                        **for employee i
        tot = tot + 1                   **tot is the total number of underfilled time slots over all
                                        **values of k
        else
            end k                       **employee j + 1 does not add to an underfilled time slot
                                        **for this value of k
      else
          end k                         **this time slot exactly filled (count[k] = max[k])
      if (tot > 0)                      **if  program (j+1) adds to underfilled time slots
        prg[i] = j                      **change program decision for this employee
      for (k=1, 42)                     ** for every time slot update k count information
        count[k] = count[k] - ts[i][j[k] + ts[i][j+1][k]
      end k
      end i                             **consider next employee
      else
      end i                             **consider next employee

  for (k=1, 42)
      if (count[k] = max[k]
      else                              **some time slots are underfilled
          go to f                       **start over with first employee
  end k

g:  continue                            **an acceptable combination has been found

   m = m + 1

   for (i = 1, numemp)                  **store each acceptable set

          tset[m][i] = prg[i]           **store each employee's program for this acceptable set

   end i                                **acceptable combination info stored in set[m][i]

   for (q=1, m)

        utstotal[q] = 0                 **computation of total time slot utilities. sum over
```

```
                                        **all nurses for time slot utilities for each
    end q                               **acceptable set of programs. An acceptable
                                        **program is one in which all time slots are exactly
                                        **filled and each nurse is included

    for (q=1, m)

        for ( i=1, numemp)

            j = set[q][i]               **set[q][i] = program decision for nurse i in
                                        **acceptable set q
            utstotal[q] = utstotal[q] + uts[i][j]       **utstotal equals sum of utilities for
                                                        **acceptable set q
         end i

        utstotal[q] = utstotal[q] / numemp      **normalized value of time slot utility
                                                **a value between zero and one.

    end q                               **time slot utilities for an acceptable set
                                        **have been computed

i = 1                                   ** find next set of acceptable combinations
j = prg[i] + 1

if (j = numprg[i] + 1)

    i = i + 1

    if (i = numemp)

       go to p                          **go to end

    else

        j = prg[i] + 1

    go to b

p: end                                  **all acceptable combinations have been
                                        **found
```

## References

1. Clancy, Thomas R. PhD, MBA, RN, FAAN. Artificial Intelligence and Nursing: The Future Is Now. JONA: The Journal of Nursing Administration 50(3):p 125-127, March 2020. | DOI: 10.1097/NNA.0000000000000855

2. Falcone, Sarah S. (2023) Why Self Scheduling is the Future of Nursing, ConnectRN blog, https://www.connectrn.com/blog/why-self-scheduling-is-the-future-of-nursing

3. Forseth, Noelle (2024) Everything You Need to Know About Nurse Self Scheduling. When I Work Blog, https://wheniwork.com/blog.

4. Hillinger, Claude (2005) The Case for Utilitarian Voting. Homo Oeconomicus 22(3).

5. Lawrence, John C (2023) Proving Social Choice Possible, Preprint: https://www.socialchoiceandbeyond2.com/proving.pdf.

6. Lawrence, John C (2024) Utilitarian Social Choice With a Minimax Provision, Preprint: https://www.socialchoiceandbeyond2.com/uscmm.pdf.

7. The Verge (2024), https://www.theverge.com/2024/3/18/24105157/nvidia-blackwell- gpu-b200-ai

8.Warren D. Smith (2023). "The case for score voting," Constitutional Political Economy, Springer, vol. 34(3), pages 297-309, September.

9. Worker-Owned Coops–The Cleveland Model, https://sustainableconsumption.usdn.org/initiatives-list/worker-owned-cooperatives-the-cleveland-model