A Utilitarian Approach to Welfare Economics

by John C Lawrence

j.c.lawrence@cox.net

j.c.lawrence@ alumni.stanford.edu

Orcid: 0000-0003-4698-8037

This is a PrePrint

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 such as the Mondragon Corporation 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' welfare in the sense of maximizing their collective satisfaction or utility with their jobs. Computational complexity has been hitherto so demanding that solutions have been difficult to attain although AI has made possible brute force algorithmic solutions to computationally complex problems such as this one in a timely manner.

Introduction

In 1951 Kenneth Arrow proposed the field of social choice. Any problem that involved individual inputs to a system which amalgamated those inputs into a social output was considered to be within the purview of social choice. His intention was that it would apply to both political and economic problems. Arrow states (1951: p. 11-12) “In the theory of consumer's choice each alternative would be a commodity bundle; ... in welfare economics, each alternative would be a distribution of commodities and labor requirements. … in the theory of elections, the alternatives are candidates.” In a companion paper we have dealt with the political implications with application to voting systems. This paper is concerned with the welfare economic implications. In a modern economy rather than "a distribution of commodities and labor requirements", it is more relevant to consider a distribution of monetary compensations and labor requirements.

Utilitarianism was founded by Jeremy Bentham (1789), an English philosopher and social reformer. Bentham defined as the "fundamental axiom" of his philosophy the principle that "it is the greatest happiness of the greatest number that is the measure of right and wrong." Utility is a synonym for happiness. Since in general it's not possible to maximize two variables simultaneously, the method described in this paper will demonstrate a mechanism which obtains just the greatest social utility regardless of the number.

Claude Hillinger (2005: pp. 295-321) has made the case for utilitarian social choice:

> "There is, however, another branch of collective choice theory, namely utilitarian collective choice, that, instead of fiddling with Arrow’s axioms, challenges the very framework within which those axioms are expressed. Arrow’s framework is ordinal in the sense that it assumes

> that only the information provided by individual orderings over the alternatives are relevant for the determination of a social ordering. Utilitarian collective choice assumes that individual preferences are given as cardinal numbers; social preference is defined as the sum of these numbers."

In this paper we are concerned with the welfare economics problem of assigning work schedules and compensation levels to employees of an enterprise in a fair way taking into account the preferences of the employees themselves. For enterprises such as the Canadian Worker Coop Federation this process is especially germane. Their website (2026) states: "Worker co-operatives are businesses that are owned and democratically controlled by the members." The goal is to provide the best possible employment conditions for the members. Mondragon Corporation (2026) is the world's largest federation of worker cooperatives, based in the Basque region of Spain since 1956. With over 100,000 employees and 120 cooperatives, it operates globally in industry, finance, retail, and knowledge. Mondragon achieves high employee satisfaction through democratic ownership, profit-sharing, and a "people-focused" mission, leading to higher productivity and lower turnover.

Nurse self scheduling (Falcone, 2023) has been responsible for better work life balance and various other benefits for nurses. 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 kinds of employees should have the same salutary benefits.

In our model we eliminate the consideration of "official" designations such as part time, full time and overtime in order to give employees complete control over their work schedules. 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. We assume that each program can have a time slot utility (Hillinger, 2005; Smith, 2023) which is a real number between 0 and 1 with 1 representing the highest utility and 0 indicating the least. Management would set the number of employees required for each 4 hour 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 in a time slot program or for a compensation of utility 0. Management would set the maximum and a minimum compensation levels. 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.

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 that employee would be assigned that program. The specification of a higher time slot utility and a lower compensation utility might give an employee a better chance to gain a more popular time slot program. Conversely, specification of a higher time slot utility and a higher compensation utility 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. 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 consider 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 and also to maximize the social utility of the employees themselves. After the computations each employee would know their compensation 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. 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. 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 involved in a solution to problems as complex as this, AI would probably be required although the programming logic is relatively straightforward.

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. 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.

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 and compensation 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 for employees would 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 for each employee. 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 possible utilities are in the following set: uts[i][j] = {0.0, 0.1, ... , 1.0}. 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 also 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 12 AM - 4 AM	Monday 4 AM - 8 AM	Monday 8 AM - 12 PM	Monday 12 PM - 4 PM	Monday 4 PM - 8 PM	Monday 8 PM - 12 AM
Tuesday 12 AM - 4 AM	Tuesday 4 AM - 8 AM	Tuesday 8 AM - 12 PM	Tuesday 12 PM - 4 PM	Tuesday 4 PM - 8 PM	Tuesday 8 PM - 12 AM
Wednesday 12 AM - 4 AM	Wednesday 4 AM - 8 AM	Wednesday 8 AM - 12 PM	Wednesday 12 PM - 4 PM	Wednesday 4 PM - 8 PM	Wednesday 8 PM - 12 AM
Thursday 12 AM - 4 AM	Thursday 4 AM - 8 AM	Thursday 8 AM - 12 PM	Thursday 12 PM - 4 PM	Thursday 4 PM - 8 PM	Thursday 8 PM - 12 AM
Friday 12 AM - 4 AM	Friday 4 AM - 8 AM	Friday 8 AM - 12 PM	Friday 12 PM - 4 PM	Friday 4 PM - 8 PM	Friday 8 PM - 12 AM
Saturday 12 AM - 4 AM	Saturday 4 AM - 8 AM	Saturday 8 AM - 12 PM	Saturday 12 PM - 4 PM	Saturday 4 PM - 8 PM	Saturday 8 PM - 12 AM
Sunday 12 AM - 4 AM	Sunday 4 AM - 8 AM	Sunday 8 AM - 12 PM	Sunday 12 PM - 4 PM	Sunday 4 PM - 8 PM	Sunday 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: 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. The data structure would look like 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	1	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**employee identification index
i	**employee identification index
j	**program identification index
m	**acceptable set index. an acceptable set is one in which each employee is **assigned a time slot program and compensation for the week such that all time **slots are filled with the exact number of employees needed and total **compensation is within budget. there may be more then one acceptable set.
q	**acceptable set index for compensation programs. For each acceptable set of **time slot programs, there may be multiple sets of compensation programs
ts[i][j][k]	**time slot information for employee i, program j, time slot k. ts[i][j][k] = {0,1}
prg[i]	**current program decision for employee i
numprg[i]	**number of submitted programs of employee i
uts[m][i][j]	**time slot utility for employee i and program j – $0 \leq$ uts[m][i][j] ≤ 1 **in acceptable set m
cmp[m][i][j][k]	**k^{th} compensation level of program j for employee i in set m
ucm[q][i][j][k]	**compensation level utilities for program j of employee i in set q - **$0 \leq$ ucm[q][i][j][k] ≤ 1
numcmp[i][j]	**number of compensation levels for program j of employee i
tset[m][i]	**set of combinations of programs that meet time slot requirements, one for **each employee. $1 \leq i \leq$ numemp
cset[q][i][j][k]	**set of compensations for each member of set q that meet total budget **requirements

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 management template. 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. 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[m][i] with *m* being the set number and *i* being the employee number. For instance, tset[2][3] = 4 means that for the second acceptable set employee 3 is assigned their program 4. We can specify a particular submitted time slot of a particular program of a particular employee in a 3-dimensional array e.g. ts[i][j][k] = {0,1} which represents the k^{th} time slot of the j^{th} program of the i^{th} employee. For instance, ts[12][3][30] = thirtieth time slot of the third program of the twelfth employee.

First we list all the programs of all employees in the following order: all the programs of employee 1; 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 starting with their first program until an exact fill or an underfill is found. Then go to the next employee. 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 10th 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 complete with an exact fill or an underfill, record the exact fills in tset[m][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[10] = 2 would indicate that for the third acceptable set, the 10^{th} employee would be assigned their second program.

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. The combined utilities for each member of tset[m][i] can then be computed. For instance, the utility of set *m* equals the sum of uts[m][i][prg[i]] for $1 \leq i \leq$ numemp.
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[m][i] 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][i]. 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 the tree like structure of Figure 1 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][i] 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][i][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[m][i] choose that member of cset[q][i][j][k] whose social utility when added to the social utility of tset[m][i] is a maximum. This determines the individual assignments of time slot and compensation level programs for the week.

The Tree Structure

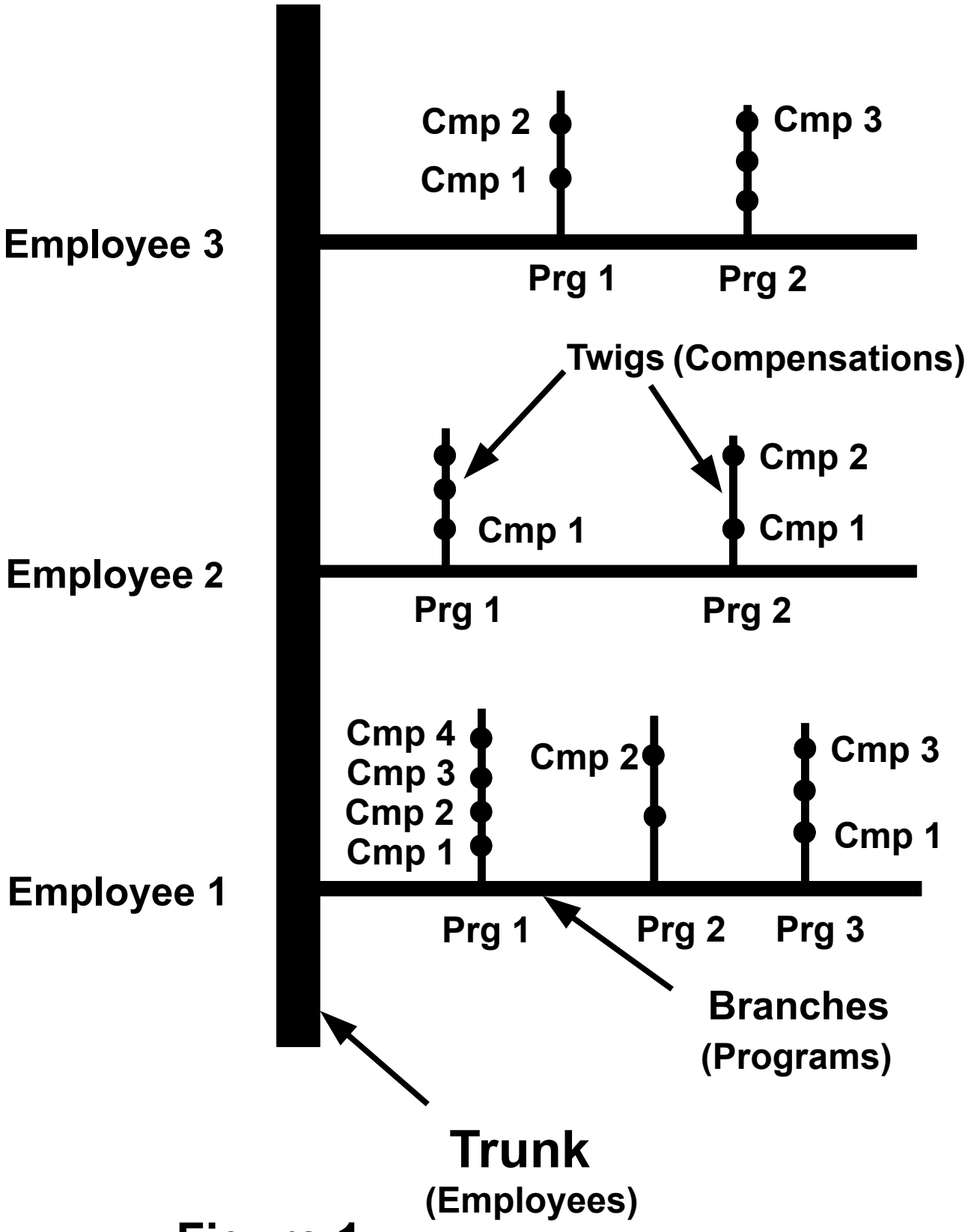

Figure 1

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 have considered 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. Utilities are associated with each time slot program and compensation level. 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 those sets 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 or social utility. The social utility is the sum of all the employees' individual utilities for the assigned programs. 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 seeks 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 time slot programs with each program having p compensation levels. There are n!/(m!)(n-m)! combinations of employees and time slot programs. Combining these with compensations levels, there are [n!/(m!)(n-m)!]/[p!{[n!/(m!)(n-m)!]-p}!] combinations so that there are this many possible paths through the tree of Figure 1.With n = 100, m = 4 and p = 4, this would compute to $1.066665066673999999 \times 10^{+25}$ calculations. The world's fastest supercomputer, El Capitan, performs at over 1.742 exaFLOPs, which translates to over 1.7 quintillion (1.7×10^{18}) calculations per second. So it would take about 4 months to solve this problem under the assumptions we have made. However, computer speeds are increasing rapidly, and it is possible that problems of this magnitude can be solved in the near future or that some short cuts in the algorithm are possible. 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 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]	**input 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[m][i]	**m = 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] i.e. prg[2] = 3 **means the (temporary) program decision for employee 2 is their third submitted **program
m	**total number of acceptable combinations of tset[m][i]

```
utstotal[m]       **total time slot utility over all employees for m^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

a:    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 b                  **go to next program of current employee
                else
      end k                                  **continue with for loop for current value of  j
```

```
                                **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. record program of current employee.
                                **go to next employee

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

      if (i = numemp)           **all employees have been considered
          go to e               **discard underfilled combinations. retain exactly filled
                                **combinations

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

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

b:    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 c             **go back to previous employee

          else

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

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

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

          go to g               **compute utility

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

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

            go to c             **go back to previous employee in order
```

```
        else
            go to a                          **start checking time slot availability with next
                                             **program of this employee

e:      for (k=1, 42)
          if (count[k] = max[k]
              continue
          else                               **some time slots are underfilled.
              go to d                        **discard this set. start over with next program of last
                                             **employee
        end k                                **all time slots have been exactly filled

f:      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]

            utstotal[q] = 0

            for ( i=1, numemp)
                utstotal[m] = utstotal[m] + uts[i][prg[i]]   **utstotal equals sum of utilities for
                                                             **acceptable set m
            end i

        go to d

        go to d                              ** find next set of acceptable combinations

g:      end                                  **all acceptable sets of combinations have been
                                             **found
```

References

1. Arrow, Kenneth J. (1951) *Social Choice and Individual Values*. New Haven: Yale University Press.

2. Bentham, J. (1789) An Introduction to the Principles of Morals and Legislation. Clarendon Press, Oxford. http://dx.doi.org/10.1093/oseo/instance.00077240

3. Canadian Worker Co-op Federation, https://canadianworker.coop/about/what-is-a-worker-co-op/

4. 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

5. 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

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

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

8. Mondragon Corporation. (2026) https://www.mondragon-corporation.com/en/

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